Cours Linux

version 1.1

Année Académique 2002-2003

Auteurs :
Frédéric BURLET,
Fabian BASTIN,
Christophe CHISOGNE,
Avec l'aide du NamurLUG

Table des matières

Remerciements

Mes remerciements aux gens du NamurLUG pour leur soutien lors de la rédaction de ce document. Plus particulièrement, à Fabian BASTIN pour la rédaction de l'introduction et pour ses remarques toujours aussi judicieuses, à Christophe CHISOGNE pour son expertise en `vi` et ses connaissances tant philosophiques qu' historiques, et à François SCHOUBBEN pour ses relectures et ses conseils en matière de structuration. Je voudrais remercier également tous les gens qui s'occupent de la rédaction des pages de manuel et info qui documentent le système linux et qui sont toujours une source d'informations inépuisable.

Chapitre 1

Introduction

1.1 À propos du cours

Ce cours sur GNU/Linux est organisé par le kot Linux, en collaboration avec le Namur LUG. Le kot Linux est un kot à projet des Facultés Universitaires Notre-Dame de la Paix de Namur (`http://www.fundp.ac.be`), qui a vu le jour début septembre 2000. Le Namur LUG[1] est un groupe d'utilisateurs de GNU/Linux à Namur, fondé dans le courant de l'année 1999, et sous l'influence duquel est née l'idée de créer un kot à projet consacré à GNU/Linux. Vous pouvez consulter le site web du LUG, à l'adresse `http://namurlug.org`.

Ce cours a pour objectif de vous initier à GNU/Linux. Il est destiné plutôt aux informaticiens (étudiants ou professionnels) qu'au grand public. En effet, GNU/Linux demande un investissement non négligeable en terme de formation pour profiter pleinement de sa puissance et de sa souplesse.

Dans un premier temps, nous vous présenterons brièvement la philosophie GNU/Linux et ses enjeux sous-jacents.

L'apprentissage du système GNU/Linux passe avant tout par une consultation de sa documentation intégrée, c'est pourquoi, nous vous donnerons les moyens d'accéder à toute cette documentation, avant d'entrer dans le vif du sujet.

Nous aborderons ensuite des aspects plus techniques, en expliquant, la description de quelques commandes utiles, suivies par quelques petites notions concernant l'interpréteur de commande **BASH**. Nous aborderons également la description de l'arborescence des répertoires du système GNU/Linux ainsi qu'un peu d'administration.

Il ne nous était évidemment pas possible d'expliciter chaque aspect de GNU/Linux.
GNU/Linux se développe tellement rapidement et est tellement vaste qu'un simple tutorial ne suffirait pas pour expliquer ce merveilleux OS en entier. D'ailleurs certains puristes risqueront d'être heurtés à la lecture de ce document. Nous nous en excusons d'avance. Le but du document est de fournir une introduction simple, sans entrer dans trop de technique. Par conséquent, plusieurs raccourcis ont été volontairement pris.

Si des questions persistent, n'hésitez surtout pas à les poser sur notre forum `http://namurlug.org/forum`.

[1] Namur Linux User Group

1.2 À propos de ce document

Signalons également que ce document a été rédigé pour servir de support aux formations données à l'Institut d'Informatique des Facultés de Namur. Il a été entièrement écrit en LaTeX. Nous avons également choisi de mettre ce document sous Licence OpenContent version 1.0 ou ultérieure. Cette licence vous donne la liberté de modifier ce document, de l'utiliser et de le redistribuer. Pour plus de détail concernant cette licence voir la copie en fin de syllabus ou encore visitez `http://www.opencontent.org/opl.shtml`. Vous pouvez avoir accés aux sources de ce document ainsi qu'une version compilée sur le site du Namur LUG (`http://namurlug.org/`), rubrique 'plat du jour'.

Le présent document n'a pas pour vocation d'être statique, mais d'être en évolution constante. Toutes remarques, suggestions, corrections peuvent être envoyées à Frédéric Burlet (`fred@namurlug.org`) ou à Fabian Bastin (`slashbin@namurlug.org`) ou encore à Christophe Chisogne (`chichi@namurlug.org`).

1.3 Qu'est-ce que Linux ?

Depuis 1999 environ, on entend beaucoup parler de Linux, y compris en-dehors du monde des informaticiens. Mais qu'est-ce que c'est exactement, Linux ?

"Linux" est une marque déposée par Linus Benedict Torvalds, son créateur. Du point de vue technique, Linux n'est rien d'autre qu'un système d'exploitation, un "operating system" particulier. Nous utiliserons le terme abrégé "OS" pour nous référer aux systèmes d'exploitation. D'un point de vue non technique, Linux est un phénomène nouveau, parfois dérangeant, qui risque bien de révolutionner l'informatique.

Au départ, Linux était un OS de type Unix, mais devant tourner sur plateforme PC plutôt que sur des stations de travail. Aujourd'hui, Linux est devenu un OS complet, puissant, rapide, d'une stabilité et d'une robustesse qui n'ont rien à envier aux plus grands OS connus.

1.3.1 D'où vient Linux ?

Mais d'où provient donc Linux ? Pour le savoir, il faut se plonger dans le volet historique, en 1991.

Cette année-là, un petit étudiant finlandais étudiait les OS à l'université d'Helsinki, en se basant sur Minix, un système d'exploitation pédagogique, forme dérivée et simplifiée des Unix commerciaux, conçu par Andrew Tannenbaum. Minix était gratuit, et son code source, disponible.

Cet étudiant s'appelait Linus Benedict Torvalds. C'était un informaticien brillant et curieux. Ses parents venaient de lui acheter le nouveau PC dernier-cri de l'époque, un splendide AT-386. Linus Torvalds n'était intéressé que par des systèmes de type Unix, mais ceux-ci était hors de prix. Le DOS prédominait sur les architectures PC, mais ne pouvait satisfaire les dessains en Linus Torvalds. Minix, écrit depuis 0 par Tannenbaum et tournant sur les processeurs 8088, était dès lors le seul point de départ valable. Torvalds se mit donc au travail en réécrivant et en étendant Minix pour les plateformes i386.

La même année, en 1991, un certain Tim Berners Lee, au CERN de Genève, inventait le World Wide Web, le désormais omniprésent WWW. Les deux événements ont convergé. Linus travaillait sur son système, qui allait devenir Linux. Mais il avait adopté une méthode de travail hors du commun : il publiait l'entièreté

de son travail (le code source de Linux) sur Internet. Plus fort encore, il invitait chacun à participer au développement de Linux !

Linux a donc été développé sur le net, par de nombreux informaticiens. Le premier noyau Linux stable est sorti en 1991.

1.4 Linux, le système d'exploitation

Linux est un système d'exploitation. Parmi ses caractéristiques :
– multitâche : on peut faire tourner plusieurs programmes en même temps,
– multi-plateformes : on le trouve sur tout type de machine,
– multi-utilisateurs : plusieurs utilisateurs peuvent se connecter en même temps,
– respect de la norme POSIX.
Il est en outre adaptable, ce qui explique qu'on le retrouve actuellement de plus en plus dans le monde de l'embarqué et des PDA.

1.4.1 Les systèmes d'exploitation

1.4.2 Linux, et les autres OS

Linux n'est évidemment qu'un OS parmi d'autres, certains étant orientés grand public (DOS, Windows 9x, Windows ME, BeOS, MacOS,...), d'autres ayant une vocation plus professionnelle (BSD, Windows NT, SunOS,...).

Linux ne se distingue guère des autres Unix, puisqu'il s'agit en fait d'un clone des Unix traditionnels. Néanmoins, le faible coût de revient et la grande portabilité de Linux lui ont accordé un succès qu'aucun Unix propriétaire n'avait connu jusqu'ici.

1.4.3 Linux, un OS viable ?

Une des grandes forces de Linux repose sur sa stabilité qui, dans le monde des serveurs, se mesure notamment par l'*uptime*, c'est-à-dire le temps avant lequel le serveur sera planté. Alors qu'il est couramment de l'ordre de 40 jours pour un système Windows NT, il n'est pas rare d'avoir un uptime oscillant entre 1 an et 2 an pour un système Linux.

S'il est fiable, Linux n'en est pas moins performant. Ainsi, Silicon Graphics commence à vendre des machines sous Linux, et tant l'US Air Force que la NASA, utilisateurs critiques s'il s'en faut, font confiance au système au pingouin.

1.5 Linux, plus qu'un OS

1.5.1 Logiciels tournant sous Linux

Contrairement à ce que certains pourraient penser, la logithèque sous Linux est très riche. La principale raison est la compatibilité Unix de Linux, permettant un portage facile de nombreuses applications. L'exemple le plus frappant est sans doute l'environnement graphique X-Window. Les secteurs les mieux représentés suivent dès lors le courant Unix, c'est pourquoi Linux est fécond en applications réseau, et est une plate-forme de développement de premier choix. Linux est également fort présent dans le monde scientifique, depuis les logiciels de composition comme LaTeX, aux outils mathématiques (Scilab, gnuplot,...), en passant par divers outils de simulation. L'utilisation de Linux par des organismes tels que la NASA illustre ce phénomène. La paternité Unix explique également la faiblesse de Linux dans les applications de type bureautique, traditionnellement développées pour des environnements plus populaires. Ce dernier point tend néanmoins à disparaître.

1.5.2 Linux vs GNU/Linux

Certains refusent le terme Linux s'il n'est pas accompagné de GNU. Derrière ce mot se cache le projet de la Free Software Foundation (`http://www.gnu.org`) de fournir un système informatique libre. Ce projet a été initié en 1984, année de fondation de la FSF. Le premier travail a consisté en la réécriture des utilitaires Unix de base, qu'il s'agisse de l'interpréteur de commandes ou du compilateur C. Linux a fourni le noyau, mais il est inexploitable seul. C'est de la réunion de ces différents travaux qu'est né le système que vous utilisez. L'inclusion du terme GNU a pour but de rappeler que Linux ne serait rien sans le travail de la FSF. Bien que nous adhérions à cet esprit, nous emploierons fréquemment le terme Linux au lieu de GNU/Linux, pour plus de facilité.

1.6 GNU/Linux, au-delà du technique

1.6.1 Logiciels libres, opensource, licences

L'inclusion des utilitaires GNU à Linux a eu pour conséquence d'incorporer la philosophie défendue par la FSF. Linux est issu d'un mode de développement ouvert, Linus Torvalds ayant soumis son code au net. La FSF a formalisé ce type d'approche dans la notion de logiciel libre, dont les idées sont les suivantes :
– liberté d'utiliser le logiciel ;
– accès au code source ;
– possibilité de modifier le code ;
– liberté de diffuser le programme, avec ou sans modifications.
Des variantes du concept de logiciel libre sont apparues, dont la plus importante est le mouvement opensource (`http://www.opensource.org`).

1.7 Pourquoi Linux ?

Choisir Linux ne relève pas du hasard ; en tant qu'OS, il a fait ses preuves sur le plan de la performance et de la stabilité. C'est un OS propice à recevoir des logiciels de grande qualité, lesquels sont d'ailleurs nombreux. Enfin, Linux est aussi un mode de pensée, une philosophie qui se démarque de la logique économique classique, pour faire place à l'ouverture et au partage de l'information.

La puissance de Linux a un prix : celle d'une complexité plus grande, nécessitant un effort certain. Heureusement, Linux jouit d'une documentation conséquente et de qualité.

1.8 Où chercher de l'aide ?

La règle de base sous Linux est un investissement personnel. L'accent dans le monde Linux est mis sur l'efficacité, au détriment du caractère intuitif. Pour tirer parti de Linux et des logiciels qu'il hébèrge, l'utilisateur sera amené à consulter par lui-même les documentations et manuels fournis.

L'utilisation optimale d'un système Linux, pour des informaticiens, requière bien plus de connaissances que pour d'autres systèmes. Plusieurs facteurs expliquent la nécessité imposée aux utilisateurs exigeants de bien se former.

Un premier facteur est le caractère hautement contrôlable, adaptable, paramétrable d'un système Linux : tous ses aspects sont susceptibles de faire l'objet de modifications. Cette possibilité est parfois fondamentale, voire critique, dans certains domaines. Mais un contrôle très fort exige de bien connaître un nombre éventuellement pharaonique de choix, d'options de configuration.

La maîtrise d'un système est proportionnelle au temps passé à le dompter. Sous Linux, la courbe d'apprentissage est assez lente au début, et s'accélère ensuite assez vite. Bien sûr, les utilisateurs travaillant sous Unix depuis des années ont un avantage énorme sur ceux qui viennent d'un monde non Unix.

Un autre facteur déterminant est la façon dont le système considère ses utilisateurs. Sous Linux, l'utilisateur est considéré comme intelligent et sachant ce qu'il fait, quoi qu'il demande (y compris la suppression du système). Les gardes-fous sont moins nombreux qu'ailleurs. L'utilisateur a donc largement intérêt à bien connaître son système, parce que ce dernier ne l'empêchera pas de faire tout et n'importe quoi.

En bref, un utilisateur de Linux est appelé à se former, et souvent à s'auto-former. C'est un des principes fondamentaux de l'utilisation de Linux : il faut chercher, gratter **soi-même** pour trouver les informations que l'on recherche. Les efforts sont récompensés par une connaissance croissante du système, qui peut aller jusqu'au code source du noyau, voire plus loin encore... Quel avantage, du point de vue "pédagogique" !

La formation, c'est bien joli. Encore faut-il savoir où trouver de l'aide. En effet, Linux est un système professionnel complexe. Devant l'immense étendue des possibilités, on peut vite s'y perdre. Voici quelques points de repère dans le paysage Linux. Quasiment toutes les questions que l'on peut se poser trouveront réponse dans un des endroits suivants.

1.8.1 Sur son disque local

L'endroit le plus rapide, c'est sur le système lui-même, à savoir sur un disque dur de sa machine.

La commande man

Une des commandes les plus souvent utilisées pour chercher de l'aide est la commande **man**. Man est l'abréviation de "manual". Les manuels informatiques sur papier étant volumineux et coûteux, ils sont fournis sous forme électronique. Dans le monde Linux/Unix, c'est la commande man qui sert à la consulter. La syntaxe est très simple. Parmi les formes les plus utilisées :

– `man xxx`
  Ceci affiche de l'aide à propos de la commande `xxx`, écran par écran.
– `man -a xxx`
  Même chose, mais cette fois la recherche ne s'arrête plus au premier "manuel" trouvé. C'est utile dans le cas où une commande fait partie de plusieurs manuels (comme "printf")
– `man n xxx`
  Le `n` indique le numéro de manuel où chercher la documentation.
– `man -t xxx | lpr`
  Avec cette commande, l'aide est convertie au format postscript et envoyée à l'imprimante par défaut (souvent une imprimante laser, pour une qualité irréprochable).
– `man man`
  Pour les plus curieux, cette commande affiche de l'aide sur la commande man.

Pour sortir d'une page man, il suffit d'appuyer sur la lettre `q`. Pour plus d'informations concernant cette commande reportez-vous au point 2.2 page 17.

La commande Info

Un autre format de documentation qui est très répandu est le format "info". C'est le format de base utilisé pour documenter la plupart des utilitaires de base du système. Souvent, la doc existe sous forme "man" et "info", mais la seconde forme fait foi en cas de litige, et la première peut être ancienne et non maintenue.

Le format info est un format texte. Il a la particularité d'autoriser une navigation (via des hyperliens) dans le document d'aide, et cela sous forme linéaire (comme un livre) ou hiérarchique (en arbre). La commande permettant de lire (et naviguer) dans les documents info est naturellement la commande **info**.

La commande info s'utilise comme la fonction man, dans la mesure où l'aide au format info sur une commande `xxx` est affichée par la commande

```
info xxx
```

Pour plus d'informations sur cette commande info, le réflexe de taper "man info" n'est guère satisfaisant. Mais comme la doc info est disponible au format info, il suffit de taper

```
info info
```

pour apprendre à l'utiliser : en appuyant ensuite sur la touche `h`, on arrive même sur un document info de type "tutoriel". Divers aspects de cette commande sont présentés à la section 2.2 page 18.

Les HOWTO, FAQ et autres /usr/doc

Une autre source extrêmement intéressante et instructive se trouve dans l'arborescence `usr/doc`. On y trouve pêle-mêle les `HOWTO`, `FAQ`, guides, documentations spécifiques à un logiciel.

Les `FAQ` sont les questions fréquemment posées (Frequently Asked Questions), accompagnées de leur réponses, et groupées par thèmes. Il est très utile (pour ne pas dire indispensable) de lire les `FAQ` avant de poser une question sur le web : si elle a déjà été posée 50 fois et qu'une réponse se trouve dans les `FAQ`, on court le risque de recevoir une réponse ne respectant pas toutes les règle de politesse.

Les `HOWTO` sont des documents concernant un sujet précis, et qui peut recouvrir plusieurs commandes. Il existe par exemple un `Sound-HOWTO` consacré au système audio de Linux. Les `HOWTO` sont rassemblés sur le site du LDP, le Linux Documentation Project (`http://www.linuxdoc.org`).

Il existe également quelques documents plus importants (de l'ordre de plus d'une centaine de pages). Ils abordent des thématiques plus larges encore, telles que l'administration système sous Linux. Ces documents, appelés **guides** en anglais, sont également disponibles sur le site du LDP. Ils se trouvent également sur le disque local des bonnes distributions Linux, même si la plupart ne les proposent hélas pas.

Pour des aides concernant un seul logiciel, une seule commande, une aide très spécifique est généralement disponible un sous-répertoire de `/usr/doc`. Ainsi, le document consacré au logiciel de boot `LILO` (pour LInux LOader), se trouve dans `/usr/doc/lilo/Manual.txt.gz`, ou quelque chose de très similaire.

Le code source

La documentation ultime d'un logiciel est bien entendu son code source. C'est la référence la plus fiable, car les docs ne sont pas forcément bien tenues à jour. Ainsi, il arrive que des options de ligne de commande soient renommées, ignorées ou supprimées, alors que la documentation y fait encore erronément allusion.

