Université Paris 8 de Vincennes à Saint-Denis
Institut d'Enseignement à Distance
2, rue de la liberté
93526 Saint Denis CEDEX 02

# Pratique, Installation et Utilisation

Aline Hufschmitt
<aline.hufschmitt@iedparis8.net>

Support de cours

29/08/2017

Pratique, Installation et Utilisation

Aline Hufschmitt

2ème édition, septembre 2016
Université Paris 8 département informatique UFR MITSIC
Institut d'enseignement à distance

# Table des matières

# 1 Introduction

L'objet de ce cours est de vous aider à installer un nouveau système d'exploitation sur votre ordinateur et de vous familiariser avec son utilisation. Ce système d'exploitation, que vous allez utiliser pendant vos trois années de Licence d'informatique, est gratuit et libre (et donc open-source).

Il s'agit d'une distribution de Linux nommée Ubuntu. L'équipe pédagogique a choisi Ubuntu pour sa popularité. Ubuntu présente l'avantage non négligeable d'être techniquement soutenu par une importante communauté internationale, ce qui vous permettra, en consultant ses forums, de trouver facilement réponse à vos questions.

Je marque ici une pause car si vous êtes novice dans le monde de l'informatique, je m'attends à ce que vous soyez quelque peu perdu. Pas d'inquiétude, c'est normal. Vous devriez être en train de vous demander :

- « *Mais, c'est quoi exactement un système d'exploitation ?* »
- « *Il est libre, open-source ? Qu'est-ce que cela signifie ?* »
- « *C'est quoi une distribution ?* »
- « *Linux, est-ce que c'est compliqué ?* »
- « *Il y a une communauté qui peut apporter de l'aide ? Mais où ça ? comment on les contacte ?* »

L'objet de ce chapitre est de répondre à toutes ces questions.

## 1.1 C'est quoi un système d'exploitation ?

Un *système d'exploitation*, appelé *Operating System (OS)* en anglais, est un ensemble de programmes dirigeant l'utilisation des éléments matériels d'un ordinateur.

Le système d'exploitation sert d'intermédiaire entre l'ordinateur et les logiciels que vous utilisez (jeux, traitement de texte, etc.). Quand un logiciel veut utiliser un périphérique, par exemple une imprimante, il envoie sa demande au système d'exploitation qui la communiquera ensuite au périphérique. Le système d'exploitation utilise des *pilotes* (ou *drivers* en anglais) qui lui indiquent comment communiquer avec chaque périphérique en fonction de sa marque et son modèle. Si les logiciels voulaient communiquer directement avec les périphériques, il faudrait que chaque programme possède les informations pour reconnaître et communiquer avec chaque type de périphérique, ce qui les rendrait très lourds !

Le système d'exploitation s'assure que chaque programme pourra, entre autre, utiliser à tour de rôle les capacités de calcul de la machine (notamment le processeur), disposer d'un espace mémoire suffisant pour travailler et saura où placer les données qu'il souhaite enregistrer sur le disque dur.

Il s'assure que les programmes ne vont pas interférer en essayant d'utiliser la même ressource au même moment, mais leur permet néanmoins de communiquer entre eux d'une façon contrôlée.

Le système d'exploitation remplit aussi un rôle de surveillance. Il peut *tuer* (stopper) une application qui ne répond plus. Il est chargé de surveiller que les ressources ne sont utilisées que par des programmes qui possèdent bien un droit d'accès. Il se charge de la gestion des fichiers (lecture, écriture, exécution) et vérifie les droits d'accès à ces fichiers par les utilisateurs et les applications. Enfin, il fournit des informations permettant de surveiller le bon fonctionnement de la machine.

Le système d'exploitation permet ainsi de dissocier les programmes et le matériel, afin notamment de simplifier la gestion des ressources. C'est le premier programme qui est exécuté au démarrage de la machine, après l'amorçage.

La base d'un système d'exploitation est généralement constituée des éléments suivants :

- Un *noyau* (*kernel* en anglais) permettant l'exécution de fonctions fondamentales du système telles que la gestion de la mémoire, des processus, des fichiers, des entrées-sorties principales, et des fonctionnalités de communication.
- Un interprèteur de commande (*shell* en anglais, littéralement «*coquille*» par opposition au noyau) permettant la communication avec le système d'exploitation par l'intermédiaire d'un langage de commandes. C'est cet interpréteur que nous utiliserons dans le chapitre 4.
- Le système de fichiers (file system en anglais) permettant d'enregistrer les fichiers et de les présenter sous forme d'une arborescence : une hiérarchie de dossiers contenant des fichiers ou d'autres dossiers.

Les systèmes d'exploitation actuels sont généralement accompagnés d'une interface graphique (sous Linux nous parlerons d'un *environnement de Bureau*) et d'un ensemble de logiciels, de vidéos, d'images, etc..

Il existe de très nombreuses familles de systèmes d'exploitation fonctionnant sur différents types d'ordinateurs, tablettes et smart-phones, TV, etc. Pour avoir une idée de leur diversité il suffit de jeter un œil à la liste des systèmes d'exploitation proposée par Wikipédia : https://fr.wikipedia.org/wiki/Liste_des_syst%C3%A8mes_d%27exploitation.

## 1.2 Le système GNU/Linux

Afin de vous aider à situer Linux par rapport à Windows et MacOS, voici quelques repères très schématiques.

En 1969, Kenneth Thompson a créé un système multi-tâches et multi-utilisateurs nommé Unix qui s'est particulièrement répandu dans les milieux académiques au début des années 1980. Unix repose sur un interpréteur ou superviseur (le shell) et de nombreux petits utilitaires, accomplissant chacun une action spécifique, commutables entre eux (mécanisme de « redirection ») et appelés depuis la ligne de commande.

En 1981 apparait MS-DOS, un système monotâche et mono-utilisateur développé par Microsoft pour l'IBM PC d'abord, puis pour les compatibles PC. À cette époque, l'informatique n'était pas encore très développée et visuellement, MS-DOS ressemblait à Unix : du texte blanc sur fond noir. Jusqu'au début des années 90, MS-DOS sera le système le plus utilisé sur les compatibles PC avant d'être remplacé notamment par Windows.

En 1984 apparait le premier ordinateur Macintosh destiné au grand public qui a la particularité

de proposer une interface graphique et une souris. Il est équipé d'un système dédié : MacOS. En 2001, à cause du retard pris parait-il dans le développement de la version X du système, des éléments de FreeBSD, un système UNIX libre, seront utilisés pour créer les couches basses de MacOS X.

C'est également en 1984 que Richard Stallman, alors chercheur en intelligence artificielle au MIT, crée le projet GNU. Le système Unix est de plus en plus cher et il souhaite réagir en proposant une alternative gratuite fonctionnant comme Unix (les commandes restant les mêmes). Entre 1980 et le début des années 90, les programmes de base du système sont élaborés (programme de copie de fichier, de suppression de fichier, éditeur de texte).

De son côté, en 1991, Linus Torvalds, étudiant à l'Université d'Helsinki (Finlande), entreprend de créer sur son temps libre un système d'exploitation basé sur Unix qu'il nomme Linux (contraction de Linus et Unix).

Les projets de Linus Torvalds et Richard Stallman sont parfaitement complémentaires. Tandis que Richard Stallman travaille sur les programmes de base, Linus Torvalds crée le cœur d'un système d'exploitation : le noyau. En 1992, les deux projets fusionnent pour créer GNU/Linux : GNU (programmes libres) et Linux (noyau d'OS)

Pour résumer, Linux et MacOS sont deux systèmes fondés sur Unix et utilisant les mêmes commandes tandis que Windows, issus de MS-DOS est une branche à part.

## 1.3 Linux, logiciel libre et Open Source

« *Alors Linux c'est un système d'exploitation ?* »

Et bien, pas tout à fait. C'est un nom générique qui désigne différents systèmes d'exploitation libres fonctionnant avec le noyau Linux et respectant la norme POSIX [1]. Pour être exact, il faudrait parler de GNU/Linux puisque comme je vous l'ai indiqué, Linux est né de ces deux projets.

Le projet GNU voulait créer un OS gratuit mais surtout un OS *libre*. En anglais libre et gratuit se disent tout deux *free* d'où une confusion. Le terme *libre* indique que l'on doit pouvoir utiliser le programme, accéder à son code source, pouvoir l'étudier, le modifier et pourquoi pas redistribuer des copies avec ou sans modification. Mais attention, un logiciel libre n'est pas forcement gratuit.

Le logiciel libre est une véritable idéologie : les partisans du libre considèrent que l'accès aux sources des programmes et la possibilité de les modifier encouragent le partage des connaissances et l'évolution de l'informatique. Un slogan du logiciel libre pourrait être « *l'union fait la force* ».

Certains logiciels sont indiqués comme étant *open-source*. Il faut savoir que ce terme est plus restrictif : un logiciel open-source est juste un logiciel dont on peut consulter le code source. Autrement dit, on peut connaître la recette de fabrication mais on n'a pas forcément le droit de modifier cette recette ou de la dupliquer pour la redistribuer. Il existe une polémique entre les partisans du *libre* et de l'*open-source* [2]

1. POSIX (*Portable Operating System Interface*, le X exprime l'héritage UNIX) est une famille de normes techniques définie depuis 1988 par l'*Institute of Electrical and Electronics Engineers* (IEEE), qui permettent de définir des standards à respecter pour les interfaces de programmation des logiciels utilisés sur les variantes d'UNIX.

2. Je vous invite à consulter ces pages pour mieux comprendre la polémique entre les partisans du *libre* et de l'*open-source* :
http://www.gnu.org/philosophy/free-software-for-freedom.fr.html
http://www.gnu.org/philosophy/open-source-misses-the-point.html.

A l'opposé du libre et de l'open-source, un programme *propriétaire* désigne un programme dont on ne peut pas consulter le code source : par exemple, Windows est propriétaire.

Un logiciel copyleft est quant à lui un logiciel libre pour lequel toute redistribution à l'identique ou modifiée doit être faite en préservant les mêmes conditions : le travail d'un auteur ne peut évoluer vers une restriction des droits d'utilisation, d'accès aux sources, de modification ou de distribution. Donc, un logiciel réalisé à partir d'éléments copyleft ne peut être que copyleft.

C'est la *licence* qui accompagne un programme qui indique si il est libre, copyleft, open-source, propriétaire, etc. Différentes licences indiquent les droits dont dispose l'utilisateur. La licence la plus courante, indiquant un logiciel libre et copyleft, est la licence GNU-GPL, il en existe 3 versions, chacune ajoutant de nouvelles règles permettant de s'assurer que les libertés fondamentales de l'utilisateur (étudier, modifier, redistribuer) seront préservées.

## 1.4 C'est quoi une *distrib* ?

Chaque système Linux est proposé sous forme d'une *distribution*. Une distribution Linux (ou GNU/Linux) est composée du noyau Linux avec ses commandes de base, accompagné d'un ensemble d'applications et d'un *environnement* ou *gestionnaire de bureau*.

Ce gestionnaire de bureau gère tous les outils graphiques permettant de contrôler l'ordinateur : sans lui vous devriez utiliser des commandes en texte blanc sur fond noir. Dans les systèmes Linux, le gestionnaire de bureau est bien séparé du système d'exploitation ce qui vous permet, si vous le souhaitez, de le changer pour en utiliser un autre ou de vous en passer carrément pour n'utiliser que les commandes texte.

Il existe un très grand nombre de distributions Linux qui sont plus ou moins connues. Les éléments qui les différencient sont les suivants :

- Leur objectif : il y a des distributions dédiées pour l'enseignement, destinées au grand public ou au contraire à des utilisateurs experts.
- Le prix : certaines distributions sont totalement gratuites, d'autres sont payantes
- Les logiciels fournis dans la distribution.
- L'environnement de bureau (Gnome, Unity, KDE)
- L'intégration : le nombre de logiciels disponibles qu'il est possible d'ajouter par la suite.
- Le type de paquets utilisés pour distribuer les logiciels (Debian, RPM)
- La notoriété : une grande communauté d'utilisateurs permet d'obtenir de l'aide plus facilement
- Le mainteneur de la distribution : une entreprise ou une communauté

Pour vous donner une idée du nombre de distributions de Linux existant actuellement, je vous propose de jeter un œil à ce graphique : https://upload.wikimedia.org/wikipedia/commons/1/1b/Linux_Distribution_Timeline.svg.

## 1.5 Obtenir de l'aide (documentation et LUG)

Maintenant que vous en savez un peu plus sur ce qu'est un système d'exploitation et sur Linux en particulier, et avant de se lancer dans l'installation de la distribution de Linux nommée Ubuntu,

sachez qu'il existe une grande communauté d'utilisateurs de Linux et plus particulièrement de Ubuntu auprès de laquelle il est possible d'obtenir de l'aide.

Si vous avez besoin d'aide, vous pouvez commencer par chercher sur le site officiel francophone de Ubuntu : https://www.ubuntu-fr.org/. Si vous ne savez pas vraiment comment faire une recherche ou bien que vous ne savez pas par où commencer, une page est spécialement dédiée aux débutants : https://doc.ubuntu-fr.org/debutant. Sinon une simple recherche Google avec le mot *Ubuntu* comme premier mot clef peut vous permettre de trouver directement la page désirée dans la documentation officielle de Ubuntu. Par exemple, je peux chercher « *ubuntu problème wifi* » pour accéder directement à la page de la documentation ou aux pages du forum qui m'aideront à résoudre mon problème de connexion wifi sous Ubuntu.

Il existe des groupes d'utilisateurs de Linux un peu partout dans le monde qui se réunissent à intervalles réguliers. On appelle ces groupes des *LUG* (*Linux User Group*), des *GUL* en français (*Groupes d'Utilisateurs de Linux*), parfois écrit *GULL* (*Groupes d'Utilisateurs de Logiciel Libres*). Différentes listes de GUL sont proposées sur Internet et vous pouvez toujours prendre contact avec l'un d'eux pour obtenir de l'aide (pour installer Linux, apprendre à l'utiliser). Le site officiel de Ubuntu propose une liste de GUL spécifiquement dédiés à cette distribution.

Il existe également des canaux IRC (Internet Relay Chat, «*discussion relayée par Internet*») dédiés à Linux ou à des distributions spécifiques, et des forums.

# 2 Installation d'une distribution Linux : Ubuntu

Dans ce chapitre, je vais vous guider pour réaliser l'installation de Ubuntu, une distribution du système Linux. A la fin de ce chapitre vous devriez avoir une version récente de Ubuntu installée sur votre machine et être en mesure de commencer à l'utiliser.

Prenez le temps de lire une fois le chapitre en entier avant de vous lancer dans les manipulations.

## 2.1 Matériel nécessaire

Pour installer Ubuntu il vous faut ... un ordinateur !

Un ordinateur portable de préférence car il sera plus facile de le transporter jusqu'à un LUG ou l'université si vous avez besoin d'aide. Et de préférence un ordinateur que vous êtes seul à utiliser car vous allez passer de nombreuses heures à travailler dessus et serez amené tout au long de la licence à faire des manipulations qui peuvent en cas d'erreur faire planter la machine.

Si vous n'avez pas d'autre choix que d'utiliser l'ordinateur familial, conseillez aux autres utilisateurs de fermer leur session avant que vous ne vous serviez de la machine et de sauvegarder leurs données régulièrement.

Pour vérifier si votre ordinateur est assez puissant pour faire fonctionner Ubuntu, vous pouvez vous reporter à cette page : https://doc.ubuntu-fr.org/exigences_minimales.

Si vous avez l'intention d'acquérir un nouvel ordinateur, profitez en pour vérifier sur le site officiel de Ubuntu qu'il sera possible d'installer Ubuntu dessus sans problème. Certaines machines présentent quelques soucis de compatibilité donc mieux vaut vérifier.

Ce cours présuppose que votre ordinateur est de type PC et que vous travaillez actuellement sous Windows, mais si vous avez un Macintosh, ne paniquez pas, je vous donnerai quelques informations supplémentaires tout au long de ce cours car l'installation de Ubuntu sera un peu différente pour vous.

Si vous êtes sur PC, pour installer Ubuntu vous pouvez soit utiliser un CD : il vous faudra alors un graveur pour créer le CD d'installation. Si vous n'avez pas de graveur sur votre ordinateur, il est possible d'installer Ubuntu à partir d'une clef USB (une clef de 1 à 2Go est suffisante).

Vous aurez éventuellement également besoin d'une autre clef USB pour effectuer les manipulations expliquées au paragraphe 2.3.5.

Durant l'installation, pensez à imprimer les instructions dont vous aurez besoin ou (mieux) gardez

près de vous une autre machine (ordinateur, tablette ou smartphone) capable d'accéder à Internet pour demander de l'aide en cas de besoin.

## 2.2 Obtenir une copie d'Ubuntu

Vous pouvez obtenir la dernière version de Ubuntu gratuitement en ligne. Il vous suffit de saisir « *Ubuntu* » dans le champ de recherche de Google pour trouver le site officiel francophone de Ubuntu (Fig. 2.1).

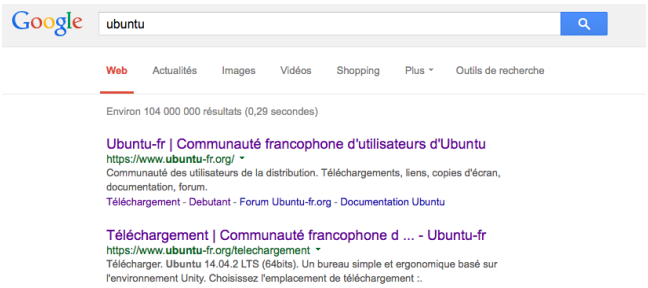


**Figure 2.1 – Deux premiers résultats proposés par Google en cherchant « *Ubuntu* ».**

Le deuxième lien https://www.ubuntu-fr.org/telechargement vous amènera directement sur la page de téléchargement. Il est également possible de commander en ligne une copie de Ubuntu sur CD (payante) à recevoir par la poste ou d'en obtenir une par l'intermédiaire d'un LUG : je ne détaillerai pas ces options puisqu'il est aussi simple de graver le CD vous-même.

### 2.2.1 Choisir et télécharger une version d'Ubuntu

Sur la page de téléchargement du site Ubuntu-fr vous pouvez voir un gros bouton : *Télécharger Ubuntu xx.xx.x LTS (64 bits)* (Fig. 2.2). A la place des *xx.xx.x* vous devez voir le numéro de version de cette distribution. Le *LTS*, si il est présent, signifie «*Long Term Support*». Cela indique que, pour cette version de Ubuntu, des mises à jour de sécurité et correctifs seront proposés pour une durée prolongée de 60 mois (5 ans). Une version non LTS ne bénéficie de ce support que pendant 9 mois, ce qui est très court et vous contraindra à installer un nouveau système au bout de cette période.

En cliquant sur *Plus d'options...* vous pouvez cocher ou décocher la case *Version LTS*. Si la version indiquée sur le bouton est LTS vous constaterez qu'il existe peut-être une version non-LTS encore plus récente de Ubuntu, mais la version LTS est préférable : vous pourrez garder cette version jusqu'à la fin de la licence sauf si vous désirez passer à un autre système entre-temps.

En cliquant sur *Plus d'options...* vous pouvez également basculer d'une version 64 bits à 32 bits en fonction du type de processeur de votre machine. Si votre machine ne supporte pas le 64 bits elle ne manquera pas de vous le faire savoir quand vous tenterez de lancer le liveCD que vous aurez gravé : vous ne pourrez pas le lancer.

Pour finir, cliquez sur le bouton *Télécharger Ubuntu xx.xx.x LTS (xxbits)* pour récupérer la version choisie. Regarder bien où votre machine enregistre ce fichier pour le retrouver ensuite.

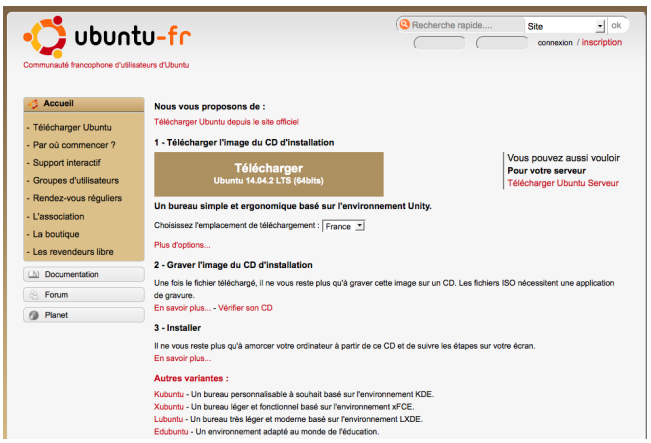


**Figure 2.2 – Page de téléchargement du site Ubuntu-fr en 2015, la dernière version de Ubuntu a changé depuis.**

### 2.2.2 Créer un Live CD ou un Live USB

Une fois le téléchargement terminé, vous vous retrouvez avec une image ISO de Ubuntu (un fichier dont le nom se termine par .iso). Cette image ISO est un unique fichier représentant le contenu complet d'un disque : CD (ou DVD) à graver ou autre support.

A partir de cette image vous pouvez, soit graver un Live CD, soit créer une clef Live USB. Le mot *live* indique que vous pourrez non seulement installer Ubuntu à partir de ce support mais également le lancer et l'utiliser directement.

Dans le cas d'un CD, vous ne pourrez pas enregistrer les données ou les modifications que vous aurez effectuées sur le système et elles seront perdues à chaque redémarrage. Un Live CD vous permettra donc juste de tester Ubuntu mais il faudra l'installer pour pouvoir vraiment l'utiliser.

Quand vous créez une clef Live USB, vous pouvez choisir un *mode persistant* ou *non-persistant*. Le mode persistant consiste à réserver de la place sur la clef pour stocker les données et modifications que vous aurez faites en utilisant Ubuntu depuis la clef. Avec le mode non persistant, les données seront perdues au redémarrage comme pour un Live CD.

#### Graver un Live CD

Pour créer un Live CD, il vous faut graver l'image de disque que vous avez téléchargée sur un CD (ou DVD) : le contenu de l'image sera copié sur le disque.

Attention, pour graver cette image vous devez choisir la fonctionnalité *graver une image de disque* dans votre logiciel de gravure. Si vous vous contenter de la fonctionnalité qui permet de graver des fichiers, au lieu de graver le contenu de l'image ISO, vous allez graver l'image elle-même et ce CD ne vous permettra pas de lancer ou installer Ubuntu par la suite.

Si vous n'avez pas de logiciel de gravure sur votre machine, vous pouvez trouver sur Internet différents logiciels gratuits permettant de graver une image ISO. Il vous suffit de saisir les mots clefs « *graveur image ISO gratuit* » dans un moteur de recherche pour en trouver quelques uns.

Si votre ordinateur est un Macintosh, vous n'avez pas besoin de graver cette image ISO sur un CD, vous pourrez l'utiliser directement pour installer Ubuntu sur une machine virtuelle. Pour savoir comment installer et utiliser une machine virtuelle, reportez-vous au chapitre 7.

### Créer une Live-USB

Pour créer une clef Live USB, il est possible de procéder de plusieurs manières. Je ne décrirai ici que l'utilisation de l'utilitaire *Unetbootin* car il est disponible à la fois sous Windows et sous MacOs. Pour avoir des détails sur les autres possibilités, je vous renvoie à la documentation de Ubuntu.

Votre clef doit être formatée en FAT32, si ce n'est pas le cas, vous pouvez la formater en faisant un clic droit sur la clef USB et en sélectionnant l'option *Formater...*. Si votre clef fait plus de 2Go vous pouvez avoir des problèmes sur les machines équipées d'un vieux BIOS : il faut alors partitionner la clef. Si vous ne savez pas comment faire, reportez vous à la section 2.3.4, j'y explique comment modifier les partitions d'un disque.

Allez sur le site `http://unetbootin.github.io/`, téléchargez *Unetbootin* en cliquant sur le bouton *Download ...* correspondant à votre système, installez le puis lancez le programme. Dans la fenêtre qui s'affiche, sélectionnez *Diskimage* pour créer la Live USB à partir de l'image de disque téléchargée. Sélectionnez *ISO* et sélectionnez l'image sur votre disque en cliquant sur le bouton *...* (Fig. 2.3).

La ligne en dessous vous propose de réserver un peu de place pour le mode persistant : les modifications que vous ferez en testant le système seront alors préservées, elles ne disparaitrons pas au redémarrage. Si vous laissez la valeur 0, vous créerez une Live USB en mode non-persistent.

Dans la case drive sélectionnez votre clef USB, puis validez pour lancer la création de la Live USB.

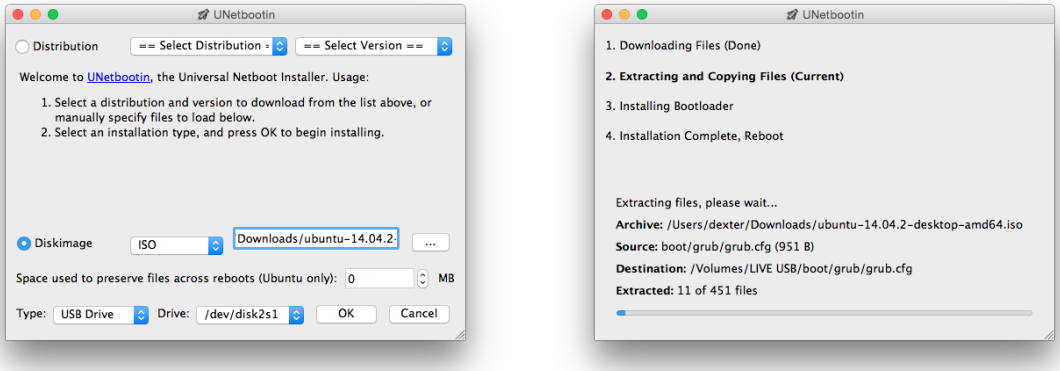


**Figure 2.3 – Utilitaire Unetbootin pour créer une clef Live USB.**

## 2.3 Avant de faire l'installation

Voici quelques petites choses à faire et auxquelles il faut réfléchir avant de commencer votre installation de Ubuntu sur un PC.

Si vous avez un Macintosh, vous n'êtes pas concerné par ce qui est décrit dans les paragraphes qui suivent et vous n'aurez pas à faire de manipulations jusqu'au prochain paragraphe 2.4. Cependant, prenez quand même le temps de lire ce qui suit car vous y trouverez quelques notions importantes.

### 2.3.1 Ubuntu en double boot ou comme unique système ?

Si vous avez le système Windows sur votre machine, vous souhaiterez peut-être le conserver. Cela est tout à fait possible : vous pouvez installer Ubuntu à côté de votre système actuel bien que cela soit un petit peu plus compliqué [1]. On appelle cela une installation en *double boot*. Le mot *boot* désigne la procédure de démarrage (ou d'*amorçage*) d'un ordinateur. *Double boot* signifie qu'il y aura deux façons de démarrer votre machine : sous Windows ou bien sous Ubuntu.

Attention, si vous voulez avoir les deux systèmes sur votre machine, l'espace disponible sur votre disque dur sera partagé entre les deux. Assurez-vous que votre disque dur dispose d'un espace libre suffisant pour en laisser une partie à Ubuntu.

L'installation en *double boot* demande de prendre quelques précautions supplémentaires avant d'installer Ubuntu, nous verrons cela un peu plus loin (§. 2.3.4).

Si vous souhaitez avoir un Windows pour l'utiliser à de très rares occasions, vous pouvez éventuellement le ré-installer dans une machine virtuelle sous Ubuntu [2] ; vous pouvez alors installer Ubuntu comme unique système. Nous verrons dans le dernier chapitre comment installer une machine virtuelle (§. 7).

### 2.3.2 Installation de Ubuntu sous Windows : déconseillé

Vous pouvez être tentés d'installer Ubuntu dans une machine virtuelle (VirtualBox, VMware, ou autre) sous Windows. Cependant, vous risquez de rencontrer différents problèmes : votre version d'Ubuntu risque d'être très lente et de ralentir peu à peu avec l'ajout de nouveaux logiciels jusqu'à devenir inutilisable.

Si votre machine est extrêmement performante, elle pourra utiliser une partie de ses performances pour faire tourner une machine virtuelle avec Ubuntu en même temps que Windows. Dans le cas contraire, si vous avez un PC ordinaire, je déconseille cette option et recommande d'installer Ubuntu selon la méthode qui sera expliquée plus loin (§. 2.5).

### 2.3.3 Sauvegarder vos données

Si vous avez choisi de remplacer votre système actuel par Ubuntu, toutes les données présentes sur l'ordinateur seront perdues. Il faut donc les sauvegarder pour ensuite les remettre sur votre nouveau système.

1. En vérité pour un débutant ça peut même devenir un sacré casse-tête mais grâce à l'abondante documentation disponible pour Ubuntu et l'importante communauté d'utilisateurs prête à aider, ce n'est pas insurmontable.
2. Attention, si vous utilisez des logiciels commerciaux sur CD ou DVD, notamment des jeux, ces derniers peuvent refuser de s'installer sur une machine virtuelle à cause des protections anti-piratage présentes sur ces supports.

Si vous avez choisi de conserver Windows, les manipulations que vous allez faire pour installer Ubuntu sont loin d'être sans risque pour vos données. Normalement tout devrait se passer sans problème mais comme on est jamais trop prudent, je vous conseille vivement de sauvegarder toutes vos données sur un disque externe avant de commencer l'installation.

D'ailleurs, je ne saurais trop vous recommander de prendre l'habitude de sauvegarder régulièrement vos données sur un disque externe pour avoir toujours une copie de vos documents si votre ordinateur tombait en panne.

Pensez à sauvegarder vos documents (textes, vidéos, photos, ...) mais également vos marque-pages internet (aussi nommés signets ou bookmarks en anglais) ou le dossier profil de Firefox si c'est votre navigateur. N'oubliez pas votre carnet d'adresse. Pensez également à noter dans un carnet vos codes de boites emails, etc.

Pour transférer d'un bloc votre carnet d'adresse, vos courriers et paramétrages de boite aux lettres sous Ubuntu, la documentation de Ubuntu [3] conseille de transférer d'abord vos données sur Thunderbird (en installant Thunderbird sous Windows), puis de sauvegarder le dossier Thunderbird pour le réimporter sous Ubuntu.

N'oubliez pas bien sur de sauvegarder ces données pour tous les utilisateurs de votre machine si vous n'êtes pas le seul à l'utiliser !

Pour être en totale sécurité assurez-vous également que vous avez un DVD d'installation ou de restauration de votre système. Sur certaines machines, il faut graver ce DVD (ou clé USB) de restauration de votre système.

### 2.3.4 Faire de la place pour Ubuntu

Si vous avez choisi d'installer Ubuntu à coté de Windows, il faut d'abord faire de la place pour Ubuntu en nettoyant et défragmentant votre disque puis en réduisant la taille de la partition occupée par Windows.

Lorsqu'un fichier est enregistré sur votre disque, si la place est suffisante, toutes les informations concernant ce fichier sont enregistrées les unes à la suite des autres, mais si le disque est déjà partiellement occupé, les fichiers suivants sont enregistrés dans les trous laissés par les fichiers qui ont été effacés [4]. Si ces trous sont petits, le fichier est découpé en morceaux qui sont enregistrés à des endroits différents du disque. La défragmentation permet de regrouper les morceaux de fichiers éparpillés sur votre disque et de remettre les fichiers proprement les uns à la suite des autres.

Avant d'aller plus loin, il ne faut en aucun cas défragmenter un disque SSD, sous peine de réduire ses performances ainsi que sa durée de vie. Si vous ne savez pas si votre disque est un SSD vérifiez dans la documentation de votre ordinateur.

3. https://doc.ubuntu-fr.org/thunderbird#importer_son_profil_de_windows_a_ubuntu

4. Sur un système Windows, la fragmentation est importante car il remplit en priorité les trous pour éviter les zones avec des clusters libres en début de disque. Un système Unix cherche en priorité un espace assez grand pour stocker le fichier entier : la fragmentation intervient quand le disque commence à être trop plein pour trouver un espace assez grand.

### Nettoyer le disque

Appuyez sur les touches ⊞ + E pour faire apparaitre une fenêtre de l'explorateur de fichiers. Dans la colonne de gauche sélectionnez *Ce PC* sous Windows 10 ou 8 ou *Ordinateur* sous Windows 7. Dans la fenêtre de droite, vous verrez surement un seul *disque local* nommé `C:` (Fig. 2.4). Si vous voyez un autre *disque local* `D:` (un disque local pas un lecteur CD ou clef USB !) vous pouvez transférer son contenu sur le disque `C:` pour le vider complètement et c'est ce *disque local* `D:` que vous utiliserez pour installer Ubuntu. Si vous n'avez que le disque local `C:`, faites un clic droit dessus et sélectionnez *propriétés*.

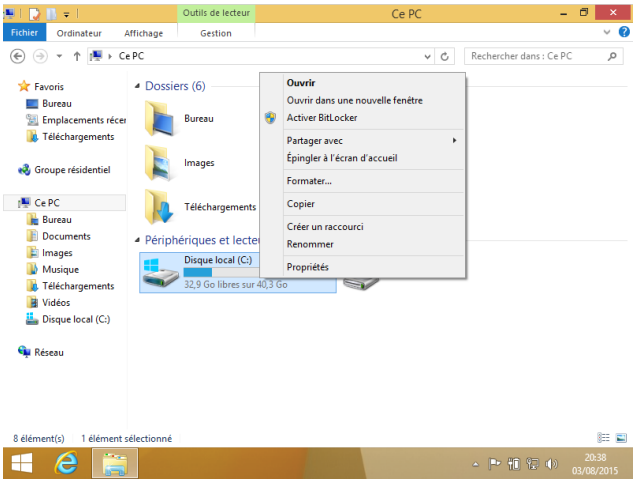


**Figure 2.4 – Afficher les propriétés de `C:` sous Windows 8.**

Dans la fenêtre qui s'affiche, dans l'onglet *Général* commencer par cliquer sur *Nettoyage de disque* et attendez que Windows ait calculé l'espace qui peut être libéré sur votre machine. Dans la nouvelle fenêtre qui s'affiche, cochez un maximum de choses à nettoyer dans la liste et valider en cliquant sur *OK* (Fig. 2.5).

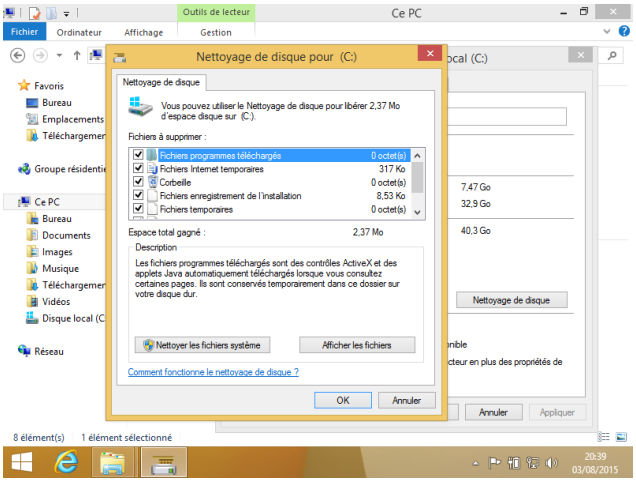


**Figure 2.5 – Nettoyage de disque sous Windows 8.**

### Vérifier le disque

Sélectionnez ensuite l'onglet *Outils* et cliquez sur *Vérifier* (*Vérifier maintenant...* sur les versions antérieures à Windows 8) (Fig. 2.6). Vous pouvez *Analyser et réparer le lecteur* directement. Atten-

tion la procédure peut durer de quelques minutes à quelques heures en fonction de la taille de votre disque et elle peut nécessiter le redémarrage de l'ordinateur.

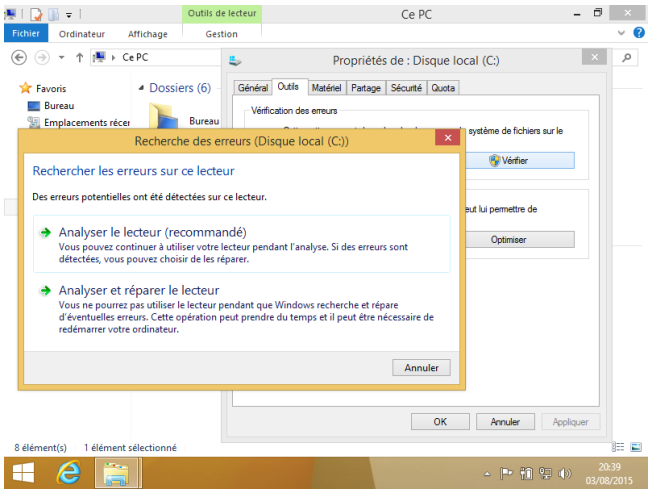


**Figure 2.6 – Vérification du disque sous Windows 8.**

## Défragmenter le disque

Enfin dans le même onglet *Outils* cliquez sur *Optimiser* (*Défragmenter maintenant...* sur les versions antérieures à Windows 8) pour lancer la défragmentation. Sélectionnez le disque `C:` (attention si c'est un disque SSD ne le défragmentez pas !) et cliquez sur *Optimiser* (Fig. 2.7). La défragmentation peut durer plusieurs heures, patientez jusqu'à la fin.

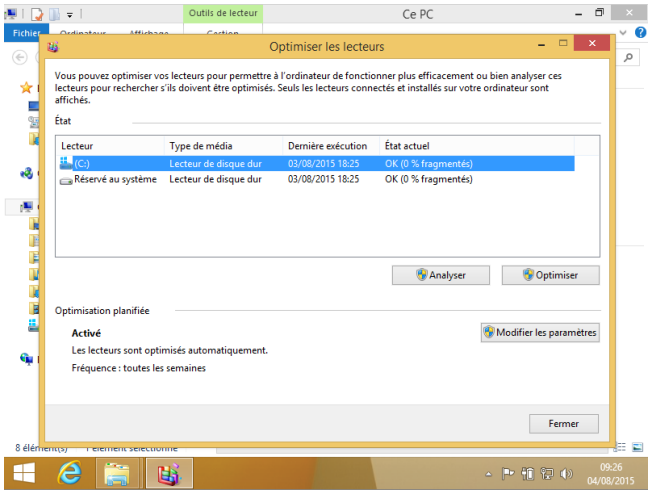


**Figure 2.7 – Défragmentation du disque `C:` sous Windows 8.**

## Réduire la place occupée par Windows

Une fois votre machine défragmentée, rendez vous dans *Gestion des disques*. Pour vous y rendre sous Windows 10 ou 8, retournez sur le bureau et faire un clic droit sur l'icône de Windows ; dans le menu contextuel, sélectionnez *Gestion du disque* (Fig. 2.8). Dans les versions antérieures de Windows, allez dans le *menu démarrer* → *Panneau de configuration (affichage classique)* → *Outils d'administration* → *Gestion de l'ordinateur*, puis cliquez sur *Stockage* → *Gestion des disques* dans la colonne de gauche.

Dans la fenêtre qui s'affiche, vous voyez les disques et partitions de votre système. Une partition, c'est un morceau de disque. En effet, il est possible de demander à la machine de considérer qu'un disque physique est découpé en plusieurs morceaux. Ça permet par exemple de stocker des choses très différentes comme deux systèmes d'exploitation différents, ou bien d'un coté un système de restauration et de l'autre le système utilisé, ou bien de séparer le système de vos données personnelles, etc. Repérez `C:` sur le graphique et faite un clic droit dessus. Dans le menu contextuel, sélectionnez *Réduire le volume.....*

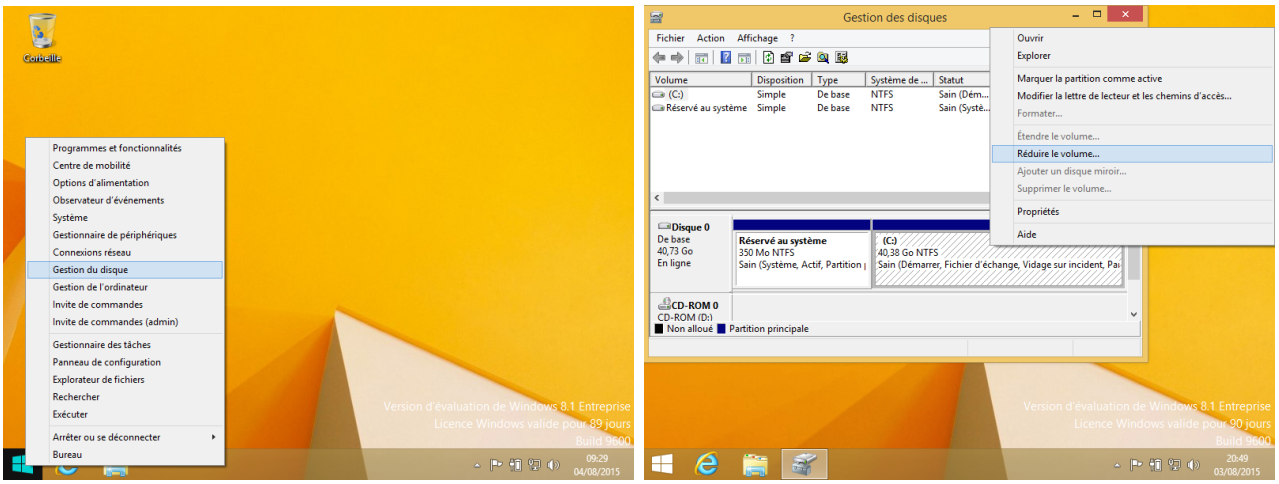


**Figure 2.8 – Diminuer l'espace alloué à Windows 8.**

Sous Windows Vista et supérieurs, la fenêtre qui s'affiche vous indique la taille du volume et l'espace libre qui peut être enlevé au volume (Fig. 2.9). Vérifiez l'espace requis pour votre version dans la documentation de Ubuntu (§. 2.1) et ajoutez y un espace suffisant pour pouvoir installer des programmes et stocker des données sous Ubuntu. Par exemple, pour la version 16.04 de Ubuntu, il est recommandé d'avoir 25Go d'espace. **Avec tous les programmes qui vous seront nécessaire pour la licence, vous pouvez compter qu'il vous faudra au minimum 40Go pour travailler confortablement**. L'espace disponible doit être supérieur à celui requis pour Ubuntu (et ses programmes/données) ; en effet, Windows aura besoin de garder un peu de place libre pour fonctionner correctement. Si l'espace libre est trop réduit, il va falloir reconsidérer votre idée d'installer Ubuntu à coté de Windows...

Sur la figure 2.9, le volume `C:` dispose d'environ 33Go d'espace libre, et l'espace nécessaire pour Linux (une vieille version) est de 15Go. Il est donc possible de prélever ces 15Go sans risque. On note cette valeur dans la case *Quantité d'espace à réduire*. Attention la valeur à indiquer dans la case est en Mo donc il faut ajouter 3 zéros. Il suffit ensuite de cliquer sur *Réduire* pour valider. Vous constatez sur le graphique qu'une zone libre est maintenant placée à la suite de `C:` (Fig. 2.9).

Notez que ces images ont été capturées il y a quelques années, sur votre machine vous devriez avoir un disque de plus grande capacité et l'espace nécessaire pour une version plus récente de Linux est désormais bien supérieur à 15Go.

### Connaître le type de partition de votre disque

Pendant que vous êtes dans *Gestion des disques*, faites un clic droit sur votre disque à gauche du graphique et jetez un œil aux *Propriétés* (Fig. 2.10). Dans l'onglet *Volumes*, notez le *Type de partition* de votre disque. Vous devez savoir si il est en GPT (*Table de partition GUID (GPT)*) ou si il utilise un MBR («*Master Boot Record*», *Secteur de démarrage principal*), ce sera utile par la suite.

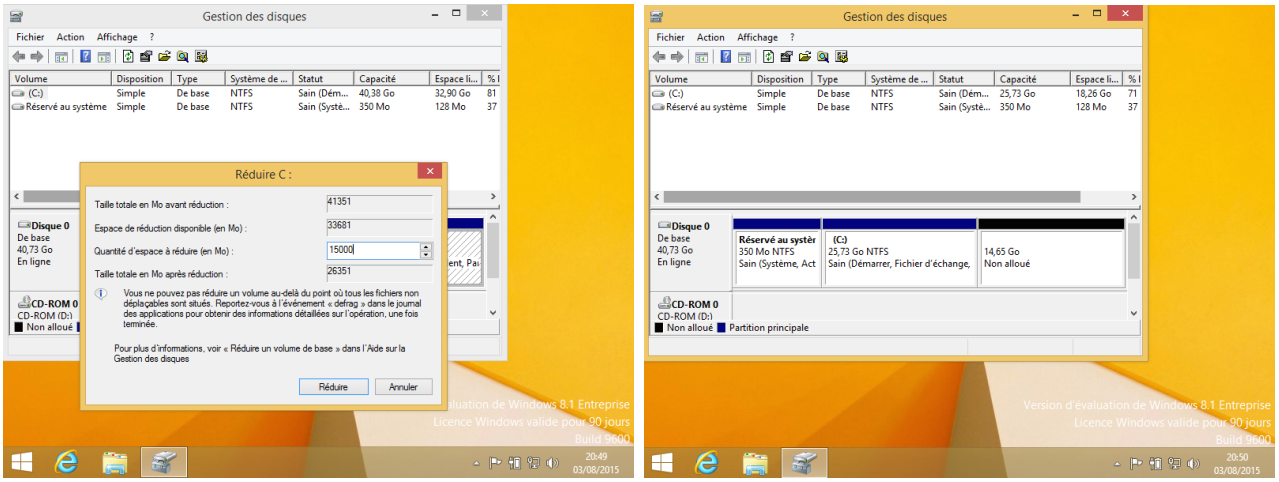


**Figure 2.9 – Choisir la quantité d'espace à prendre pour Ubuntu.**

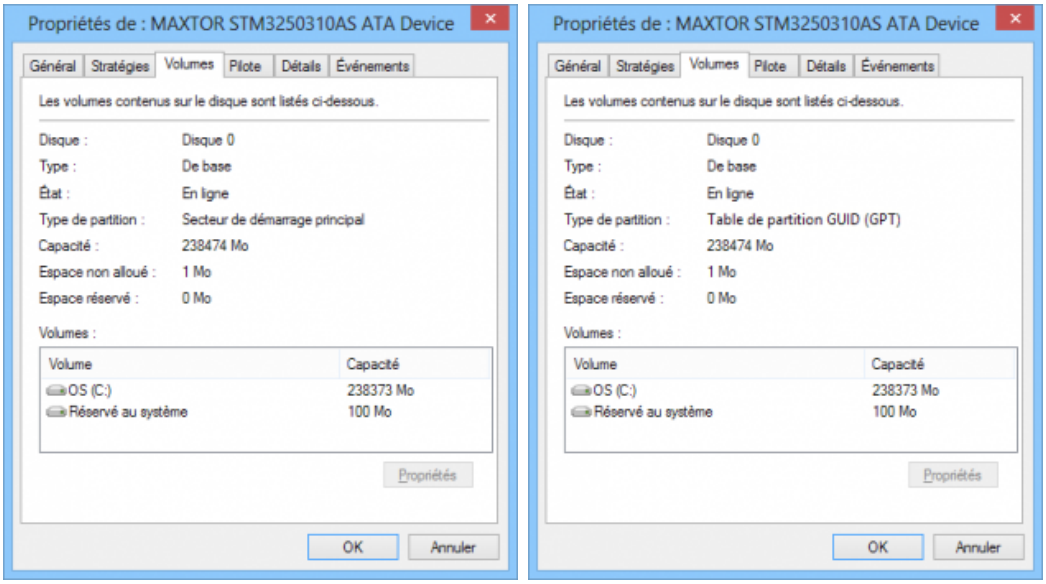


**Figure 2.10 – Type de partition du disque dur.**

### 2.3.5 Précautions pour ne pas endommager le démarrage de Windows

Nous avons libéré de l'espace pour installer Ubuntu sans risque à côté de Windows, mais restent des précautions à prendre pour prévenir tout problème concernant le démarrage de ces deux systèmes.

Pour trouver le système d'exploitation à démarrer lors de l'allumage de votre ordinateur, il y a ce qu'on appelle le *BIOS* (*Basic Input Output System*, «*système élémentaire d'entrée/sortie*»). Le BIOS est un ensemble de fonctions, contenu dans la *mémoire morte (ROM)* de l'ordinateur : une mémoire qui ne peut être modifiée et qui ne disparait pas quand on coupe l'alimentation.

Ce BIOS commence par identifier tous les éléments matériels de l'ordinateur (il vérifie que tout est là), puis il examine l'ordre de démarrage des périphériques (faut-il examiner le contenu d'un CD qui est dans le lecteur avant d'examiner le contenu du disque dur ?). Quand il a trouvé le périphérique à examiner en premier il cherche une information lui indiquant si il y a un système d'exploitation à démarrer, dans le cas contraire il passe au périphérique suivant.

Sur un disque dur, cette information se situe dans le premier secteur du disque que l'on nomme *MBR* (*Master Boot Record*, «*zone amorce*»). Ce MBR contient des informations sur le disque, la plus importante indiquant si le disque contient un système *amorçable* : votre Windows par exemple.

Si cette information est présente, le BIOS va lire une *routine* d'amorçage présente dans le MBR : un tout petit bout de code lui indiquant où se situent les instructions à lire pour démarrer le système d'exploitation.

Ces instructions constituent un *boot loader* [5] (un *chargeur d'amorçage*), un groupe d'instructions dont le rôle est d'aller chercher votre système d'exploitation pour le démarrer.

Sur certains PC actuels, c'est le *micrologiciel EFI* (et non pas le BIOS) qui est utilisé pour lancer le chargeur d'amorçage : l'EFI lit le *GPT* du disque (*GUID Partition Table*) un ensemble d'informations placé à partir du second secteur du disque, juste après le MBR, pour déterminer l'emplacement de la routine d'amorçage. Ce GPT est plus grand que le MBR et permet de stocker bien plus d'informations. Le MBR est toujours présent pour permettre à un ordinateur utilisant BIOS de démarrer quand même sur un disque utilisant un GPT.

Tout cela est peut être un peu compliqué, mais pour résumer, vous devez retenir qu'il se passe finalement beaucoup de choses avant même que votre système ne démarre.

Le *boot loader* de votre disque est prévu pour le moment pour ne démarrer que Windows. Quand on va ajouter Ubuntu, on va ajouter un programme d'amorçage nommé *GRUB* (*GRand Unified Bootloader*) qui affichera un menu et vous permettra de choisir si vous souhaitez démarrer avec Windows ou Ubuntu. Il peut arriver qu'à l'installation de GRUB des informations nécessaires pour trouver le boot loader chargé du lancement de Windows soient écrasées, et dans ce cas il ne démarre plus ! Pour vous assurer que vous pourrez réparer cela en cas de besoin, je vous conseille de prendre les précautions suivantes.

### Si vous êtes sous Windows 7/Vista/XP ou antérieur

Si le système était pré-installé sur l'ordinateur ET que le type de partition de votre disque dur utilise un MBR (§. 2.3.4) ET que vous n'avez jamais installé une distribution de Linux avec succès ET que vous n'avez jamais installé (ou réinstallé) Windows vous-même avec un CD non-fourni avec l'ordinateur, alors il faut faire une sauvegarde du MBR pour le restaurer si Windows ne démarre plus après l'installation de Ubuntu.

Reportez vous à la page suivante de la documentation de Ubuntu et imprimez-la au cas où vous auriez besoin d'effectuer la restauration du MBR : https://doc.ubuntu-fr.org/tutoriel/comment_sauvegarder_le_mbr.

### Si vous êtes sous Windows 8

Si Windows 8 était pré-installé sur votre ordinateur, il est fortement recommandé (par la documentation de Ubuntu) de créer une clef USB de réparation du démarrage *Boot Repair Disk* ; les instructions sont données sur cette page : http://sourceforge.net/p/boot-repair-cd/home/fr/.

Un des avantages de l'installation à partir d'un Live USB en mode persistant 2.2.2 est que l'on peut installer Boot Repair depuis le Ubuntu et le garder sur la clef.

Ensuite reportez vous à la page suivante de la documentation de Ubuntu et imprimez là pour suivre les recommandations données pour installer Ubuntu sans endommager votre Windows : https://doc.ubuntu-fr.org/uefi.

5. On le nomme aussi *boot strap loader*. *Bootstrap*, qui signifie littéralement «*tirant de botte*», est une référence au Baron de Münchhausen.

Dans cette documentation il est question d'aller dans le BIOS et de désactiver des choses dedans. Si vous ne comprenez pas ce que cela signifie, vous pouvez vous reporter au paragraphe 2.4.1 qui parle d'une autre modification à faire dans le BIOS.

## 2.4 Tester Ubuntu avec un Live CD ou une Live USB

Avant d'éteindre votre ordinateur, placer le CD de Ubuntu que vous avez gravé (liveCD) dans le lecteur de votre machine. Éteignez votre machine. Débranchez tous les disques externes de sauvegarde, les clefs USB (à part la clef liveUSB si vous en utilisez une) et retirez les cartes mémoires qui pourraient se trouver dans un lecteur de l'ordinateur. Je vous conseille de tout débrancher car cela diminuera le risque de commettre une erreur pendant l'installation et d'endommager le contenu de ces disques.

Maintenant redémarrez votre machine. Avec un peu de chance votre ordinateur va détecter le CD et vous devriez voir un message indiquant le démarrage du liveCD de Ubuntu. Vous devriez alors arriver sur un écran vous demandant de choisir entre essayer Ubuntu ou l'installer (Fig. 2.11).

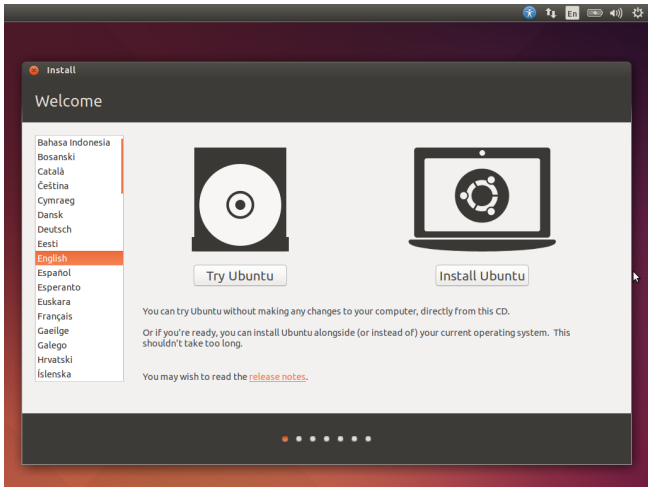


**Figure 2.11 – Démarrage du liveCD**

Si ce n'est pas le cas et que votre machine démarre directement le système Windows, il va falloir passer par une étape supplémentaire : il va falloir modifier l'ordre de boot dans le BIOS.

Si vous êtes sous Mac vous pouvez tester un liveCD Ubuntu (ne l'installez pas, sinon vous risquez d'endommager votre système MacOS), mais vous risquez de rencontrer des difficultés pour connecter ce Ubuntu lancé depuis le liveCD à Internet. Pour démarrer sur le CD gardez la touche Alt enfoncée au démarrage. Il se peut que votre Mac affiche un disque « *Windows* » : il s'agit en fait du liveCD, votre Mac ne l'a juste pas bien reconnu, cliquez dessus pour le démarrer.

### 2.4.1 Modifier l'ordre de boot dans le BIOS

Comme je vous l'ai dit précédemment (§. 2.3.5), le BIOS s'occupe de choisir l'ordre dans lequel les périphériques sont examinés. Si vous n'avez pas pu démarrer sur le Live CD ou la Live USB c'est

que votre BIOS doit examiner le disque dur en premier et, comme il y a trouvé un système à lancer, il n'a pas cherché plus loin.

Pour modifier l'ordre dans lequel les périphériques sont examinés, reportez vous à cette page de la documentation de Ubuntu : https://doc.ubuntu-fr.org/tutoriel/modifier_ordre_amorcage_du_bios.