La documentation étant généralement le parent pauvre des programmeurs, il arrive même parfois qu'elle n'existe pas. Le code source est alors la seule référence. Sur les canaux très spécialisés, et se référant à la série culte de la guerre des étoiles, les internautes Linuxiens confirmés aiment à répondre un vibrant "Read The Source, Luke" aux questions pouvant être résolues de cette manière.

1.8.2 Sous forme non électronique

La documentation n'existe pas que sous forme électronique. Il existe de bons livres très bien faits. Nous vous recommandons les éditions O'Reilly, pour leurs nombreux ouvrages de qualité consacrés aux systèmes Unix/Linux.

Les être humains sont également une source d'information très appréciée. Après avoir pris connaissance de documents et n'avoir pas trouvé de réponse, il est sans doute possible de contacter un copain, une connaissance, voire même un "guru" local qui accepterait d'éclairer une lanterne.

Enfin, sachant que le support est un critère déterminant pour de nombreuses sociétés, il existe encore la possibilité de souscrire à une formule de support/help desk auprès d'une société. Les distributeurs Linux le font de plus en plus, comme Red Hat, Mandrake ou SuSE, qui proposent une aide par téléphone ou mail.

Bien entendu, ce support est souvent payant. Mais ce n'est pas choquant dans le monde du logiciel libre, au

contraire. Les **services** sont une des manières de gagner de l'argent grâce aux logiciels libres, en fournissant des éléments de valeur ajoutée. D'autres moyens sont la vente de documentation, de livres reliés, les formations, les certifications, etc.

1.8.3 Sur Internet

Linux est né en même temps qu'Internet, et s'est développé parallèlement. Les sites le concernant sont donc nombreux ; parmi les plus intéressants, citons :
– `linuxdoc.org` : le site de référence pour rechercher de la documentation ;
– `freshmeat.net` : site référençant les applications sous Linux ;
– `lealinux.free.fr` : des documentations diverses, en français ;
– `linuxprinting.org` : site consacré à l'impression sous Linux ;
– `linuxfr.org` : pour connaître les dernières nouvelles sous Linux, en français ;
– `traduc.org` : traduction de documentation ;
Nous vous invitons également à profiter des services offerts : mailing-lists, forums, IRC,... Vous y trouverez généralement les réponses à vos questions. Néanmoins, avant de vous y aventurer, apprenez les règles de base de la netiquette, sous peine de recevoir des répliques assez peu obligeantes. En particulier, évitez de poser une question dont la réponse se trouve déjà dans une `FAQ`.

Chapitre 2

Les commandes utiles

2.1 Un peu de théorie...

2.1.1 Notion de système de fichiers

Avant d'aborder les commandes utiles, nous allons expliquer les notions de système de fichiers, de fichiers et de répertoires parfois mal comprise par les débutants. Si vous maîtrisez déjà bien ces notions, vous pouvez passer cette section.

Lorque vous enregistrez des données sur un support (par exemple, le disque dur), elles sont rangées dans une structure que l'on appelle *système de fichiers*. Le *système de fichiers* est donc un objet sur le support permettant d'y structurer, d'y ranger des données. Le système de fichiers utilisé par Linux est l'**ext2**, sous Windows 9x le système de fichier est du type **FAT16** ou encore **FAT32**. Il existe plusieurs types de systèmes de fichiers : chacun étant caractèrisé par sa propre manière de ranger, d'organiser les fichiers sur un support. Avant de pouvoir mettre des données sur un support, vous devez au préalable le **formater**. **Formater** un support signifie le découper, le partager afin que l'ordinateur puisse y repérer sa position, savoir où il se trouve. Mais ce n'est pas parce que l'ordinateur sait se repérer qu'il sait comment il doit écrire les données sur le support, ni comment les ranger efficacement. C'est pourquoi après avoir formater le support on 'installe' un système de fichiers qui nous permettera de ranger et, également, d'accéder aux données. Les données que l'on écrit sur le disque ou que l'on range dans le système de fichiers sont appelées *fichiers*. On en distingue plusieurs types dont les *répertoires* et les *fichiers de données*. Les systèmes de fichiers permettent de ranger les fichiers généralement sous forme d'arbre et ce, de manière complexe. Un arbre (en informatique) peut être considéré comme un arbre habituel dont les intersections entre les branches s'appellent les *noeuds*, les feuilles des *feuilles* et les branches des *branches*. Dans un système de fichiers les noeuds sont les répertoires et les feuilles, des fichiers. Chaque noeud (répertoire) est père (contient) de feuilles (des fichiers). Les répertoires peuvent être considérés comme des tiroirs dans lesquels on range des fichiers. Par abus de langage – je devrai plutôt dire : par définition – les répertoires comme les fichiers de données sont appelés *fichiers*. Un répertoire est un fichier particulier possédant des caractèristiques propres. Par conséquent, quand nous utiliserons le terme *fichier*, nous parlerons des fichiers de données **et** des répertoires.

Notes : pour les gens qui connaissent déjà le système Zindow$ de Microsoft, nous pouvons dire qu'un répertoire est un dossier dans lequel on y range des fi chiers.

2.1.2 Quelques remarques préliminiaires

Quelques règles sont à respecter lorsque vous introduisez vos commande :

1. Séparer les différents mots par des espaces. Ainsi lors de la saisie de cd .. pour descendre dans l'arborescence de répertoire, il est impératif que vous mettiez un espace entre cd et ..
2. Le shell est sensible à la casse. Vérifier donc bien à tapez vos commandes en distinguant minuscules et majuscules.
3. Les autres mots faisant partie de la commande sont des *paramètres*. Un paramètre est une chaine de caractères passée à une commande dans le but d'obtenir un résultat spécifique. Ces derniers peuvent commencer par un tiret (-), on parle alors d'options, sinon, il s'agit d'arguments. On distingue deux types d'options : les *options courtes* et les *options longues*. Les options courtes sont les chaines de caractères commençant par un tiret (`-`) et les options longues sont celles qui commencent par un double tiret (`--`). Les options sont utilisées pour diriger le résultat de la requête. Les arguments quant à eux sont en général les noms de fichiers passés en paramètre de la commande. Généralement, les commandes admettent que l'on utilise plusieurs options et plusieurs arguments sur une même ligne. Signalons que les options courtes peuvent être regroupées entre elles. Par exemple, la commande **ls** que nous verrons plus tard admet entre autre les options `-a`, `-l` et `-h`. Ainsi au lieu d'écrire `ls -a -l -h`, le regroupement vous permet d'écrire `ls -alh`.

2.2 Plus d'aide !

Une bonne utilisation d'un système Linux passe tout d'abord par une bonne connaissance des commandes console. Certes, toutes les commandes ne seront pas décrites vu leur nombre. Nous vous présentons ci-dessous les commandes les plus utilisées avec les paramètres principaux. Toutefois, si cela ne vous satisfait pas, vous pouvez consulter de la documentation directement intégrée au système Linux par l'intermédiaire de l'outil d'aide *man* (pour *manual*). Nombre d'utilitaires de Linux se voient associer une ou plusieurs pages de documentation consultables directement via cette commande.

Sa syntaxe est :

```
man <options> <nom_de_commande>
```

man attend comme argument le nom de la commande pour laquelle vous désirez des explications. Par exemple :

```
$ man ps
```

vous donnera tous les renseignements concernant la commande *ps*. Il existe sous *man* des touches pour circuler à travers des pages de manuel ; ce sont les suivantes :

Touche	Effet
PgUP	Page précédente
PgDwn	Page suivante
Enter	Ligne suivante
Space	Page suivante
b	Page précédente
q	Quit

Á la commande *man* peut être également associé des options, nous en présentons un résumé :

-P ***pager*** : Spécifie un pager pour afficher la doc concernant la commande. Par défaut, *man* utilise **less** ;

-a : Par défaut, *man* quittera après avoir affiché la première page de manuel qu'il aura trouvée. En utilisant cette options vous forcez *man* à afficher toutes les pages de manuel qui matchent avec <nom_de_commande>, pas simplement la première ;

-h : Affiche un résumé des options de man. Pas de <nom_de_commande> à spécifier ;

-k ***string*** : Recherche après *string* dans TOUTES les pages de manuel. Pas de <nom_de_commande> à spécifier ;

-t : Fait de la page de manuel un fichier imprimable. Une redirection vers un fichier est nécessaire car par défaut le fichier résultat est affiché à l'écran ;

Chaque sujet est structuré de la même manière, nous distinguons de manière générale 8 parties :

NAME	Nom de la commande ;
SYNOPSIS	Reprend la syntaxe de la commande (sous forme de la syntaxe BNF en général) ;
DESCRIPTION	Explications condensées des résultats de la commande, servant plus d'aide mémoire que de descriptions précises ;
FILES	Indique les fichiers sur lesquels agit la commande.
DIAGNOSTICS	Explication des messages d'erreurs susceptibles d'être affichés par la commande ;
BUGS	Pour certaines commandes, des erreurs sont connues et répertoriées. Cela évite des surprises ;
EXAMPLES	Des exemples d'utilisation de la commande ;

Vous constaterez qu'au cours de l'utilisation de Linux certaines commandes ne possédent pas de manuel. Toutefois, vous pouvez généralement obtenir de la doc sur ces commandes en tapant en mode console le nom de la commande suivie de *--help* (parfois -h). Un résumé simple, clair et concis concernant la commande sera affiché à l'écran. Souvent, les explications fournies par les pages de manuel sont tellement condensées qu'elles ne sont plus claires. Toutefois, il existe une autre commande pouvant vous fournir de l'information. C'est la commande **info**. Sa syntaxe est la suivante :

```
info <nom_de_commande>
```

Cette commande donne en général des informations plus précises et, parfois, agrémentées d'exemples sur la commande dont vous cherchez des explications. Vous pouvez vous déplacez dans les pages avec les touches de direction. Passez aux pages suivantes avec la touche `n`, aux pages précédentes avec la touche `p` et enfin, vous pouvez quitter en utilisant la touche `q`.

Lors de la rédaction de ce document, nous avons veillé à respecter une syntaxe en ce qui concerne la définition des commandes. La syntaxe que nous avons choisie est un sous-ensemble de la syntaxe `BNF` (Backus Naus Form). Le tableau 2.1 présente un résumé du fonctionnement de la syntaxe.

TAB. 2.1 – La syntaxe BNF

Syntaxe	**Signification**
[x]	x optionnel
<quelquechose>	paramètres ou arguments ou options
x*	0, 1 ou plusieurs occurences de x
x+	1 ou plusieurs occurences de x

Les commandes décrites ci-dessous ne présentent aucunement la totalité des commandes mais une synthèse des commandes les plus utilisées avec les options les plus utilisées. Si vous êtes assoiffés de connaissance, n'hésitez pas à lire les pages **info** ou de manuel ;-)

2.3 Opérations sur les répertoires

2.3.1 Lister les fichiers d'un répertoire

Cette première commande est utilisée tant pour les fichiers de données que pour les répertoires. Pour lister les fichiers d'un répertoire, vous pouvez utiliser la commande **ls**. Sa syntaxe est la suivante :

```
ls [<options>] [<fichiers>]
```

Si le nom de `<fichiers>` n'est pas spécifié, la commande liste les fichiers du répertoire courant. Á la commande **ls** peut être également associé des options :

`-a` **ou** `--all` : Liste tous les fichiers, y compris ceux cachés (avec un nom commençant par `.`) ;

`-A` **ou** `--almost-all` : Identique à l'option `-a` mais omet les entrées `.` et `..` ;

`-B` : Ne liste pas les backups-fichiers (se terminant par ~) ;

`-c` : Trie par date. Agrémenté de l'option `-l` c'est plus clair ;

`-C` : Liste par colonne ;

`--color[=WHEN]` : Utilise des couleurs pour distinguer les fichiers. `WHEN` peut être égal à 'never', 'always', 'auto' ;

`-h` **ou** `--human-readable` : Affiche les tailles de fichiers en format lisible par l'homme (p.e., 1k, 234M, 2G) ;

-l : Affiche une liste long format ;

-m : Rempli la largeur de la fenêtre en affichant les fichiers en les séparant par une virgule ;

-F ou `--classify` : Ajoute un des symboles suivants après le nom de fichier :
- **@** : Pour les liens symboliques ;
- ***** : Pour les exécutables ;
- **/** : Pour les répertoires ;

-R : Liste les sous-répertoires récursivement ;

-s : Affiche la taille de chaque fichier (en blocks) ;

-x : Liste les fichiers par lignes au lieu de par colonnes ;

-1 : Liste un fichier par ligne ;

2.3.2 Créer un répertoire

Pour créer un répertoire, la commande à utiliser est la suivante :

```
mkdir <nom_de_repertoire>+
```

où `<nom_de_repertoire>+` est une liste non vide de noms de répertoire séparés par un ou des espace(s).

Exemple :

```
$ mkdir toto
$ mkdir lulu B212
```

2.3.3 Supprimer un répertoire

Pour supprimer un répertoire, la commande à utiliser est la suivante :

```
rmdir <nom_de_repertoire>+
```

où `<nom_de_repertoire>+` est une liste non vide de noms de répertoire séparés par un ou des espace(s). Si le répertoire est non vide la commande affiche un message d'erreur.

Exemple :

```
$ rmdir toto
$ rmdir lulu B212
```

2.3.4 Afficher le répertoire courant

Il existe une commande pour afficher au terminal le chemin du répertoire courant. la commande est la suivante :

```
pwd
```

Cette commande ne requiert ni options, ni arguments.

2.3.5 Changer de répertoire

Pour changer de répertoire, la commande à utiliser est la suivante :

```
cd [<nom_de_repertoire>]
```

Le `<nom_de_repertoire>` est ici optionnel. Dans le cas où le nom de répertoire est absent, la commande **cd** vous dirige vers votre répertoire utilisateur, sinon, dans le répertoire du chemin spécifié.

Des astuces :

`cd ..`	Permet de passer dans le répertoire parent
`cd -`	Revient dans le répertoire précédent le dernier changement de répertoire

Exemple :

```
$ cd /bin
$ pwd
/bin
$ cd
$ pwd
/home/fabian
$ cd - ; pwd
/bin
```

2.3.6 Espace occupé par un répertoire

La commande **ls** contrairement à la commande **dir** sous MS-DOS n'indique pas la taille des fichiers, ni celle des répertoires (sauf si l'option `-l` est spécifiée). Il existe une commande pour connaître l'espace disque occupé par un répertoire et ses fichiers. Cette commande est la commande **du**. Sa syntaxe est la suivante :

```
du <options>* <nom_de_rep_ou_de_fich>*
```

En fait la commande **du** à elle seule donne la taille du répertoire courant ainsi que des sous-répertoires de ce dernier et cela récursivement. La liste peut parfois être longue et illisible. Toutefois cette commande agrémentée de quelques options peut s'avérer bien utile :

`-a` ou `--all` : Affiche la taille pour tous les fichiers, pas simplement les répertoires ;

`-c` ou `--total` : Affiche un total de la taille de tous les arguments après que ces derniers aient été visité ;

`-h` ou `--human-readable` : Affiche les tailles des telles sortes qu'elles soient lisibles pour l'homme, c'est-à-dire, ajoute un M pour MégaByte, G pour GigaByte, k pour kiloByte (en puissance de 1024).

`--max-depth=DEPTH` : Montre le total pour chaque répertoire (et fichier si `--all`) qui sont situés à maximum DEPTH niveau(x) "en dessous" du répertoire courant ;

`-s` : Affiche simplement un total pour chaque argument ;

`-S` ou `--separate-dirs` : Donne la taille de chaque répertoire séparément, excluant la taille des sous-répertoires ;

`-X` *fichier* : Exclus les fichiers dont le nom correspond à ***fichier***.

Exemple :
Si vous voulez connaître la taille de chacun des répertoires et fichiers situés dans votre répertoire *home* :

```
$ du -a -h --max-depth=1 ~/
```

Note :
`~/` est un raccourci pour désigner votre répertoire *home*. Par exemple si vous êtes un utilisateur portant le nom *fabian*, `~/` désignera le répertoire `/home/fabian`. Le répertoire *home* est votre répertoire personnel, il contient votre profil (ensemble des fichiers de configuration des applications que vous utilisez), vos fichiers personnels, ... C'est également le répertoire par défaut dans lequel vous entrez quand vous vous logguez.

2.4 Opérations sur les fichiers

2.4.1 La copie de fichiers

La commande permettant la copie de fichiers est la commande **cp**. Sa syntaxe est la suivante :

```
cp <options>* <source>+ <destination>
```

Cette commande copie la source vers une destination qui peut-être un fichier, si la source est un fichier, un répertoire si la source est une liste de fichiers. Par défaut, cette commande écrase les fichiers de destination sans demande de confirmation. Á cette commande peut être associé des options :

`-a` ou `--archive` : Préserve tant que possible les attributs et la structure des fichiers originale ;

`-b` ou `--backup` : Fait un backup de chaque fichiers qui pourraient être écrasés ou déplacés ;

`-f` ou `--force` : Enlève les fichiers de destinations existants ;

`-i` ou `--interactive` : Mode interactif, demande si un fichier de destination existant doit être écrasé ou non. Il est fortement conseillé de mettre cette option dans les alias de votre *.bashrc* ;

`-R` ou `--recursive` : Copie les répertoires récursivement ;

`-u` ou `--update` : Ne pas copier les fichiers dont la destination existante est identique ou plus nouvelle que la source ;

Exemple-1 : Copier les fichiers *~/.bashrc* et *~/.bash_profile* dans le répertoire */tmp*

```
$ cp ~/.bashrc ~/.bash_profile /tmp
```

Exemple-2 : Copier le contenu du répertoire *~/.gnome* dans le répertoire */tmp*

```
$ cp ~/.gnome/ /tmp
```

Exemple-3 : Avec l'option `-R`, le répertoire et les sous-répertoires sont copiés récursivement

```
$ cp -R ~/.gnome/ /tmp
```

Exemple-4 : Copier un fichier sous un autre nom

```
$ cp .bashrc toto
```

Notez la différence entre l'exemple 2 et l'exemple 3.