Si jamais votre PC utilise *UEFI* (*Windows secure boot*), au lieu de vous donner accès au BIOS, la touche enfoncée (indiquée dans la page de documentation ci-dessus) va faire apparaître un message contenant les mots « *Secure boot* », « *UEFI* » ou « *wrong signature* ». Reportez vous alors aux instructions données au paragraphe 2.3.5.

### 2.4.2 Tester le son et le wifi

Maintenant que votre machine accepte de booter sur le CD (ou sur votre live USB) et que vous êtes arrivé à l'écran indiqué un peu avant (Fig. 2.11), commencez par choisir votre langue dans la colonne à gauche et cliquez ensuite sur *Essayer Ubuntu*.

Ubuntu démarre et vous voici sur le bureau. Cliquez sur le logo en haut à droite (zone de notification) correspondant à votre connexion internet (Fig. 2.12). Vous devez voir les réseaux wifi disponibles, vous pouvez choisir le vôtre et saisir la clef wifi quand elle vous sera demandée. Si vous êtes connecté par câble, la connexion devrait déjà être active (représentée par deux flèches ↑↓).


**Figure 2.12 – Indicateur de réseau sous Ubuntu.**

Si les réseaux wifi n'apparaissent pas, vérifiez qu'il n'y a pas un bouton (ou une combinaison de touches) sur votre machine pour activer et désactiver le wifi. Sinon, cliquez sur l'icône en haut du menu de gauche pour ouvrir le tableau de bord (Fig. 2.13). Tapez « *drivers* » dans le champ de recherche pour trouver l'utilitaire d'installation de pilotes additionnels. Cet utilitaire vous permettra de chercher un éventuel pilote propriétaire pour votre carte wifi et de l'installer.


**Figure 2.13 – Trouver l'utilitaire d'installation de pilotes additionnels sous Ubuntu.**

Quand vous avez réussi a connecter Ubuntu à Internet, vous pouvez tester le son et les autres périphériques éventuels de votre machine. Pour tester le son, vous pouvez cliquer sur l'indicateur de volume dans la zone de notification (non loin de l'icône réseau en haut à droite de l'écran),

puis sur le bouton lecture pour lancer le logiciel Rythmbox et choisir une chaîne de radio au hasard (Fig. 2.14).

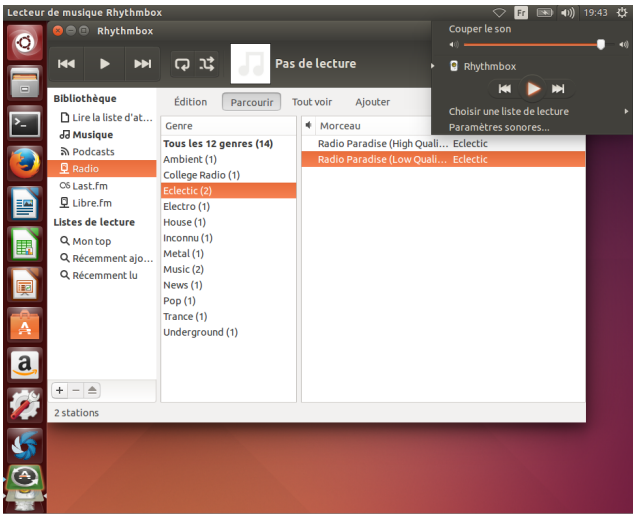


**Figure 2.14 – Tester le son avec le logiciel Rythmbox sous Ubuntu.**

Si pendant vos saisies vous remarquez que les caractères affichés ne correspondent pas aux touches de votre clavier, cliquez sur l'icône de langue à côté de l'icône réseau et sélectionnez *Paramètres de saisie du texte...* (Fig. 2.15). Cliquez sur le + en dessous du cadre *sources d'entrées à utiliser* et choisissez la langue correspondant à votre clavier. Enfin, n'oubliez pas de vérifier que la nouvelle langue est bien sélectionnée : si c'est le français, l'icône représente maintenant les lettres *FR*.

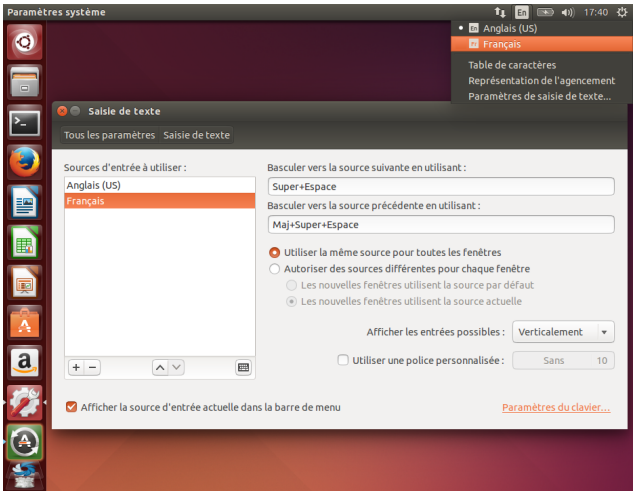


**Figure 2.15 – Modifier le mode de saisie au clavier.**

Continuez à explorer Ubuntu. Comme vous êtes sur le liveCD vous ne pouvez pas installer de logiciel et les données enregistrées disparaitront au redémarrage, mais vous pouvez vous familiariser avec l'interface graphique. Quand vous êtes prêt à installer Ubuntu, passez à la suite.

## 2.5 Installer Ubuntu

Maintenant que vous êtes prêts à démarrer l'installation, cliquez sur l'icône *Installer Ubuntu* et allons y !

Les étapes décrites ici sont celles observées lors de l'installation de Ubuntu 14.04 LTS. Il est possible que les étapes ne soient pas exactement les mêmes avec les prochaines versions d'Ubuntu. Si elles varient beaucoup et que vous hésitez sur les choix à faire pendant l'installation, n'hésitez pas à me contacter pour me le signaler et obtenir de l'aide.

Si vous êtes sous Mac, je vous recommande d'installer Ubuntu dans une machine virtuelle. Sautez au chapitre 7 pour voir comment installer votre système Ubuntu dans une machine virtuelle *VirtualBox*.

### 2.5.1 Étape 1 : Sélection de la langue

Ici rien de difficile, choisissez la langue avec laquelle vous vous sentez le plus à l'aise et passez à l'étape suivante.

### 2.5.2 Étape 2 : Préparation de l'installation

Si une liste des éléments requis est affichée, assurez-vous que toutes les coches sont vertes : que vous avez suffisamment de mémoire sur votre machine, que l'ordinateur est bien relié au secteur (si c'est un portable) et qu'il est bien connecté à Internet (ça vous permettra d'obtenir directement les dernières mises à jour).

Prenez soin de cocher les deux cases qui sont affichées dans tous les cas pour installer les mises à jour pendant l'installation et pour télécharger les logiciels tiers qui sont bien utiles pour lire les fichiers Flash, MP3 et autres médias.

« *Pourquoi il les installe pas sans demander ?* »

Ces logiciels ne sont pas des logiciels libres, vous pourriez ne pas être d'accord avec les conditions d'utilisation de ces produits indiquées dans leur licence, Ubuntu a donc besoin de votre accord avant de télécharger ces logiciels.

### 2.5.3 Étape 3 : Type d'installation

Le logiciel d'installation détecte automatiquement si un système est déjà présent sur votre ordinateur. Il vous propose alors d'installer Ubuntu à côté de votre système actuel ou de tout effacer pour installer Ubuntu à la place (Fig. 2.16).

Un autre choix s'offre également à vous : choisir vous-même où placer Ubuntu en sélectionnant *Autre chose*. Vous allez faire ce choix et aller à l'étape suivante car c'est l'occasion d'en apprendre un peu plus sur Linux.

#### Disques et partitions

L'écran suivant vous présente une image de votre disque (Fig. 2.17). La fenêtre du milieu présente les différents disques de votre machine et la manière dont ils sont partitionnés.

Linux a pour habitude de nommer les périphériques avec des codes qui permettent de savoir à quoi l'on a affaire (`sda1`, `sdb4`, `hda1`, ...). C'est un peu l'équivalent des `A:`, `B:`, `C:`, `D:` sous Windows,

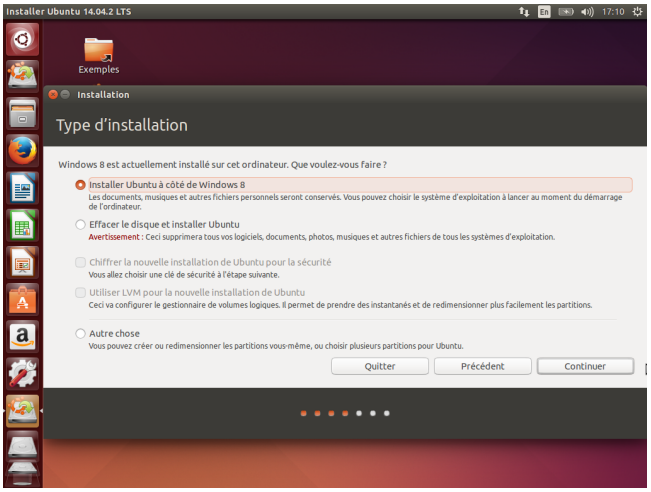


**Figure 2.16 – Installation de Ubuntu. Étape 3 : Type d'installation.**

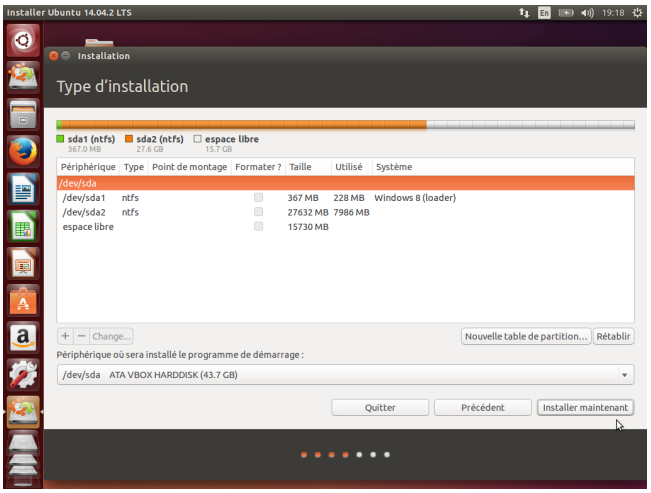


**Figure 2.17 – Installation de Ubuntu. Partitionner manuellement son disque.**

à part que sous Windows on ne sait pas si `C:` est une partition ou un disque tout entier et à partir de `D:` on ne sait pas vraiment ce qui est représenté...

Voici un petit résumé pour identifier la signification de ces codes :

*Sous Windows :*

- `A:` représente le premier lecteur de disquette[6].
- `B:` représente le second lecteur de disquette.
- `C:` représente le premier disque, ou première partition visible[7] du premier disque (c'est là que Windows est installé).
- `D:` représente le second disque, ou deuxième partition du premier disque si il y en a un.
- etc.
- La première lettre inutilisée par les disques (`D:` si il y a un seul disque avec une seule partition) est donnée au premier lecteur de CD/DVD.
- Les lettres suivantes sont attribuées aux périphériques branchés sur les USB.

6. Il est rare d'avoir un tel lecteur sur un ordinateur récent et remarquez que du coup la lettre `A:` n'est pas utilisée...

7. Une partition de restauration peut être cachée, elle n'est donc pas représentée par une lettre.

*Sous Linux :*

- `/dev/` représente un répertoire dans l'arborescence des fichiers 3.2 dans lequel sont stockés tous les périphériques (*device*). Les systèmes Unix représentent absolument tout sous la forme d'un fichier que l'ont peut lire ou sur lequel on peut écrire. Dans ce dossier les fichiers représentent donc les différents périphériques connectés à la machine.
- `/dev/fd0` représente le premier lecteur de disquette («*Floppy Disk*»).
- `/dev/fd1` représente le second lecteur de disquette.
- `/dev/sda` représentait à l'origine le premier disque SCSI. On trouve aujourd'hui le code `sd` également pour les disques SATA, SSD (*Solid-State-Drive*, «*mémoires flash*») et tous les disques/clefs USB.
- `/dev/sdb`, `/dev/sdc`, ... représentent les disques suivants quand il y en a plusieurs.
- `/dev/scd0` ou `/dev/sr0` désigne le premier lecteur CD/DVD SCSI
- `/dev/hda` désigne le disque maître sur le contrôleur IDE primaire.
- `/dev/hdb` désigne le disque esclave sur le contrôleur IDE primaire.
- `/dev/hdc` et `/dev/hdd` désignent respectivement les disques maître et esclave sur le second contrôleur.

Les numéros ajoutés après ces codes (`/dev/sda1`, `/dev/sda2`, ...) indiquent les différentes partitions d'un disque.

### S'assurer qu'on a de l'espace libre

Si vous avez plusieurs disques sur votre ordinateur. Repérer le disque sur lequel vous souhaitez faire l'installation. Si ce disque est vide vous pouvez commencer à créer les partitions comme expliqué un peu plus loin.

Sinon,

- Si vous avez Windows Vista ou supérieur, vous devez déjà avoir préparé un espace libre (comme sur la figure 2.17) : vous pouvez passer à la création des partitions.
- Si vous n'avez pas pu réduire la partition Windows (c'était le cas sous Windows XP) et souhaitez le conserver, vous devez réduire la partition maintenant. Sélectionnez la partition contenant le système Windows : vous devez avoir repéré sa taille et son format précédemment (§. 2.3.4). Cliquez sur *change...* (surtout pas -). La colonne *Utilisé* indique la taille occupée par des données, vous ne devez surtout pas réduire la partition en dessous de ce nombre sinon les données seront écrasées. Réduisez la partition en laissant suffisamment d'espace libre en plus de l'espace utilisé pour que Windows puisse fonctionner correctement. Vous devez prélever la taille recommandée pour le fonctionnement de Ubuntu.
- Si vous souhaitez effacer tout le contenu du disque pour installer Ubuntu (toute information non sauvegardée au préalable sera perdue), vous pouvez sélectionner les partitions existantes et les supprimer avec le bouton -.

### On partitionne !

Dans l'espace libre nous allons créer plusieurs partitions. Pour commencer, cliquez sur le +.

La première partition que nous allons créer est un *espace d'échange* ou *swap*. Quand la mémoire de l'ordinateur est saturée, pour éviter un plantage, le système peut décider de décharger une partie de cette mémoire en écrivant les informations sur le disque dur. C'est à ça que sert le swap : c'est une zone où le contenu de la mémoire est recopié quand il n'y a plus assez de place. Le swap sert également quand on veut placer un ordinateur en hibernation. Comme la mémoire ne peut garder les informations que si elle est sous tension, pour placer l'ordinateur en hibernation le contenu de la mémoire est recopié dans le swap.

Puisque le swap doit pouvoir stocker tout le contenu de la mémoire, il doit avoir une taille au moins équivalente à celle de la mémoire. Autrefois, quand les ordinateurs avaient peu de mémoire, le swap devait même pouvoir contenir plusieurs fois la taille de la mémoire et on le plaçait en début de disque[8]. Aujourd'hui, les mémoires sont bien plus grandes et il arrive que le swap ne soit jamais utilisé à part pour l'hibernation. Sur ma machine j'ai donc décidé de placer le swap à la fin de l'espace libre et je lui ai donné la même taille que la mémoire, c'est à dire 2Go (Fig. 2.18).

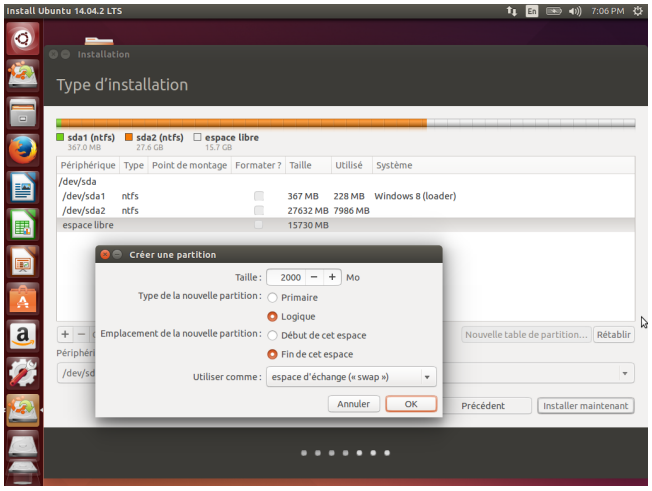


**Figure 2.18 – Installation de Ubuntu. Ajout d'une partition swap.**

La documentation de Ubuntu donne la recommandation suivante pour le choix de la taille du swap :

> *« Votre ordinateur dispose de 1 Gio de RAM ou plus ?*
> *Allouez un espace d'échange de 1× à 1,5× la taille de votre RAM ;*
> *Votre ordinateur dispose de moins de 1 Gio de RAM ?*
> *Allouez un espace d'échange de 1,5× à 2× la taille de votre RAM. »*

La deuxième partition servira à installer Ubuntu et nous allons lui attribuer tout l'espace libre encore disponible. Cette partition sera au format `ext4` et le point de montage sera `/` (Fig. 2.19).

*« C'est quoi ext4 ? »*

Il s'agit d'un format au même titre que `ntfs` ou `fat32`. Si tout cela vous est étranger, voici quelques repères concernant des formats courants que vous pouvez rencontrer :

**Fat32** C'est un format qui présente l'avantage d'être reconnu par la majorité des produits (ordinateurs, consoles de jeux, boitiers multimédias, télévisions, ...). Il est très bien accepté par Windows, MacOS et Linux. Son seul inconvénient est d'être limité à des fichiers de 4Go maximum (et la capacité totale du disque ne doit pas dépasser 2To).

8. Un disque dur possède un bras qui se déplace à la surface du disque pour lire son contenu. La lecture de données situées vers le début du disque est plus rapide car elle demande des mouvements plus réduits de ce bras.

**NTFS** C'est un format propriétaire de Microsoft qui succède à Fat en supprimant ses limitations. Sur les systèmes MacOS, la lecture d'un disque NTFS ne pose pas de problème mais l'écriture nécessite quelques manipulations supplémentaires. Ce format est correctement reconnu en lecture et écriture sur les versions récentes de Linux.

**exFAT** C'est un format propriétaire de Microsoft. La limite de taille de fichier et de disque est bien plus élevée. Il est compatible avec les dernières version de MacOS et peut être supporté par Linux en installant un pilote propriétaire additionnel.

**HFS+** Aussi nommé *MacOS étendu*, c'est le format utilisé sur les ordinateurs Macintosh. Il n'a pas de limite de taille de fichier. Il n'est pas du tout compatible avec Windows mais l'est avec Linux.

**ext4** C'est le format utilisé par Linux. Sous Windows et MacOS il est nécessaire d'installer un programme additionnel pour pouvoir lire et écrire sur un disque avec ce format. Les formats `ext2` et `ext3` sont d'autres formats plus anciens qui ont précédé `ext4`.

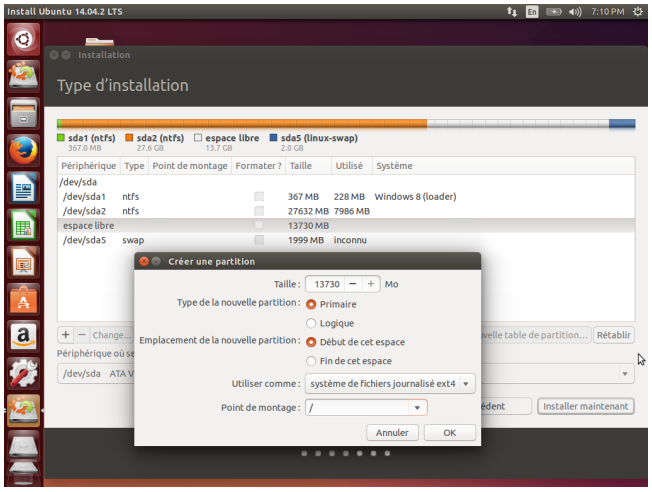


**Figure 2.19 – Installation de Ubuntu. Ajout d'une partition pour /.**

« *Et c'est quoi un point de montage ?* »

Comme je l'ai dit précédemment, les systèmes basés sur Unix considèrent que tout est fichier. Un répertoire (*directory* en anglais, nommé *dossier* sur d'autres systèmes) est également un fichier qui a juste la particularité de contenir d'autres fichiers. Même les périphériques sont représentés comme des fichiers à l'intérieur d'une arborescence dont la racine est nommée `/`.

Chaque fois qu'un fichier est contenu dans un autre (un répertoire donc), on le signale en ajoutant un symbole `/`. Par exemple, `/home/martin/image/monlogo.gif` indique que depuis la racine du disque `/` on trouve un répertoire `home` qui contient un répertoire `martin` qui contient un répertoire `image` qui contient une image `monlogo.gif`.

Comme un ordinateur peut posséder plusieurs disques et plusieurs partitions par disque, cette arborescence peut se retrouver éclatée en plusieurs parties. Pour savoir quelle partie de cette arborescence se situe sur quelle partition, on précise le *point de montage*. Comme ici je n'ai créé qu'une seule partition, je précise que c'est la racine `/` que je vais y placer.

En présence d'un autre disque, j'aurais pu y placer par exemple le répertoire `/home` qui contient les comptes de tous les utilisateurs (c'est à dire leurs données personnelles) pour les séparer du reste du système. Dans ce cas, tout les répertoires placés à la racine `/` auraient été dans la première partition, sauf `/home` et tous ses sous-répertoires qui auraient étés sur la seconde partition.

La liste déroulante présente les points de montage les plus courants mais le champ de texte montre qu'on à la possibilité de choisir le point de montage que l'on veut et qu'il est possible de répartir l'arborescence des fichiers comme on le souhaite sur les différents disques et partitions.

« *Je vois type de partition primaire et logique, c'est quoi ?* »

Un disque ne peut contenir au maximum que 4 *partitions primaires* (le MBR placé sur le premier secteur du disque n'a que 4 emplacements pour stocker des adresses de partitions). Si on veut plus de partitions sur un disque il faut alors remplacer une des partitions primaires par une *partition étendue*. Cette *partition étendue* est, elle, capable de contenir plusieurs *partitions logiques* (aussi appelée *partitions secondaires*). Si vous choisissez *partition logiques*, l'outil de partitionnement créera automatiquement une partition étendue pour la contenir.

Ces partitions logiques ne se distinguent pas des autres pour un utilisateur ou pour un programme utilisateur. Le BIOS, lui ne reconnait directement que les partitions primaires et certains systèmes d'exploitation (comme Windows) exigent d'être placés sur une partition primaire. Linux, lui, peut être installé sur les deux types de partitions.

On peut considérer qu'une partition étendue est une partition primaire d'un type particulier, mais quand on compte les partitions sur un disque, on considère généralement les partitions étendues à part et on ne les comptabilise pas parmi les partitions primaires. Par exemple, sur la figure 2.20 on voit un disque qui contient 3 partitions primaires (sda1, sda2, sda3) et une partition étendue (sda4). Cette partition étendue contient 6 partitions logiques, ce qui fait un total de 9 partitions (3 primaires et 6 logiques) visibles par l'utilisateur de la machine. Le système Windows XP est installé sur sda2 car Windows exige d'être placé sur une partition primaire. En revanche nous voyons que Ubuntu est installé sur une partition logique.

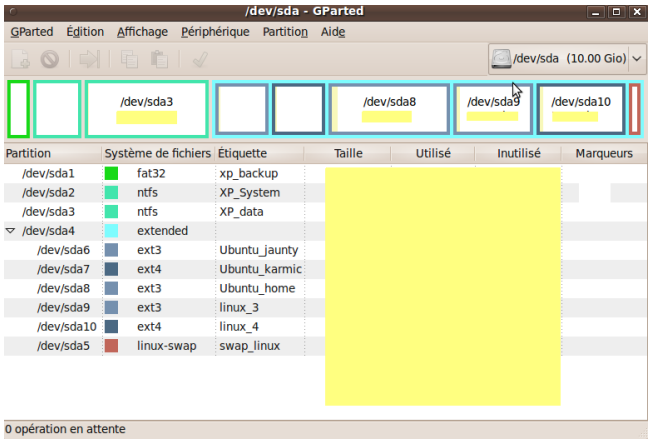


**Figure 2.20 – Représentation d'un disque et de ses partitions.**

Maintenant, vous devriez avoir vos deux partitions (swap et ext4 pour /) en plus (éventuellement) des partitions Windows (Fig. 2.21). Vous pouvez passer à l'étape suivante en cliquant sur *installer maintenant*.

### 2.5.4 Étape 4 : Emplacement géographique

Rien de compliqué ici. Indiquez votre emplacement qui sera utilisé pour régler votre fuseau horaire.

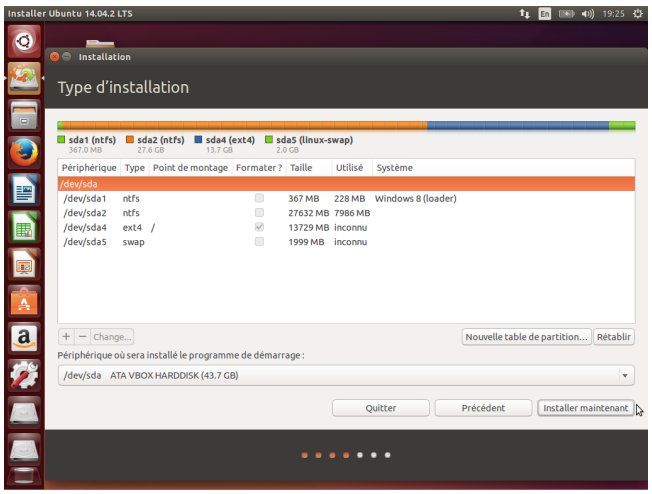


**Figure 2.21 – Installation de Ubuntu. Résultat final du partitionnement.**

### 2.5.5 Étape 5 : Disposition du clavier

Si vous êtes sur un PC vous ne devriez rien avoir à changer au réglage par défaut, mais vérifiez quand même que le clavier indiqué correspond bien au vôtre.

Si vous êtes sous Mac avec un clavier Français, choisissez le clavier : *Français - Français (Macintosh)*.

Le cadre de texte en dessous est là pour vous permettre de tester votre clavier. Si vous avez un clavier non conventionnel, vérifiez que les lettres, symboles, majuscules affichées correspondent bien aux touches que vous avez actionnées. Dans le cas contraire, vous n'avez pas sélectionné le bon clavier.

### 2.5.6 Étape 6 : Identité

Maintenant il va vous falloir donner votre nom, choisir un nom d'utilisateur et un mot de passe. Comme pour tout autre ordinateur vous n'êtes pas obligé de lui donner votre identité, un pseudonyme fera très bien l'affaire.

Sachez juste que le nom d'utilisateur et le mot de passe seront nécessaires fréquemment : pour installer un programme, modifier un réglage, etc. Choisissez donc un nom rapide à taper et un mot de passe que vous ne risquez pas d'oublier.

Le nom de votre machine sera utile pour vous identifier sur un réseau donc choisissez un nom qui ne soit pas trop long et pas trop commun.

On vous propose *ouvrir la session automatiquement* : cela signifie que la machine ne vous demandera pas votre mot de passe avant d'afficher votre bureau et vos documents. Si vous souhaitez sécuriser votre machine pour éviter qu'un tiers[9] n'accède directement à vos données, évitez cette option.

9. Mon chat sait démarrer l'ordinateur et faire des manipulations indésirables en piétinant le clavier et le touchpad de mon Mac mais il n'a pas encore réussi à taper le mot de passe ;-).

### 2.5.7 Étape 7 : Importation des données de Windows

Cette option peut éventuellement apparaitre si Ubuntu a détecté votre système Windows. Choisissez ce que vous désirez importer et validez.

La collecte d'informations est terminée, patientez quelques minutes pendant que l'installation se déroule. A la fin un message vous demande de redémarrer la machine pour quitter le liveCD et passer sur la version d'Ubuntu qui a été installée sur votre machine.

## 2.6 Le GRUB au démarrage

Si vous avez conservé Windows, au démarrage de la machine vous devez voir un écran noir avec le menu de démarrage de GRUB proposant les différentes options de démarrage : Ubuntu, Windows et différentes autres options de démarrage (Fig. 2.22). Si vous avez installé Ubuntu comme unique système, il vous faudra garder la touche *Majuscule* enfoncée pour faire apparaitre le menu de GRUB. Vous pouvez utiliser les touches indiquées pour sélectionner et démarrer le système de votre choix.


**Figure 2.22 – Menu de GRUB proposant les différents démarrages possibles.**

Vous pouvez modifier la manière dont se comporte GRUB : le temps pendant lequel le menu est affiché, le système démarré par défaut, etc. Vous trouverez des informations sur ces points dans la documentation de Ubuntu, mais attention, une mauvaise manipulation sur GRUB peut endommager le démarrage de votre système.

Vous pouvez également afficher le menu de GRUB dans votre machine virtuelle. Cliquez dans la fenêtre au départ pour être sûr que les saisies clavier soient bien capturées par la machine virtuelle et appuyez sur *Majuscule*. Si vous venez d'installer Ubuntu et que vous n'avez pas encore de données personnelles dessus, c'est l'occasion de faire des expérimentations avec GRUB ! Vous ne risquez rien, si vous plantez totalement le démarrage de Ubuntu, au pire il suffira de le réinstaller.

# 3 Premiers pas avec Ubuntu

Vous devriez maintenant avoir une copie de Ubuntu installée sur votre ordinateur. Dans ce chapitre nous allons découvrir ce système, à commencer par le gestionnaire de bureau Unity qui est assez différent de l'interface graphique de Windows (ou Mac) à laquelle vous êtes habituée. Nous nous familiariserons avec différents logiciels fournis dans cette distribution et découvrirons quelques spécificités de Linux, notamment le principe des dépôts avec la logithèque.

Si vous avez choisi de taper votre mot de passe pour ouvrir votre session, vous voyez un écran d'accueil vous demandant de le saisir. Faites le et vous arrivez sur le bureau de *Unity*.

## 3.1 Organisation de l'interface graphique

Unity est l'environnement de bureau de Ubuntu. Vous devez voir une barre en haut de l'écran que l'on nomme simplement *barre de menus* et à gauche une colonne présentant diverses icônes que l'on nomme *le lanceur*.

A droite de la *barre de menus* se situe la *zone des indicateurs* ou *zone de notifications*. Vous vous êtes déjà familiarisés avec l'*indicateur de réseau*, l'*indicateur de saisie de texte* et l'*indicateur audio* (§. 2.4.2). Vous pouvez voir également un indicateur de niveau de batterie, une horloge (qui affiche le calendrier en cliquant dessus), et enfin l'*indicateur de session*.

En cliquant sur l'indicateur de session, vous trouverez les options pour arrêter votre session (vous retournez alors à l'écran d'ouverture de session avec la demande de mot de passe), mettre l'ordinateur en veille ou l'arrêter. Vous y trouverez aussi une option permettant d'accéder aux paramètres système (paramétrage de l'affichage, du clavier, de la souris, des comptes utilisateurs, du son, du réseau, etc.).

Au niveau du *lanceur*, nous avons déjà vu la première icône qui permet de lancer le *tableau de bord*. Différentes icônes en bas du tableau de bord vous permettent de filtrer les résultats d'une recherche selon différent thèmes courants. D'autres options de filtrage sont disponibles en cliquant sur *Filtrer les résultats* à droite.

Utilisez le champ de recherche pour rechercher le *Terminal*. Ce Terminal est tellement important qu'il sera l'objet de tout le chapitre 4 et vous en aurez besoin très souvent. Vous pouvez donc ajouter son icône au lanceur avec un cliqué-glissé. Une autre solution est de démarrer le programme en cliquant dessus, puis de faire un clic droit sur son icône dans le lanceur et de choisir *Conserver dans le lanceur*.

Pour supprimer une icône il suffit de faire un clic droit dessus et de choisir *Retirer du lanceur*. Pour la déplacer il suffit d'utiliser un cliqué-glissé. Pour lancer une application, un seul clic sur une icône du lanceur suffit. Chaque application ouverte est signalée par un triangle blanc à gauche de

son icône dans le lanceur. Plusieurs triangles à gauche d'une icône indique que plusieurs fenêtres de l'application sont ouvertes. Une coche supplémentaire à droite signale que la fenêtre de l'application est affichée au premier plan.

Pendant que le Terminal est ouvert, approchez votre curseur de la barre de menus en haut de l'écran. Vous voyez apparaitre la barre de menu de la fenêtre (Fichier, Edition, Affichage, ...). Les utilisateurs de MacOS ne seront pas trop dépaysés, à part le fait que le menu n'apparait qu'au survol. Par contre, les utilisateurs de Windows seront un peu déroutés de voir cette barre de menu placée en haut de l'écran et non en haut de la fenêtre : c'est une habitude à prendre.

Les boutons ×, − et □ sont placés à gauche (et non à droite comme sous Windows) sur la fenêtre du programme en mode fenêtre. Le bouton □ permet de maximiser la fenêtre : dans ce cas elle occupe toute la zone centrale de l'écran. Les boutons ×, − et □ sont alors superposés à la barre de menus et n'apparaissent qu'au survol de la souris. En cliquant de nouveau sur □ on peut restaurer la fenêtre. Le bouton − permet de minimiser l'application ; pour la restaurer il suffit de cliquer sur son icône dans le lanceur. Le bouton × ferme l'application.

Le raccourci clavier `alt+tab` vous permet de passer rapidement d'une fenêtre ouverte à l'autre, `ctrl+Super+D` vous permet de réduire toutes les fenêtres d'un coup pour accéder au Bureau (§. 3.4.1).

## 3.2 Arborescence des fichiers sous Linux

Nous avons déjà évoqué un peu l'arborescence des fichiers dans le chapitre précédent (§. 2.5.3). Pour rappel, dans les systèmes Unix, tout est considéré comme un fichier. Un répertoire est également un fichier mais qui a la particularité de contenir d'autres fichiers.

Pour que l'utilisateur puisse reconnaître un fichier, chaque fichier a un nom (*filename* en anglais). Un nom de fichier peut contenir quasiment tous les caractères sauf le *slash* `/` qui sert de délimiteur entre les répertoires. Pour le repérer dans l'arborescence des fichiers, chaque fichier peut être repéré par son *chemin d'accès* (*pathname* en anglais), une suite de répertoires séparés par le caractère `/`.

Un caractère `/` au début d'un chemin d'accès représente la racine de l'arborescence : si il est présent on dit que le chemin est *absolu*. Si le chemin d'accès ne commence pas par `/`, alors c'est un chemin *relatif* et il indique que ce chemin doit être parcouru depuis la position courante.

Il arrive que l'on emploie indifféremment *filename* ou *pathname* mais attention leur signification est différente (§. 3.1). La confusion vient du fait que si le fichier est dans le répertoire courant, son chemin d'accès relatif se réduit à son nom de fichier.

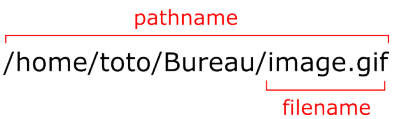


**Figure 3.1 – Différence entre *pathname* et *filename*.**

A sa création un répertoire contient déjà deux fichiers spéciaux `.` et `..` («*point*» et «*point-point*» en français, «*dot*» et «*dot-dot*» en anglais). Le fichier `.` désigne le répertoire courant et `..` désigne le répertoire parent. Ces fichiers spéciaux ne sont pas visibles quand on utilise l'explorateur de fichiers mais nous pourrons les voir dans le prochain chapitre quand nous utiliserons le *terminal* (nommé parfois la *console*).

### 3.2.1 Explorateur de fichiers

L'explorateur de fichiers est accessible en cliquant sur l'icône *Fichiers* du lanceur (Fig. 3.2). Dans la colonne de gauche divers raccourcis vous permettent d'accéder à votre *Dossier personnel* et à différents dossiers qui ont été créés pour vous. Dans *Périphériques* vous pouvez voir les différents périphériques branchés à votre machine et un raccourci vers *Ordinateur* qui vous amène à la racine de l'arborescence des fichiers (§. 2.5.3).

**Figure 3.2 – Icône de l'explorateur de fichiers sous Ubuntu.**

En cliquant sur *Saisir l'emplacement* dans le menu *Lancer* ou avec le raccourci `Ctrl+L` vous pouvez voir apparaitre le chemin d'accès. Si vous êtes dans *Ordinateur* vous voyez en effet que le chemin d'accès est `/`. Si vous allez maintenant dans votre dossier personnel vous voyez que son chemin d'accès est `/home/`$votrePseudo$ ($votrePseudo$ représente le nom que vous avez choisi).

Vous pouvez expérimenter en modifiant directement le chemin d'accès en haut de la fenêtre. En tapant `/home/`$votrePseudo$`/Images`, vous pouvez afficher votre dossier d'images. Tentez de taper un chemin fantaisiste comme `/home/`$votrePseudo$`/tardis` et vous constaterez qu'un message d'erreur signale que ce chemin n'existe pas. Maintenant depuis votre Dossier personnel, effacez le chemin affiché et tapez à la place `Images` (sans `/` devant) : vous venez de saisir un *chemin relatif* et vous vous retrouvez dans le répertoire `/home/`$votrePseudo$`/Images`.

Pour vous éviter de commettre des erreurs en effaçant des fichiers importants mais aussi pour ne pas encombrer la fenêtre en affichant des fichiers que vous n'avez pas forcément envie de voir, l'explorateur masque certains fichiers. Ces fichiers ont la particularité d'avoir un nom qui commence par `.`, c'est d'ailleurs à cela que l'explorateur les reconnait et qu'il sait qu'il doit les cacher. Utilisez le raccourci `Ctrl+H` pour les faire apparaitre. Pour les cacher à nouveau il suffit de saisir le raccourci à nouveau.

### 3.2.2 Répertoires principaux sous Unix et Linux

Voici maintenant une petite liste des répertoires les plus importants que vous trouverez sur la plupart des systèmes Unix ou Linux et qui vous seront utiles par la suite.

**`/`** Prononcé *root* ou *slash*, ce chemin représente la *racine* de l'arborescence des répertoires.

**`/home`** Comme je l'ai évoqué au chapitre précédent, `/home` est un répertoire contenant le répertoire personnel de chaque utilisateur.

**`/media`** Dans les distributions récentes de Linux[1], ce répertoire est utilisé comme *point de montage* pour les périphériques d'entrées/sorties tels que les CD/DVD, clefs USB ou disques dur externes. C'est à dire que si vous branchez une clef USB sur votre ordinateur, son contenu sera représenté par un répertoire lui même placé dans ce répertoire.

1. Ce répertoire n'est pas présent sur tous les systèmes Unix. Par exemple, sous MacOs, c'est le répertoire `/Volumes` qui sert de point de montage pour les périphériques d'entrées/sorties.

**/boot** Ce répertoire contient les fichiers nécessaires au démarrage de Linux. Remarquez la présence des fichiers de configuration de *GRUB*. Le fichier qui commence par `vmlinuz` contient une image du noyau et le fichier qui commence par `initrd` est une pseudo-partition utilisée lors du boot.

**/bin /sbin** Ces répertoires contiennent respectivement les commandes ordinaires et les commandes d'administration exécutées par l'administrateur du système (aussi nommé *root*). Nous utiliserons certaines de ces commandes dans le chapitre suivant.

**/etc** Ce répertoire (dont le nom signifie Et cetera !) contient les scripts de configuration pour tout le système.

**/dev /proc** Ces répertoires n'existent pas sur le disque dur. Il sont générés par le noyau au démarrage de l'ordinateur.
Dans `/dev`, chaque périphérique détecté est représenté par un fichier. Vous pouvez par exemple trouver des fichiers `/dev/ram` (pour la mémoire), `/dev/sda` ou `/dev/hda` (pour les disques).
Dans`/proc`, on trouve des fichiers d'informations et de statistiques qui permettent de consulter et parfois modifier les caractéristiques des processus en cours d'exécution et les paramètres du noyau. Un répertoire est créé pour chaque processus en fonctionnement avec son identifiant (PID) comme nom.

**/lib** Ce répertoire contient des *bibliothèques* («*library*»). Une bibliothèque est un ensemble de fonctionnalités tellement utiles et fréquemment utilisées qu'on les regroupe dans un fichier que l'on peut ensuite importer dans un programme qui en a besoin. On trouve dans ce répertoire des *bibliothèques statiques* (dont le nom se terminent par `.a`) qui sont importées dans le code du programme et compilées[2] avec lui, et des *bibliothèques dynamiques* (dont le nom se terminent par `.so` suivi d'un numéro de version.) qui contiennent du code ajouté au programme en cours d'exécution.

**/tmp** Ce répertoire est prévu pour stocker les fichiers temporaires.

**/usr** Ce répertoire contenait autrefois les fichiers des utilisateurs («*user*» en anglais), mais aujourd'hui il sert à stocker une copie des répertoires de la racine contenant tout ce qui n'est pas indispensable au moment du boot. Les répertoires `/usr/bin` et `/usr/sbin` contiennent d'autres programmes et outils d'administration du système, `/usr/lib` d'autres bibliothèques. Dans `/usr/local` on trouve une fois de plus des répertoires `bin`, `sbin`, `etc`, etc. ; ils contiennent les commandes non standard rajoutées par dessus l'installation du système.

**/usr/share/doc** Ce dossier contient les documentations de tous les programmes installés.

**/usr/share/man** Ce dossier contient les pages du *manuel* pour tout le système. Les fameuses *pages man* que nous apprendrons à utiliser au prochain chapitre.

**/usr/include** C'est là que vous trouverez les fichiers à inclure dans les programmes C que vous écrirez (dans le cours de Langage impératif C).

**/var** Ce répertoire contient les fichiers variables (qui changent souvent). Le répertoire `/var/log` contient les journaux (en anglais «*log file*» ou plus simplement «*log*») : des fichiers contenant des informations horodatées sur les activités du système. Le fichier `/var/log/auth.log` par exemple vous permet de voir qui a ouvert ou tenté d'ouvrir une session sur votre Linux. Vous y reconnaîtrez par exemple une phrase comme « *password check failed...* » indiquant une tentative

2. La compilation consiste à transformer un programme écrit par un humain et lisible par un humain (code source) en un fichier binaire (composé de 0 et de 1) exécutable par une machine.

de connexion avec un mauvais mot de passe : ce qui peut permettre par exemple de démasquer une tentative d'intrusion ou vous suggérer de choisir un mot de passe plus facile à taper/mémoriser !

## 3.3 Les dépots et la logithèque

Sous Linux, on peut obtenir des logiciels grâce aux *dépôts*. Vous pouvez voir les dépôts comme un principe similaire à l'App store de Apple, le Windows store de Microsoft, etc. à la différence que les dépôts contiennent des *paquets*. Un *paquet* est un morceau de logiciel. Certains *paquets* sont utilisés par plusieurs logiciels donc au lieu de stocker des logiciels complets et d'avoir des morceaux qui sont dupliqués plusieurs fois, les logiciels sont découpés en *paquets* uniques.

Un *paquet* est une archive (un fichier compressé) qui contient des fichiers informatiques ainsi que les instructions nécessaires pour les installer sur un système d'exploitation particulier en tenant compte des paquets déjà présents et en s'assurant que l'ensemble fonctionnera correctement après l'installation. Il existe plusieurs formats de paquets (Debian, RPM, ...), Ubuntu utilise des paquets Debian.

Pour installer ou désinstaller ces paquets, on peut utiliser un *gestionnaire de paquets* graphique ou en lignes de commandes. Le gestionnaire de paquets se connecte à différents *dépôts* pour connaître la liste des paquets disponibles et nous les proposer. Un paquet peut nécessiter la présence d'un autre paquet : on appelle cela une *dépendance*. Quand on choisit d'installer un logiciel, le gestionnaire vérifie la liste des paquets à installer et leurs dépendances pour installer tout ce qui est nécessaire.

La *logithèque* est le gestionnaire de paquet graphique par défaut de Ubuntu. Vous pouvez accédez à la logithèque en cliquant sur son icône (Fig. 3.3) dans la lanceur. Il est possible d'éditer la liste des dépôts utilisés par la logithèque dans *Édition/sources de logiciels...* sous Ubuntu 14.04 ou dans *Logiciels Ubuntu/Logiciels & mises à jour* sous Unbuntu 16.04. Dans l'onglet *Autres logiciels*, seuls les dépôts maintenus par la Fondation Ubuntu sont activés : cochez les dépôts qui ne sont pas encore activés (Fig. 3.4). Si au moment de fermer la fenêtre, Ubuntu vous propose une mise à jour du cache, acceptez en cliquant sur *Activer*.

**Figure 3.3 – Icône de la logithèque dans le lanceur de Unity.**

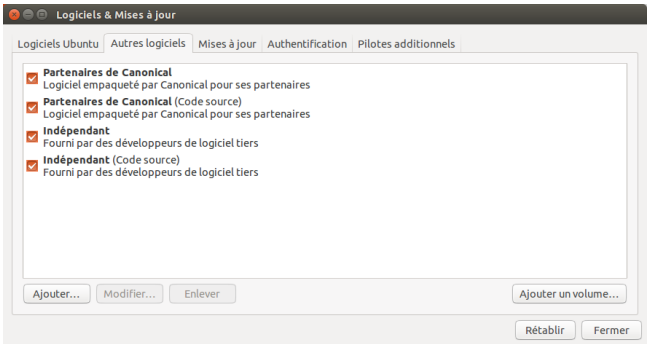


**Figure 3.4 – Logiciels & Mises à jour : activer tous les dépôts.**

Maintenant que tous les dépôts sont activés, vous pouvez installer les paquets nécessaires pour suivre les différentes cours de l'année. Pour trouver un paquet, cliquez dans le cadre de recherche en

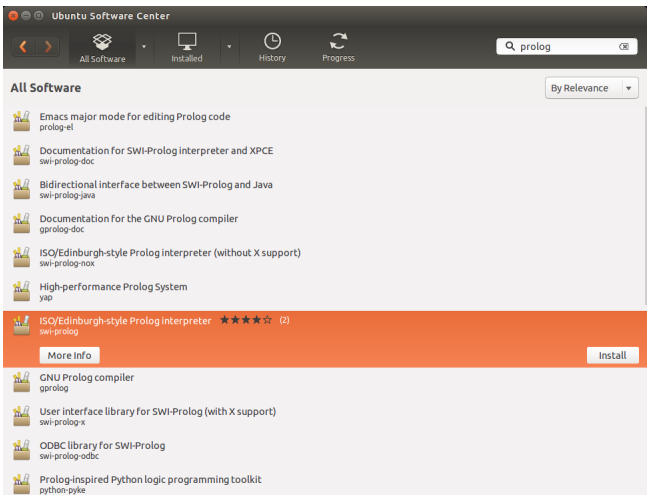


**Figure 3.5 – Liste des paquets correspondant à une recherche avec le mot clef « *prolog* ».**

haut à droite et saisissez le nom du paquet ou un mot clef (Fig. 3.5). Chaque paquet est présenté avec un titre et le nom du paquet (écrit en petit en dessous). En cliquant sur un titre, vous pouvez voir apparaitre deux boutons pour avoir plus d'informations sur ce paquet ou pour l'installer. Installez les paquets suivants :

**`dia gnome`** Un logiciel pour dessiner des schémas nécessaire pour le cours de programmation fonctionnelle.

**`emacs`** Un éditeur de code incontournable, à part si vous préférez utiliser l'éditeur Vi [3]. Vous pouvez installer la version GUI avec une interface graphique ou la version Terminal qui fonctionnera dans la console (voir le chapitre 4 pour découvrir ce qu'est la console).

**`Vim`** Une édition presque totalement compatible de l'éditeur Vi. Un éditeur de code incontournable, à part si vous préférez utiliser l'éditeur Emacs.

**`HexChat`** Un client IRC facile à utiliser dont nous verrons le fonctionnement un peu plus loin (§. 3.5.3).

Le *gestionnaire de mises à jour* utilise aussi les dépôts auxquels vous avez accès pour vérifier périodiquement si vous disposez des dernières versions de vos logiciels et bibliothèques.

## 3.4 Les manipulations à connaître

Je vais vous présenter maintenant quelques manipulations courantes dont vous aurez besoin fréquemment et qui seront utiles pour réaliser et envoyer les exercices des différents cours à distance.

### 3.4.1 Les raccourcis clavier

Vous n'êtes pas obligés d'utiliser des raccourcis clavier. Comme leur nom l'indique, ils servent juste à gagner du temps et il existe d'autres façons d'obtenir le même résultat en utilisant les menus.

3. L'utilisation de Emacs ou Vi demande un temps d'apprentissage non négligeable. Cependant, je vous recommande de faire cet effort. Si vous voulez travailler dans le domaine de l'informatique, il est bon de maitriser les bases d'au moins un de ces deux logiciels.

Cependant, ils font la différence entre un utilisateur averti et les autres. Voici une liste courte des raccourcis incontournables :

| | |
|---|---|
| **alt+tab** | permet de basculer rapidement d'une fenêtre d'une application à l'autre. |
| **Ctrl+Super**[4]**+D** | permet de réduire toutes les fenêtres pour afficher le bureau. |
| **Ctrl+C** | permet de copier un élément (bout de texte, image, fichier, répertoire, etc.). |
| **Ctrl+V** | permet de coller un élément que l'on a copié au préalable. |
| **Ctrl+X** | permet de couper un élément et de le conserver dans le presse papier. En utilisant l'action *coller*, cet élément pourra être placé ailleurs. L'élément est perdu si il n'a pas été collé avant l'action *couper* ou *copier* suivante. |
| **Impr**[5] | Prendre une capture de tout l'écran. |
| **Alt+Impr** | Capturer l'image d'une seule fenêtre. |
| **Maj+Impr** | Capturer l'image d'une zone à sélectionner avec le pointeur. |

Vous pouvez consulter la liste complète les raccourcis disponibles en cliquant sur l'indicateur de session et en sélectionnant *Paramètres système...* → *Clavier*, onglet *Raccourcis*. Pour modifier un raccourci existant, cliquez sur la combinaison de touches actuelle indiquée à droite et appuyez sur la nouvelle combinaison de touches.

Pour les utilisateurs d'un clavier Mac je propose d'utiliser `Fn+F12` pour remplacer la touche `Impr` dans *Capture d'écran*.

### 3.4.2 Faire une copie d'écran

Pour ceux qui l'ignorent encore : une copie d'écran c'est une image de ce qu'il y a à l'écran. Cela permet de montrer à d'autres ce que l'on voit, pour demander de l'aide, se faire dépanner ou pour les besoins d'un tutoriel ou cours comme ici.

Dans certains cours, y compris celui-ci, on vous demandera de faire une copie d'écran pour montrer le résultat que vous avez obtenu après une manipulation. Pour faire une copie d'écran, vous pouvez utiliser les raccourcis donnés au paragraphe précédent (§. 3.4.1). Vous pouvez, au choix, capturer tout l'écran, une seule fenêtre ou bien une zone particulière de l'écran. Dans ce dernier cas, un pointeur en croix s'affichera. Cliquez en haut à gauche de la zone que vous voulez capturer puis glissez le pointeur pour dessiner une zone rectangulaire ; relâcher le clic pour valider la capture. Enregistrez la capture dans votre compte (dans un dossier de votre `home`), vous pourrez ensuite l'envoyer par email ou la joindre à un exercice.

### 3.4.3 Organiser ses données

Pour vous y retrouver dans tous vos cours, vos exercices et données personnelles, prenez l'habitude d'organiser vos données. Je vous conseille de créer un répertoire[6] *Licence-info* (ou autre nom qui

4. Sous Windows, la touche `Super` est la touche avec le logo . Sous mac c'est la touche `cmd` de droite.

5. `Impr` désigne la touche Imprime écran (`Print Screen` ou `Prt Sc`). C'est généralement la première touche située à droite de la touche F12. Sur un clavier Mac, cette touche n'existe pas, il faut modifier le raccourci par défaut : cliquez sur l'indicateur de session et en sélectionnant *Paramètres système...* → *Clavier* onglet *Raccourcis* rubrique *Capture d'écran* pour utiliser une autre touche.

6. Dans le gestionnaire de fichiers, faites un clic droit dans la fenêtre là où vous souhaitez créer un répertoire. Sélectionnez *nouveau dossier* et donnez lui un nom.

vous conviendra) dans votre home (`/home/`*votrePseudo*`/Licence-info`) pour ranger tout ce qui concerne la Licence à distance. Créez un répertoire par cours et dans chacun de ces répertoires créez 3 répertoires : *cours*, *exercices-en-cours*, *exercices-rendus* (ou autre nom qui vous convient).

Dans *exercices-en-cours* vous pourrez placer tous vos tests, différentes versions plus ou moins abouties de vos exercices, alors que dans *exercices-rendus* vous pourrez mettre les exercices achevés et correctement présentés, prêts à être envoyés ou déjà envoyés. En effet, certains professeurs ne veulent pas recevoir tous les exercices en même temps, mais s'assurer que chaque chapitre est bien assimilé avant de passer au suivant : cela ne doit pas vous empêcher d'avancer et de stocker les exercices prêts à être envoyés. Dans le cas ou un professeur perdrait des exercices suite à une défaillance de la plateforme, vous aurez toujours une copie prête à être renvoyée.

Pensez à nommer vos exercices à rendre avec votre nom et éventuellement votre numéro d'étudiant (exemple : `Martin_Miggs_186514_chap1.txt`), ou bien à mettre votre nom à l'intérieur du fichier si c'est un fichier texte. Comme dans une salle de classe réelle, une copie sans nom ne pourra pas être notée...

Vous remarquerez que je propose des noms avec des tirets. Il n'y a pas si longtemps, les seuls caractères qui permettaient de nommer des fichiers en étant sûr qu'ils seraient lisibles sur les différents systèmes étaient des lettres, nombres ou tirets (`-`, `_`) et les noms devaient être assez courts. Aujourd'hui ce n'est plus vraiment nécessaire mais il est bien de garder l'habitude de nommer les fichiers avec des noms courts et sans trop de caractères spéciaux.

Si vous voulez nommer des fichiers ou dossiers avec une date (exemple : `photo_11-01-2010`, `photo_13-05-2015`, `photo_15-09-2012`, etc...), prenez l'habitude de commencer par l'année, puis le mois, puis le jour pour qu'elles soient classées dans l'ordre chronologique (exemple : `2010-01-11_photo`, `2012-09-15_photo`, `2015-05-13_photo`, etc...).

Si vous numérotez des exercices (exemple : `exo_1`, `exo_10`, `exo_11`, `exo_2`, `exo_3`, ...), sous Unity et KDE, ils seront placés dans le bon ordre mais pas forcément sur les autres gestionnaires de bureaux ou systèmes. Pensez à ajouter des `0` pour qu'ils soient toujours classés dans le bon ordre (exemple : `exo_01`, `exo_02`, `exo_03`, `exo_10`, `exo_11`, ...).

### 3.4.4 zipper / dézipper

Si votre exercice est constitué de plusieurs fichiers (un texte d'explication, plusieurs fichiers contenant un programme, des images, etc.), il sera sans doute nécessaire de tout rassembler dans une archive compressée. On appelle parfois cela un *fichier zip* du nom du format `.zip` qui est un format de compression.

Pour créer une archive compressée sous Ubuntu, c'est très simple. Dans le gestionnaire de fichiers, regroupez les différents documents dans un dossier[7]. Profitez en pour donner à ce dossier un nom qui vous identifie (nom, prénom, numéro étudiant). Faites un clic droit sur le dossier et sélectionnez *Compresser...*.