2.4.2 Déplacement/Renomage de fichiers

La commande permettant le déplacement et/ou le renomage de fichiers est la commande **mv**. Sa syntaxe est la suivante :

```
mv <options>* <source>+ <destination>
```

Si le dernier argument de cette commande est un répertoire, alors **mv** déplace tous les fichiers listés avec le même nom dans ce répertoire. Sinon, si seulement deux noms fichiers (ou de répertoires) sont donnés, il renomme le premier par le second. Par défaut, la commande **mv** écrase le fichier destination sans demande de confirmation. Á cette commande peut être associé des options :

`-b` ou `--backup` : Fait un backup de chaque fichier qui pourrait être écrasé ou déplacé ;

`-f` ou `--force` : Enlève les fichiers de destination existants et n'averti jamais l'utilisateur ;

`-i` ou `--interactive` : Mode interactif, demande si un fichier de destination existant doit être écrasé ou non sans regarder aux permissions. Il est fortement conseillé de mettre cette option dans les alias de votre *.bashrc* ;

`-u` ou `--update` : Ne pas déplacer les fichiers dont la destination existante est identique ou plus nouvelle que la source ;

Exemple-1 : Renome le fichier *cours.html* en *chapitre1.html*

```
$ mv cours.html chapitre1.html
```

Exemple-2 : Déplace les fichiers *page1.html* et *page2.html* dans le répertoire *http-pub/*

```
$ mv page1.html page2.html http-pub/
```

2.4.3 Effacement de fichiers

La commande permettant l'effacement de fichiers est la commande **rm**. Sa syntaxe est la suivante :

```
rm <options>* <fichier>+
```

Cette commande efface les fichiers spécifiés en arguments. Par défaut, cette commande efface les fichiers sans demande de confirmation. Á cette commande peut-être associées des options :

`-d` ou `--directory` : Efface les répertoires, même si ils ne sont pas vides (en super-user seulement) ;

`-f` ou `--force` : Ignore les fichiers non existants et n'avertit pas l'utilisateur de la suppression ;

`-i` ou `--interactive` : Mode interactif, demande si un fichier existant doit être effacé ou non. Il est fortement conseillé de mettre cette option dans les alias de votre *.bashrc* ;

`-r` ou `-R` ou `--recursive` : Enlève le contenu des répertoires récursivement ;

Pour indiquer que les arguments qui suivent la commande ne sont pas des options, vous devez utiliser l'option `--`. Ainsi pour effacer un fichier dont le nom serait *-f* dans le répertoire courant, vous pouvez taper :

```
$ rm - -f
```

Exemple : Suppression avec demande de confirmation. Dans ce cas-ci on refuse puisque *.bashrc* est un de nos fichiers de configuration ;-)

```
$ rm -i .bashrc
rm : remove '.bashrc'? n
```

2.5 Commandes plus complexe

2.5.1 Comparer le contenu de deux fichiers

Pour comparer le contenu de deux fichiers, il existe la commande **diff**. Sa syntaxe est la suivante :

```
diff <options>* <de_fichier> <a_fichier>
```

Plusieurs options sont également disponibles. Pour plus d'informations concernant ces dernières, veuillez consulter les pages info ou exécuter **diff** avec l'option `--help`.

2.5.2 Concaténation de fichiers

Pour concaténer deux fichiers l'un à la suite de l'autre et afficher le résultat à l'écran il existe la commande **cat**. Sa syntaxe est la suivante :

```
cat <options> <fichier>+
```

Vous pouvez rediriger vers un fichier le résultat de la concaténation en utilisant une redirection (voir section 3.3). Ce programme accepte les options suivantes :

`-b` ou `--number-nonblank` : Les lignes de sorties non blanches sont numérotées ;

`-E` ou `--show-ends` : Affiche le caractère $ à la fin de chaque ligne ;

`-n` ou `--number` : Numérote toutes les lignes de sorties ;

`-T` ou `--show-tabs` : Affiche le caractère de tabulation comme ^I ;

`-v` : N'affiche pas les caractères non imprimables

2.5.3 Chercher un fichier

Il existe plusieurs méthodes pour trouver un fichier dans l'immense arborescence du sytème de fichier. Nous verrons dans cette section trois commandes : **find**, **which** et **type**

find

La commande **find** est très utile, elle offre plusieurs critères de recherche : type de fichier, le propriétaire, la taille, les permissions,...

Cette commande posséde la syntaxe suivante :

```
find <repertoire_de_depart> <option>* <action>*
```

find accepte les options suivantes :

-name <nom_de_fichier> : Tous les fichiers dont le nom correspond au `<nom_de_fichier>` sont sélectionnés ;

-iname <nom_de_fichier> : Idem que l'option précitée mais ignore la distinction entre majuscule et minuscule ;

-type <lettre> : Permet de sélectionner des fichiers ou des répertoires selon leur type. Le type est spécifié par la lettre *d* pour répertoire, *f* pour fichier et *l* pour lien symbolique ;

-size [+|-] <taille> : Limite la recherche aux fichiers d'une certaine taille. Un signe $-$ ou $+$ peut être utilisé en préfixe pour sélectionner les fichiers dont la taille est respectivement inférieure ou supérieure à la `<taille>` donnée (par défaut la taille est comptée en bloc de 512 octets). Pour plus de facilités vous pouvez préciser une unité de mesure pour la taille. Pour ce faire, vous faites suivre le chiffre par la lettre *k* pour des kilo-octets ou la lettre *c* pour des octets ;

-perm <mode> : Limite la recherche en fonction des permissions des fichiers.

-user <username> : Cette option permet de sélectionner des fichiers selon leur propriétaire ;

Une fois que **find** a trouvé les noms de fichiers correspondant aux critères de recherche, il est possible de définir des actions à exécuter pour chaque entrée. Ces actions sont utilisées comme des options :

-exec <commande> : Exécute une commande pour chaque fichier. La chaîne `{}` sera remplacée par le nom de fichier en cours. Notez qu'en *bash*, le point-virgule qui suit la commande doit être précédé d'un anti-slash (\) pour qu'il ne remarque pas la fin de la commande `find` elle-même.

-ok : Identique à l'option `-exec` mais demande confirmation avant d'exécuter chaque commande.

-print : Affiche le nom de chaque fichier. C'est l'action par défaut.

L'exemple d'utilisation ci-dessous recherche depuis le répertoire racine tous les fichiers dont le propriétaire est *fabian* et qui sont des fichiers PostScript et nous allons exécuter la commande `ls` sur chacun d'entre eux :

```
$ find / -type f -user fabian -iname '*.ps' -exec ls {} \; 2>
/dev/null
```

Nous faisons une redirection des erreurs vers un fichier spécial (`/dev/null`). Ce fichier réagit comme un trou noir quand on lui envoie des données ; il absorbe ces dernières dès leur réception. Nous faisons cette redirection à cause des permissions sur certains répertoires. En effet, certains d'entre eux ne sont pas accessibles par l'utilisateur et si cette redirection n'est pas faite, le fichier trouvé risque de passer sous nos yeux sans que l'on ne s'en aperçoive.

which

Montre le chemin d'accès à une commande. Ce programme se réfère à la variable d'environnement *PATH*. Si le fichier ne se trouve pas dans un des chemins définis dans *PATH*, alors cette commande renvoie un message d'erreur. Sa syntaxe est la suivante :

```
which <nom_de_programme>*
```

Exemple :

```
$ which ping ls
/bin/ping
/bin/ls
```

type

type est une commande interne au système qui permet de localiser des fichiers binaires. Sa syntaxe est la suivante :

```
type <nom_de_programme>*
```

Exemple :

```
$ type ls ping
ls is aliased to 'ls --color=always -FA'
ping is /bin/ping
```

2.5.4 Chercher une expression dans un fichier

La commande permettant de rechercher une expression dans un fichier est la commande **grep**. La syntaxe est la suivante :

```
grep <options>* <pattern> <fichier>*
```

La commande **grep** recherche toute occurence du `<pattern>` dans la liste de fichiers. Si la liste de fichiers est vide, la commande affiche toute occurence du pattern provenant de l'entrée standard. De ce fait, les deux commandes suivantes sont équivalentes[1] :

```
cat toto | grep 'lulu' ≡ grep 'lulu' toto
```

Le pattern à identifier doit impérativement se trouver entre cote(' '). La commande **grep** accepte les options suivantes :

`-A <NUM>` ou `--after-context=<NUM>` : Affiche `<NUM>` lignes de contexte après avoir affiché la ligne de correspondance ;

`-B <NUM>` ou `--before-context=<NUM>` : Affiche `<NUM>` lignes de contexte avant d'avoir affiché la ligne de correspondance ;

`-C` ou `--context[=<NUM>]` : Affiche `<NUM>` lignes (défault=2) de contexte ;

`-c` ou `--count` : Affiche le nombre d'occurences du pattern dans chaque fichier ;

`-i` ou `--ignore-case` : Ignore la casse dans le pattern et dans le fichier d'entrée ;

`-n` ou `--line-number` : Préfixe chaque ligne de sortie avec le numéro de la ligne du fichier d'entrée ;

[1] Voir chapitre 3 section 3.3.8 pour plus de détails

Exemple : Rechercher toutes les lignes qui ont au moins une occurence du pattern 'http' dans le fichier */etc/services*

```
$ grep -i 'http' /etc/services
www        80/tcp      http    # WorldWideWeb HTTP
https      443/tcp             # MCom
https      443/udp             # MCom
```

2.6 Les pagers

Un *pager* est un programme qui permet d'afficher à l'écran le contenu d'un fichier texte. Nous vous présentons dans cette section plusieurs *pagers*.

2.6.1 less

Le pager **less** est un des pagers les plus complets et complexes. Nous allons vous présenter dans cette section qu'une partie infime des fonctionnalités de **less**. Vous pouvez toutefois approfondir la question en lisant les pages de manuel concernant cette commande. Vous pouvez également obtenir une description complète de son utilisation en tapant `h` lors de son exécution. Sa syntaxe est la suivante :

```
less <options>* <nom_de_fichier>+
```

Nous ne parlerons pas des options ici. Notez que vous pouvez ouvrir plusieurs fichiers à la fois. Les touches pour se déplacer dans le document sont celles de la commande **man** puisque **man** utilise par défaut **less** pour afficher les pages de manuel (voir section 2.2 page 17). Si vous avez spécifié plusieurs fichiers à la fois, vous pouvez consulter le suivant en tapant `:n` et le précédent en tapant `:p`. Une autre fonction utile est la recherche de mots. Cette dernière peut être invoquée en tapant la touche `/` suivie du mot à rechercher. La touche `n` permet de trouver l'occurence suivante tandis que la touche `N` permet d'en trouver la précédente.

2.6.2 more

Le pager **more** posséde moins de fonctionnalité que **less**. Vous pouvez utiliser celles que nous avons décrites pour **less**. Pour plus d'information concernant cette commande vous pouvez consulter les pages de manuel ou bien taper `h` lors de son utilisation. Sa syntaxe est similaire à celle de **less**.

2.7 Accès à d'autres disques

2.7.1 Monter un système de fichier

Dans tout système Unix, chaque périphérique (ou nom de partition) est identifié par un fichier. Les disques durs, les lecteurs de disquettes et CD-ROM ne sont pas identifiés par `C:`, `A:` ou `D:`. Pour pouvoir accéder à un périphérique ou une partition, on dit qu'il faut monter (ou encore attacher) son système de fichier sur un point de montage (un répertoire) situé dans l'arborescence des répertoires. Typiquement, chaque périphérique ou nom de partition est identifié comme suit :

- /dev/fd0 pour une disquette ;
- /dev/cdrom qui est un lien symbolique vers un fichier de périphérique ;
- /dev/hd*b*`1` pour le second (*b*) disque dur (hd) première partition (`1`) ;
- /dev/sd0 pour un disque dur ou un autre périphérique de stockage SCSI.

La commande pour monter un système de fichier est la commande **mount**. Cette commande posséde la syntaxe suivante :

```
mount <options>* [<nom_de_partition>] <point_de_montage>
```

L'option la plus couramment utilisée est l'option `-t` qui permet de spécifier le type de système de fichier. Les différents systèmes de fichiers sont nombreux. Les plus courants sont `msdos`, `ext2`, `vfat`, `iso9660`, `autofs`, `smbfs`, ...

Remarque : `msdos` correspond à un système de fichier FAT16 uniquement avec des noms courts alors que le `vfat` gère le FAT16 et le FAT32 avec des noms longs.

Pour monter une disquette formatée msdos, la ligne de commande serait la suivante :

```
$ mount -t vfat /dev/fd0 /mnt/floppy
```

où `vfat` désigne un système de fichier utilisant les noms longs, `/dev/fd0` désigne le premier lecteur de disquette et `/mnt/floppy` l'endroit dans l'arborescence des répertoires où le système de fichier de la disquette doit être monté. Pour accéder au contenu de la disquette, il vous suffit donc d'aller dans `/mnt/floppy`. Vous pouvez y copier, renommer, déplacer, créer ou détruire des fichiers ou répertoires.

La possiblité de monter un système de fichier par un utilisateur dépend de la configuration du fichier */etc/fstab* (voir pages de manuel pour plus de détail).

2.7.2 Démonter un système de fichier

Après avoir monté un périphérique (ou une partition), il est IMPÉRATIF que vous le (ou la) démontiez (essentiellement pour les disquettes et CD-ROM avant de pouvoir lire le contenu d'un autre support). D'ailleurs, en ce qui concerne le CD-ROM, le bouton 'Eject' du lecteur est désactivé lors du montage. Vous ne pouvez pas sortir le CD tant que le lecteur n'est pas démonté.

La commande pour démonter un périphérique (ou une partition) est **umount**. La syntaxe de cette commande est la suivante :

```
umount <nom_de_partition>
```

Si vous obtenez un message comme quoi le périphérique est occupé, vérifiez que vous n'êtes pas simplement dans un des répertoires du périphérique monté.

Pour démonter une disquette quelque soit son format, la ligne de commande est la suivante :

```
$ umount /dev/fd0
```

2.7.3 Espace occupé d'une partition

La commande pour voir l'espace occupé (ou libre) sur chacune des partitions montées est **df**. Sa syntaxe est la suivante :

```
df <options>* <fichier>*
```

En fait, la commande **df** affiche l'espace disque disponible sur le système de fichier qui contient le nom de `<fichier>` (avec chemin complet) en argument. Si aucun nom de fichier est donné, l'espace disponible sur tous les systèmes de fichiers montés est affiché. L'espace par défaut est donné par blocs de 1k. Les options suivantes sont disponibles :

`--block-size=<SIZE>` : Utilise des blocs de taille `<SIZE>` ;

`-h` ou `--human readable` : Affiche les tailles en format lisible pour l'homme ;

`-l` ou `--local` : Limite le listing aux systèmes de fichiers locaux ;

`-t` ou `-type=<TYPE>` : Limite le listing aux systèmes de fichiers de type `<TYPE>` ;

Exemple : Déterminer l'espace occupé par chacune des partitions montées

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/hda1       2.6G  1.3G 1.2G 53% /
/dev/hda2       2.8G  1.5G 1.1G 57% /home
```

2.8 Divers

Quelques petites commandes supplémentaires :

Nom de commande	Description
`clear`	Efface l'écran ;
`set`	Défini une variable d'environnement ;
`stty`	Change et imprime les différents paramètres du mode console ;
`cal`	Affiche un calendrier du mois courant ;
`date`	Permet d'afficher et de modifier le date et l'heure de système ;
`reset`	Réinitialise l'affichage de la console ;

2.9 L'arborescence du système Linux

Quelque soit la distribution que vous installez, vous remarquez qu'il n'y a pas ou peu de différence entre les arborescences des systèmes Linux. En fait chaque distributeur de Linux essaie de respecter la norme **FHS** (*File Hierarchy Standard*). Nous allons vous présenter d'une manière générale à quoi correspond chacun des répertoires.

/ C’est le réperoire racine. C’est de lui que partent tous les autres répertoires, c’est la base du système de fichiers ;

/bin C’est ici que doivent se trouver tous les fichiers binaires utilisables par l’administrateur d’une part, et, par les utilisteurs d’autre part ;

/sbin C’est le répertoire qui contient les binaires réservés à l’administrateur ;

/etc Il contient les fichiers de configuration propres au sytème local ;

/boot Ce répertoire contient les fichiers statiques du chargeur de démarrage ;

/dev Il contient les fichiers de périphériques. Par exemple, nous trouverons */dev/psaux* pour les souris *ps2* ;

/home Il contient les fichiers personnels des utilisateurs. Ces fichiers sont triés selon une hiérarchie telle que */home/fabian* est le répertoire spécifique pour l’utilisateur *fabian* ;

/lib Ce répertoire contient les bibliothèques partagées essentielles au système lors du démarrage. Un sous-répertoire **modules/** contient les modules du noyau ;

/root Répertoire propre à l’administrateur système ;

/tmp Réperoire utilisé par des applications qui ont besoin de créer des fichiers temporaires ;

/var Ici figurent des données variables. On y trouve traces de logs, d’impressions, de mails,... ;

/mnt Répertoire contenant les points de montage d’autres systèmes de fichiers. Tout répertoire vide convient pour être un point de montage ;

/usr Répertoire où vont se placer les applications secondaires.

Chapitre 3

L'interprèteur de commande : BASH

3.1 Introduction

Sous Linux, vous avez le choix entre une multitude d'interpréteur de commande (*sh*, *csh*, *ksh*, *tcsh*, ...). Dans cette partie nous allons expliquer BASH (Bourne Again SHell) : l'interprèteur de commande sous Linux. L'invite de BASH sous Linux se présente généralement comme suit[1] :

```
[<nom_utilisateur>@<ordinateur> <répertoire>]$
```

ou

```
[root@<ordinateur> <répertoire>]#
```

Notez que l'invite peut être personnalisée.

Entre crochets figurent quelques informations sur l'état actuel du shell (ou invite BASH). Au début, vous trouvez le nom de l'utilisateur qui est ici soit `<nom_utilisateur>`, soit `root`. Après le caractère @, vous voyez le nom abrégé de l'ordinateur qui a été défini à la configuration du réseau : un nom du type *machine.quelque.part.be* deviendra *machine* simplement. Après l'espace, vous lisez le nom du répertoire actuel, et non le chemin complet : si vous vous trouvez dans le répertoire */home/Fabian/documents*, seul s'affichera *documents*.

Le caractère qui suit le crochet fermant, $ ou #, précise vos privilèges d'accès : dans le cas d'un utilisateur classique figurera le signe $, l'administrateur *root* recevra un dièse #.

Attention, connectez-vous seulement en tant que root que lorsque vous êtes sûr et certain de ce que vous voulez faire. Le root est la personne privilégiée, qui, dans un système Unix, peut réaliser tout ce qu'elle veut, sans contrôle d'accès ! Je crois que nous le répèterons jamais assez.

Si un utilisateur classique veut exécuter une commande pour laquelle il ne possède pas de droit d'accès, le système lui répondra par un message du style : *Only root can do that* ou *Permission denied*.

BASH offre plusieurs services à l'utilisateur permettant à ce dernier d'être plus efficace dans son travail. Encore une fois, nous ne pouvons aborder tous les aspects de BASH tant ses fonctionnalités sont nombreuses.

[1] Cela varie d'une confi guration à l'autre.

Ainsi, dans ce chapitre, nous avons choisi de parler du mécanisme des *alias* qui est un mécanisme permettant de définir des raccourcis aux appels de commandes ; du mécanisme des *redirections* et des *filtres* qui permettent de contrôler des flux de données ; du *regroupement de commandes* et, enfin, de la *substitution de commandes*.

3.2 Le mécanisme des alias

Un alias est une abréviation pour une commande, voire pour plusieurs mots de la ligne de commande.

Un alias est défini par la commande **alias** (qui l'eut cru ! ! !) :

```
alias <nom>='<texte>'
```

Le nom d'un alias peut commencer par un caractère spécial. Vous ne devez pas mettre d'espace avant et après le signe égal.

Exemple :

```
$ alias ls='ls --color -Fa'
$ alias rm='rm -i'
```

Pour qu'un alias soit pris en compte à chaque ouverture de session, vous devez modifier le fichier `.bashrc` se trouvant dans votre répertoire utilisateur. Si vous appelez la commande **alias** sans aucun paramètre, vous obtenez la liste des alias déjà définis. La commande **unalias** supprime les alias que vous avez définis. Vous devez spécifier en argument le nom d'un alias pour que cet alias soit supprimé.

Exemple :

```
$ alias
alias ls='ls --color -Fa'
alias rm='rm -i'
$ unalias ls
$ alias
alias rm='rm -i'
```

3.3 Redirection et fi ltre

La redirection des entrées-sorties est une des principales caractéristiques du shell, dont la puissance n'a d'égale que la simplicité. La réputation de Linux en tant que système d'exploitation souple et performant est due en grande partie aux possibilités offertes de rediriger, directement ou indirectement, les entrées-sorties.

Lorsque quelque chose est tapé au clavier, la machine intercepte le signal et envoie ce signal vers un canal particulier appelé *le canal d'entrée standard*. Ce signal est alors "aiguillé" sur un autre canal afin d'être affiché à l'écran. Ce canal s'appelle le *canal de sortie standard*. Ce canal est aussi utilisé par les commandes afin d'afficher leur résultat. Parfois, les commandes fournissent des erreurs comme résultat, ces erreurs sont envoyées vers l'écran par un autre canal que l'on appele le *canal d'erreur*. Le canal d'entrée, de sortie standard et d'erreur sont respectivement numérotés 0,1 et 2.

Le principe de la redirection est en général de rediriger des informations envoyées à l'écran (ce qui sort du canal de sortie standard ou d'erreur) dans un fichier, un fichier qui sera ultérieurement consultable et/ou modifiable.

3.3.1 Redirection de la sortie standard

La syntaxe est la suivante :

```
<commande> > <fichier_de_sortie>
```

Tout ce que la commande `<commande>` transmettera sera automatiquement placé dans le fichier `<fichier_de_sortie>` en question. Cette syntaxe est équivalente à la suivante :

```
<commande> 1> <fichier_de_sortie>
```

Le chiffre 1 étant là pour nous rappeler que nous redirigeons la sortie standard, celle qui correspond au canal numéro 1. Les messages d'erreurs passant par le canal d'erreur, numéroté 2, seront quant à eux affichés à l'écran.

En fait, deux situations sont à envisager. Dans la première, le fichier `<fichier_de_sortie>` n'existe pas encore. Dans ce cas, il sera créé. Dans la seconde, un fichier de même nom existe déjà et dans ce cas son contenu sera écrasé par les données transmises par la commande.

Exemple : Redirection du résultat de la commande `ls -l`

```
$ ls -l > toto
```

Pour visualiser le résultat de la commande contenu dans le fichier *toto*, il vous suffit de prendre soit votre éditeur préféré ou bien utiliser *less*.

3.3.2 Double redirection de sortie

Il n'est pas toujours judicieux d'écraser l'ancien fichier par des données de la commande. Dans ce cas, vous pourrez utiliser le signe >> (> doublé). Si le fichier de sortie n'existe pas, il est créé sinon, au fichier existant est ajouté la sortie de la commande.

3.3.3 Redirection d'entrée

Certaines commandes attendent leurs données depuis le clavier. Mais il est possible de les lire depuis un fichier en utilisant le signe < suivi du nom de fichier. La syntaxe est la suivante :

```
<commande> < <fichier_d'_entrée>
```

Le signe "plus petit" (<) indique que l'on fournit le fichier `<fichier_d'_entrée>` à la commande. Cette syntaxe est équivalente à la suivante :

```
<commande> 0< <fichier_d'_entrée>
```

Une exception existe cependant pour l'éditeur. Il n'est pas possible de lui affecter une redirection d'entrée-sortie sans provoquer d'importants problèmes.

Exemple : La commande **tr** prend comme arguments deux chaines de caractères. Elle a pour but de remplacer dans un fichier toute occurrence de la première par la seconde. Par défaut cette commande attend des entrées venant du clavier. Ainsi si vous tapez `tr 'r' 'g'` suivi de [entrée], cette commande attendra des informations venant du clavier[2]. Nous pouvons rediriger l'entrée standard, c'est-à-dire donner les informations non plus à partir du clavier mais d'un fichier en faisant :

```
$ tr 'r' 'g' < toto
```

Cette commande aura pour effet de remplacer toute occurrence du caractère "r" apparaissant dans le fichier *toto* par le caratère "g" et d'afficher le résultat à la sortie standard (ici l'écran).

3.3.4 Combinaison de redirections

Il est possible de combiner les redirections, c'est-à-dire, prendre un fichier en entrée et de mettre le résultat dans un fichier de sortie ; nous obtenons une syntaxe du genre :

```
<commande> < <fichier_d'_entrée> > <fichier_de_sortie>
```

Exemple : Reprenons l'exemple de la commande **tr**. Pour obtenir le résultat de la commande dans un autre fichier, il suffit de faire une redirection de la sortie standard :

```
$ tr 'r' 'g' < toto > lulu
```

La commande **tr** prends en entrée le contenu du fichier *toto* grâce à une redirection d'entrée et fournit comme résultat un fichier *lulu* grâce à une redirection de sortie.

3.3.5 Redirection des erreurs

Parallèlement aux canaux d'entrée et de sortie standard, chaque programme dispose encore de la possibilité d'envoyer à l'écran des messages d'erreurs par un canal de sortie séparé. C'est d'ailleurs la raison pour laquelle des messages d'erreur sont envoyés à l'écran alors qu'une redirection de sortie a été spécifiée dans la ligne de commande. Pour rediriger le canal standard d'erreur, vous utiliserez pour cela la commande :

```
<commande> 2> <fichier_sortie_erreur>
```

3.3.6 Redirection de la sortie standard et d'erreur

Notez que vous pouvez rediriger à la fois la sortie standard et la sortie d'erreur vers un même fichier. Il existe deux formats de syntaxe :

```
<commande> &> <fichier_de_sortie>
```

ou

```
<commande> >& <fichier_de_sortie>
```

[2] Pour sortir de la commande faites [Ctrl]-[D]

Les deux commandes précitées sont équivalentes à la suivante :

```
<commande> > <fichier_de_sortie> 2>&1
```

Attention, `<commande> 2>&1 1> <fichier_de_sortie>` ne donne pas le même résultat, pourquoi ?[3]

3.3.7 Les filtres

Un filtre (ou une commande filtre) est un programme sachant écrire et lire des données par les canaux d'entrée et de sortie standard.

3.3.8 Relier les commandes par un tube

Pour établir une liaison directe ente le canal de sortie standard d'une commande et le canal d'entrée standard d'une autre, il existe le signe | (le tube, en anglais *pipeline*). Toutes les données renvoyées par la commande placée à la gauche de ce signe à travers le canal de sortie standard sont renvoyées au canal d'entrée standard de la commande placée à droite. Voici ci-dessous quelques informations supplémentaires sur les tubes :

- Le signe tube (|) peut être placé à plusieurs reprises dans une même ligne de commande. Il n'y a aucune limite à dépasser ;
- La première commande d'un tube ne doit pas obligatoirement être un filtre ;
- La dernière commande d'un tube ne doit pas non plus forcément être un filtre ;
- A la fin d'un tube, la dernière commande peut également être dotée d'une redirection de sortie si le résultat ne doit pas apparaître à l'écran mais être stocké dans un fichier ;

3.3.9 Principales commandes de filtre

Les commandes présentées dans cette section ont comme caractèristiques qu'elles peuvent lire des données autrement que par le canal d'entrée standard. Si vous ajoutez le nom d'un fichier comme paramètre à l'une de ces commandes, elles liront les données dans ce fichier et non pas par l'intermédiaire du canal d'entrée standard.

- Trier le contenu d'un fichier : **sort**
  La commande **sort** lit les données ligne par ligne par le canal d'entrée standard et retourne ces données triées par le canal de sortie standard.
- Compter le nombre de lignes, de mots et de caractères dans un fichier : **wc**
  La commande **wc** lit les données ligne par ligne par la canal d'entrée standard et retourne comme résultat trois nombres entiers désignant respectivement le nombre de lignes, de mots et de caractères lus.

[3] Solution : En effet, la première commande fait tout d'abord un redirection de la sortie standard vers un fichier. Le symbole à la fin de la commande, indique qu'il faut rediriger la sortie d'erreur vers la sortie standard. Comme cette dernière a été redirigée vers un fichier, les erreurs y seront redirigées également. Dans le second cas, la première chose que l'on fait, c'est de rediriger les erreurs vers la sortie standard, qui est initialement l'écran avant de rediriger la sortie standard. Par conséquent tout ce qui est sortie standard uniquement sera redirigé vers le fichier.

- Paginer l'affichage d'un fichier : **more**
  La commande **more** lit du texte par le canal d'entrée standard et retourne ces données page à page à l'écran. Cette commande est expliquée plus en détail dans le chapitre 2, section 2.6.
- Recherche d'une ligne précise dans un fichier : **grep**
  La commande **grep** contient comme premier paramètre un critère de recherche. Cette commande cherche, parmi toutes les lignes d'un fichier ou d'un canal d'entrée standard, celles qui contiennent le critère et retourne ces lignes par le canal de sortie standard. Le critère doit être placé entre apostrophe (’). Cette commande est expliquée plus en détail dans le chapitre 2, section 2.5.4.

Exemple-1 : La commande **ps** (voir chapitre 5) fournit tous les processus tournant sur une machine. Ces processus sont nombreux ce qui complique la recherche à l'écran. Grâce aux filtres, on peut limiter le résultat de la recherche. La ligne suivante permet de récuperer dans le résultat fournit par la commande **ps** les lignes qui contiennent au moins une occurrence du mot ’bash’ :

```
$ ps aux | grep 'bash'
```

Exemple-2 : Nous pouvons étendre la recherche précédente au mot ’getty’ en ajoutant un tube supplémentaire :

```
$ ps aux | grep 'bash' | grep 'getty'
```

Exemple-3 : Comme nous voulons garder ce résultat dans un fichier, nous faisons une redirection de la sortie standard vers un fichier du nom *resultat* :

```
$ ps aux | grep 'bash' | grep 'getty' > resultat
```

3.4 Regroupement de commandes

Dans cette section, il s'agit de lancer un groupe de commandes.

L'utilisation de parenthèses en ligne de commande permet de créer un sous-shell qui va prendre en charge l'exécution des commandes placées entre ces parenthèses. Les différentes commandes sont séparées par des points-virgules, ce qui garantit leur ordre de traitement. Toutes les commandes sont traitées en tâche de fond. Les parenthèses permettent par conséquent de grouper des commandes dans la mesure où un traitement en arrièrer plan est possible pour l'ensemble.

Le regoupement de commandes est également possible avec des accolades {}. Deux différences fondamentales sont cependant à prendre en compte :

1. L'accolade d'ouverture doit toujours être placée, tout comme une commande, en première position d'une ligne ou comme premier mot après un point-virgule ;
2. Pour le traitement de commandes placées entre accolades, aucun sous-shell n'est lancé. Les commandes internes placées entre les accolades agissent sur le shell actif et non sur un sous-shell.

Exemple-1 : L'utilisation des parenthèses lance un sous-shell

```
$ cd ; pwd
/home/linus
$ ( cd .. ; pwd )
/home
```

```
$ pwd
/home/linus
```

Exemple-2 : L'utilisation d'accolades agit sur le shell actif

```
$ cd; pwd
/home/linus
$ { cd ..; pwd; }
/home
$ pwd
/home
```

3.5 Les substitutions

Il existe deux types de substitution : la substitution *de caractères* et la substitution *de commandes*.

3.5.1 La substitution de caractères

La substitution de caractères est réalisée par l'intermédiaire de caractères spéciaux. Il en existe trois :

***** L'étoile est le caractère qui peut remplacer n'importe quelle chaîne de caractère, ainsi `f*n` peut représenter aussi bien la chaîne de caractères 'fabian' que 'fzzzzzzzn'.

? Le point d'interrogation représente quant à lui, un et un seul caractère. Ainsi `f?n` peut représenter aussi bien la chaîne de caractère 'fan' que 'fzn'.

[ ] Les crochets sont des délimiteurs représentant l'un des quelconques caractères qu'ils contiennent. Ainsi `[bar]` est la représentation soit du caractère 'b', soit du caractère 'a', soit du caractère 'r'. Un tiret peut être utilisé pour spécifier une plage entière de caractères. Par exemple, `[d-m]` représente l'un des caractères compris entre la lettre 'd' et la lettre 'm'.

3.5.2 La substitution de commande

Lorsque vous exécuter une commande, il vous est possible d'indiquer au shell d'exécuter dans un permier temps certaines sous-commandes et de les remplacer, dans le texte de votre commande, par leur sortie standard. Une fois la sous-commande effectuée, le shell exécute votre commande : vous pouvez donc utiliser la sortie standard d'une commande comme argument d'une autre commande. La substitution de commande s'effectue grâce aux délimiteurs `` (des quotes inversées) ou encore, en encadrant la commande concernée par $().

Le petit exemple ci-dessous affiche à l'écran le nombre de fichier qu'il y a dans le répertoire courant :

```
$ echo "Nombre de fichiers dans le repertoire `ls |wc -l`"
```

elle est identique à la commande :

```
$ echo "Nombre de fichiers dans le repertoire $(ls |wc -l)"
```

Autre exemple : Cette ligne de commande permet de faire un `ls -l` sur le binaire `cp` de copie de fichier et ce, sans se déplacer dans le répertoire contenant `cp`.