Dans le menu qui s'affiche, vous voyez que le format proposé par défaut est `.tar.gz`, un format issu du monde unix : `.tar` est un format d'archive (tous les fichiers sont réunis en un seul) et `.gzip` est un format de compression. Si la personne à qui vous souhaitez envoyer le document travaille

7. Il est possible de sélectionner directement des fichiers et de les compresser, mais lorsque le destinataire décompressera l'archive, les fichiers se retrouveront en vrac dans le répertoire de destination. Créer un dossier est préférable pour garder tous vos documents rassemblés et éviter d'agacer le destinataire (c'est à dire la personne qui va vous noter).

sous Linux (ou connait Linux) vous pouvez utiliser ce format. Sinon, préférez le format `zip` qui est plus connu des utilisateurs de MacOS et Windows.

## 3.5 Logiciels utiles pour communiquer

Pour suivre vos cours et obtenir de l'aide en cas de besoin, vous devez savoir naviguer sur Internet, envoyer des emails ou utiliser un chat. Je vous donne ici quelques indications sur les outils dont vous disposez.

### 3.5.1 Navigateur

Le navigateur utilisé par Ubuntu est Firefox. Son icône est déjà présente dans le lanceur. Vous pouvez bien sûr installer un (ou plusieurs) autre(s) navigateur(s) si vous le souhaitez. Je ne m'attarderais pas sur son utilisation, si vous n'avez jamais utilisé de navigateur, mais sur quelle planète vivez vous et comment avez-vous eu accès à ce cours ? !

### 3.5.2 Courrier électronique

L'icône du logiciel de mail n'est pas dans le lanceur au départ. Utilisez le tableau de bord pour chercher « *mail* », vous verrez que Ubuntu propose *Thunderbird* comme logiciel de courrier par défaut. Ajoutez son icône au lanceur pour y accéder plus vite la prochaine fois et démarrez le.

Suite à votre inscription à Paris8, vous disposez d'une boite email fournie par la faculté. Nous allons configurer cette boite email pour que vous puissiez recevoir facilement votre courrier.

Quand Thunderbird vous propose de créer une adresse email, passez cette étape pour configurer vos boites existantes. Indiquez votre nom : c'est sous ce nom que vos correspondants recevront vos messages alors évitez les pseudonymes pour que les enseignants puissent vous identifier. Indiquez votre adresse email qui vous a été fournie par la fac, c'est une adresse du type `prenom.nom@etud.univ-paris8.fr` qui vous a été fournie quand vous avez activé votre compte numérique. Si ce n'est pas fait, faites le à cette adresse : `https://numerique.univ-paris8.fr/index.php?page=act`. Indiquez ensuite votre mot de passe et cliquez sur continuer(Fig. 3.6).

Dans la fenêtre suivante il vous faudra surement modifier les éléments présentés dans les différents champs car Thunderbird ne sera pas capable de détecter les réglages automatiquement. Les informations doivent être les suivantes (Fig. 3.7) :

| Serveur entrant : IMAP | |
|---|---|
| **Nom d'hote du serveur :** | imap.etud.univ-paris8.fr |
| **Port :** | 993 |
| **SSL :** | SSL/TLS |
| **Authentification :** | Mot de passe normal |
| Serveur sortant : SMTP | |
| **Nom d'hote du serveur :** | smtp.etud.univ-paris8.fr |
| **Port :** | 587 |
| **SSL :** | STARTTLS |
| **Authentification :** | Mot de passe normal |

Votre identifiant entrant et sortant correspond à votre identifiant de boite mail qui vous a été donné à sa création. C'est normalement la première lettre de votre prénom suivie de votre nom, sans espace, point, ni autre caractère.


**Figure 3.6 – Création de la boite email fournie par Paris8.**

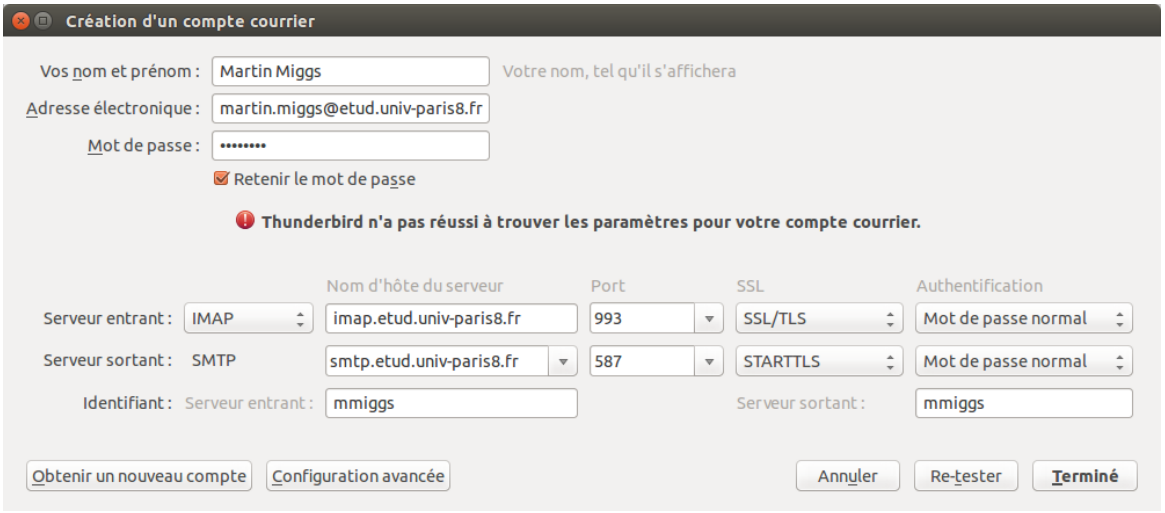


**Figure 3.7 – Configuration de la boite email fournie par Paris8.**

### 3.5.3 Tchat IRC

Vous avez surement déjà utilisé un logiciel de tchat (MSN, skype, etc.) mais connaissez-vous *IRC* ? Si ce n'est pas le cas, il est temps de découvrir de quoi il s'agit. *IRC* (*Internet Relay Chat*) est un protocole de communication textuelle qui permet de dialoguer entre deux utilisateurs ou en groupe et qui permet aussi de faire du transfert de fichiers, mais avec IRC, pas de conversation audio ou vidéo.

« *Mais c'est quoi ce truc super limité ?!* »

Certes cela peut sembler limité mais IRC a été créé en 1988 et est toujours très utilisé. Il existe des salons de discussion dédiés pour presque tous les sujets informatiques [8] et il existe même un *chat*

8. Il y a notamment différents canaux IRC dédiés à Ubuntu sur le serveur freenode : https://doc.ubuntu-fr.org/salons_irc.

*officieux* réservé aux étudiants de la licence à distance de paris8 !

Dans la logithèque vous avez installé le logiciel *HexChat*, il est temps de le démarrer. Utilisez le tableau de bord pour chercher « *chat* », glissez l'icône de HexCaht dans le lanceur pour y accéder plus rapidement la prochaine fois et démarrez le.

Au démarrage HexChat doit afficher la liste des réseaux IRC (Fig. 3.8). Si ce n'est pas le cas, vous pouvez l'afficher en allant dans le menu *HexChat/Liste des réseaux...*. Choisissez un pseudo : les second et troisième choix sont là au cas ou votre pseudo serait déjà utilisé.

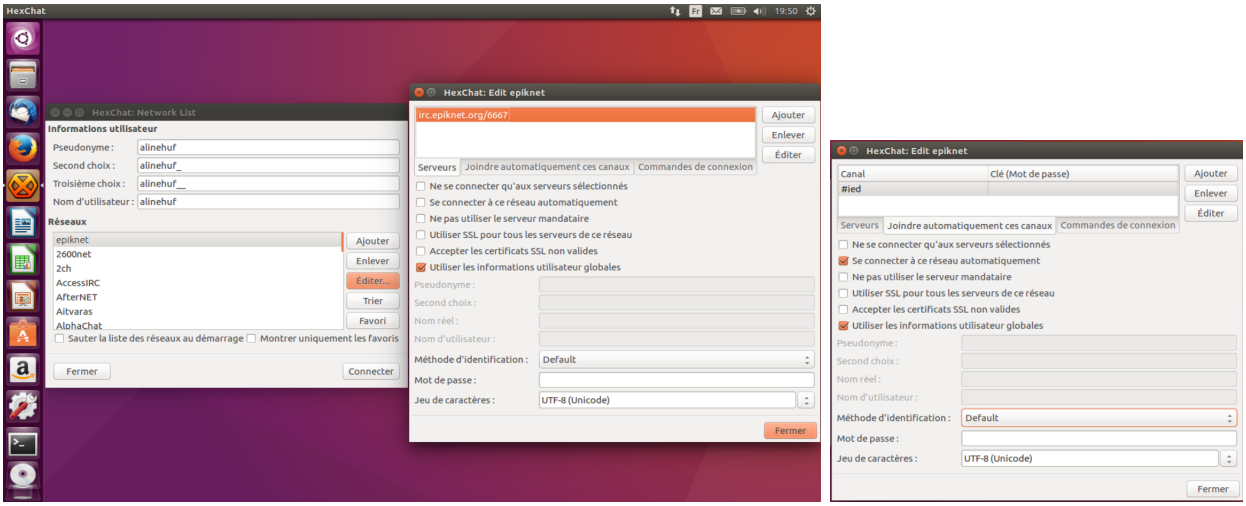


**Figure 3.8 – Configuration de HexChat sous Ubuntu.**

En bas de la fenêtre des réseaux, cliquez sur *Ajouter* et indiquez `Epiknet` comme nom de réseau (validez en appuyant sur Entrer). Cliquez ensuite sur *Éditer* pour paramétrer le serveur. Dans la fenêtre *Serveur pour Epiknet* indiquez `irc.epiknet.org/6667` (`irc.epiknet.org` c'est le serveur, `6667` le numéro de port qu'il utilise). Vous pouvez cocher la case *Se connecter à ce réseau automatiquement* pour aller sur le Tchat de l'IED automatiquement au démarrage de HexChat. Vérifiez aussi que *UTF-8 (Unicode)* est bien indiqué dans le champ *jeu de caractères*. Maintenant, cliquez sur l'onglet *Joindre automatiquement ces canaux*, cliquez sur *Ajouter* et indiquez `#ied`.

Dans *Paramètres/Préférences* vous pouvez régler les *Alertes* pour avoir un signal visuel ou auditif quand un nouveau message arrive. Dans *Général* vous pouvez choisir les messages à afficher en cas d'absence ou quand vous quittez le salon de discussion. Je vous laisse découvrir les autres réglages possibles.

Sous MacOS, vous pouvez utiliser Xchat Azure, disponible dans l'Apple Store. La configuration se fait dans *Fichier/Liste des réseaux...*.

Avant de vous connecter, voici quelques recommandations sur le bon usage d'IRC. L'usage est de se connecter à un salon de discussion et de rester connecté tant que votre ordinateur est allumé. Du coup, les autres utilisateurs ne sont pas forcement tout le temps derrière leur écran prêts à voir votre message et à vous répondre. Il peuvent aller faire un tour, et revenir 15 minutes, 1 heure ou plusieurs heures plus tard, apercevoir votre message et décider d'y répondre. De même quand vous répondez à un message, celui qui a posé la question peut, entre temps, s'être éloigné de son ordinateur. Par contre, une fois la conversation engagée, les choses se déroulent comme sur un chat classique.

En bref, soyez patients et restez connectés. Si le salon est abondamment fréquenté, vous aurez plus de chance d'y croiser quelqu'un qui est bien derrière son ordinateur. Participez ! C'est en vous aidant mutuellement que vous ferez vivre ce chat officieux, vous nouerez des contacts et vous rendrez vos études à distance beaucoup moins solitaires. Le chat officieux est un peu comme une salle de travail. On est parfois attiré par une conversation entre d'autres étudiants, on lève les yeux de son travail et c'est l'occasion d'apprendre/découvrir quelque chose, d'être informé d'une actualité importante sur l'IED ou d'avoir la réponse à une question qu'on n'avait même pas pensé à se poser.

Le chat fonctionne 24h/24 et si vous êtes insomniaque, vous y croiserez peut-être d'autres étudiants même à une heure tardive.

On peut utiliser différentes commandes sur IRC pour effectuer différentes tâches que l'on soit *opérateur* (administrateur), *voice* (assistant opérateur) ou simple utilisateur du canal. Vous trouverez les différentes commandes utilisables avec une petite recherche sur Internet, mais voici celles qui peuvent vous être utiles dans l'immédiat :

**/msg** *pseudo*
: permet d'envoyer un message privé à l'utilisateur qui se nomme*pseudo*.

**/me** *message*
: permet d'afficher un message avec votre pseudo. Par exemple si Martin tape `/me fait un test`, il verra s'afficher « *Martin fait un test* ».

**/away** *raison*
: permet d'afficher un message d'absence si quelqu'un s'adresse à vous.

**/nick** *autre-pseudo*
: permet de changer son pseudo en cours de session.

**/msg Themis REGISTER** *mot-de-passe email-valide*
: permet de réserver son pseudo pour que personne d'autre ne puisse l'utiliser. Themis vous enverra un mot de passe par email. Il faut ensuite confirmer la réservation avec `/msg Themis confirm` *code-reçu-par-email*. Vous pourrez préciser votre mot de passe dans la configuration du canal. Sinon, Themis exigera votre mot de passe à chaque démarrage. Il faudra alors lui répondre avec la commande `/msg Themis IDENTIFY` *mot-de-passe*.

## 3.6 Éditeur de texte vs. Traitement de texte

Vous avez surement déjà utilisé un *traitement de texte* comme Word (Microsoft), Page (Apple) ou LibreOffice, mais connaissez vous la différence entre un *traitement de texte* et un *éditeur de texte* ?

Un traitement de texte est utile pour mettre en page du texte, ajouter des couleurs, utiliser différentes fontes ou polices typographiques, etc. Ce type d'éditeur dispose d'une interface WYSIWYG (« *What you see is what you get* », ce qui signifie littéralement « *ce que vous voyez est ce que vous obtenez* ») permettant de visualiser directement ce que sera le résultat final. Le fichier qui est enregistré par ce type de logiciel contient non seulement le texte que vous avez tapé, mais aussi des informations sur la structure (entête, pied de page, marges) du document et, séparément ou non, de présentation (taille, couleur, etc).

Un éditeur de texte n'est pas prévu pour mettre en page du texte. Le fichier enregistré ne contient généralement que du *texte pur* (ou *texte brut*). L'éditeur de texte peut afficher des couleurs : il met certains mots ou caractères en couleur pour faciliter la lecture. Il permet aussi, par exemple,

de grossir/rétrécir le texte pour le confort du lecteur. Cependant, ces informations ne sont pas enregistrées dans le fichier final.

On peut parfois utiliser l'expression *éditeur de code* pour désigner un *éditeur de texte* : c'est parce qu'on utilise parfois un éditeur de texte pour écrire du code informatique. Comme le code informatique est du texte pur, un éditeur de texte peut faire l'affaire pour le consulter. Cependant un *éditeur de code* propose des fonctionnalités supplémentaires qui simplifient l'écriture et la modification du code. L'éditeur de code permet entre autre de mettre en couleur les mots clefs de différents langages informatiques, d'indenter automatiquement le code (décaler certaines lignes pour en faciliter la lecture), de commenter/dé-commenter un ensemble de lignes et propose bien d'autres fonctionnalités prévues pour améliorer le confort de travail du programmeur (complétion automatique des mots clefs, fermeture automatique des parenthèses ou accolades ouvertes, etc.).

Souvent les éditeurs de texte proposent quelques fonctionnalités de base comme la mise en couleur et certains peuvent être enrichis grâce à des pluggins pour les transformer en véritables éditeurs de code (comme gEdit) : il est donc souvent difficile de définir si un logiciel est à classer parmi les éditeurs de texte ou de code, gEdit par exemple peut être classé dans les deux catégories.

Sous Ubuntu, vous disposez de *libreOffice* comme traitement de texte et de *gedit* comme éditeur de texte (ou éditeur de code basique). Vous avez aussi installé emacs ou GVim (§. 3.3) qui sont de vrais éditeurs de code.

### 3.6.1 Fichiers de texte brut

Les fichiers de texte pur sont généralement enregistrés avec l'extension `.txt`. Cependant les fichiers contenant du code informatique (`.c` pour le langage C, `.py` pour le langage python, etc.) sont également écrits en texte pur : ils ne contiennent pas d'information sur la mise en page du code.

Ouvrez le fichier `comptine.txt`[9] avec gedit. Vous pouvez constater qu'il contient quatre lignes de texte pur. L'éditeur présente le texte sans marges et sans représentation d'une page éventuelle sur laquelle le texte pourrait être positionné.

Vous pouvez aussi constater que le texte est présenté dans une fonte à chasse fixe : chaque caractère à la même taille que les autres et les caractères se retrouvent parfaitement alignés les uns au dessus des autres d'une ligne à l'autre. Quand on programme, ceci est utile pour localiser une erreur. Un interpréteur ou un compilateur peuvent vous indiquer une erreur en précisant la ligne et la colonne où elle se situe.

Ouvrez maintenant le fichier *comptine.c*. C'est un petit programme (sans intérêt) en langage C. Vous constatez que cette fois le texte est présenté avec des couleurs. Ces couleurs ne sont pas indiquées dans le fichier : gedit détecte qu'il s'agit d'un programme C grâce à l'extension `.c` à la fin du nom du fichier et prend l'initiative d'ajouter des couleurs pour améliorer la lisibilité du code. Renommez le fichier en remplaçant `.c` par `.txt` et ouvrez le à nouveau : vous constatez que la couleur a disparu.

Ouvrez le fichier `comptine.py`. C'est un autre petit programme (aussi sans intérêt) en Python cette fois. Vous voyez que la mise en couleur n'est pas tout a fait la même, elle s'est adaptée en fonction de l'extension `.py` du fichier. Vous pouvez, en utilisant le menu *Affichage/Mode de coloration...*, choisir un autre langage : vous faites croire à gedit que ce programme est dans un

9. Ce fichier et les autres fichiers cités par la suite vous ont été fournis avec ce cours.

autre langage et il va utiliser une autre mise en couleur en fonction des mots clefs de cet autre langage.

### 3.6.2 Fichiers de texte formaté

Un texte formaté est un texte qui contient des informations de mise en page : ce n'est plus du texte pur car certaines informations contenues dans le fichier ne sont pas affichées telles quelles mais utilisées pour la mise en forme.

Pour mieux comprendre, cliquez sur le fichier `comptine.html` pour l'ouvrir en laissant Ubuntu choisir le programme à utiliser pour le lire. Il s'agit d'une page web et c'est surement le navigateur Firefox qui s'est ouvert pour afficher la même comptine de quatre lignes que nous avons vu précédemment.

Maintenant, ouvrez ce même fichier avec gedit : vous devez constater la présence de balises (des codes entourés des caractères < et >) qui n'étaient pas visibles dans le navigateur. Les balises `<p>` et `</p>` signalent le début et la fin d'un paragraphe. Le couple de balises `<b>...</b>` permet de mettre le texte en gras. Vous constatez donc que ces balises sont utilisées pour définir la structure du texte aussi bien que sa mise en forme.

Les documents `comptine.docx` et `comptine.odt` ont été créés respectivement avec *Word* (Microsoft) et avec *libreOffice*. Si vous avez Word sous la main, vous pouvez ouvrir le premier document et constater qu'il contient toujours la même comptine. L'autre document contient également cette comptine.

Si vous tentez d'ouvrir l'un de ces documents avec gedit vous devriez voir des caractères bizarres et obtenir un message d'erreur signalant que ce document contient des caractères non valides. Ces documents contiennent des informations de mise en forme en plus du texte, mais apparemment, ces informations ne se résument pas à des balises comme dans un document html.

Pour comprendre de quoi sont composés ces documents, ouvrons le logiciel emacs. Remarquez en passant que la page affichée par défaut dans emacs propose un *emacs tutorial* (le contenu est en français) pour apprendre à utiliser ce logiciel [10]. Ouvrez le document `comptine.odt` (`File/Open file...`) : emacs vous affiche une liste de fichiers, c'est le contenu de `comptine.odt`. `comptine.odt` est donc une archive contenant plusieurs fichiers.

Si vous cliquez sur `content.xml` vous verrez qu'il contient la comptine avec de nombreuses balises décrivant la structure du document. Appuyez sur `Ctrl+X` suivi de la touche ← pour revenir à la liste des fichiers. Dans le fichier `style.xml` vous trouverez des balises décrivant la mise en forme.

Ouvrez maintenant le document `comptine.docx` : vous constatez que le contenu est similaire. La structure du document est contenue dans `word/document.xml`, les styles de mise en forme dans `word/style.xml`.

### 3.6.3 Textes formatés, sans édition

Il existe également des formats de textes formatés qui ne sont pas éditables (ils ne peuvent pas être modifiés, ou ne sont pas prévus pour l'être). Ces formats sont conçus pour s'assurer qu'un

10. Ce tutoriel contient tout ce qu'il faut pour prendre en main Emacs. A l'occasion, prenez le temps de lire ce tutoriel et de découvrir Emacs. Je vous y encourage.

document aura exactement la même mise en page, exactement le même aspect sur toutes les plateformes. En effet, si vous ouvrez un fichier Word sur différentes versions du logiciel Word, sur openOffice, libreOffice ou un autre logiciel capable de lire ce format de fichiers, vous constaterez que la mise en page peut varier et être parfois altérée de manière indésirable. Ceci n'arrive pas avec les formats PostScript ou PDF par exemple.

PostScript (extension `.ps`) est un langage de description de page mis au point par Adobe. La plupart des éléments d'un document PostScript sont décrits sous forme vectorielle. Au lieu de définir chaque pixel, un format vectoriel décrit des formes abstraites (cercle, carré, etc.) ce qui permet une description plus concise et permet de ne pas perdre en qualité quand le document est agrandi.

Si vous tentez d'ouvrir le fichier `comptine.ps` avec gedit vous constaterez que c'est un fichier texte, mais il contient un code qui décrit une page et ce qui doit être affiché ou imprimé dessus.

Le PDF (*Portable Document Format*, extension `.pdf`) est un autre langage de description de page également mis au point par Adobe. Si vous ouvrez le document `comptine.pdf` avec gedit vous constaterez que la description n'est pas lisible : en fait, le pdf est un format compressé [11].

### 3.6.4 Latex

Je ne pouvais pas parler des éditeurs et traitements de texte sans vous glisser un mot concernant *LaTeX* (à prononcer «*Latek*» et surtout pas «*latec-se*»). LaTeX est un langage et un système de composition de documents créé par Leslie Lamport en 1983 et destiné à simplifier l'utilisation du système logiciel de composition de documents *TeX* (prononcez Tek) créé par le mathématicien et informaticien Donald Knuth en 1977.

LaTeX est un outil très puissant pour créer des documents (articles, rapports, mémoires de thèse) à l'aspect professionnel [12]. L'utilisateur de Latex, écrit son texte et utilise des balises pour décrire la structure de son document et sa mise en forme. Vous pouvez voir dans le dossier `exemple_latex` un document `lorem-ipsum.tex`. Vous pouvez l'ouvrir avec gedit pour voir ce qu'il contient. Vous pouvez constater qu'il contient un texte avec des commandes commençant par `\` et finissant parfois par des accolades (parfois des crochets) encadrant des paramètres précisant l'action de la commande. Voici la signification de quelques unes d'entre elles :

**`\documentclass`** définit le type de document, le format des pages et la taille moyenne du texte.

**`\begin{document} ... \end{document}`**
délimitent le début et la fin du document

**`\maketitle`** génère un titre à partir du titre indiqué dans `\title`, du nom d'auteur quand il y en a un, et de la date si elle est fournie (sinon il prend automatiquement la date du jour).

**`\section`** **`\subsection`**
indiquent les titres de sections et de sous-sections.

**`\tableofcontents`** génère automatiquement une table des matières à partir des titres utilisés dans le document.

11. PDF est un format ouvert, il est possible de connaître la manière dont un PDF est codé en consultant la norme `ISO 32000-1:2008 PDF`
12. Ce support de cours a été créé avec LaTeX.

| | |
|---|---|
| **\label** | définit un repère auquel on pourra faire référence ensuite. |
| **\ref** | permet de placer une référence à un repère placé ailleurs dans le texte. |
| **%** | signale un commentaire. Le texte précédé de ce caractère est ignoré par LaTeX jusqu'au prochain saut de ligne. |

Un document LaTeX doit être compilé pour transformer ce code en une mise en page soignée. Pour cela, il est possible d'utiliser un logiciel graphique qui fera ce travail pour vous, comme TexMaker que vous pouvez trouver dans la logithèque et qui vous présentera d'un coté le fichier avec ses balises et de l'autre le résultat obtenu (Fig. 3.9).

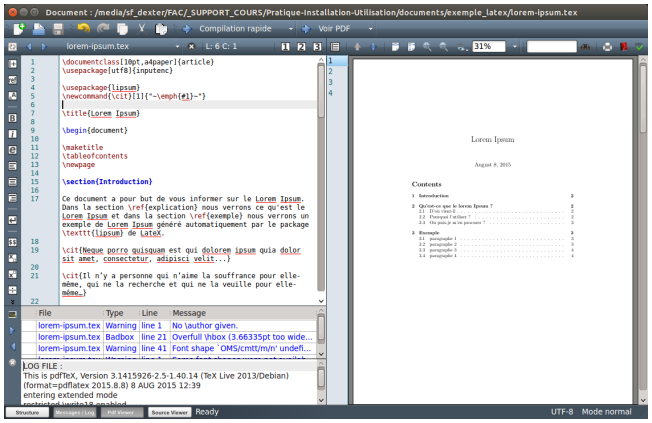

**Figure 3.9 – Interface graphique de TexMaker.**

Sinon, il faut utiliser un éditeur de texte brut et taper les commandes de compilation à la main. Pour vous faire comprendre comment cela fonctionne, je vais vous montrer comment le faire à la main. Commencez tout de même par installer TexMaker qui installera en même temps que lui-même tous les paquets nécessaires pour compiler un document LaTeX .

Ouvrez le logiciel *Terminal* en cliquant sur l'icône que je vous ai fait ajouter dans le lanceur (§. 3.1) et placez vous dans le dossier `exemple_latex`. Pour cela, faite un petit bond vers le chapitre suivant (§. 5.1.1) pour savoir comment vous déplacer dans l'arborescence des fichiers avec le terminal [13].

Pour compiler le document latex et en faire directement un fichier PDF, utilisez la commande suivante (`pdflatex` suivi du nom du fichier) :

```
$ pdflatex lorem-ipsum.tex
```

LaTeX va créer différents fichiers : `lorem-ipsum.aux` contient les *labels* qui ont été récoltés tout au long du document, `lorem-ipsum.toc` («*toc*» pour «*table of content*») va récolter les titres qui ont été rencontrés et `lorem-ipsum.log` est un journal dans lequel LaTeX inscrit se qui s'est passé pendant la compilation (c'est en gros ce que l'on voit s'afficher dans le terminal).

Dans le fichier pdf qui a été généré, vous constatez avec déception que la table des matières est vide et dans l'introduction les numéros des sections sont remplacés par des ??. C'est tout à fait normal : LaTeX a collecté les titres et les labels tout en générant le PDF, il n'avait donc pas fini de les collecter quand il a dû commencer à créer le document (Fig. 3.10).

Relancer la commande une deuxième fois : à la seconde exécution il utilise le contenu des fichiers `.aux` et `.toc` qui existent déjà et vous constatez que votre PDF contient maintenant une jolie table

13. Si vraiment vous n'êtes pas à l'aise avec le terminal, je vous conseille d'étudier le chapitre 4 avant de revenir à cette section concernant LaTeX.

des matières générée automatiquement et que le texte de l'introduction fait référence aux bons numéros de sections (Fig. 3.11).