```
$ ls -l `which cp`
-rwxr-xr-x    1 root    root      32272 Jul 18 21 :49 /bin/cp*
```

En fait le résultat de la commande `which cp` est `/bin/cp`. Ce résultat remplace la commande se trouvant entre les quotes inversées pour fournir la commande `ls -l /bin/cp` qui, elle-même, fournit le résultat qui en suit.

3.5.3 Les caractères d'échappement

Bien sur, qui dit *caractères de substitution*, dit *caractères d'échappement*. Les caractères d'échappement permettent de considérer les caractères de subsitution comme des caractères en tant que tels et non comme des caractères de subsitutions. Les caractères d'échappement sont au nombre de trois :

\ le *backslash* permet de protèger un et un seul caractère. Ainsi `\*` permet de considérer le caractère `*` comme une étoile ;

’’ Les simples quotes, quant à elles, protègent tous les caractères qu'elles encadrent. Ainsi `echo '*.*'` affichera à l'écran `*.*` (les caractères de substitutions ne sont pas interprétés). Contrairement à `echo *.*` qui afficherai tous les fichiers du répertoire courant ;

"" Les doubles quotes protègent tous les caractères qu'elles encadrent à l'exception du dollard ($), du backslash (\) et des quotes inversées (`). Voyez l'exemple suivant : `echo “*.*$(ls)”`

Chapitre 4

Les éditeurs : Emacs et vi

Introduction

Dans ce chapitre, nous allons vous présenter deux éditeurs de texte parmi les plus couramment utilisés sur les systèmes Unix.

Dans un premier temps, nous vous présenterons `vi` : un éditeur de texte qui se veut simple, compact et efficace.

Dans un second temps, nous vous présenterons `emacs` (développé par *Richard Stallman*, fondateur de la Free Software Foundation). Cet éditeur est plus lourd que son petit frère pré-cité mais possède également l'avantage d'être simple et efficace.

Nous ferons remarquer à nos chers lecteurs que ces éditeurs ne sont pas facile à utiliser au premier abord, mais qu'ils offrent des commandes rapides, courtes, puissantes et très appréciables une fois assimilées.

A propos de cet article

Cet article a été rédigé pour servir au plus grand nombre, mais il est orienté vers un public d'informaticiens ou d'étudiants en informatique plutôt qu'au grand public. Il devrait être plus adapté aux novices en matière de Unix/Linux, avec évidemment une forte orientation vers Linux.

I L'éditeur `vi`

Cette partie du cours présente l'éditeur par excellence du monde Unix/Linux, à savoir vi. *Rapide, puissant et compact,* vi *est l'éditeur de tout administrateur qui se respecte. Son utilisation n'étant pas vraiment naturelle, il faut apprendre à le dompter. Une fois cet effort accompli, il devient difficile de s'en passer. Cet article vous apprendra les parties qui me paraissent les plus utiles.*

4.1 Édition de textes sous Linux

Une des tous premiers utilitaires nécessaire sur tout système d'exploitation est un éditeur de texte. Par "texte", j'entends bien sûr les textes "bruts", sans formatage.

Sur les systèmes de type Unix, on est certain de trouver le célèbre éditeur `vi`, même sur les systèmes très anciens : `vi` est livré en standard sur tous les Unix, ce qui en fait l'éditeur le plus utilisé à base installée la plus large.

Il en existe un autre, `GNU/Emacs`, très performant et très versatile. Écrit par Richard Stallman lui-même. `Emacs` sait éditer des textes, lire des mails ou des news, lancer un shell, etc. Mais il n'est pas toujours installé, car il occupe beaucoup plus d'espace disque.

4.1.1 Qu'est-ce que vi ?

`vi`, est une abréviation pour "VIsual editor". C'est une extension de `ed` et `ex`, deux éditeurs plus anciens ne pouvant éditer les fichiers que ligne par ligne. Mais `vi` est tout de même un éditeur plein écran.

`vi` est rapide, puissant et compact : il tient sur une disquette de boot, capacité que lui envient pas mal d'autres éditeurs. Sous Linux existe une version améliorée de `vi`. Il s'agit de `vim`, pour "VI iMproved".

Il est tout de même nettement conseillé d'utiliser `vim` plutôt que `vi`, car il dispose d'un certain nombre de fonctionnalités supplémentaires que d'aucun jugerait indispensable d'un point de vue confort. Néanmoins, en cas de gros problème nécessitant de rebooter sur disquette, il vaut mieux savoir utiliser `vi` pur, étant donné que plus rien d'autre n'est possible dans ce cas.

4.1.2 Fonctionnalités de vi

Dans la suite, le terme "vi" référera à "vim", par facilité et abus de langage. Les fonctionnalités de vi sont bien plus élevées que celles attendues d'un éditeur texte par les utilisateurs de systèmes non Unix/Linux. Parmi celles-ci, on trouve notamment

- Edition en mode (texte) plein écran, pas ligne par ligne
- Indépendance par rapport au terminal (via stty)
- Edition de fichier : insertion, modification, etc
- Edition de fichiers multiples
- Recherche/Remplacement avec des expressions régulières
- Coloration syntaxique automatique, en fonction du type de fichier édité (code source en C, fichier HTML, PHP, SGML, etc)
- Lancement de commandes shell

– Macros, abréviations
– Répétition de commandes
– Undo infini
– Insertion du contenu d'un fichier externe à partir de la position du curseur
– Fenêtrage multiple (oui, en mode texte !)

4.1.3 Cas où vi est particulièrement adapté

L'administration sous Linux consiste souvent à éditer des fichiers de configuration en mode texte. C'est à ce niveau que l'on a un contrôle maximal sur ce qui peut se passer sur son système : il n'y a pas d'interface graphique qui cache les détails.

L'outil idéal de modification des fichiers de configuration en mode texte est incontestablement `vi`. Une fois qu'il est suffisamment maîtrisé, il n'est pas impossible d'avoir terminé une édition avec `vi` alors que la même édition sous `Emacs` en est encore au stade du chargement de l'éditeur.

Pour l'édition de scripts, `vi` est aussi un maître-achat. Il existe d'ailleurs un nombre considérable de formats de fichiers textes pour lesquels `vi` utilise une coloration syntaxique appropriée.

Les seuls cas où l'on pourrait se dire que `vi` n'est pas l'éditeur idéal, c'est sans doute dans le cadre de projets plus importants. Éditer plus d'une vingtaine de fichiers simultanément devient vite lourd. Dans ces cas, le besoin de plugins se fait sentir, et `Emacs`, l'éditeur le plus personnalisable au monde, remplit parfaitement cette tâche.

4.2 Concepts de base sous vim

Avant de commencer à expliquer les commandes utilisées sous `vi`, il est indispensable d'avoir bien compris les quelques principes de base de `vi`. Ils surprennent souvent le débutant par leur caractère peu répandu.

4.2.1 Edition de fichiers

Pour lancer vim, rien de plus simple. Il suffit de taper son nom :

```
vim
```

Pour éditer un fichier, il suffit de passer son nom en paramètre :

```
vim monfichier.txt
```

L'édition se passe comme l'on peut s'y attendre. Le fichier à éditer est chargé en mémoire. vim utilise le terme ”buffer” pour se référer aux fichiers édités, car ils sont placés en mémoire dans des ”buffers”. Ensuite, après avoir apporté toutes les modifications voulues au buffer, on peut l'écrire sur disque (sans quoi elles sont bien sûr perdues). L'écriture peut bien sûr se faire sous un autre nom, afin de garder intacte la version originale du fichier. Tout ceci est un fonctionnement classique bien connu.

4.2.2 Modes utilisés par vim

Les modes de travail utilisés par `vi` sont nettement moins naturels. Ils rebutent souvent le débutant au premier abord. Mais une fois le pas franchi, ce système s'avère diablement efficace.

`vi` possède trois grands modes, entre lesquels on switche souvent :

1. Le mode **saisie** ou **insertion**. C'est le mode que tout le monde connaît : lorsque l'on frappe des touches au clavier, les lettres sont insérées telles quelles dans le fichier texte. C'est le fonctionnement classique des autres éditeurs.
2. Le mode **commande** est plus inhabituel. C'est le mode dans lequel on donne des commandes simples à effectuer sur le texte : déplacement dans le buffer, suppression d'éléments, remplacement d'un mot, etc. Dans ce mode, chaque lettre tapée lance la commande correspondante !
3. Le mode **ex**, ou **ligne de commandes**. Mode similaire au précédent, il permet d'entrer des commandes, mais plus complexes cette fois. Dans ce mode, on tape une commande complexe après l'invite (:), et on la valide par la touche `[Entrée]`. Ces commandes groupent notamment les recherches/remlacement par expression régulière, l'écriture des buffers dans des fichiers, la sortie de vim, etc.

4.2.3 Changer de mode

Les deux premiers modes sont les plus utilisés au début. Au lancement de `vi`, on se trouve en mode commande. Pour pouvoir entrer du texte, il faut donc passer en mode insertion.

Pour passer en mode insertion lorsque l'on est en mode commande, il suffit d'introduire la commande `i` (insert) pour passer en mode insertion et insérer des caractères à partir de la position sous le curseur. Une alternative est de donner la commande `a` (append) pour passer en mode insertion et insérer des caractères à partir de la position qui suit celle sous le curseur.

Après avoir entré le texte souhaité, il faut revenir au mode commande, y compris si la commande que l'on veut entrer est un simple déplacement dans le texte ! Pour sortir du mode insertion, il suffit d'appuyer sur la touche `[Escape]`. On peut alors entrer des commandes simples (déplacement dans le texte, suppression de la ligne courante, etc).

Le troisième mode devient vite indispensable pour profiter de la puissance de `vi`. Pour y entrer (à partir du mode commande), on tape le caractère d'invite (:). `vi` passe alors en mode ligne de commande. Il attend que l'utilisateur tape sa commande complexe validée par la touche `[Entrée]`. `vi` exécute alors la commande complexe (recherche, remplacement, etc), puis revient tout seul en mode principal, le mode commande.

Il faut évidemment constamment passer du mode commande au mode insertion, ce qui paraît abominablement lourd, a priori. Cependant, les avantages contrebalancent largement ces inconvénients. Une fois l'habitude prise, ce sont les modes plus classiques d'édition de texte qui paraissent contre-naturels !

Comme d'habitude sous Linux, il faut mettre les mains dans le cambouis : lancer `vim`, essayer les quelques commandes, chipoter un peu pour bien assimiler le concept, etc. Ce n'est qu'au fil du temps passé à s'exercer que l'investissement consenti devient extrêmement rentable.

4.2.4 Comment sortir de vim ?

Une question angoissante est : comment sortir de vim ? La manière classique est celle du mode ligne de commande (ex). Cela consiste à taper la ligne suivante, en validant par `[Entrée]` :

```
:q!
```

Cette commande ”complexe” (:) fait sortir de `vi` (`q`), sans enregistrer les modifications (`!`).

Une autre manière classique est d'utiliser la commande simple

```
ZZ
```

Cette commande (en mode commande, naturellement) fait sortir de `vi`, mais en enregistrant les éventuelles modifications.

4.2.5 Principes généraux de vim

Dans le mode ligne de commande (ex), `vi` utilise un certain nombre de conventions qu'il est très utile de connaître.

Une première convention est l'utilisation du caractère `!`. En ajoutant ce caractère directement après la commande, on obtient une version légèrement modifiée de la commande. Ainsi, `:q` sort de `vi`, sauf si le buffer a été modifié, tandis que `:q !` sort de `vi` même si le buffer a été modifié (càd avec perte des modifications, dans ce cas).

Une seconde convention est similaire à son équivalent sous Emacs : c'est la notion de répétition. En préfixant une commande par un nombre `X` (dans le mode commande ou ligne de commande/ex), on obtient le même effet que si on avait entré `X` fois la commande sans ce nombre. Ainsi, pour la commande `dd` qui supprime la ligne courante, la commande `5dd` a pour effet de supprimer 5 lignes (comme si on avait tapé 5 fois la commande `dd`).

Après avoir rapidement passé en revue ces différents principes, on peut raisonnablement se lancer dans l'exploration de `vi`. Il suffira de donner une liste des commandes les plus utilisées, et de faire des essais sur machine pour voir si l'on a bien compris. Sous Linux, la meilleure façon de faire est bien souvent l'auto-formation par l'action (de préférence après présentation des concepts par une personne compétente).

4.3 Utilisation basique de vim

Il est temps de passer à la pratique, en appliquant les principes vus ci-dessus. Une première façon de faire est de lancer le tutoriel de **vim**. Pour ce faire, lancer, au niveau du shell, la commande

```
vimtutor
```

Si cette commande n'est pas disponible, ce n'est pas très grave. Il suffit de copier le fichier texte

```
/usr/share/vim/tutor/tutor
```

dans son répertoire personnel, puis de lancer **vim** en lui passant l'emplacement de la copie comme fichier à éditer (premier paramètre de la commande vim).

Une autre façon est de lire la documentation système (un peu longue), et de se mettre à la tâche. Pour ce genre de solution, autant lire d'abord ce qui suit avant de revenir à la doc complète : ce sont les formes les plus utilisées de vim.

4.3.1 Commandes de déplacement

Les premières choses à connaître sont les commandes de déplacement au sein d'un fichier. Parmi les très nombreuses, voici les plus utilisées. Il en existe encore d'autres, mais elles ne sont pas vraiement indispen-

sables dans une utilisation standard (sauf pour les utilisateurs très avancés).

Déplacement d'une unité

Pour se déplacer d'un caractère (en mode commande), les commandes suivantes sont disponibles :
- **h** pour aller au caractère à gauche (flèche gauche)
- **j** pour descendre à la ligne suivante (flèche bas)
- **k** pour remonter à la ligne précédente (flèche haut)
- **l** pour aller au caractère à droite (flèche droite)

Ceci a l'air fort primitif, mais est indispensable sur les claviers qui ne disposent pas des touches fléchées ! Pour les autres claviers, l'utilisation des touches fléchées est possible.

Déplacement au sein d'une ligne

Voici comment se déplacer à l'intérieur de la ligne courante :

- **0 (zéro)**, ou la touche "home" pour aller en début de ligne
- **$** ou la touche "end" pour aller en fin de ligne
- **w** (word) pour aller au mot suivant
- **b** (back) pour aller au mot précédent
- **f***x* (forward *x*) pour aller au prochain caractère *x*
- **F***x* fait la même chose que **f***x*, mais en reculant vers la gauche pour trouver le caractère *x*
- **t***x* (to *x*) pour aller au caractère précédant le prochain caractère *x*
- **T***x* fait la même chose que **t***x*, mais vers la gauche

Les commandes **f** et **t** sont un peu spéciales. Elles acceptent comme argument un caractère. Par exemple : **fp** déplace le curseur sous le prochain caractère 'p' de la ligne. C'est pratique pour un déplacement plus rapide au sein d'une ligne, mais surtout en association avec la commande **'d'** de suppression. Ainsi, **df.** supprime (**d**) tout depuis la position du curseur jusqu'au premier caractère '.' de la ligne (**f.**). Ceci correspond à la suppression de la fin de phrase courante.

La commande **t** est utile aussi. Lorsqu'on édite du HTML, il est souvent utile de supprimer le texte du curseur jusqu'au prochain début de tag, non inclusif. Pour réaliser cela, la commande est **dt**<, ce qui signifie supprimer les caractères à partir du curseur jusqu'au prochain caractère '<' de la ligne, non inclusivement.

Déplacements entre lignes

- **G** sert à se déplacer à la dernière ligne du fichier
- *x***G** sert à se déplacer à la ligne *x* (paramètre numérique *x*) Exemple : **1G** sert à se déplacer à la ligne 1, càd en début de fichier
- **:***x* pour se déplacer à la ligne *x* (paramètre numérique *x*)
- `[CTRL]`+**F** (forward) pour se déplacer d'un écran vers le bas
- `[CTRL]`+**B** (backward) pour se déplacer d'un écran vers le haut
- **%** (sur une parenthèse) pour se déplacer sur la parenthèse correspondant à la paire (ouverture-fermeture). Très utile pour la programmation en C, et surtout dans des langages fonctionnels, connus pour leur utilisation massive des parenthèses.

– **{** ou **}** pour se déplacer au bloc précédent ou suivant. La notion de bloc varie selon le type de fichier édité. C'est par exemple un paragraphe en HTML, ou le corps d'une fonction en langage C.

4.3.2 Commandes de changement de mode

En mode commande, lorsque l'on se trouve sur un caractère donné, on peut passer en mode insertion avec, principalement :
– **a** pour insertion après le curseur
– **i** pour insertion avant le curseur
– **I** pour insertion en début de ligne
– **A** pour insertion en fin de ligne
– **o** pour insertion sur la ligne suivante
– **O** pour insertion à la ligne précédente

4.3.3 Commandes de suppression

Toujours en mode commande, voici quelques commandes utiles de suppression.
– **x** pour supprimer le caractère sous le curseur
– **X** pour supprimer le caractère avant le curseur (backspace)
– **dw** (delete word) pour supprimer la fin du mot courant
– **dd** pour supprimer la ligne courante
– *x***dd** pour supprimer *x* lignes. Ex : **5dd** pour supprimer 5 lignes
– **dG** (delete, EOF) pour supprimer du curseur jusqu'à la fin du fichier
– **df***x* (delete forward param *x*) pour supprimer du curseur jusqu'au prochain caractère *x* (paramètre) de la ligne courante (ce caractère *x* inclus)
– **dt***x* (delete to *x*) pour supprimer du curseur jusqu'au prochain caractère *x* (paramètre) de la ligne courante (ce caractère *x* exclus)

4.3.4 Commandes de modification

En mode commande, voici les commandes principales pour substituer du texte :
– **cw** (change word) pour changer la fin du mot en cours (suppression de la fin du mot et passage en mode insertion)
– **cc** pour changer la ligne entière (suppression et passage en mode insertion)
– **s** (substitute) pour substituer le caractère (suppression et entrée en mode insertion)
– *x***s** (paramètre *x*) pour substituer *x* caractères
– **~** pour mettre la lettre sous le curseur en majuscule
– *x***~** (paramètre *x*) pour mettre les *x* caractères suivants en majuscule

4.3.5 Commandes "Fichiers"

– **ZZ** ou **:x** pour quitter vim et sauvegarder les changements éventuels
– **:w** pour enregistrer le buffer courant ("write")
– **:w** *nom* (paramètre *nom*) pour enregistrer le buffer sous un nouveau nom ("write as")

– **:w !** *nom* (paramètre *nom*) pour enregistrer le buffer sous un nouveau nom, et l'écraser s'il existait déjà.
– **:q** pour quitter (échoue si un buffer a été modifié)
– **:q !** pour quitter `vi` (changements perdus si buffer modifié)
– **:e** *nom* (paramètre *nom*) pour éditer un fichier supplémentaire (en gardant les anciens en mémoire).

4.3.6 Commandes couper-copier-coller

– **yy** ou **Y** pour copier la ligne en cours dans le tampon
– **p** pour coller le contenu du tampon après le curseur
– **P** pour coller le contenu du tampon avant le curseur
– **dd** pour couper la ligne en cours dans le tampon
– **"***x***yy** (*x* paramètre de type caractère) pour copier la ligne en cours dans le tampon portant le nom *x*
– **"***x***p** pour coller le contenu du tampon de nom *x* (paramètre)
– **"***x***P** idem, mais juste avant la position actuelle, pas juste après
– **y + commandes de déplacement** pour mettre dans le tampon ce qui suit le curseur jusqu'à l'endroit où arrive la commande de déplacement. Exemple : **yw** pour y placer le mot courant, **y0** (zéro) pour y placer le début de ligne jusqu'au curseur.

4.4 Commandes plus avancées

vi possède aussi un certain nombre de fonctionnalités bien plus avancées. Les plus utiles parmi celles-ci sont présentées ci-dessous.

4.4.1 Recherches et remplacement

Pour chercher une occurence, il suffit de définir le terme recherché.
– **/***nom* pour rechercher la 1ère occurence de *nom*, en paramètre. *nom* peut être simplement le texte cherché, ou une expression régulière.
– **n** pour chercher l'occurence suivante
– **N** pour chercher l'occurence précédente
– **:set ic** pour ignorer la casse lors des recherches (ne pas différencier majuscules et minuscules)
– **set noic** pour ne pas ignorer la casse lors des recherches (recherche case-sensitive, celle par défaut)
Pour remplacer une occurence, on précise en plus le terme de remplacement lorsque l'occurence est trouvée :
– **:s/***avant***/***après* pour remplacer la première occurence *avant* dans la ligne courante par *après*
– **:s/***avant***/***après***/g** pour remplacer toutes les occurences *avant* de la ligne courante par *après*
– **:%s/***avant***/***après***/g** pour faire un rechercher/remplacer global, sur tout le fichier. "%" signifie toutes les lignes. On peut spécifier d'autres étendues (bloc en cours, de la ligne x à la ligne y, . . .).

4.4.2 Sélection

Il est possible de définir un bloc sur lequel effectuer des actions plus tard. C'est la sélection telle que nous la connaissons habituellement :

– **v** pour définir un bloc qui commence sous le curseur et s'étend jusqu'à l'endroit où l'on déplace le curseur avec les commandes de déplacement.
– **V** pour passer en "mode visuel", c-à-d définir un bloc composé de lignes entières, commençant à la ligne courante et s'étendant jusqu'à la ligne où l'on se déplace.

Une fois la sélection effectuée, on peut effectuer des actions similaires à celles du couper-coller, ainsi que entre autres :

– **d** pour couper la sélection dans un tampon
– **y** pour copier la sélection dans un tampon
– **p** pour coller la sélection après le curseur
– **P** pour coller la sélection avant le curseur
– **:s** pour effectuer des rechercher/remplacer limités au bloc (voir ci-dessus).

4.4.3 Coloration syntaxique

– **:syntax on** pour activer la coloration syntaxique
– **:syntax off** pour la désactiver
– **:set background=dark** pour que la coloration choisisse des couleurs adaptées à un fond sombre (cas de la console texte)
– **:set background=light** pour que la coloration choisisse des couleurs adaptées à un fond clair (cas d'un xterm par exemple)

4.4.4 Insertion d'un autre fichier texte

:r*filename* pour inclure le contenu du fichier *filename* à partir de la position du curseur.

4.4.5 Lancer un sous-shell

Pour lancer une commande shell, taper **:!***commande*, en remplaçant *commande* par la commande désirée. C'est pratique dans de nombreux cas. Exemples :

– `:! sgml2html -l fr -c latin1 -s 1 fichier.sgml` pour générer les pages html correspondant au fichier source linuxdoc que l'on édite
– `:! sort` pour trier les lignes du fichier par ordre alphabétique. Ce type de commande peut même agir sur un bloc, s'il a été sélectionné avant.
– `[CTRL]`+**Z** : mettre `vi` en arrière-plan. Cela correspond à la combinaison de touches standard du shell pour mettre un processus en arrière-plan. Pour le remettre au premier plan, utiliser la commande **fg** ou les raccourcis du genre %n (où n est le numéro du job).

4.4.6 Edition multi-fichiers

– **:e** *filename* pour éditer *filename* en plus, en gardant les autres buffers
– **:b** *filename* pour passer au buffer *filename* (affichage)
– **:b** *n* pour passer au buffer numéro *n*
– **:ls** pour lister les buffers ouverts
– `[CTRL]-^` pour switcher entre deux buffers (dits "alternate").

II L'éditeur `emacs`

4.5 Description de la bête

Avant de commencer la description de l'utilisation d'`emacs`, vous remarquerez que vous pouvez lancer `emacs` tant à partir du mode console que d'une session graphique. Les présentations sont légèrement différentes mais les fonctionnalités restent les mêmes. Vous trouverez sur certains systèmes une version X de `emacs` appelée `xemacs`. Cette version, plus lourde mais autrefois plus conviviale est à présent surpassée par la version 21 de `emacs` Les propos qui suivent sont toutefois valables tant pour `emacs` que pour `xemacs`.

Avant d'entrer dans le vif du sujet, nous introduirons les notations suivantes :

C-x signifie que vous devez appuyer simultanément sur les touches [Ctrl] et [x] ;

M-p signifie que vous devez appuyer simultanément sur la *Meta-Key* et [p]. Habituellement, la Meta-Key correspond à la touche [Alt], mais ce n'est pas toujours le cas. Ainsi, suivant la configuration du clavier, il arrive fréquemment que ce soit la touche Windows qui remplisse ce rôle. Si vous ne parvenez à trouvez la Meta-Key, vous pouvez simuler son fonctionnement en appuyant sur la touche [Esc] suivi de la touche [p] ;

C-M-p signifie que vous devez appuyer simultanément sur les touches [Ctrl], [Alt] et [p]. Si votre touche [Alt] n'a pas d'effet, appuyer sur la touche [Esc] puis la relâcher équivaut à maintenir la touche [Alt] enfoncée.

Pour tester si c'est votre touche [Alt] ou [Esc] qui est opérationnelle, pressez simultanément dans un premier temps [Alt] et [x] ; si **M-x** apparaît dans le *mini-buffer,* c'est que la touche [Alt] est opérationnelle. Faites suivre de **C-g** pour annuler l'action (**C-g** est la combinaison de touche qui permet d'annuler une interaction avec le *mini-buffer*). Si la première combinaison n'affiche rien dans le *mini-buffer,* essayez en appuyant sur la touche [Esc] suivie de [x] ; dans ce cas, vous allez voir apparaître dans le *mini-buffer* le message **M-x**. Faites également suivre de **C-g** pour annuler l'opération.

L'interface d'`emacs` est divisée en trois parties (il existe une quatrième partie : la barre de menu dont nous n'aborderons pas l'utilisation dans le cadre de ce cours). Une première dans laquelle vous pouvez manipuler du texte, une seconde au fond de l'écran appelée *ligne d'état* et, enfin, une troisième juste en-dessous de la précédente appelée *mini-buffer*.

La *ligne d'état* indique le nom du fichier ainsi que le mode dans lequel vous vous trouvez (ce mode se trouve entre parenthèses sur la ligne d'état). Vous y trouverez également le numéro de la ligne sur laquelle vous vous trouvez (c'est le chiffre qui, dans la ligne d'état, suit immédiatement la lettre 'l' majuscule). A l'extrême droite apparaît le mot `Bot`, qui vous signale que vous voyez uniquement la fin du fichier. Sinon, apparaît le mot `All` qui signifie que vous voyez tout ou `Top` qui signifie que vous voyez uniquement le début ou encore, un pourcentage qui augmente au fur et à mesure que l'on 'déroule' le fichier.

Le *mini-buffer* est utilisé par `emacs` pour communiquer avec vous. Et c'est à partir de ce *mini-buffer* que vous communiquerez à `emacs` certaines commande à exécuter.

4.6 Commandes d'édition élémentaire

Avant d'effecuter des changements dans un texte, il faut savoir s'y déplacer. A défaut des *touches fléchées*, vous pouvez utiliser les raccourcis clavier suivants :

C-f avance d'un caractère (*forward*) ;
C-b recule d'un caractère (*backward*) ;
C-n passe à la ligne suivante (*next*) ;
C-p passe à la ligne précédente (*previous*) ;

Si vous maintenez les touches enfoncées, le système répétera l'action jusqu'à relâchement de ces dernières.

À partir du moment où vous insérez un ou plusieurs caractères, deux astérisques sont affichées à gauche dans la *ligne d'état*, et ce, pour vous indiquer que vous avez modifié le fichier.

En plus des quatre premières opérations sus-citées sur les caractères, nous pouvons signaler les suivantes :

[Backspace] efface le caractère précédent ;
[Delete] même effet que **[Backspace]** ;
C-d efface le caractère sous le curseur ;
C-t permute les deux caractères précédents.

Sur certains systèmes, `emacs` est configuré de telle sorte que la touche **[Delete]** ait le même effet que la combinaison de touche **C-d**. C'est l'une des facettes d'`emacs` : il est configurable à souhait[1]

Vous pouvez également effectuer des opérations sur les mots :

M-b se déplace en début de mot, celui directement à gauche (*back*) ;
M-f se déplace en début de mot, celui directement à droite (*forward*) ;
M-[Backspace] efface à gauche du mot ;
M-d efface à droite du mot ;
M-t permute deux mots autour du curseur ;
M-u met un mot en majuscule ;
M-l met un mot en minuscule.

Les combinaisons des touches **C-a** et **C-e** déplacent le curseur respectivement au début et à la fin de la ligne courante. Vous pouvez vous rendre à une ligne spécifique en tapant **M-x goto-line** suivi d'**[Entrée]** puis du numéro de ligne à laquelle vous voulez vous rendre. D'autres petites commandes utiles effectuent des opérations sur les lignes :

C-o Ouvre une ligne au-dessus du curseur pour insertion de texte ;
C-x C-o Ferme/Efface toutes les lignes blanches par-dessus et par-dessous le curseur ;
C-k Efface une ligne à partir du curseur jusqu'à la fin de celle-ci ;

[1] Pour plus d'information sur la confi guration d'`emacs` référez-vous au document écrit par *Fabian Bastin* sur le site du Namur-LUG ou bien aux pages d'aide relatives à `emacs`, intégrées au système.

Pour avancer d'une page, pressez **C-v**. Si vous devez reculer d'une page, **M-v**. Vous pouvez généralement utiliser aussi les touches [PageUp] et [PageDown] pour effectuer ces actions.

Nous terminerons cette première section par de petites opérations sur les pages :

C-x [ou **C-<**	Déplacement au début du fichier ;
C-x] ou **C->**	Déplacement à la fin du fichier ;
C-x l	Indique le nombre total de lignes du fichier ;

Signalons que les commandes de déplacement et d'édition élémentaires peuvent être également utilisées sous bash.

4.7 Editer plusieurs fi chiers à la fois

Quand vous ouvrez un fichier avec `emacs`, ce dernier le charge en mémoire. Le contenu du fichier se trouvant en mémoire s'appelle un *buffer*, et c'est celui-cio qu'`emacs` affiche. contenu du fichier que vous avez ouvert. Lorsque vous faites des modifications de texte, ce n'est pas le fichier qui est modifié mais le *buffer*. Les modifications effectuées ne sont prises en compte que lorsque vous enregistrez le *buffer*.

Pour ouvrir un second fichier, employez le raccourci **C-x C-f**. Dans le *mini-buffer* vous verrez apparaître un message du type `Find file : ~/`. Pour spécifier un nom de fichier, la syntaxe est identique à celle employée par le shell et la complétion est active (utilisez la touche `[Tab]` ou encore `[Space]`). Si le fichier que vous spécifiez n'existe pas il sera créé, sinon il est ouvert dans un deuxième buffer. En d'autres termes, si vous ouvrez plusieurs fichiers à la fois, le premier ne sera pas écrasé par l'ouverture du second.

Pour passer d'un buffer à l'autre vous pouvez utiliser la combinaison de touche **C-x b**. Quand vous faites cela, le *mini-buffer* vous demande vers quel buffer vous souhaitez passer. Par défaut, le buffer est le dernier qui était actif avant l'ouverture du nouveau fichier. Il vous suffit d'appuyer sur la touche `[Entrée]` pour mettre à exécution le changement. Si vous avez oublié les fichiers qui sont actifs, pressez simplement la touche `[Tab]` et ces derniers vous seront listés par `emacs`.

Si vous ne voulez pas enregistrer les modifications effectuées dans un buffer, vous pouvez en 'tuer' l'édition par **C-x k** suivi de la touche [Entrée]. Attention, cette fonctionnalité n'a pas pour but d'effacer le fichier que vous éditez mais simplement d'arrêter l'édition du buffer en le supprimant de la liste des buffers. Quand vous faites cela, `emacs` va vous demander quel est le buffer dont vous voulez arrêter l'édition. Par défaut, c'est le buffer courant. Si c'est le cas, pressez simplement la touche `[Entrée]`. En cas de modificiation du buffer depuis le dernier enregistrement, `emacs` demandera confirmation. Pour confirmer, tapez 'yes' et pressez `[Entrée]`.

Pour enregister les modifications apportées au buffer, utilisez simplement **C-x C-s** : `emacs` enregistre le buffer actif. La combinaison des touches **C-x C-w** vous permet d'enregistrer le buffer actif sous un autre nom que celui porté par le fichier original.

4.8 Terminer une session

Quand vous avez fini de travailler avec `emacs`, vérifiez que vous avez bien enregistré tous les fichiers. Vous pouvez quitter `emacs` avec **C-x C-c**. Parfois **C-x C-c** vous posera une question[2] ou deux via le *mini-buffer* avant de quitter ; ne paniquez pas, répondez-y en fonction.

4.9 Copier, coller, supprimer des parties de texte

Comme tout bon éditeur, `emacs` permet de faire du copier-coller. Pour pouvoir réaliser cette action, vous devez définir le début et la fin du bloc concerné. Le début du bloc est appelé **mark** et la fin de ce dernier est appelé **point**. Pour marquer le début du bloc, placez le curseur au bon endroit puis faites **C-[Espace]**. Vous verrez apparaître dans le *mini-buffer* le message `Mark set`. Par défaut, il n'y pas de point lumineux ou de surbrillance pour indiquer que le marquage a été fait à un endroit précis ; de toute manière vous savez où vous avez déposé la **mark**. Vous pouvez toutefois activer la surbrillance en faisant appel à la fonction transient-mark[3]. Cette dernière ne fonctionne qu'en mode graphique pour les versions inférieures à la version 21. Le marquage de fin correspond simplement à la position courante du curseur dans le texte. Une zone de texte peut-être également sélectionnée à l'aide de la souris. Pour ce faire, placez la souris au début du texte à sélectionner et appuyez sur le bouton de gauche. Déplacez ensuite la souris jusqu'à la fin de la zone à sélectionner tout en maintenant le bouton gauche enfoncé. Après avoir posé la **mark** et déplacé le curseur pour indiquer la fin du bloc, vous avez déterminé le bloc de texte prêt à être copier-coller.

Avant de coller le bloc séléctionné, vous devez le copier. Vérifiez que vous avez bien délimité ce dernier, puis, pour le copier, faites **M-w**. `Emacs` enregistre le bloc en mémoire. Ensuite, pour le coller, placez-vous juste à l'endroit désiré puis faites **C-y**.

Pour déplacer un bloc de texte (*couper-coller*), d'un point à un autre, sélectionnez tout d'abord la zone de texte concernée, suivi de **C-w**. **C-w** coupe la zone de texte qui doit être déplacée. Ensuite placez-vous à l'endroit où vous voulez coller le bloc, suivi de **C-y**.

4.10 Chercher et remplacer des expressions

4.10.1 Chercher une expression

Nous allons voir dans cette section deux manières de rechercher du texte avec `emacs`. Notez qu'il existe d'autres manières que nous n'aborderons pas ici.

La façon la plus facile de rechercher une chaîne de caractères est d'utiliser la fonction *isearch* (*incrementale search*). Cette méthode procède, comme son nom l'indique, à une recherche incrémentale, *i.e.* qu'au fur et à mesure que vous introduisez la chaîne de caractères à chercher, `emacs` place le curseur sur la première occurence de la chaîne entrée.

Vous pouvez utiliser les raccourcis suivants pour faire appel à cette fonction :

[2] Les questions concernent uniquement l'enregistrement des modifications au cas où des buffers modifiés n'auraient pas été engregistrés.

[3] Voir point 4.12 pour plus de détail sur l'appel de fonction.

C-s Pour effectuer une recherche incrémentale vers l'avant ;
C-r Pour effectuer une recherche incrémentale vers l'arrière ;

Nous pouvons avoir dans un texte plusieurs occurences d'un même mot. Pour accéder à l'occurence suivante du mot que vous cherchez, faites à nouveau **C-s**. Si, lors de votre recherche, vous arrivez à la fin du buffer, `emacs` vous dira `Failing i-search` : soit il n'a pas trouvé d'occurence du mot recherché ou bien, soit il est à la fin du buffer et il n'y a plus d'occurence du mot. Pour continuer la recherche faites à nouveau **C-s**, `emacs` recommencera la recherche à partir du début du buffer. Vous pouvez procéder de même avec **C-r**.

4.10.2 Chercher et remplacer une expression

Il est parfois utile dans un texte de savoir remplacer une chaîne de caractères par une autre (surtout si il y a mille occurences de la première !).

Pour effectuer cette action faites **M-%** (ou **M-x query-replace**. `Emacs` vous demandera tout d'abord la chaîne de caractères à remplacer, ensuite la chaîne de caractères de remplacement. Vous pourrez, lors de l'exécution de la fonction, confirmer ou infirmer le remplacement de la chaîne de caractère en tapant respectivement 'y' ou 'n'. La fonction **M-x replace-string** fonctionne de manière similaire, à la différence près qu'aucune confirmation n'est demandée.

4.11 Répéter une commande et annuler une commande

Pour répéter une commande, vous devez spécifier à `emacs` le nombre de fois que vous voulez l'exécuter, suivi de la commande à répéter.

Faites **C-u**, suivi du nombre, suivi de la commande (raccourci clavier) pour effectuer la répétition de la commande désirée.

Exemple :
Si vous voulez effacer 10 caractères l'un à la suite de l'autre, faites **C-u 10 C-d**.
Si vous voulez descendre de 10 lignes dans le texte faites **C-u 10 C-n**.
Si vous voulez écrire 100 fois la lettre 's', faites **C-u 100 s**.

`Emacs` offre, tout comme `vi`, la possibilité d'annuler l'effet des dernières commandes effectuées (et ce à l'infini). Pour annuler l'effet de la dernière commande, faites **C-_**. Comment feriez-vous pour annuler les 10 dernières actions effectués ?

4.12 Les commandes explicites : *M-x* <commande-explicite>

Tous les raccourcis clavier étudiés jusqu'à présent sont des appels à des fonctions internes à `emacs`. Par exemple, **C-p** est une manière de dire à `emacs` que l'on veut exécuter la fonction interne `previous-line`. Toutes ces fonctions internes peuvent être appelées par leur nom en utilisant **M-x**. Si vous avez oublié que `previous-line` est lié à **C-p**, vous pouvez taper simplement **M-x previous-line** suivi de **[Entrée]**. Ces deux manières de faire ont le même effet.

Il existe beaucoup de fonctions internes à `emacs`. Toutes ne peuvent pas par défaut être invoquées à l'aide de raccourci clavier. Par contre, toutes ces fonctions peuvent être appelées à l'aide de **M-x**. Notez que la complétion sur les fonctions fonctionne également.

4.13 Les modes en emacs

4.13.1 Le mode fondamental

Les buffers d'`emacs` sont associés à des *modes*. La raison d'être de ces modes est due aux différentes manières d'éditer un texte. Par exemple, on n'écrit pas un mail comme on pourrait écrire un programme.

Le mode le plus basic est le mode *fundamental* qui ne possède aucune commande particulière. Lorsque l'on tape **C-x b** et qu'on spécifie que l'on veux ouvrir le buffer 'foo', `emacs` en ouvre un nouveau. Le mode par défaut est le mode *fundamental*. Dans un cas contraitre, nous pouvons l'invoquer en faisant **M-x fundamental-mode**. Tous les modes peuvent être appelés via la commande **M-x <nom-de-mode>-mode** qui fait basculer le buffer courant dans ce mode.

Pour obtenir de l'aide sur le mode, faites **C-h m**.

4.13.2 Les modes de programmation

Si vous ouvrez avec `emacs` un fichier dont l'extension est *.c*, il passera en mode C (*c-mode*). Si vous ouvrez un fichier dont l'extension est *.for* ou *.f*, vous serez en mode fortran (*fortran-mode*). Ce ne sont que quelques exemples parmis tant d'autres.

Pour obtenir de l'aide sur le mode de programmation dans lequel vous programmez, faites **C-h m**.

4.14 L'aide en ligne : *C-h*

Où trouver de l'aide est l'éternelle question ! Si vous voulez en apprendre plus sur `emacs`, vous pouvez consulter la documentation disponible sur votre système (man, info, `/usr/doc/emacs/*`). Nous trouvons, fourni avec `emacs`, un tutorial sur lequel vous pouvez agir. Pour faire appel à ce tutorial, faites **C-h t**.

Plusieurs types d'aide sont disponibles :

C-h f Pour obtenir de l'aide sur une fonction ;
C-h k Pour obtenir de l'aide sur un raccourci clavier ;
C-h m Pour obtenir de l'aide sur les modes d'éditions ;
C-h C-h Pour obtenir de l'aide sur l'aide ;

Vous pouvez également étudier la configuration d'`emacs` en vous référant au document 'Configurer `Emacs`' écrit par Fabian Bastin. Ce document se trouve sur le site du Namur LUG (`http://namurlug.org`).

Chapitre 5

Administration

5.1 Introduction

Le système Linux (comme tout système Unix) est un système d'exploitation *multi-utilisateur* et *multi-tâche*. Un système multi-utilisateur signifie que plusieurs utilisateurs peuvent employer la même machine en même temps. Un système multi-tâche est un système qui permet à chaque utilisateur d'exécuter simultanément plusieurs programmes.

D'une part, lorsque l'on a plusieurs personnes qui peuvent utiliser une même machine, il est nécessaire d'établir une *politique de gestion de ces utilisateurs*. Cette politique de gestion concerne plusieurs aspects qui feront l'objet de ce chapitre.

Tout d'abord, nous verrons quelques simples commandes concernant la gestion du système en général. Ensuite, nous verrons comment un utilisateur peut gérer ses propres fichiers et les partager aux autres utilisateurs. Cette notion de partage nous conduira vers la notion de groupe. Nous parlerons de la manière de gérer ces groupes.

D'autre part, lorsque l'on a un utilisateur qui peut exécuter plusieurs programmes en même temps, il est nécessaire également d'établir une *politique de gestion des processus* sous-jacents à l'exécution des programmes. Et c'est cette politique de gestion que nous détaillerons par la suite.

Finalement, nous expliquerons comment utiliser les gestionnaires de packages **rpm** pour les fichiers `.rpm` et, **dpkg** pour les fichiers `.deb`.

5.2 Commandes de gestion générale

5.2.1 Passer en superutilisateur

L'administration du système est généralement destinée au *superutilisateur*. Le superutilisateur ou `root` est la personne qui possède tous les droits et privilèges sur la machine. Pour vous connecter en tant que root, à l'invite "login" vous tapez `root` suivi de **[Enter]**. Ensuite, vous entrez le mot de passe qui lui est réservé. Il est généralement déconseillé de se connecter en tant que `root`. La mèthode que nous préfèrons est de se connecter en tant que simple utilisateur et de faire la commande `su`. Cette dernière vous demandera le mot de passe du `root` avant de vous accorder les privilèges. Pour arrêter l'effet de la commande `su`,

vous appuierez simultanément sur les touches **[Ctrl]** et **[d]**. Vous pouvez également, avec cette commande, possèder les privilèges `root` le temps d'exécution d'une commande. Pour effectuer une telle action, utilisez la commande `su` avec l'option `-c` suivie du nom de la commande. Par exemple, vous pouver arrêter votre système en tapant :