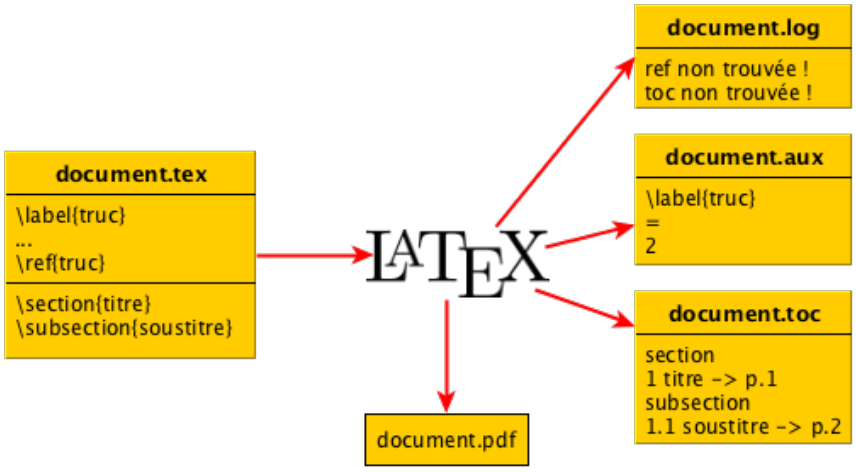


**Figure 3.10 – Première compilation d'un document avec pdflatex.**

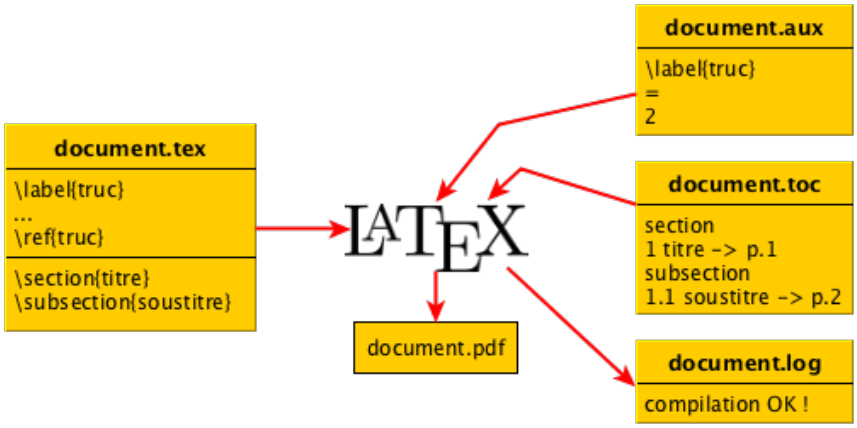


**Figure 3.11 – Seconde compilation d'un document avec pdflatex.**

On peut aussi générer un fichier PostScript avec LaTeX. Pour cela il faut commencer par compiler le document avec la commande `latex` pour générer une image du document : un fichier `.dvi` (on compile 2 fois pour générer la table des matières et les références). Ensuite on peut utiliser un outil pour générer le fichier PostScript : par exemple la commande `dvips` (Fig. 3.12).

```
$ latex lorem-ipsum.tex
$ latex lorem-ipsum.tex
$ dvips lorem-ipsum.dvi
```

Vous constatez qu'un fichier `.ps` a été généré et il contient le même document que dans le fichier `.pdf` précédent.

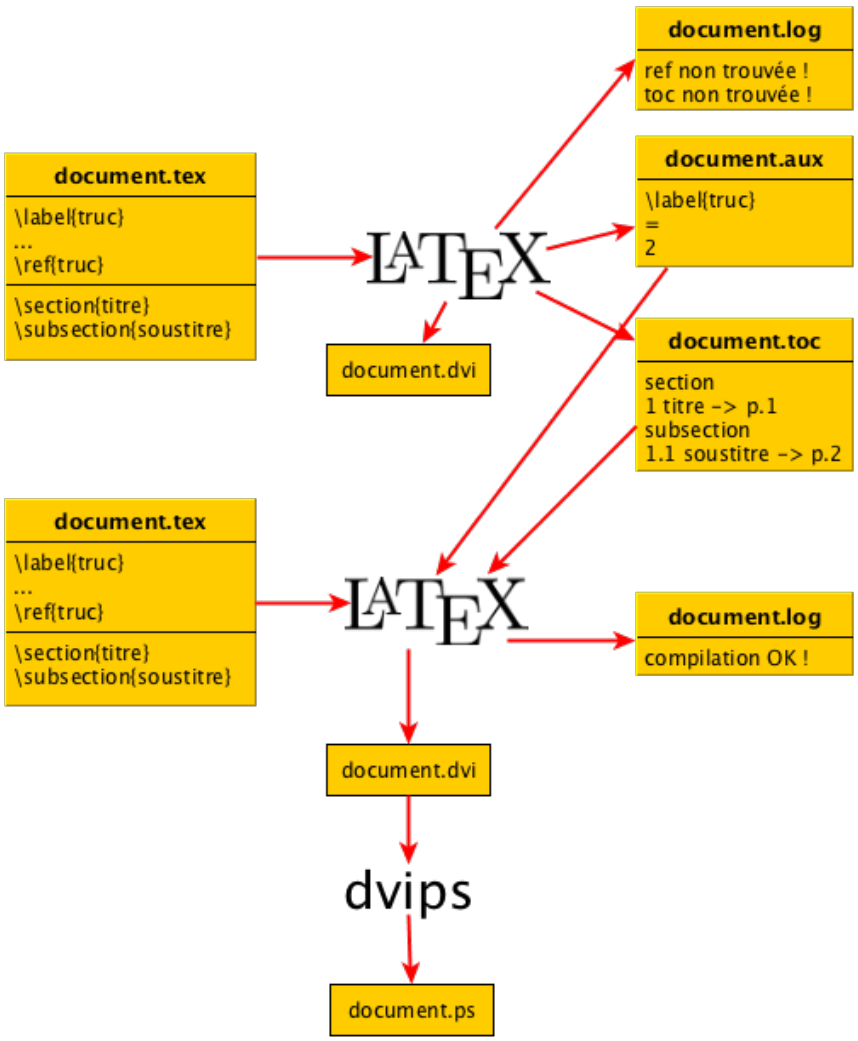


**Figure 3.12 – Générer un fichier PostScript à partir d'un fichier LaTeX avec `latex` et `dvips`.**

## 3.7 Tester un autre gestionnaire de bureau : KDE

Comme je vous l'ai indiqué, sous Linux, le gestionnaire de bureau est indépendant (du reste[14]) du système d'exploitation. Il est donc possible d'installer un autre gestionnaire de bureau que Unity et c'est ce que nous allons faire pour visualiser la différence. Il existe différents gestionnaires de bureaux (*Gnome*, *KDE*, *XFCE*, etc.). *Kubuntu* et *Xubuntu* sont respectivement des versions de Ubuntu utilisant *KDE* et *XFCE*.

Le gestionnaire de bureau se charge d'afficher le Bureau avec l'arrière plan, les barres de menu et les icônes. Il gère les fenêtres avec les boutons (fermer, agrandir, réduire). Il présente un écran d'ouverture de session (avec la demande de mot de passe) et affiche aussi l'écran *splash*, cet écran avec un logo, lors du démarrage de la machine. Il peut être aussi accompagné de divers logiciels, c'est le cas de KDE par exemple.

XFCE est un gestionnaire de bureau réputé pour être très léger et adapté aux machines qui ont de faibles capacités. Si votre machine a des capacités très réduites, installer Xubuntu plutôt que Ubuntu peut être un choix judicieux. Au contraire, KDE (Fig. 3.13), souvent conseillé aux utilisateurs habitués

14. Si l'on considère que le gestionnaire de bureau fait partie du système d'exploitation...

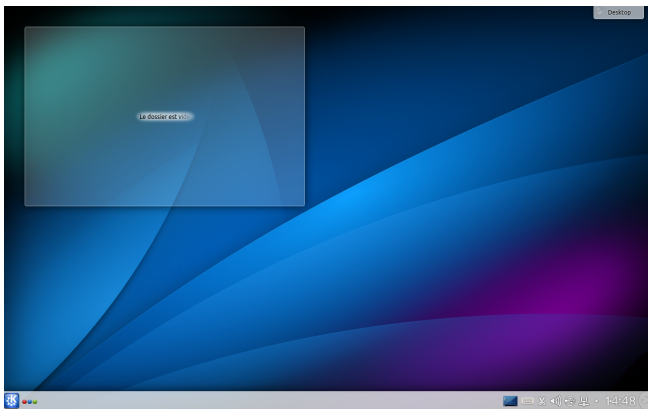

**Figure 3.13 – Le bureau de KDE.**

à Windows à cause de la ressemblance de son interface [15] est beaucoup plus lourds. KDE n'est pas qu'un environnement de bureau, c'est un projet de logiciel libre historiquement centré autour de cet environnement de bureau. Les logiciels faisant partie de ce projet sont facilement reconnaissables au fait qu'il y a toujours un « *K* » dans leur nom, souvent placé au début : navigateur Konqueror, client IRC Konversation, logiciel de mail Kmail, lecteur audio Amarok, et beaucoup d'autres.

J'ai choisi de vous faire tester XFCE. On aurait pu choisir un autre gestionnaire, ça n'a pas vraiment d'importance ici. Il s'agit juste de vous montrer que l'on peut changer totalement l'apparence extérieure de Ubuntu et tant qu'à faire autant éviter d'installer KDE qui pourrait ralentir votre système (surtout si vous travaillez dans une machine virtuelle) en installant tous ses logiciels associés.

Hélas, le paquet nécessaire pour installer XFCE n'est aps disponible dans la logithèque utilisée sous Ubuntu 16.04LTS. Si vous êtes sur une version plus ancienne de Ubuntu, vous pouvez utiliser la logithèque pour chercher « *xubuntu-destop* » et l'installer, sinon, voici comment faire avec la console.

Je vous expliquerais plus en détails les manipulations que je vais vous faire faire maintenant à la section 6.4. Pour le moment essayez simplement d'appliquer les instructions. Si quelque chose ne se passe pas comme décris et que vous ne savez pas comment résoudre le problème, attendez d'avoir suivi le cours jusqu'à la section indiquée pour revenir ensuite vers ce chapitre pour tester XFCE.

Vous allez cliquer sur l'icône du *Terminal* que je vous ai faite ajouter dans le lanceur. Dans le terminal, juste après le texte déjà affiché (que l'on nomme *prompt*) vous allez saisir la commande [16] `sudo apt-get install xubuntu-desktop` comme sur la figure 3.14. Validez la commande en appuyant sur la touche *Entrer*. L'installation se prépare et après avoir affiché quelques lignes, un message vous demande de confirmer que vous souhaitez faire l'installation (Fig. 3.15). Saisissez la lettre « *o* » puis appuyez sur la touche *Entrer* pour poursuivre. L'installation se déroule, puis à la fin le *prompt* est de nouveau affiché (Fig. 3.16). Vous pouvez fermer la fenêtre du Terminal, tout s'est bien passé.

Fermez votre session pour retourner à l'écran d'ouverture de session. Cliquez sur le logo Ubuntu, en haut à droite du cadre où l'on vous réclame votre mot de passe (Fig. 3.17). Dans la fenêtre qui s'ouvre vous pouvez choisir l'autre gestionnaire de bureau Xfce ou Xubuntu. Une fois ce choix fait, connectez vous normalement. Après le chargement du nouvel environnement de bureau, vous

15. Personnellement, je ne vois pas trop la ressemblance entre l'interface KDE et celle de Windows actuellement. La ressemblance était peut être plus marquée avec les anciennes versions de Windows (95, 98, XP)...

16. Attention, évitez les copier/coller, certains caractères invisibles pourraient être copiés en même temps et la commande collée serait erronée

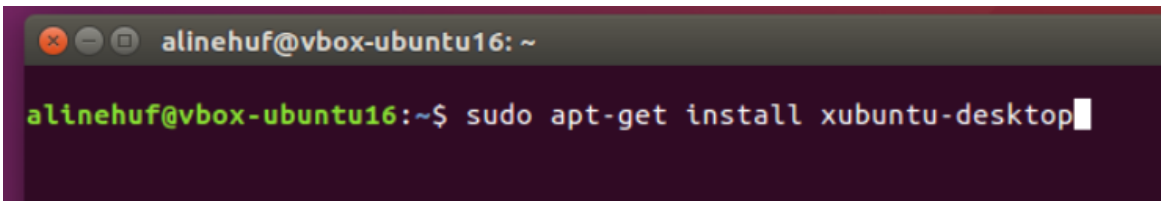


**Figure 3.14 – Commande pour installer xubuntu-desktop depuis la console**

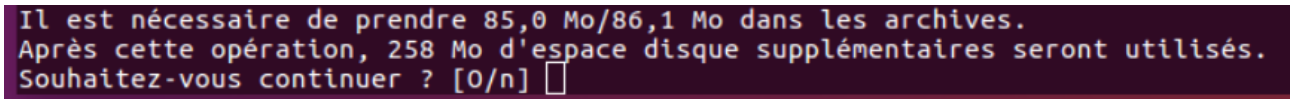


**Figure 3.15 – Message demandant de confirmer l'installation.**

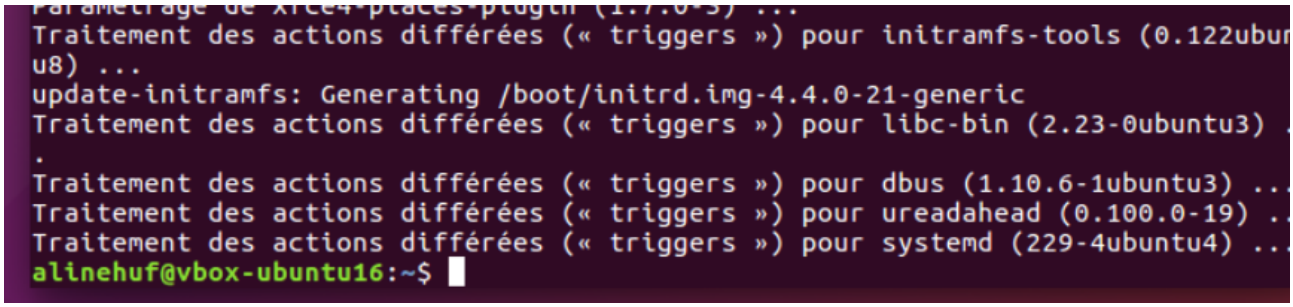


**Figure 3.16 – Fin de l'installation, le *prompt* est affiché de nouveau.**

faites face à un bureau similaire à celui de la figure 3.18. Un message vous demande de choisir une configuration : choisissez « *Utiliser les paramètres par défaut* » pour pouvoir tester XFCE.

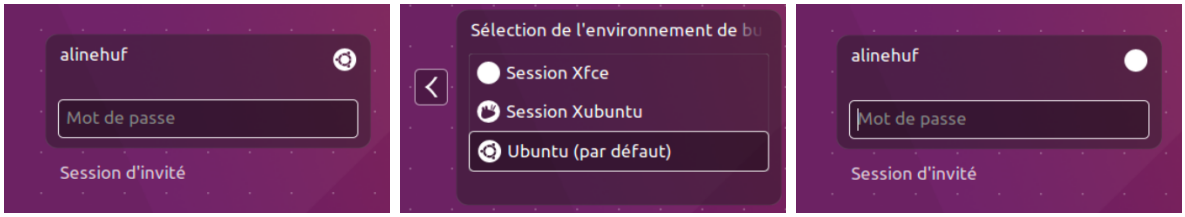


**Figure 3.17 – Choix d'un autre gestionnaire de bureau à l'ouverture de session.**

L'aspect de ce gestionnaire de bureau est différent de Unity mais vous y retrouvez les mêmes éléments. Une barre d'outils en haut, avec une zone de notification comme sur Unity. Juste avant une zone affiche des carrés bleutés au survol qui représentent les différents bureaux utilisables. A l'extrême gauche se situe le lanceur d'application.

Sur le bureau vous pouvez voir différentes icônes : *Système de fichiers* vous permet d'explorer le contenu de votre disque à partir de la racine / tandis que *Répertoire personnel* vous permet d'accéder directement à vos données.

En bas de l'écran, un dock semblable à celui de MacOS permet de ranger des icônes pour accéder rapidement à différents programmes/fonctionnalités.

Je vous laisse découvrir cet environnement et vous renseigner dans la documentation de Xubuntu sur ses différents menus et fonctionnalités. Si vous l'aimez vous pouvez garder cet environnement. Sinon, vous pouvez fermer votre session et revenir à Unity. Si vous êtes curieux, vous pouvez tester *w9wm* le gestionnaire de bureau préféré de Jean Méhat. Attention ce gestionnaire de bureau est particulièrement minimaliste, mais si vous voulez passer pour un super geek c'est ce qu'il vous faut !

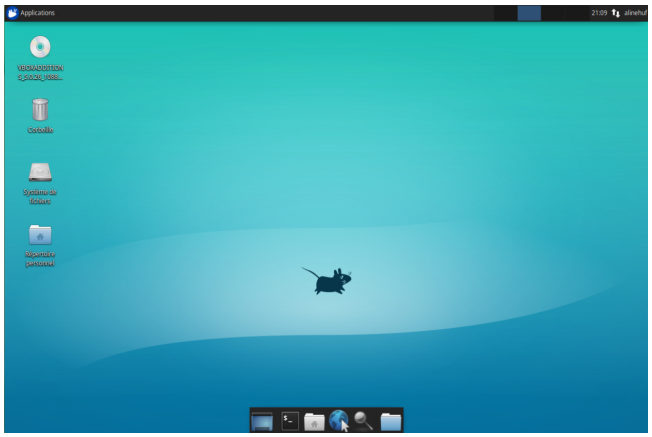


**Figure 3.18 – Le bureau de XFCE.**

# 4 Découverte du terminal

Dans ce chapitre nous allons découvrir le *terminal* (aussi appelé parfois *console*). Le terminal est un nom générique pour désigner la fenêtre d'affichage d'un *shell*. Le shell est un *interprète de commande*, un programme capable d'exécuter des commandes de l'utilisateur ; il sert d'interface entre l'utilisateur et le système d'exploitation.

Nous allons voir ici quelques informations générales sur l'utilisation du terminal et du shell. Nous verrons aussi comment se présente une commande (un ordre que le shell exécutera) et comment il est possible d'obtenir des informations détaillées sur la manière de l'utiliser et sur ce qu'elle permet de faire. Dans les chapitres suivants, je ne ferais qu'effleurer les possibilités offertes par les différentes commandes et se sera à vous d'aller vous renseigner si vous désirez en savoir plus.

Je vous recommande vivement de tester les différentes manipulations à mesure de leur présentation.

## 4.1 Les différents shells

Il existe plusieurs sortes de shell couramment utilisé sur les systèmes Unix, voici les plus connus : shell Bourne (`sh`), Bash (`bash`), Korn shell (`ksh`), C shell (`csh`), Tenex C shell (`tcsh`), et Z shell (`zsh`).

Le *shell Bourne* est l'ancêtre de tous les shells que je viens d'énumérer. Il porte le nom de son créateur Steve Bourne et est apparu en 1976 comme shell par défaut du système Unix version 7. C'est de lui que dérivent presque tous les interprètes de commandes utilisés aujourd'hui (§. 4.1).

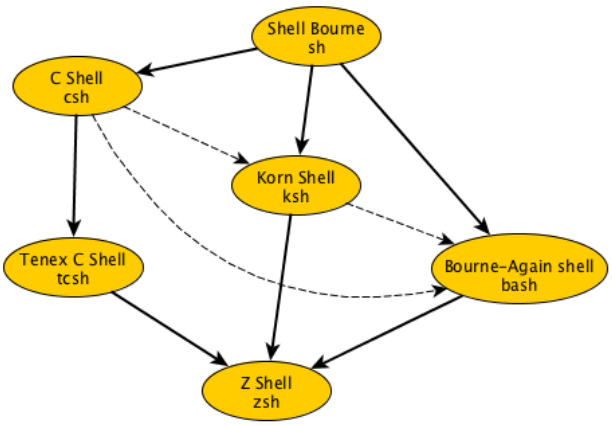


**Figure 4.1 – Les shells Bourne, Bash, Korn shell, C shell, Tenex C shell, et Z shell.**

La commande permettant de lancer ce shell se nommait `sh` et cette commande se retrouve

dans le répertoire /bin de n'importe quel système Unix ou Linux (ceux qui ont un Mac peuvent le vérifier).

Il faut faire attention avec la commande sh car depuis, elle a été utilisée parfois comme alias pour lancer le shell bash. Quand on souhaite écrire un script shell destiné à sh, il faut donc penser que les commandes peuvent être transmises au Shell Bourne ou Bash selon les systèmes, ce qui peut amener à de mauvaises surprises si les commandes utilisées n'ont pas le même sens pour ces deux shell.

Le *Bourne Again Shell*, nommé plus fréquemment *shell bash*, dérive du *shell Bourne* et lui apporte de nombreuses améliorations provenant entre autre du *Korn shell* et du *C shell*. Ces améliorations portent principalement sur l'interface utilisateur et comprennent, entre autre, l'utilisation d'un historique des commandes et l'autocomplétion.

Certains utilisateurs de ubuntu, aiment particulièrement le *Z shell* car il dérive lui aussi du *shell Bourne* et regroupe la plupart des fonctionnalités les plus pratiques des shells que je viens de citer.

Quelque soit le shell proposé par défaut, vous pouvez facilement en changer en utilisant la commande chsh («*CHange SHell*») (vous verrez plus loin comment obtenir des informations sur cette commande en utilisant le manuel). Cependant, le shell désiré doit être mentionné dans le fichier /etc/shells. Si ce n'est pas le cas, il faudra installer ce shell au préalable via la logithèque ou en ligne de commande comme vous apprendrez à le faire un peu plus loin dans ce cours.

## 4.2 Présentation du Terminal

Sous Ubuntu, le terminal se présente sous la forme d'un fond sombre avec du texte blanc [1] dans une fonte à chasse fixe [2]. Au démarrage, il affiche une *invite de commande* ou *prompt* : un texte indiquant le nom de l'utilisateur, le nom de la machine et le répertoire courant. Le *prompt* se termine par un caractère spécial ($ dans le shell bash, % dans le shell csh, etc.) : un prompt minimaliste peut se réduire à cet unique caractère suivi d'un espace. Comme son nom l'indique, l'invite de commande vous invite à taper une commande. Un curseur vous indique l'endroit ou saisir votre commande. Ce prompt peut être personnalisé pour afficher la date et l'heure, par exemple, ou mettre le texte en couleur (Fig. 4.2).

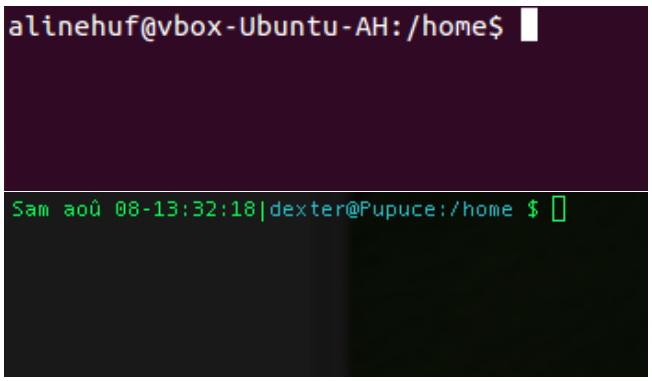


**Figure 4.2 – Deux exemples de prompt de terminaux sous Ubuntu et sous MacOS.**

1. Il est possible de personnaliser cette apparence, vous pouvez donc rencontrer des shells très différents.
2. Une fonte à chasse fixe est un ensemble de caractères typographiques qui ont tous la même taille. Les caractères d'une machine à écrire sont à chasse fixe.

### 4.2.1 Pourquoi le terminal ?

« *Mais pourquoi a-t-on inventé un truc aussi laid ?* »

Dans les années 1970, pour interagir avec les premiers ordinateurs sous Unix, il fallait utiliser un *télétype*, une sorte de machine à écrire qui communiquait avec l'ordinateur et qui imprimait sur un rouleau de papier. Il était possible d'écrire mais pas d'effacer et pour passer à l'instruction suivante on passait à la ligne.

Au début des années 1980, quand sont apparus les premiers terminaux composés d'un écran de 24 lignes par 80 colonnes, c'était la pointe de la technologie. Comme on pouvait afficher plusieurs lignes de texte et effacer des caractères, les premiers éditeurs de texte (Emacs, Vi) sont apparus.

Les interfaces graphiques se sont généralisées dans les années 1990. Une interface nommée X11 s'est imposée et est toujours utilisée aujourd'hui. Elle permet de transmettre au système d'exploitation les actions effectuées sur le clavier et la souris, et affiche sur l'écran une interface graphique avec des fenêtres. X11 est toujours présent, actuellement, entre les gestionnaires de bureau (Gnome, KDE, etc.) et le système d'exploitation.

Si l'utilisation du terminal et des commandes textuelles a persisté malgré l'apparition des interfaces graphiques, c'est qu'elle permet de faire certaines choses qui ne sont pas faisables avec l'interface graphique ou qui serait bien plus longues.

Si vous voulez, par exemple, compter **rapidement** le nombre d'images (`.jpg`) présentes dans le répertoire courant, vous pouvez utiliser la commande suivante :

```
$ ls *.jpg | wc -l
```

Vous avez vu également que les gestionnaires de bureau peuvent être très différents les uns des autres et on peut de prime abord être un peu perdu (mais ou se trouve tel programme ? Comment faire telle action sur ce gestionnaire de bureau ?). Pire, vous pouvez tomber sur un serveur qui n'a pas d'interface graphique du tout ! Les commandes, elles, n'ont quasiment pas changées depuis les années 1970. Ce sont les mêmes sur tous les systèmes hérités d'Unix : des commandes courtes, abrégées de l'anglais.

On peut placer des commandes dans un fichier (un script shell) et les exécuter pour rapidement effectuer un ensemble de tâches complexe. Par exemple : éliminer les images en double dans mon dossier photo, renommer le dossier avec la date du jour, placer le dossier dans le répertoire *archive* de mon disque dur externe et libérer la place sur mon ordinateur. Je prend 5 minutes pour écrire ce script shell : les fois suivantes je pourrai le lancer pour faire ça en une seconde. Ceci n'est pas faisable avec une interface graphique : il n'existe pas de moyen simple pour enregistrer une suite d'actions effectuées avec la souris. De plus, les gestionnaires de bureau comme KDE et Gnome (Unity est une sur-couche de Gnome) sont incompatibles entre eux. Les commandes, elles, restent les mêmes et sont toujours disponibles.

### 4.2.2 Tester Linux sans interface graphique

Il est possible de voir à quoi ressemble Ubuntu sans interface graphique. Pour cela, il suffit d'appuyer sur les touches `Ctrl+Alt+F1`. Vous voyez alors le premier terminal. Il y en a 6, accessibles avec `Ctrl+Alt+ F1` à `F6` depuis l'interface graphique. Si vous êtes sur une interface terminal, vous pouvez passer à une autre avec `Alt+F`$n$ (avec $n$ le numéro de console). Pour revenir à l'interface graphique utilisez `Alt+F7`.

Dans un terminal, vous devez voir en haut de l'écran une ligne indiquant le numéro du terminal (`tty`$n$, abréviation de télétype), la version du système et une ligne qui vous invite à indiquer votre nom d'utilisateur (Fig. 4.3). Si vous le faites (validez avec la touche *entrée*), vous recevrez alors une demande de mot de passe. Si vous tapez votre mot de passe, vous ne verrez rien s'afficher : c'est normal, c'est une sécurité. Si un intrus regarde votre écran par dessus votre épaule, il ne saura ainsi même pas combien de caractères vous avez tapés. Une fois connecté, vous pouvez commencer à saisir des commandes.


**Figure 4.3 – Exemple d'interface terminal sous Ubuntu.**

### 4.2.3 Le terminal en mode graphique

Il est possible d'utiliser le terminal en mode graphique. Sous Ubuntu, le logiciel Terminal reproduit dans une fenêtre l'apparence d'un terminal et vous permet d'utiliser le shell. Démarrez le terminal en cliquant sur l'icône que je vous ai fait ajouter dans le lanceur (§. 3.1).

Vous allez saisir votre première commande `echo "bonjour"` :

```
$ echo "bonjour" # une commande qui affiche bonjour
bonjour
```

Dans ce cours, je vous présenterai toujours les commandes à saisir avec le `$` du prompt au début pour distinguer les lignes de commande des résultats retournés par le shell. Ce caractère est affiché par le terminal pour vous inviter à écrire, vous ne devez pas l'écrire vous même. La commande `echo` permet d'afficher quelque chose et `"bonjour"` est une simple chaîne de caractères[3] contenant les caractères à afficher. Après avoir tapé cette commande, le shell a dû vous répondre « *bonjour* » !