```
$ su -c halt
```

5.2.2 Redémarrer/arrêter le système

Appuyer sur le bouton *power* pour arrêter une machine Linux n'est pas la meilleure idée. Si tel est le cas, vous remarquerez que, lorsque vous la redémarrez, le système exécute au moment du démarrage un programme du nom `fsck`. Cela signifie que le système n'a pas pu démonter les disques correctement.[1] De plus, il est risqué que vous perdiez ou endommagiez des fichiers de configuration. Pour arrêter le système proprement, vous pouvez utiliser la commande **halt**. L'arrêt du système consiste tout d'abord à tuer tous les processus utilisateurs puis, d'arrêter tous les services qui ont été lancé au démarrage, de démonter les disques préalablement monter, pour finalement arrêter le noyau.

Parfois, il est nécessaire de redémarrer une machine, parce que l'on a, par exemple, installé un nouveau noyau. Pour redémarrer le système, vous pouvez utiliser la commande **reboot** ou encore la célèbre séquence de touche **[Ctrl]+[Alt]+[Delete]**.

Sur certaines distributions ces commandes ne fonctionnent que quand vous êtes `root`.

5.3 Gestion des utilisateurs

5.3.1 Un peu de théorie

Imaginons le scénario classique d'une machine hébergeant plusieurs utilisateurs. Chaque utilisateur possède son propre répertoire sur la machine dans lequelle il peut stocker ses données personnelles. On appelle ce répertoire le *home directory* ou le *répertoire home* de l'utilisateur. Le répertoire contenant tous les utilisateurs est généralement le répertoire **/home**.[2] Ainsi le home directory de l'utilisateur **Linus** se nommera **/home/Linus**. C'est dans ce répertoire qu'est défini le *profil de l'utilisateur* **Linus**. Le profil de l'utilisateur est l'ensemble des fichiers de configuration des programmes que l'utilisateur utilise. De plus, seul l'utilisateur **Linus** est autorisé à entrer dans son répertoire, y copier, lire, déplacer des fichiers ou répertoires, à l'exception du super-utilisateur.

Un utilisateur est identifié par un numéro que l'on appelle l'UID (*User IDentificator*). L'utilisateur `root` possède toujours l'UID 0. Ce numéro sert au système afin de contrôler les actions d'un utilisateur. Ce principe d'identification empêche par exemple un simple utilisateur d'exécuter une commande réservée au `root` uniquement.

L'utilisateur `root` possède quelques commandes afin de gérer les utilisateurs : il peut en ajouter, en supprimer, leur changer leur mot de passe. Ces sont ces points que nous allons aborder maintenant.

[1] Pour le montage/démontage voir point 2.7 page 27.

[2] Ce répertoire est modifi able. Par exemple sur FreeBSD c'est le répertoire **/usr/local/home**.

5.3.2 Ajouter un nouvel utilisateur

La commande `adduser` a pour effet d'ajouter un nouvel utilisateur au système, en créant son home directory, en modifiant le fichier **/etc/passwd**, en lui demandant son mot de passe et quelques diverses informations. Le fichier **/etc/passwd** est le fichier regroupant tous les utilisateurs du système. Une ligne de ce fichier décrit le nom de login de l'utilisateur, son UID, son home directory et le shell à lancer après la procédure de login. La syntaxe de cette commande est la suivante :

```
adduser <nom_d_utilisateur>
```

où `<nom_d_utilisateur>` est le nom utilisé pour le login et le nom du home directory du nouvel utilisateur.

5.3.3 Enlever un nouvel utilisateur

Pour enlever un utilisateur de votre machine, vous devez utiliser la commande `userdel`. Elle a pour but d'enlever du fichier **/etc/passwd** la ligne correspondant à l'utilisateur passé en argument. Avec l'option `-r`, le home directory de cet utilisateur est également supprimé. Sa syntaxe est la suivante :

```
userdel <nom_d_utilisateur>
```

5.3.4 Changer le mot de passe

Un utilisateur un peu soucieux de la sécurité de ses données change régulièrement son mot de passe. La commande pour changer le mot de passe est `passwd`. Sa syntaxe est la suivante :

```
passwd [<nom_d_utilisateur>]
```

Tout utilisateur doit exécuter cette commande sans nom d'utilisateur en argument pour changer son mot de passe. Quant à l'utilisateur `root`, si il spécifie un nom d'utilisateur en argument, il pourra alors changer le mot de passe de cet utilisateur.

5.4 Gestion des groupes

5.4.1 Un peu de théorie…

Un groupe est un ensemble d'utilisateurs. Les groupes sont utilisés pour réunir des utilisateurs qui, soit travaillent sur un même projet, soit ont besoins de permissions spécifiques que d'autres utilisateurs ne doivent pas avoir ou encore, pour établir une hiérarchie au sein des utilisateurs. Les exemples suivants illustrent très bien l'utilisation de groupe :

– Nous pourrions très bien imaginer les utilisateurs **Linus** et **Alan** dans un même groupe **kernel** destiné au développement du kernel ;
– Sur la Debian GNU/Linux (une distribution Linux), par mesure de sécurité, les utilisateurs n'ont pas accès au périphérique audio. Le périphérique audio (**/dev/dsp**) appartient au groupe **audio**. Pour que les utilisateurs puissent avoir accès à ce périphique afin de pouvoir écouter leurs morceaux préférés, il faut

que ces utilisateurs soient ajoutés au groupe `audio`. (Voir section 5.5 sur les permissions pour plus de détail).
– Dans une institution comme une école, si des professeurs et des étudiants ont accès à la même machine, mettre les professeurs dans un groupe **professeurs** et les étudiants dans un groupe **étudiants** permet de distinguer les différents utilisateurs. De plus, ce technique permet aux premiers de protèger leurs données des seconds.

Un groupe est identifié par un numéro : le GID (*Group IDentificator*). Un groupe peut ne pas contenir d'utilisateur.

5.4.2 Ajouter un groupe

La commande `addgroup` a pour effet d'ajouter un nouveau groupe. Elle a pout but de modifié le fichier **/etc/group**. Le fichier **/etc/group** est le fichier regroupant tous les groupes du système. Une ligne de ce fichier décrit le nom du groupe et son GID. La syntaxe de cette commande est la suivante :

```
addgroup <nom_du_groupe>
```

où `<nom_du_groupe>` est le nom du groupe à ajouter.

5.4.3 Supprimer un groupe

Pour enlever un utilisateur du système, vous devez utiliser la commande `groupdel`. Elle a pour but d'enlever du fichier **/etc/group** la ligne correspondant au groupe passé en argument. La syntaxe de cette commande est la suivante :

```
groupdel <nom_du_groupe>
```

5.4.4 Ajouter un utilisateur dans un groupe

La commande `adduser` vue précédemment permet d'ajouter un utilisateur existant dans un groupe existant. Elle a pour effet de modifier le fichier **/etc/group** en ajoutant à fin de la ligne correspondant au groupe l'utilisateur devant être inscrit. La syntaxe de cette commande est la suivante :

```
adduser <nom_d_utilisateur> <nom_de_groupe>
```

où `<nom_d_utilisateur>` est le nom de l'utilisateur devant être ajouté au groupe `<nom_de_groupe>`.

Signalons que la commande `groups` permet de savoir dans quel(s) groupe(s) vous êtes inscrits et que la commande `id` en donne une description plus détaillée.

5.4.5 Supprimer un utilisateur d'un groupe

Il n'existe pas vraiment d'outil pour supprimer un utilisateur d'un groupe mise à part la bonne vieille méthode d'édition de fichier ! Le fichier à éditer est **/etc/group**. Chaque ligne débute par le nom du goupe

suivi de numéros et des utilisateurs inscrits dans ce groupe. Il suffit d'effacer le nom de l'utilisateur dont on souhaite la désinscription.

5.5 Les permissions d'accès

5.5.1 Un peu de théorie…

Lorsque vous tapez la commandes `ls` avec l'option `-l`, celle-ci vous renvoie une ligne du type :

```
-rw-r--r--    1 linus  users      4990 Sep 28 20:17 cours.html
```

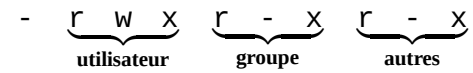
Si nous commençons par la droite, nous voyons le nom de fichier (`cours.html`), suivi de la dernière date de modification (`Sep 28 20:17`) et de la taille de ce fichier exprimée en octet (`4990`). Observons maintenant le reste de la ligne en commençant par la gauche. Tout d'abord, nous voyons une série de symboles et de lettres (`-rw-r--r--`). Cette série de symboles et de lettres sont *les permissions d'accès aux fichiers*. Les permissions d'accès sont des autorisations d'accès en lecture, en écriture ou encore, en exécution sur le fichier que le (ou un des) possesseur(s) du fichier peut(peuvent) imposer. Cette série de symboles est immédiatement suivie d'un chiffre. Ce chiffre décrit le nombre de liens dur du fichier (ici `1`) ou le nombre de fichiers que contient un répertoire (dans le cas d'un répertoire). Ensuite, apparaissent le nom de l'utilisateur (`linus`) et du groupe (`users`) possédant le fichier.

Une conséquence immédiate que nous pouvons tirer de cette description est que si un fichier n'appartient pas à l'utilisateur ou au groupe indiqué, l'accés au fichier par un autre utilisateur sera limité. Par exemple, si vous faites `ls -l /etc/inittab`, vous obtenez une ligne du type (la taille et la date du fichier n'ont pas beaucoup d'importance pour notre explication) :

```
-rw-r--r--    1 root     root         1856 Jul 18 21:48 /etc/inittab
```

Cette ligne vous indique que le fichier appartient à l'utilisateur et au groupe `root`. Si vous êtes loggué en tant que simple utilisateur (donc n'étant pas `root` et n'appartenant pas au groupe `root`), faire un `rm -f` du fichier *ne devrait pas l'effacer* : cela dépend des permissions d'accès[3].

Les permissions d'accès sont structurées en trois groupe de trois lettres ou symboles plus une. Le premier symbole sert à indiquer à l'utilisateur de quel type de fichier il s'agit. Si il s'agit d'un simple fichier de données, nous aurons le symbole `-`, si il s'agit d'un répertoire, nous y trouverons la lettre `d` (pour directory). Le premier groupe de trois lettres indique les permissions d'accès du propriétaire du fichier (le nom d'utilisateur indiqué comme étant le possesseur). Le second indique les permissions d'accès du groupe possédant le fichier. Et le dernier indique les permissions d'accès des utilisateurs qui ne sont ni le nom d'utilisateur, ni le nom de groupe indiqués. Nous pouvons attribuer à un fichier plusieurs types de méthode d'accès. Nous verrons les trois principales : la lecture symbolisée par la lettre `r` (pour read), l'écriture symbolisée par la lettre `w` (pour write) et l'exécution symbolisée par la lettre `x` (pour execute). Le signe `-` étant présent pour symboliser l'absence de l'une des trois premières.



[3] Si vous exécutez la commande avec les même permissions d'accès, le système refusera d'effacer le fichier. Un message `permission denied` sera affiché.

Nous entendons par accès en lecture, le droit de lire le fichier. Attention, ce n'est pas parce que vous avez le droit de lire un fichier contenant un programme écrit en bash que vous pouvez exécuter le programme[4]. Il faut pour cela que vous possédiez la permission d'exécution du fichier (donc du programme). Si ce qui précède n'est pas clair, voici un petit exemple : supposons que nous ayons un fichier contenant un programme écrit en bash, par exemple, `prog`. Ce fichier contient comme première ligne une ligne du type `#!/bin/bash`. Cette première ligne indique que lorsque nous voulons exécuter le programme, l'interpréteur à utiliser est **bash**. Imaginons, maintenant que les permissions d'accés soient les suivantes (peu importe l'utilisateur et le groupe) : `-rwxr-----`. Ces permissions dénotent que l'utilisateur peut lire, écrire et exécuter ce fichier, que le groupe peut uniquement le lire et n'a pas le droit de l'exécuter. En d'autres termes, le groupe à la droit d'en regarder le contenu mais pas de l'exécuter. Tandis que les autres n'en ont aucun droit d'accès. Le droit d'accés en écriture signifie que le fichier peut être modifié de son contenu.

Pour les répertoires la signification est un peu différente. Le droit d'accés en lecture sur un répertoire signifie que vous pouvez lire le contenu d'un répertoire, à cela près, qu'avant d'en pouvoir lire le contenu, vous devez être capable d'y entrer. Pour pouvoir entrer dans un répertoire, ce dernier doit possèder une permission d'accès en exécution. Le droit d'accés en écriture sur un répertoire signifique que vous pouvez y ajouter et y enlever des fichiers.

Une ligne comme la première sus-citée se décrit comme suit :

– il s'agit d'un fichier de donnée ;
– `linus`, propriétaire du fichier, possède les droits d'accès en lecture et en écriture ;
– les membres du groupe (dont `linus` fait partie) possèdent uniquement le droit en lecture ;
– les autres, qui ne sont ni `linus`, ni faisant partie du groupe `users` (à l'exception du `root` bien entendu) possèdent uniquement le droit en lecture.

Nous allons voir dans cette section comment nous pouvons changer les permissions d'accès d'un fichier, ainsi que le nom de l'utilisateur et du groupe possèdant ce fichier.

5.5.2 Changer les permissions d'accès

La commande **chmod** est la commande à utiliser pour changer les permissions d'accès d'un fichier. Un utilisateur quelconque a le droit de changer des permissions d'accès sur un fichier si il est le propriétaire de ce fichier. Dans le cas contraire, le système l'en avertira par un message : `operation not permitted`.

Il existe deux types de syntaxe pour cette commande : la première que nous appellerons la *méthode lettrée*, et la deuxième que nous appellerons la *méthode octale*.

La méthode lettrée

Comme nous l'avons déjà signalé, il existe trois groupes de trois lettres ou symboles. Chacun de ces groupes désignent une personne ou un groupe de personnes en particulier. La méthode lettrée attribue à chacun de ces groupes une lettre particulière : **u** pour l'utilisateur possédant le fichier, **g** pour le groupe et **o** pour les autres (pour *other* en anglais). Elle attribue également Pour chacune des permissions, la lettre **r** pour la lecture, **w** pour l'écriture et **x** pour l'exécution. Pour ajouter une permission, nous utiliserons le symbole **+**, pour en enlever, nous utiliserons le symbole **-**.

[4] Voir chapitre sur la programmation en bash pour plus de détail

La syntaxe de cette commande est la suivante :