A la fin de la commande j'ai placé un commentaire. Pour indiquer au shell qu'il s'agit d'un commentaire, j'ai ajouté devant le caractère `#`. Tout caractère qui est placé après le symbole `#` est totalement ignoré par le shell jusqu'au prochain saut de ligne. Dans ce cours, j'utiliserai les commentaires pour expliquer le rôle d'une commande quand ce sera nécessaire. Pour tester cette commande, il est inutile de recopier ce commentaire.

Il est possible d'enchaîner plusieurs commandes en les séparant par un point virgule qui joue le même rôle qu'un saut de ligne (`echo` ajoute un saut de ligne après le message affiché ce qui explique que les résultats sont présentés sur deux lignes différentes) :

```
$ echo bonjour ; echo "au revoir"
```

3. Une chaîne de caractères est tout simplement une suite de caractères manipulée d'un bloc. Elle peut contenir des espaces. Une chaîne de caractères peut être délimitée par des guillemets ou des quotes simples.

```
bonjour
au revoir
```

Le shell est un programme qui tourne en boucle : il affiche une invite de commande et attend de recevoir un retour à la ligne pour vérifier si la commande est complète. Si ce n'est pas le cas, le shell affiche un *prompt secondaire* (ici le caractère >). Si la commande est complète, il exécute la commande, puis affiche de nouveau l'invite, etc.

Il peut considérer que la commande est incomplète quand, par exemple, il a détecté la présence de parenthèses ou de guillemets non refermés.

```
$ echo 'debut du message,
> fin du message'
debut du message,
fin du message
$ (echo bonjour
> echo au revoir)
bonjour
au revoir
```

Vous constatez qu'il est donc également possible d'enchaîner deux commandes en les mettant entre parenthèses et en sautant une ligne.

## 4.3 Pour utiliser la console

Différentes facilités sont disponibles dans le shell Bash pour écrire les commandes ou retrouver une commande que vous avez déjà saisie auparavant.

### 4.3.1 Les raccourcis clavier

Si vous avez tenté d'utiliser les raccourcis que vous connaissez (comme `Ctrl+C` et `Ctrl+V`), vous vous êtes peut-être rendu compte qu'ils ne fonctionnaient pas dans la fenêtre du terminal. Voici quelques raccourcis à connaître pour utiliser le terminal :

| | |
|---|---|
| **Maj+Ctrl+C** | pour copier du texte. |
| **Maj+Ctrl+V** | pour coller du texte. |
| **Ctrl+A** | pour placer le curseur en début de ligne. |
| **Ctrl+E** | pour placer le curseur en fin de ligne |

### 4.3.2 L'historique des commandes

Quand vous tapez des commandes, elles sont enregistrées dans le fichier `.bash_history`. Vous pouvez donc rappeler rapidement des commandes que vous avez déjà saisies. Les flèches du clavier (↑ et ↓) permettent de faire défiler les commandes de l'historique. Pour voir l'historique de toutes les commandes, vous pouvez utiliser la commande `history` :

```
$ history
```

Le raccourci Ctrl+R permet de lancer un outil de recherche d'une ligne de commande dans l'historique. Après avoir saisi le raccourci clavier, vous voyer un message du type (reverse-i-search)‘’:. Commencez à taper un morceau de texte présent dans la commande recherchée : l'outil de recherche affiche la première commande correspondant à la recherche. Si la commande n'est pas celle que vous cherchez, appuyez sur Ctrl+R à nouveau pour faire défiler toutes les autres commandes correspondant à la recherche.

### 4.3.3 L'auto-complétion

Les commandes peuvent être longues à taper, surtout quand il faut saisir un long chemin d'accès vers un fichier. Pour accélérer les choses, vous pouvez utiliser la touche tab (touche de *tabulation* portant le symbole →| ). Commencez à taper cd /h dans la console (la commande cd permet de changer de répertoire), puis appuyez sur la touche tab et constatez que /h a été complété automatiquement en /home/ : on appelle cela l'*autocomplétion*. Désormais, vous pourrez utiliser la touche tab pour compléter les chemins d'accès et les noms de commandes.

Si vous appuyez 2 fois sur la touche tab après avoir saisi le début d'une commande (ou d'un chemin), le shell vous affichera la liste de toutes les commandes (ou chemins) commençant par les caractères saisis. Si cette liste est trop longue, il affichera à la place un message du type « *Display all xxx possibilities ? (y or n)* ». Si vous appuyez sur y il vous affichera les premiers éléments : appuyez sur la touche espace pour voir la suite ou Q pour quitter.

### 4.3.4 Stopper une commande

Quand vous souhaitez arrêter une commande qui vous demande d'entrer du texte, vous pouvez utiliser le raccourci Ctrl+D qui place l'information EOF («*End Of File*», «*fin de fichier*») à la suite du texte à traiter et qui forcera donc la commande à terminer son travail. Par exemple, lancez l'interprète du langage python avec la commande python : Python affiche un message de démarrage puis son prompt >>> vous invitant à saisir une commande : notez qu'il est différent de celui du shell. Comme vous ne connaissez aucune commande en python pour le moment, appuyez sur Ctrl+D pour lui dire qu'il n'y a plus rien à lire : l'application s'arrête et le shell affiche de nouveau son prompteur.

Quand vous lancez une commande qui semble ne plus répondre ou tourner en boucle, vous pouvez utiliser le raccourci Ctrl+C (qui envoie un signal SIGINT) pour forcer l'arrêt. N'utilisez cet arrêt en force que si la commande ne peut pas être arrêtée proprement avec Ctrl+D.

## 4.4 Commandes et paramètres

Comme vous venez de le voir, une commande est un mot (groupe de lettres sans espace) qui représente un ordre à exécuter. Cette commande peut être suivie de 0, 1 ou plusieurs paramètres précisant ce qui doit être fait. La commande date, par exemple, peut être appelée dans aucun paramètre. Essayez la et elle vous affichera la date et l'heure.

Il existe plusieurs types de paramètres que l'on peut passer à une commande : une donnée (le nom d'un fichier, d'une variable d'environnement, du texte entre guillemets, etc.) ou ce que l'on nomme un paramètre (ou option) court ou long. Un paramètre court commence par le symbole -

suivi d'une seule lettre[4]. Un paramètre long est la version longue d'un paramètre, il commence par `--` suivi d'un mot. Le paramètre long est généralement plus lisible mais il est plus long à taper.

Par exemple l'option `-u` passé à la commande date, la force à afficher la date au format universel (utc). L'option `--utc` (« *Universal Time Coordinated* ») est la version longue de ce même paramètre :

```
$ date
samedi 8 août 2015, 15:21:45 (UTC+0200)
$ date -u
samedi 8 août 2015, 13:21:48 (UTC+0000)
$ date --utc
samedi 8 août 2015, 13:21:51 (UTC+0000)
```

L'intérêt des paramètres courts c'est qu'il est possible de les concaténer (les juxtaposer). Prenons par exemple la commande `ls` qui permet de lister les fichiers d'un répertoire. Si on lui transmet le paramètre `-a` (all), on pourra voir les fichiers cachés (dont le nom commence par `.`). Si on lui transmet le paramètre `-l` (long), on pourra voir une liste détaillée des fichiers (avec des informations dont nous verrons plus tard la signification). Il est possible de réunir ces deux paramètres en un : `-al` pour voir une liste détaillée de tous les fichiers.

```
$ ls
index.html
$ ls -a
.  ..  .htaccess  index.html
$ ls -l
total 0
-rw-rw-r-- 1 martin muggles 0 août  8 15:38 index.html
$ ls -al
total 8
drwxrwxr-x  2 martin muggles 4096 août  8 15:38 .
drwxr-xr-x 25 martin muggles 4096 août  8 15:37 ..
-rw-rw-r--  1 martin muggles    0 août  8 15:38 .htaccess
-rw-rw-r--  1 martin muggles    0 août  8 15:38 index.html
```

Attention, presque tout sous Unix est *sensible à la casse* : cela signifie qu'il y a une différence de sens entre les caractères majuscules et minuscules. Une même commande peut avoir deux paramètres différents l'un en majuscules, l'autre en minuscules et les deux paramètres auront un effet totalement différent.

Un paramètre peut nécessiter l'ajout d'une information supplémentaire. Prenons par exemple la commande `tail` qui permet d'afficher les dernières lignes d'un fichier texte, l'option `-n` permet d'indiquer le nombre de lignes à afficher : il faut donc préciser ce nombre juste après. On peut ajouter ou ne pas ajouter d'espace entre le paramètre et le nombre saisi, cela n'a pas d'importance. Si vous utilisez le paramètre long correspondant `--lines` il faut par contre ajouter le signe `=` avant de saisir le nombre :

```
$ tail -n 2 comptine.txt
7, 8, 9, dans mon panier neuf ;
10, 11, 12, elles seront toutes rouges. »
$ tail -n2 comptine.txt
7, 8, 9, dans mon panier neuf ;
10, 11, 12, elles seront toutes rouges. »
```

4. Il y a des exceptions. Nous verrons avec la commande `find` qu'il existe des options constituées d'un mot précédé du caractère `-`.

```
$ tail --lines=2 comptine.txt
7, 8, 9, dans mon panier neuf ;
10, 11, 12, elles seront toutes rouges. »
```

## 4.5 RTFM ! Le manuel

« *Mais comment mémoriser toutes ces commandes et ces paramètres ? !* »

Et bien en réalité, on ne mémorise que les commandes qu'on utilise très souvent. Les autres, on les oublie et c'est pour cela que le *manuel* (aussi nommé *les pages man*) est si précieux. Il y a donc une seule commande qu'il ne faut pas oublier, c'est `man` !

Si vous demandez de l'aide sur des forums ou des chats, vous recevrez peut être le message « *RTFM !* » qui signifie « *Va lire le p... de manuel !* » (« *Read The F... Manual* »). L'expression traduit généralement l'agacement de votre interlocuteur qui considère que votre question aurait reçu une réponse immédiate si vous aviez seulement pris la peine de faire une recherche dans le manuel.

Pour vous éviter cette remarque désobligeante, prenez toujours la peine de faire une petite recherche avant de poser une question. Si les explications obtenues dans le manuel ne sont pas suffisamment claires pour vous, indiquez-le dans votre demande d'aide : vos interlocuteurs prendront alors la peine de vous expliquer plus en détail ce que le manuel essaye de vous dire.

### 4.5.1 Utiliser le manuel

La manière la plus simple d'utiliser le manuel est de passer en paramètre à la commande `man` le nom de la commande dont on souhaite obtenir le mode d'emploi : chaque commande possède une page de manuel à son nom. Et comme `man` est une commande on peut lui demander le manuel du manuel !

```
$ man man
```

Quand vous avez ouvert une page du manuel, pour vous déplacer, vous pouvez utiliser les touches suivantes. Pour prendre connaissance de la liste de tous les raccourcis clavier (recommandé) appuyez sur `H` :

| | |
|---|---|
| ↑ ↓ | pour vous déplacer ligne par ligne. |
| **/** | permet de faire une recherche. Taper le mot recherché et appuyez sur `Entrer`. Pour chercher l'occurrence suivante, appuyez sur `N`. Pour chercher l'occurrence précédente, appuyez sur `Maj+N`. |
| **Q** | pour quitter le manuel à tout moment. |

C'est la commande `less` qui est utilisée de manière transparente pour afficher les pages du manuel. Vous pouvez donc utiliser également `man less` pour connaitre la liste des raccourcis claviers utiles.

Une page de manuel contient différentes *rubriques* et commence généralement par les trois suivantes :

| | |
|---|---|
| NOM | Le nom de la commande et une description très brève de ce qu'elle fait. |

**SYNOPSIS** Explique comment saisir la commande dans le shell.

**DESCRIPTION** Explique en détail ce que fait la commande, comment l'utiliser, quels sont les différents paramètres utilisables et leur effet.

Le manuel contient différentes *sections* et il est possible de faire une recherche dans le manuel en précisant la section où l'on désire chercher. Par exemple, pour chercher le manuel de la fonction C `printf` il faudra saisir :

```
$ man 3 printf
```

Les sections que vous utiliserez sans doute le plus sont la section 1 qui contient toutes les commandes shell et la section 3 qui contient le manuel de toutes les fonctions fournies par les bibliothèques des programmes. C'est dans la section 3 que vous trouverez le manuel des fonctions du langage C.

### 4.5.2 Aide fournie par les commandes elles-mêmes

Certaines commandes fournissent directement une aide quand elles détectent une erreur de votre part. Testez par exemple la commande `ls` avec l'option `-e` qui n'existe pas :

```
$ ls -e
ls : option invalide -- 'e'
Saisissez « ls --help » pour plus d'informations
```

Voilà ! un message vous invite cordialement à utiliser l'option `--help` pour obtenir de l'aide. Testez la commande `ls --help` et vous verrez des explications s'afficher dans la console sur l'utilisation de la commande `ls`. Ces informations ne sont pas toujours semblables à celles du manuel et sont souvent plus succintes.

L'option `--help` ou sa version courte `-h` est disponible pour la plupart des commandes.

Pour les commandes qui attendent forcement un argument, il est possible de saisir la commande seule, sans arguments pour avoir un rappel très bref de son usage :

```
$ grep
Utilisation : grep [OPTION]... MOTIF [FICHIER]...
Exécutez « grep --help » pour obtenir des renseignements complémentaires.
$ wget
wget : URL manquante
Utilisation : wget [OPTION]... [URL]...
Utilisez « wget --help » pour obtenir plus de renseignements.
```

### 4.5.3 Autres informations sur les commandes : whatis et whereis

D'autres commandes permettent d'obtenir des informations sur les commandes. La commande `whatis` permet d'avoir rapidement la première ligne du manuel d'une commande (nom + ligne expliquant son utilité) dans chaque section ou elle apparait :

```
$ whatis man
man (1)      - Interface de consultation des manuels de référence en ligne
man (7)      - Macros pour la mise en forme des pages de manuel
```

```
man (1posix) - display system documentation
```

La commande `whereis` permet de savoir où se situe, dans l'arborescence des fichiers, le fichier contenant une commande (il peut y avoir plusieurs versions de la même commande), les sources de cette commande (le programme lisible par un humain avant sa transformation en fichier binaire exécutable) et la documentation :

```
$ whereis passwd
passwd: /usr/bin/passwd /etc/passwd /usr/bin/X11/passwd
/usr/share/man/man1/passwd.1ssl.gz /usr/share/man/man1/passwd.1.gz
/usr/share/man/man5/passwd.5.gz
```

# 5 Commandes pour gérer les fichiers et leur contenu

Dans ce chapitre nous allons utiliser le shell pour se déplacer dans l'arborescence des fichiers, créer, modifier ou supprimer des fichiers et des dossiers. Vous ne verrez dans ce chapitre que quelques commandes qui sont fréquemment utilisées. L'effet de chaque commande peut être modifié ou ajusté avec des instructions additionnelles, nous en verrons quelques unes mais là encore ce chapitre ne prétend pas être exhaustif. Je vous conseille vivement d'explorer les pages du manuel pour approfondir votre connaissance des commandes évoquées.

Nous verrons également les notions de *redirections* et de *pipes* («*tuyaux*»). Enfin, nous aborderons la notion complexe d'*expression régulière*.

Ce chapitre (le suivant aussi) sera sans doute très difficile pour ceux qui n'ont jamais tapé une commande ou jamais programmé. Je vous conseille de tester abondamment les commandes à mesure de leur présentation pour comprendre leur fonctionnement et d'explorer la documentation pour découvrir les possibilités qu'elles offrent.

## 5.1 Se déplacer, s'informer, gérer les fichiers

Nous allons voir ici des commandes de base pour se déplacer dans l'arborescence des fichiers, pour obtenir des informations sur les fichiers et les disques et pour gérer les fichiers (création, modification, suppression, etc.). Au passage nous aborderons des notions importantes sur les *liens symboliques*, *liens physiques* et sur les *inodes*.

### 5.1.1 Se déplacer, obtenir des informations sur les répertoires et leur contenu

#### pwd

Pour savoir où vous vous trouvez dans l'arborescence des fichiers, vous pouvez utiliser la commande `pwd` («*Print Working Directory*») : le chemin affiché commence par le symbole `/`, c'est un chemin absolu depuis la racine.

### cd

Pour vous déplacer, vous pouvez utiliser la commande cd (« *Change Directory* ») suivie du chemin vers le répertoire où vous souhaitez vous rendre. Ce chemin peut être relatif ou absolu (pensez à utiliser tab pour taper plus rapidement les chemins d'accès). Pour faire référence au répertoire parent, vous pouvez utiliser le fichier spécial .. :

```
$ pwd                    # où suis-je ?
/home/martin
$ cd Documents/exo_1     # je descends dans un sous-répertoire
$ pwd                    # je verifie que je suis arrivé au bon endroit
/home/martin/Documents/exo_1
$ cd ../exo_2            # je remonte, puis descend dans une autre branche
$ pwd
/home/martin/Documents/exo_2
$ cd ../../Images        # je remonte de 2 niveaux, puis descend dans Images
$ pwd
/home/martin/Images
```

Votre *home* (votre répertoire personnel placé dans /home/) étant un dossier que vous aurez souvent besoin de prendre comme référence, il existe un symbole particulier pour le représenter c'est le ~ (tilde). Tapez la commande cd ~/Images/ puis la commande pwd : constatez que vous êtes dans /home/*votrePseudo*/Images/.

Comme votre répertoire personnel est le centre de votre petit monde, la commande cd sans argument vous ramène dans votre répertoire personnel.

Sachez également que vous pouvez revenir rapidement au répertoire précédent avec la commande cd - (un tiret à ne pas confondre avec ~ le tilde) :

```
$ pwd                    # où suis-je ?
/boot/grub               # dans /boot/grub
$ cd /usr/bin            # je change de répertoire
$ pwd
/usr/bin                 # je suis maintenant dans /usr/bin
$ cd -                   # je reviens au répertoire précédent
/boot/grub
```

Pour rappel : un chemin absolu commence par / qui représente la racine. Si le chemin commence par un autre caractère, c'est un chemin relatif qui est interprété par rapport à la position courante.

### ls

La commande ls (abréviation de «*LiSt*») vous permet de lister le contenu d'un répertoire. Vous pouvez la tester sur votre répertoire personnel. Comme nous l'avons vu précédemment (§. 4.4), il est possible d'utiliser les paramètres -a pour voir tous les fichiers y compris .., . et les fichiers cachés et -l pour avoir une liste détaillée des fichiers. L'option -F permet de signaler les répertoires en ajoutant le caractère / à la fin de leur nom. Si votre terminal est en couleur, vous remarquerez que les répertoires et les fichiers ne sont pas colorés de la même manière pour les différencier plus facilement.

Si vous souhaitez afficher le contenu d'un autre répertoire que le répertoire courant, vous pouvez

le passer comme dernier argument de la commande `ls` :

```
$ ls -alF /usr/bin/
```

#### file

La commande `file` permet de connaitre le type d'un fichier. Les noms de fichiers sous Linux, comme sous Windows ou MacOS peuvent se terminer par une extension (un point suivi de 2-3 lettres) indiquant au système quelle application utiliser pour les lire. Cependant, il est tout à fait possible de ne pas donner d'extension à un fichier ou bien d'en donner une qui ne correspond pas au contenu réel du fichier. C'est une astuce très basique utilisée par certains pirates pour vous envoyer un virus déguisé, par exemple, en une inofensive image avec une extension `.jpg`...

Voici quelques exemples de réponses obtenues avec file en lui donnant différents noms de fichiers :

```
$ file compile_test.sh
compile_test.sh: Bourne-Again shell script text executable
$ file LICENSE
LICENSE: ASCII English text, with very long lines
$ file Src
Src: directory
```

Maintenant, voici un exemple de fichier dont l'extension ne correspond pas à son contenu. L'extension `.a` signale généralement une archive, mais le fichier `archive.a` suivant n'est qu'un simple fichier texte, en voici la preuve :

```
$ file archive.a
archive.a: ASCII text
```

#### du

La commande `du` («*Disk Usage*») permet d'avoir des informations sur l'utilisation du disque. Le paramètre `-h` permet de voir les tailles dans un format plus facile à lire (en Ko, Mo, Go, etc.). Le paramètre `-a` permet de voir la taille de tous les fichiers, pas seulement les répertoires, tandis que `-s` permet au contraire d'avoir juste la taille globale.

#### df

La commande `df` («*Disk Free*») permet de connaître l'état d'occupation d'un disque (espace utilisé et espace libre). Là aussi, le paramètre `-h` permet d'afficher les tailles dans un format humainement lisible.

### 5.1.2 Les méta-caractères, échappements et substitutions de commandes

Il existe des caractères spéciaux qui sont interprétés par le shell avant même qu'il ne tente d'exécuter une commande. Ces caractères sont interprétés et remplacés par leur signification, ensuite seulement la commande est exécutée. Les *méta-caractères* permettent de désigner rapidement un

groupe de fichiers plutôt qu'un seul, ce sont les caractères ?, *, et []. Les *caractères d'échappement* \, "..." et '...' permettent au contraire d'éviter qu'un caractère soit interprété, il est alors utilisé tel quel. Les caractères `...` permettent d'insérer le résultat d'une commande directement dans une chaîne de caractères.

### ?

Le caractères ? peut remplacer n'importe quel caractère sauf le point situé au début d'un nom : on ne peut donc pas utiliser le ? pour remplacer le premier caractère des fichiers spéciaux ., .. ou des fichiers cachés. Voici un exemple d'utilisation où t?t? permet de désigner à la fois titi et toto :

```
$ ls          # je demande la liste de tous les fichiers
bar  foo  titi  toto
$ ls t?t?     # je demande les fichiers correspondant au modèle
titi  toto
```

### *

Le caractères * (wildcard) permet de remplacer un ou plusieurs caractères. Vous pouvez l'utiliser, par exemple, pour afficher tous les fichiers qui commencent par la lettre a ou qui finissent par .jpg. Si ces fichiers sont des répertoires, leur contenu sera listé :

```
$ ls /usr/bin/ama* # fichiers qui commencent par "ama" dans /usr/bin
/usr/bin/amarok                    /usr/bin/amarokmp3tunesharmonydaemon
/usr/bin/amarok_afttagger          /usr/bin/amarokpkg
/usr/bin/amarokcollectionscanner
$ ls ~/M*           # fichiers commençant par "M" dans mon home
/home/martin/Modèles:                    # ce fichier est un répertoire
                                            # mais il est vide

/home/martin/Musique:
beep.mp3  sonnerie.mp3
$ ls Images/*.jpg  # fichiers finissant par ".jpg" dans Images/
Images/image1.jpg  Images/image2.jpg
```

### [ ]

Les caractères [] permettent de désigner un ensemble de caractères, par exemple [1-4] désigne les chiffres de 1 à 4, [A-C] les lettres de A à C, [134] désigne le chiffre 1, 3 ou 4, [AC] désigne le caractère A ou C :

```
$ ls              # listing de tous les fichiers
bar  foo  img1  img2  img3  img4  imgA  imgB  imgC  imgD  titi  toto
$ ls img[1-4]     # uniquement les fichiers entre img1 et img4
img1  img2  img3  img4
$ ls img[14]      # uniquement les fichiers img1 ou/et img4
img1  img4
$ ls t[io]t[io]   # uniquement les fichiers titi, toto, toti ou tito
titi  toto
$ ls img[A-D]     # uniquement les fichiers entre imgA et imgD
```

```
imgA  imgB  imgC  imgD
```

Si jamais les caractère *, ?, [ ou ] font partie du nom du fichier que vous recherchez ou de la commande que vous êtes en train de taper, il peut être gênant que le shell tente de les interpréter comme des méta-caractères. Pour éviter cela il existe des *caractères d'échappement*.

### \, "..." et '...'

Le caractère \ («*anti-slash*», «*backslash*» ou «*contre-oblique*») permet d'empêcher l'interprétation d'un caractère. Pour empêcher l'interprétation d'un ensemble de caractères, il est possible d'ajouter un *quote simple* ' ou un *quote double* " avant et après le groupe.

Dans l'exemple qui suit, le caractère * indique un nom de fichier contenant plusieurs caractères, il est donc remplacé par tout les noms de fichiers correspondant à ce modèle. Les mots suivants, « *est* », « *une* » et « *étoile* » sont ensuite affichés.

```
$ echo * est une étoile
Bureau Documents Images Modèles Musique Public Vidéos est une étoile
```

Si le but est d'afficher la phrase sans interpréter le caractère * , il faut précéder celui-ci du symbole \ ou bien utiliser des quotes doubles ou simples :

```
$ echo \* est une étoile
* est une étoile
$ echo "* est une étoile"
* est une étoile
$ echo '* est une étoile'
* est une étoile
```

### `...` ou $(...)

La substitution de commandes dans une chaîne de caractères est une autre facilité offerte par le shell. Elle permet de capturer la sortie d'une commande et de l'utiliser directement dans une autre commande. Pour signaler qu'une commande doit être interprétée et remplacée par son résultat avant d'évaluer le reste, il suffit de l'encadrer par des caractères ` («*backquote*»).

```
$ echo "le répertoire courant est `pwd`"
le répertoire courant est /home/martin/bin
```

Il est possible également d'utiliser le symbole $ devant l'expression entre parenthèses. Cette syntaxe est plus moderne et évite les problèmes qui peuvent survenir quand l'expression contient elle-même des backquote ou autres quotes :

```
$ echo "le répertoire courant est $(pwd)"
le répertoire courant est /home/martin/bin
```

### 5.1.3 Créer, modifier, supprimer des fichiers et des répertoires

#### touch

La commande touch permet de modifier la date de dernière modification d'un fichier.

```
$ ls -l              # listing détaillé
total 68
-rw-rw-r-- 1 alinehuf alinehuf 68294 août  17 12:11 image.png
$ touch im*          # je touche le fichier...
$ ls -l              # le listing montre que la date a changé
total 68
-rw-rw-r-- 1 alinehuf alinehuf 68294 oct.   9 17:02 image.png
```

Si le fichier n'existe pas, il est créé. On peut donc éventuellement utiliser cette commande pour créer un fichier. Si vous voulez passer en paramètre un fichier dont le nom comporte un espace, vous devez mettre ce nom entre guillemets, sinon chaque espace sera considéré comme un séparateur et plusieurs fichiers seront créés :

```
$ ls -Q              # -Q permet de visualiser les noms avec des guillemets
"le texte.txt"
$ touch le texte.txt # j'oublie de mettre des guillemets autour du nom
$ ls -Q
"le"  "le texte.txt"  "texte.txt" # j'ai créé d'autre fichiers accidentellement
```

#### mkdir

La commande mkdir («*MaKe DIRectory*») permet de créer un ou plusieurs répertoires. A sa création chaque répertoire contient les fichiers spéciaux (.. et .). Pour créer directement un chemin (une suite de répertoires contenus les uns dans les autres), il est possible d'utiliser le paramètre -p :

```
$ mkdir dossier
$ mkdir dossier1 dossier2
$ mkdir -p fac/python/exos/
```

#### mv

Pour déplacer un fichier ou un dossier, on peut utiliser la commande mv («*MoVe*») en indiquant le fichier (ou répertoire) à déplacer en premier paramètre et la destination en second paramètre. La commande mv est également utilisée pour renommer un fichier :

```
$ touch exo.txt          # je crée le fichier exo.txt
$ ls                     # je vérifie qu'il est bien là
exo.txt
$ mkdir chap1            # je crée un répertoire chap1/
$ mv exo.txt chap1/      # je déplace mon exo.txt dedans
$ ls                     # là où je suis il n'y a plus que le répertoire
chap1
$ ls chap1/              # dans le répertoire il y a mon fichier
exo.txt
$ mv chap1/exo.txt chap1/exo_1.txt  # je renomme exo.txt
```

```
$ ls chap1/              # dans le répertoire le fichier est renommé
exo_1.txt
```

### cp

La commande cp («*CoPy*») permet de copier un fichier en indiquant le nom du fichier et le nom de la copie. En précisant un chemin d'accès différent pour l'original et la copie, on peut placer la copie où on le souhaite pourvu que le répertoire existe. Si le second paramètre est un répertoire (et non un chemin complet avec un nom de fichier), le fichier sera copié sous le même nom dans ce répertoire. Pour copier un répertoire et tout son contenu, il faudra utiliser le paramètre -r («*Recursive*»).