```
chmod <permissions> <nom_de_fichier>+
```

Quelques exemples suffiront à expliquer l'utilisation de la commande.

avant	`----------`	Il n'y a aucune permission de définie sur ce fichier ;
commande	`chmod u+rwx`	Nous mettons pour l'utilisateur et uniquement pour l'utilisateur les permissions de lecture, d'écriture et d'exécution ;
après	`-rwx------`	
avant	`----------`	Il n'y a aucune permission de définie sur ce fichier ;
commande	`chmod u+rwx,g+rw,o+r`	Nous mettons pour l'utilisateur les permissions de lecture, d'écriture et d'exécution ; pour le groupe les permissions de lecture et d'écriture ; pour les autres uniquement la permissions de lecture.
après	`-rwxrw-r--`	
avant	`-r-xr-xr--`	
commande	`chmod u+rwx,g+rw,o+r`	Nous mettons pour l'utilisateur les permissions de lecture, d'écriture et d'exécution ; pour le groupe les permissions de lecture et d'écriture ; pour les autres uniquement la permissions de lecture.
après	`-rwxrwxr--`	Nous pouvons remarquer au vu des résultats que les anciennes permissions ne sont pas écrasées, elles sont seulement mises à jour puisque le `x` du groupe est toujours présent ;

Signalons qu'avec cette méthode, nous pouvons utiliser la lettre supplémentaire `a` (pour all) à la place des lettres `u`, `g` et `o` pour indiquer un changement simultané.

Nous ajoutons également que l'option `-R` permet de faire des modifications récursivement sur les contenus des répertoires.

La méthode octale

La deuxième méthode est une méthode un peu moins intuitive. Elle est basée sur le tableau suivant :

	Utilisateur	Groupe	Autres
Lecture	400	40	4
Ecriture	200	20	2
Execution	100	10	1

La commande a utilisé est toujours `chmod`. Elle prends comme arguments la somme des permissions sélectionnées dans le tableau et le(s) fichier(s) sur lequel vous voulez les appliquer. Les permissions initiales sont complètement écrasées : cette méthode ne met donc pas à jour les permissions, elle les applique. L'exemple suivant illustre nos propos, notez la différence avec le dernier exemple du point précédent :

avant	`-r-xr-xr--`	
commande	`chmod 764`	Nous mettons pour l'utilisateur les permissions de lecture, d'écriture et d'exécution ; pour le groupe les permissions de lecture et d'écriture ; pour les autres uniquement la permissions de lecture. La sélection dans le tableau nous donne : $400+200+100+40+20+4 = 764$
après	`-rwxrw-r--`	

L'option `-R` est également disponible.

5.5.3 Changer de propriétaire

Chaque utilisateur inscrit dans le système fait partie d'au moins un groupe. Un utilisateur peut-être seul dans un groupe, mais lorsque plusieurs utilisateurs travaille sur un même projet, il devient alors nécessaire de les réunir dans un seul et même groupe. Un des objectifs étant de faciliter les échanges de fichiers.

Sous Unix, en gros, l'échange de fichiers revient à en changer son propriétaire. La commande **chown** permet d'effectuer une telle action. Sa syntaxe est la suivante :

```
chown <nouvel_utilisateur> <fichier>*
```

Exemple :

```
$ ls -l
-rw-r--r--    1 linus  users      4990 Sep 28 20:17 cours.html
$ chown rms cours.html
$ ls
-rw-r--r--    1 rms    users      4990 Sep 28 20:17 cours.html
```

L'échange de fichier se fait uniquement entre utilisateur d'un même groupe.

L'option `-R` permet des modifications récursives du contenu d'un répertoire.

5.5.4 Changer de groupe

Maintenant un utilisateur peut faire partie de plusieurs projets, donc de plusieurs groupe à la fois. Il est alors possible à cet utilisateur de changer le groupe auquel appartient un fichier dont il est le possesseur.

La commande permettant de changer de groupe est la commande **chgrp**. Sa syntaxe est pratiquement identique à celle de **chown** :

```
chgrp <nouveau_groupe> <fichier>*
```

Exemple :

```
$ ls -l
-rw-r--r--    1 linus  linux    104560 Oct 3 19:00 libc.c
$ chgrp hurd
$ ls -l
-rw-r--r--    1 linus  hurd     104560 Oct 3 19:00 libc.c
```

L'utilisateur doit faire partie du groupe pour lequel il y a changement.

L'option `-R` est également disponible.

5.6 Gestion des processus

5.6.1 Un peu de théorie…

Tout programme qui est exécuté sur une machine crée, via le système d'exploitation, un processus. Un processus est une représentation de l'exécution d'un programme dans la mémoire de l'ordinateur. Ce n'est pas le programme lui-même. Chaque processus se partage la mémoire de l'ordinateur, chacun possède son propre territoire, sa propre parcelle de mémoire et il n'est pas question qu'un processus étranger pénètre le territoire d'un autre processus. Le cas échéant, un bon système d'exploitation *tue* le processus importun. D'ailleurs, un des rôles du système d'exploitation est de gérer les processus d'une machine afin que chaque programme puisse s'exécuter correctement.

Il existe toute une théorie sur les processus et leur méthode de fonctionnement mais elle dépasse largement le cadre de ce cours. Nous signalerons seulement quelques points de cette théorie qui vous seront utiles dans la pratique. Tout d'abord, à tout processus correspond un numéro unique, appelé PID (*Process IDentificator*). Ce numéro sert au système d'exploitation afin de bien gérer les différentes exécutions. Ensuite, il est important de savoir qu'un processus peut être père de plusieurs fils, ainsi que les fils eux-même,… En effet, si un processus père devient défaillant, tuer les fils ne servira pas à faire disparaître le père. Finalement et très brièvement, un processus peut entrer dans plusieurs états lors de son exécution, il peut être arrêté, suspendu, activé et réactivé et ce, afin de mieux partager les ressources du système avec les autres processus.

5.6.2 Lister les processus

La commande pour lister l'ensemble des processus tourant sur la machine est la commande **ps**.

Cette commande est utile car si, un jour, un programme boucle ou plante assez désagréablement (cela arrive), vous pourrez tuer le processus associé à ce programme. Grâce à la commande **ps**, vous pourrez connaître le numéro du processus (PID) défaillant afin de le tuer.

La syntaxe de cette commande est la suivante :

```
ps <options>
```

Selon le numéro de version, cette commande admet plusieurs type d'options. Il faut distinguer les options Unix qui peuvent être groupées et doivent être précédées d'un tiret, les options BSD qui peuvent être groupées et qui ne doivent pas être précédées d'un tiret, et, finalement, les options longues GNU qui sont précédées quant à elles de deux tirets. Ces différentes options peuvent être mélangées entre elles. Nous ne présenterons ici que les options qui nous semble vraiment intéressantes tant elles sont nombreuses. Les plus curieux complèteront leur connaissance en consultant la page de manuel.

Tout d'abord, nous avons les options de sélection de processus :

`-A` : Sélectionne tous les processus
`T` : Sélectionne tous les processus du terminal courant
`a` : Sélectionne tous les processus des terminaux, même ceux des autres utilisateurs
`r` : Sélectionne tous les processus dans un état actif
`x` : Sélectionne les processus sans contrôler leur terminaux

Exemple :

```
$ps T
 PID TTY      STAT   TIME COMMAND
1234 pts/3    S      0:00 -bash
1311 pts/3    R      0:00 ps T
```

Ensuite, nous avons les options de sélection par liste. Ces options requiert un argument :

`-C` : Sélectionne les processus par nom de commande
`-t` : Sélectionne par terminal
`-u` : Sélectionne les processus de l'utilisateur spécifié

Exemple :

```
$ps -t pts/0
 PID TTY          TIME CMD
 730 pts/0    00:00:00 bash
1217 pts/0    00:00:01 xdvi.bin
1221 pts/0    00:00:08 vim
```

Afin d'obtenir des informations plus détaillées sur chacun des processus, des `options de mise en forme sont disponibles`. Les plus utilisées sont les suivantes :

`-l` : Affichage long format
`u` : Affichage orienté utilisateur

Signalons que le groupement d'options le plus utilisé est `aux`.

5.6.3 Tuer un processus

Afin de tuer le processus récalcitrant, vous avez à votre disposition la commande **kill**. Sa syntaxe est la suivante :

```
kill [-s <signal>] <PID>
```

Pour tuer un processus de manière douce, il suffit de donner comme argument à la commande, uniquement le PID du processus que vous désirez tuer. Par exemple, si le processus numéro 234 est bloqué ou boucle, alors `kill 234` l'arrêtera et l'enlèvera de la liste des processus.

Parfois un simple `kill` ne suffit pas, vous pouvez alors tuer le processus de manière forte en utilisant l'option `-s 9` ou bien `-9`.

Vous ne pouvez tuer que des processus dont vous êtes propriétaires.

5.7 Installer et désinstaller des programmes

5.7.1 La notion de package

Un *package* regroupe l'ensemble des fichiers qui sont nécessaires à l'installation d'un programme. Il existe trois types de package et ils sont caractérisés par leur extension. Tout d'abord, nous avons les packages avec l'extension `.rpm`. Ce type de package a été créé par la compagnie RedHat. D'autres compagnies, comme Mandrake et Suse, ont repris ce format de package et l'ont adapté à leurs besoins. Ensuite, nous avons les packages d'extension `.deb` : package provenant de la distribution Debian. Enfin, nous avons le format `.tar.gz` ou encore `.tgz` qui est le format utilisé pour livrer les sources. Signalons qu'il existe des sources qui sont livrées en format rpm ou deb.

Il existe des programmes qui gére la création des packages, leur installation, leur désinstallation et leur dépendance. Ainsi pour le format `.rpm`, il existe le programme **rpm**. Pour les packages de format `.deb`, il existe **dpkg** et, pour le format `.tgz`, il existe **tar**.

Nous nous limiterons dans cette section d'expliquer le fonctionnement du programme **rpm** qui permet d'installer des packages d'extension `.rpm`.

5.7.2 (Dés)installer des packages RPM

Le **Red Hat Package Manager** est un outil puissant d'installation des packages sur votre système. Le format RPM (fichier d'archive ***.rpm**) possède des informations sur :

– les autres packages à installer pour que le programme fonctionne bien ;
– les composants avec lesquels il entrerait en conflit si ceux-ci sont déjà installés.

L'installation à l'aide du paramètre -i

Pour installer un package, utilisez la syntaxe :

```
rpm -i <nom_de_package>
```

Pour installer un package, vous devez vous connecter en tant que **root**. Vous pouvez utiliser la commande **su** pour bénéficier des droits de l'administrateur.

Les fichiers contenus dans le package sont installés et le package est enregistré dans la base de données. Vous pouvez combiner le paramètre `-i` avec les paramètres `-v` et `-h` en les concatènant : `-ivh`. Les options `-v` et `-h` ensemble affichent une jolie barre de progression lorsque le package est en cours d'installation. Si le package est déjà installé, **rpm** vous le signale en affichant un message d'erreur.

La syntaxe devient :

```
rpm -ivh <nom_de_package>
```

Le cas échéant, il arrive que d'autres packages ou fichiers soient nécessaires, les informations correspondantes étant spécifiées dans le package à installer. Dans ce cas, **rpm** affiche un message d'erreur et vous demande d'installer les packages dont dépend celui que vous voulez installer. Un problème parfois rencontré est celui des dépendance cyclique entre package : vous ne pouvez installer un package *A* que si un

package *B* est installé et vice-versa. Pour contourner le problème, il vous suffit de mentionner dans la même commande les deux noms de packages :

```
rpm -ivh package_A package_B
```

D'autres conflits peuvent aussi interrompre l'installation. Un fichier ne doit appartenir qu'à un seul package, par exemple. Vous pouvez contourner le contrôle mais cela peut avoir des conséquences dommageables dans la base de données et vous risquez de ne plus pouvoir mettre à jour certains packages par la suite.

Désinstallation avec le paramètre -e :

Le paramètre -e permet de supprimer des packages déjà installés et inscrits dans la base de données. Les packages sont désignés par leur nom abrégé. La syntaxe est la suivante :

```
rpm -e <nom_de_package_abrégé>
```

Ne désinstallez que des packages qui ne sont pas réclamés par d'autres packages. Cette dépendance est définie dans la base de données. Le cas échéant, la désinstallation est interrompue et la dépendance vous est signalée dans un message d'erreur. Le processus peut-être exécuté de force grâce à certaines options mais nous déconseillons cette méthode.

Mise à jour avec le paramètre -U

Si vous vous êtes procuré une nouvelle version d'un package, vous n'êtes pas obligé de désinstaller l'ancienne version pour installer la nouvelle. Utilisez la même syntaxe que pour l'installation mais avec le paramètre -U :

```
rpm -U <nom_de_package>
```

ou

```
rpm -Uvh <nom_de_package>
```

Lors d'une mise à jour, **rpm** tient compte de vos fichiers de configuration. S'ils n'ont pas été modifiés depuis l'installation, ils sont remplacés par les nouveaux. Dans le cas contraire, une copie de sauvegarde des fichiers de configuration existants est créée avec l'extension **.rpmorig* ou **.rpmsave*.

Lecture d'informations avec le paramètre -q :

Un package RPM contient de nombreuses informations telles que la liste des fichiers qui y sont contenus ou la description du package. Voici quelques moyens de déterminer ces informations.

Le paramètre -q vous permet de vous renseigner sur des packages installés :

```
rpm -q <nom_de_package_abrégé>
```

En ajoutant le paramètre -l, vous obtenez la liste de tous les fichiers du packages :

```
rpm -ql <nom_de_package_abrégé>
```

L'option -i fournit la description du package ainsi que d'autres informations :

```
rpm -qi <nom_de_package_abrégé>
```

Avec le paramètre -f, vous pouvez déterminer à quel package appartient un fichier :

```
rpm -qf <nom_de_fichier>
```

Pour obtenir des informations à propos d'un package non encore installé, utilisez l'option -p :

```
rpm -qpl <nom_de_package>
```

La combinaison de paramètres -qa est également intéressante : elle permet d'afficher la liste des packages installés.

```
rpm -qa <nom_de_package_abrégé>
```

Vérification avec le paramètre -V :

Le programme **rpm** offre la possibilité de contrôler si des packages ont été modifiés depuis leur installation. Utilisez le paramètre -V avec la syntaxe suivante :

```
rpm -V <nom_de_package_abrégé>
```

Les différents fichiers modifiés sont affichés. Aucune réponse n'est donnée à la commande s'il n'y a pas de différences avec la version initialement installée. Ce sont les fichiers de configuration que l'on retrouve le plus souvent dans cette liste.

Open Content Licence v1.0

Version 1.0, July 14, 1998.

This document outlines the principles underlying the OpenContent (OC) movement and may be redistributed provided it remains unaltered. For legal purposes, this document is the license under which OpenContent is made available for use.

The original version of this document may be found at http ://opencontent.org/opl.shtml

LICENCE

Terms and Conditions for Copying, Distributing, and Modifying

Items other than copying, distributing, and modifying the Content with which this license was distributed (such as using, etc.) are outside the scope of this license.

1. You may copy and distribute exact replicas of the OpenContent (OC) as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty ; keep intact all the notices that refer to this License and to the absence of any warranty ; and give any other recipients of the OC a copy of this License along with the OC. You may at your option charge a fee for the media and/or handling involved in creating a unique copy of the OC for use offline, you may at your option offer instructional support for the OC in exchange for a fee, or you may at your option offer warranty in exchange for a fee. You may not charge a fee for the OC itself. You may not charge a fee for the sole service of providing access to and/or use of the OC via a network (e.g. the Internet), whether it be via the world wide web, FTP, or any other method.
2. You may modify your copy or copies of the OpenContent or any portion of it, thus forming works based on the Content, and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions :
   (a) You must cause the modified content to carry prominent notices stating that you changed it, the exact nature and content of the changes, and the date of any change.
   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the OC or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License, unless otherwise permitted under applicable Fair Use law.

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the OC, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the OC, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. Exceptions are made to this requirement to release modified works free of charge under this license only in compliance with Fair Use law where applicable.
3. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to copy, distribute or modify the OC. These actions are prohibited by law if you do not accept this License. Therefore, by distributing or translating the OC, or by deriving works herefrom,

you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or translating the OC.

4. BECAUSE THE OPENCONTENT (OC) IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE OC, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE OC "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK OF USE OF THE OC IS WITH YOU. SHOULD THE OC PROVE FAULTY, INACCURATE, OR OTHERWISE UNACCEPTABLE YOU ASSUME THE COST OF ALL NECESSARY REPAIR OR CORRECTION.
5. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MIRROR AND/OR REDISTRIBUTE THE OC AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE OC, EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.