### rm

Pour effacer un fichier, on peut utiliser la commande rm («*ReMove*») suivie du nom du (ou des) fichier(s) à effacer. L'option -i permet d'afficher un message de confirmation avant que chaque fichier ne soit définitivement effacé. Il est possible de configurer la commande rm pour qu'elle demande systématiquement cette confirmation avant d'effacer un fichier (§. 6.2.8). Si au contraire, les demandes de confirmation vous ennuient, il est possible de forcer l'effacement sans confirmation avec -f. L'option -v («*Verbose*») permet d'afficher le détail de l'activité de rm. Pour effacer un répertoire, il faudra ajouter le paramètre -r.

```
$ rm -vr fac
répertoire supprimé : «fac/python/exos»
répertoire supprimé : «fac/python»
répertoire supprimé : «fac»
```

### rmdir

Pour effacer un répertoire, on peut aussi utiliser la commande rmdir («*ReMove DIRectory*»). Attention, avec rmdir un répertoire ne peut être effacé que s'il est vide. Pour supprimer tout le contenu d'un répertoire, on peut utiliser le wildcard * [1] (§. 5.1.2). Attention, toutes les opérations d'effacement sont irréversibles, et avec le wildcard on peut effacer beaucoup de choses d'un coup : soyez donc très prudents.

```
$ cp mesTextes/ copyMesTextes/
cp: omission du répertoire «mesTextes/»
$ cp -r mesTextes/ copyMesTextes/ # il faut -r pour copier un répertoire
$ ls
copyMesTextes  mesTextes
$ rm mesTextes/
rm: impossible de supprimer «mesTextes/»: est un dossier
$ rm -r mesTextes/  # il faut -r pour supprimer un répertoire
$ ls
copyMesTextes
$ rmdir copyMesTextes/ # rmdir ne peut supprimer qu'un répertoire vide
```

1. Les fichiers spéciaux . et .. ne peuvent être effacés d'un répertoire (sinon il ne serait plus possible de trouver le répertoire parent ou de faire référence à lui-même en étant dans un répertoire). Ils ne sont détruits par le système que lorsque le répertoire est lui-même détruit. Un répertoire qui ne contient que ces deux fichiers spéciaux est considéré vide.

```
rmdir: échec de suppression de «copyMesTextes/»: Le dossier n'est pas vide
$ rm copyMesTextes/*    # je supprime tout le contenu du répertoire
$ ls -a copyMesTextes
.  ..                   # maintenant il est vide
$ ls
copyMesTextes
$ rmdir copyMesTextes/ # je peux le supprimer avec rmdir
```

### 5.1.4 Liens physiques et symboliques

Sous Windows ou MacOS, vous étiez surement habitués à utiliser des raccourcis : des liens permettant de faire un renvoi vers un fichier placé ailleurs dans l'arborescence des fichiers (sous forme d'une icône sur le Bureau de Windows pour lancer un programme par exemple). Sous Linux de tels liens sont nommés *liens symboliques*.

Sous Linux, il existe également un autre type de lien, inconnu sous Windows : les *liens physiques* (aussi nommés *lien en dur*, *hard link* ou *liens logiques*). Un *lien physique* est un autre nom qui pointe sur le même contenu alors qu'un *lien symbolique* est un autre nom qui pointe sur le nom original du fichier (Fig. 5.1). Si l'on efface le fichier original, le lien symbolique pointera alors vers un élément qui n'existe plus alors que le lien physique, pointera toujours vers le même contenu.

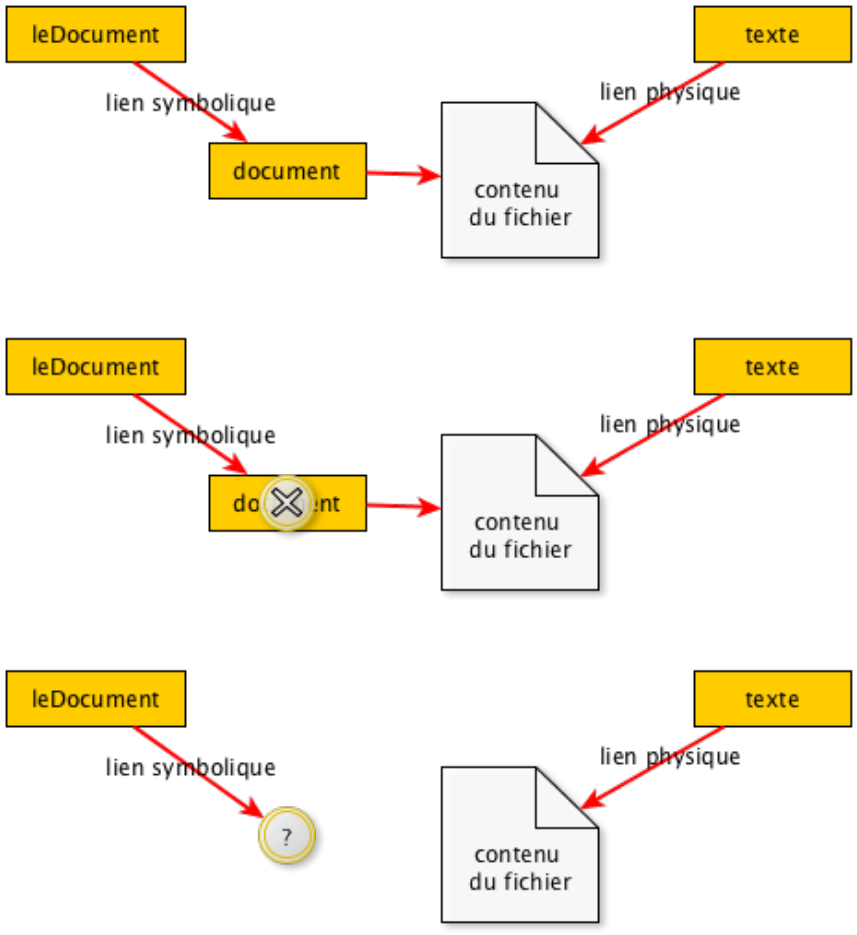


**Figure 5.1 – Différence entre lien symbolique et physique.**

Chaque fichier créé dans l'arborescence des fichiers possède des meta-données : des informations

comme la date de création, la date de dernière modification, etc. Ces méta-données comprennent également un compteur qui indique combien de noms sont utilisés pour le désigner. Quand on efface un fichier, on efface en réalité le nom qui le désigne : tant qu'il existe un autre nom pointant sur le même contenu, le fichier n'est pas effacé.

#### ln -s

Pour créer un lien symbolique, il faut utiliser la commande `ln` («*LiNk*») avec l'option `-s`. En listant les fichiers avec `ls -l`, on peut voir que le lien pointe sur le nom original du fichier :

```
$ pwd
/home/martin/Documents
$ ln -s ../Images/image1.jpg img.jpg
$ ls -al
total 0
lrwxrwxrwx  1 martin martin   20 août  10 19:37 img.jpg -> ../Images/image1.jpg
```

#### ln

Pour créer un lien physique, il faut utiliser la commande `ln` sans l'option `-s`. Le nouveau nom pointe alors sur le même contenu et la commande `ls -l` ne signale pas qu'il y a un lien, pourtant remarquez le nombre 2 placé en deuxième position, juste après l'étrange `-rw-rw-r--` dont nous verrons la signification plus tard : il indique qu'il y a deux noms pointant sur le même contenu :

```
$ ln ../Images/image1.jpg picture.jpg
$ ls -al
total 0
lrwxrwxrwx  1 martin martin   20 août  10 19:37 img.jpg -> ../Images/image1.jpg
-rw-rw-r--  2 martin martin    0 août  10 16:01 picture.jpg
```

### 5.1.5 Les inodes

Les méta-données des fichiers dont nous avons parlé dans la section précédente sont contenues dans un *inode*. Un inode est une structure de données (vous comprendrez mieux cette notion après le cours de C), un groupe d'informations permettant d'identifier chaque fichier de manière unique et décrivant ses propriétés. Ces informations indiquent le type du fichier (répertoire, lien, fichier ordinaire), des dates importantes (création, dernière modification, dernière consultation), le nombre de noms qui le désignent (liens physiques), le propriétaire du fichier, son groupe et leurs droits d'accès (nous verrons ces notions plus tard). Chaque fichier est représenté par un inode qu'il conserve jusqu'à sa destruction. Chaque inode possède un numéro unique qui permet également d'identifier le fichier.

#### ls -i

Nous pouvons vérifier qu'un lien physique pointe bien vers le même contenu que le nom original en affichant le numéro inode du fichier pointé par ces noms avec l'option `-i` de la commande `ls`. Vous constaterez que ce numéro est identique :

```
$ ls -i picture.jpg
```

```
709029 picture.jpg
$ ls -i ../Images/image1.jpg
709029 ../Images/image1.jpg
```

Par contre le lien symbolique créé avant, pointe sur un autre inode :

```
$ ls -i img.jpg
705323 img.jpg
```

### 5.1.6 Chercher des fichiers

#### locate

La commande `locate` permet de localiser rapidement tous les fichiers dont le nom contient un mot clef. La commande `locate` ne consulte qu'une base de données des fichiers. Pour avoir une liste à jour il faut lancer la commande `sudo updatedb` ; `sudo` indique que vous souhaitez exécuter une commande en tant qu'administrateur (super-utilisateur) de la machine. Votre mot de passe vous sera demandé pour bien vérifier que vous êtes administrateur. Généralement `updatedb` est lancé automatiquement une fois par jour pour actualiser la base de donnée, mais vous pouvez avoir besoin de lancer la commande à la main si vous chercher des fichiers créés très récemment.

#### find

La commande `find` permet d'effectuer une recherche à un emplacement donné en premier paramètre. Divers filtres peuvent être utilisés pour la recherche et le plus intéressant est qu'il est possible d'appliquer directement un traitement au résultat de la recherche.

Voici par exemple quelques un des filtres qui peuvent être utilisés :

**-name** recherche sur le nom du fichier,

**-user** recherche sur le propriétaire du fichier,

**-type** recherche sur le type du fichier (d=répertoire, f=fichier normal, etc.),

**-size** recherche sur la taille du fichier,

**-atime** recherche par date de dernière consultation du fichier,

**-mtime** recherche par date de dernière modification du fichier,

**-ctime** recherche par date de création du fichier.

```
$ find ~/cours/python/ -name px23-1.py  # cherche le fichier nommé px23-1.py
$ find / -user martin                   # fichier appartenant à martin
$ find . -mtime 0               # fichier modifié il y a moins de 24 heures
$ find ~ -size +10M             # fichier de plus de 10Mo
$ find ~/Images/ -size -5k      # fichier de moins de 5Ko
$ find . -atime -7              # fichier consulté il y a moins de 7 jours
$ find / -name Images -type d   # un répertoire nommé "Images"
* find . -mmin +0 -mmin -5    # fichier modifié entre 1 et 5 minutes auparavant
```

Pour plus d'informations sur les filtres existant et leurs utilisations possibles utilisez `man find`.

Il est possible de faire subir différents traitements aux fichiers trouvés. Voici quelques un des traitements possibles :

**-print** affiche le résultat de la recherche (le chemin du fichier trouvé), c'est l'option par défaut,

**-printf** permet d'imprimer le résultat mais avec un formatage, `printf` est le nom d'une fonction C, reportez vous à la page de manuel (section 3 pour les fonctions C) pour savoir comment l'utiliser.

**-delete** efface définitivement tous les fichiers trouvés (à manipuler avec précaution),

**-exec** exécute une commande pour chaque fichier trouvé ( représente le nom du fichier, le point virgule signale la fin de la commande),

**-ok** fait la même chose que `-exec` mais une confirmation est demandée pour chaque fichier à traiter.

```
$ find . -mtime 0 -print          # imprime le résultat de la recherche
$ find . -mtime 0 -printf "%p - %u\n" # imprime le nom du fichier (%p),
                                      # un tiret, le nom de l'utilisateur (%u)
                                      # et un saut de ligne (\n).
$ find /tmp/ -mtime 300 -delete  # efface définitivement les fichiers trouvés
$ find ~/Images -type f -name "a*" -exec cp "{}" "{}.bak" \; # cherche les
                                  # fichiers commençant par "a" présents
                                  # dans ~/Images et en fait une copie
```

Je ne peux ici, qu'effleurer quelques une des possibilités offertes par `find`, je vous renvoie encore à la documentation pour en savoir plus.

## 5.2 Utiliser les redirections et les pipes

Les commandes utilisées en console, peuvent afficher deux sortes d'informations : le résultat de leur travail et des messages d'erreur. Afin de récupérer le résultat d'une commande sans être gêné par les éventuels messages d'erreur ou d'avertissement, ces deux types d'informations ne sont pas transmises sur la même sortie : il existe une sortie standard et une sortie d'erreur. Avec les commandes que nous avons utilisées jusqu'à maintenant, toutes les informations provenant des deux sorties étaient affichées dans la console, mais il est possible de rediriger ces informations.

### 5.2.1 Redirection de la sortie standard : > et >>

Le symbole `>` permet de rediriger le résultat d'une commande dans un fichier. Par exemple, la commande suivante permet d'enregistrer le résultat de la commande `ls` dans le fichier `result-ls.txt`. Si ce fichier n'existe pas il sera créé :

```
$ ls > result-ls.txt
```

Cependant, si le fichier existe déjà son contenu sera remplacé par le résultat envoyé par `ls`. Pour éviter d'écraser le contenu du fichier, il est possible d'utiliser le symbole `>>` qui permet d'ajouter le résultat d'une commande dans un fichier déjà existant. Si le fichier n'existe pas encore, il est créé :

```
$ ls >> result-ls.txt
```

Si on exécute une commande en provoquant volontairement une erreur (on demande à `ls` d'afficher les fichiers `.jpg` alors qu'il n'y en a pas), on peut constater que les messages d'erreur, eux, sont toujours affiché dans la console.

```
$ ls *.jpg > result-ls.txt
ls: impossible d'accéder à *.jpg: Aucun fichier ou dossier de ce type
```

Remarque : en ne redirigeant rien dans un fichier, il est possible de créer un fichier vide.

```
$ ls    # ce répertoire est vide
$ > nouveau.txt
$ ls    # le répertoire contient maintenant un fichier vide
nouveau.txt
```

### 5.2.2 Redirection des erreurs : 2> et 2>>

Si l'on veut envoyer les messages d'erreur dans un fichier on peut utiliser les redirections équivalentes aux précédentes : `2>` pour envoyer les erreurs dans un fichier en écrasant son contenu, `2>>` pour ajouter les erreurs à la suite du contenu du fichier.

Si l'on veut envoyer le résultat d'une commande et les erreurs dans deux fichiers différents, il est possible de cumuler les deux instructions :

```
$ ls *.jpg . > result-ls.txt 2> error-ls.txt
```

Ici, `ls` retournera une erreur si il n'y a pas de fichier finissant par `.jpg` et la placera dans le fichier `error-ls.txt` puis listera le contenu du répertoire courant (`.` désigne le répertoire lui même) et placera ce résultat dans le fichier `result-ls.txt`.

### 5.2.3 Redirection vers /dev/null

Il peut arriver parfois que l'on veuille juste voir le résultat d'une commande, sans être gêné par les messages d'erreurs, ou l'inverse. Il est alors possible de rediriger une des sorties dans le fichier spécial `/dev/null`. Une sortie ainsi redirigée est simplement envoyée dans les limbes.

```
$ ls *.jpg *.txt 2> /dev/null
```

### 5.2.4 Redirection des deux sorties 2>&1 et 1>&2

Si vous avez commencé à écrire une commande pour envoyer la sortie standard dans un fichier, pour ne pas retaper le nom du fichier, vous pouvez ajouter l'instruction `2>&1` pour envoyer les erreurs au même endroit. Par exemple :

```
$ ls *.jpg . > result-ls.txt 2>&1

$ ls *.jpg . >> result-ls.txt 2>&1
```

Si la sortie standard est ajoutée au contenu d'un fichier avec `>>` notez que l'instruction `2>&1` ne change pas : les erreurs seront aussi ajoutées au fichier et n'écraseront pas son contenu initial.

Si vous aviez commencé par rediriger la sortie d'erreur, vous pouvez utiliser 1>&2 pour envoyer la sortie standard au même endroit.

```
$ ls *.jpg . 2> result-ls.txt 1>&2

$ ls *.jpg . 2>> result-ls.txt 1>&2
```

Il est facile de se mélanger avec ces instructions. Il suffit d'inverser la position des différents caractères pour que l'instruction prenne un tout autre sens... Alors souvenez vous : on indique en premier le numéro de la sortie que l'on veut rediriger (1), puis le chevron qui indique la redirection (>), enfin on indique (&2) qui signifie "comme pour la sortie 2".

Notez qu'il existe des redirections qui permettent de rediriger les deux sorties en même temps : &> et &>>. Bien qu'elles soient plus simples, on rencontre souvent les redirections précédentes et il est bon de les connaitre.

```
$ ls *.jpg . &> result-ls.txt

$ ls *.jpg . &>> result-ls.txt
```

### 5.2.5 Redirection de l'entrée standard <

De même qu'il est possible d'envoyer le résultat d'une commande dans un fichier, il est possible d'utiliser le contenu d'un fichier comme entrée d'une commande. Par exemple, je peux envoyer le résultat de la commande du dans un fichier puis donner ce fichier à la commande sort -nr qui va le trier numériquement en ordre inverse puis afficher le résultat dans la console.

```
$ du > result-du.txt
$ sort -nr < result-du.txt
```

Cependant, si c'est pour réutiliser le résultat d'une commande immédiatement dans une autre commande, les redirections ne sont pas forcement des plus pratiques : il vaut mieux utiliser un pipe (§. 5.2.6).

### 5.2.6 Connecter la sortie d'une commande sur l'entrée d'une autre : le pipe |

Il est possible de transmettre directement le résultat d'une commande à la suivante. Pour réaliser cela, on utilise ce qu'on appelle un *pipe* (« *un tuyau* », prononcez « *paille-pe* »). Le pipe est représenté par le caractère | et est placé entre les différentes commandes qui doivent utiliser le résultat l'une de l'autre.

Voici par exemple un enchaînement de commandes :

```
$ du | sort -nr | head
```

La commande du fournit des informations sur l'usage du disque, la commande sort -nr trie ces informations numériquement en ordre inverse et head n'affiche que les 10 premières lignes. Nous verrons le détail de ces deux dernières commandes un peu plus loin.

Bien que cela soit assez rare, on peut vouloir transmettre également les messages d'erreur à la

commande suivante. Pour cela, on peut utiliser l'instruction `2>&1 |`.

Il peut arriver que l'on veuille exécuter plusieurs commandes et envoyer le résultat cumulé à une autre commande. Cela est possible en regroupant les deux commandes entre parenthèses, et en utilisant le `;` ou le saut de ligne pour les séparer, avant de d'envoyer le résultat à une autre commande avec un pipe :

```
$ (ls Documents/ ; ls Images/) | sort
$ (ls Documents/
> ls Images/) | sort
```

On peut procéder de même pour envoyer le résultat de plusieurs commandes dans un fichier, on peut les entourer de parenthèses avant d'indiquer la redirection :

```
$ (echo "listing des fichiers" ; ls .) > listing.txt
$ (echo "listing des fichiers"
> ls) > listing.txt
```

### 5.2.7 Enchaîner les commandes si tout se passe bien : &&

L'instruction `&&` permet d'enchaîner deux commandes, mais de ne réaliser la seconde que si la première s'est bien passée :

```
$ touch /home/test && echo "creation de fichier réussie"
touch: impossible de faire un touch «/home/test»: Permission non accordée
$ touch ~/doc/test && echo "creation de fichier réussie"
creation de fichier réussie
$ ls ~/doc/
test
```

## 5.3 Consulter, trier, traiter le contenu des fichiers

Nous allons voir maintenant des commandes qui permettent d'afficher, trier et transformer le contenu de fichiers passés en paramètre ou d'informations transmises via l'entrée standard.

### 5.3.1 Consulter des fichiers

#### cat

Pour visualiser le contenu d'un fichier texte dans la console, vous pouvez utiliser la commande `cat` («*ConcATenate*») : tout le contenu du fichier sera affiché d'un bloc dans la console. L'option `-n` permet de numéroter les lignes. Si vous indiquez plusieurs fichiers en paramètre, les contenus de tous les fichiers seront affichés les uns à la suite des autres. Ceci peut-être utile quand vous désirez utiliser tout le texte retourné par `cat` comme donnée à traiter avec une autre commande, pour rechercher une information dans plusieurs fichiers à la fois par exemple, ou pour rassembler plusieurs textes dans un seul fichier.

```
$ cat fichier1.txt fichier2.txt
```

#### less

La commande `less` permet d'afficher le fichier sur tout l'écran de la console et de se déplacer ou faire une recherche dans le texte. C'est cette commande qui est utilisée par défaut pour afficher les pages du manuel, vous pouvez donc utiliser les raccourcis clavier que nous avons vu précédemment (§. 4.5) pour vous déplacer et faire une recherche dans le texte.

#### head & tail

Les commandes `head` et `tail` («*tête*» et «*queue*») permettent respectivement d'afficher les première ou les dernières lignes d'un fichier. L'option `-n` suivie d'un nombre permet de choisir le nombre de lignes à afficher. Avec `tail`, l'option `-f` permet de suivre l'évolution du fichier : cela permet de suivre un fichier log et de voir les lignes s'afficher dès qu'elles sont ajoutées par le système. Le paramètre `-s` suivi d'un nombre permet d'indiquer le nombre de secondes souhaité entre chaque rafraichissement de l'affichage.

```
$ tail -n3 -s5 -f /var/log/syslog
```

### 5.3.2 Traiter, trier, comparer des fichiers

#### sort

La commande `sort` permet de trier ligne par ligne, le contenu d'un fichier ou des données passées en entrée. L'option `-o` suivie d'un chemin permet d'écrire le résultat dans un autre fichier, `-r` permet de faire le tri en ordre inverse et `-n` permet de faire le tri en fonction des nombres trouvés dans chaque ligne. Il existe de nombreuses autres options, comme d'habitude, reportez vous au manuel pour en savoir plus.

#### uniq

La commande `uniq` permet de supprimer les lignes identiques qui se suivent dans un fichier. Pour enlever les lignes doublons dans un fichier, il faut donc penser à trier le fichier pour que les lignes identiques soient placées côte à côte avant d'appliquer `uniq`.

#### wc

La commande `wc` («*Word Count*») permet de compter le nombre d'octets[2] (option `-c`), de mots (option `-w`), ou de lignes (option `-l`) dans un fichier texte. Cette commande permet par exemple de compter le nombre de résultats retournés par une autre commande :

```
$ wc -l < liste.txt
40
$ sort liste.txt | uniq | wc -l
20
```

2. Pour un texte composé uniquement de caractères codés sur 1 octet, l'option `-c` donne le nombre de caractères. Vous verrez dans les prochains cours que certains caractères (les caractères accentués par exemples) peuvent occuper plusieurs octets de données.

Ici, nous voyons que le fichier `liste.txt` ne contient que 20 lignes uniques.

### cut

La commande `cut` permet de sélectionner une partie de chaque ligne d'un fichier. L'option `-c` permet d'indiquer un intervalle de caractères à découper, `-f` permet d'indiquer les champs à découper et `-d` permet d'indiquer le caractère à considérer comme séparateur entre ces champs.

Les groupes de caractères ou de champs peuvent être définis de différentes manières :

**-3** indique qu'il faut prendre du début jusqu'au 3ème.
**3-** indique qu'il faut prendre du 3ème jusqu'à la fin.
**3-5** indique qu'il faut prendre du 3ème jusqu'au 5ème.
**3,5** insique qu'il faut prendre le 3ème et le 5ème.

Par exemple pour ne garder que la première colonne d'information retournée par la commande `ls -l` on peut découper le premier champ délimité par un espace ou bien découper les 10 premiers caractères :

```
$ ls -l | cut -d " " -f 1
$ ls -l | cut -c 1-10
```

Pour prendre le premier champ de chaque ligne d'un fichier `.csv` utilisant la virgule comme séparateur, les trier et les placer dans un autre fichier :

```
$ cut -d , -f 1 adresses.csv | sort > noms.txt
```

### diff

La commande `diff` permet de comparer deux fichiers ligne par ligne. Elle ne donne en sortie que les lignes qui diffèrent en indiquant leurs numéros. Cette commande est très pratique quand vous avez par exemple plusieurs versions d'un fichier et que vous ne savez pas trop ce qui diffère d'une version à l'autre : la commande diff permet d'afficher directement ces différences.

```
$ diff comptine.txt comptine2.txt
0a1
> comptine :
2,3c3,4
< 4, 5, 6, cueillir des cerises ;
< 7, 8, 9, dans mon panier neuf ;
---
>    4, 5, 6, cueillir des cerises ;
>    7, 8, 9, dans mon panier neuf ;
5,6d5
<
< fin
```

Ces informations indiquent les modifications à apporter à `comptine.txt` pour obtenir `comptine2.txt` :

**0a1** Après la ligne 0 de `comptine.txt` il faudrait ajouter (a=add) la ligne 1 de `comptine2.txt`. À la ligne suivante, le texte précédé du caractère `>` indique le contenu à ajouter.

**2,3c3,4** Les lignes 2 à 3 de `comptine.txt` devraient être remplacées (c=change) par les lignes 3 à 4 de `comptine2.txt`. Les lignes suivantes indiquent le contenu des deux lignes à remplacer précédé de <, puis les lignes de remplacement précédées de >.

**5,6d5** Les lignes 5 à 6 de `comptine.txt` devraient être effacées (d=delete) car elles n'existent pas après la ligne 5 de `comptine2.txt`. Les lignes suivantes indiquent les deux lignes à supprimer précédées de <

#### grep

La commande `grep` permet de rechercher une expression correspondant à un motif donné à l'intérieur d'un fichier. Par exemple il est possible de rechercher une ligne faisant référence à `Ethernet` dans la liste des périphériques PCI :

```
$ lspci | grep Ethernet
```

L'option `-i` permet de rendre `grep` insensible à la casse, `-n` permet d'afficher le numéro de la ligne dans le fichier, `-v` permet de rechercher les lignes qui ne contiennent pas l'expression donnée, `-r` permet de faire une recherche dans tous les fichiers et sous-répertoires d'un répertoire donné (la commande rgrep est équivalente à `grep -r`) et `-E` permet de passer une expression sous forme d'une expression régulière (la commande egrep est équivalente à `grep -E`). Sans l'option `-E`, sous linux, grep est quand même capable de comprendre des expressions régulières, mais il faut respecter une syntaxe particulière : nous en reparlerons un peu plus loin.

## 5.4 Découvrir les expressions régulières et les utiliser

Comme nous venons de le voir, `grep` peut utiliser une expression régulière pour effectuer une recherche. Une *expression régulière* (abrégé en *regex*), aussi appelée *expression rationnelle*, permet de décrire la composition d'une expression sans avoir besoin de l'écrire directement. Par exemple, si je veux chercher un numéro de téléphone dans un fichier avec une commande comme `grep`, il serait fastidieux de lister toutes les combinaisons possibles de caractères correspondant à un numéro valide pour les chercher ensuite dans le fichier. A la place, je peux décrire avec une expression régulière ce qu'est un numéro de téléphone français valide[3] (isolé sur une ligne) et l'utiliser pour faire ma recherche :

```
^0[1-9]([-. ]?[0-9]{2}){4}$
```

Pareil avec un email :

```
^[a-z0-9._-]+@[a-z0-9._-]{2,}\.[a-z]{2,4}$
```

### 5.4.1 Les expressions régulières

« *Au secours ! Mais c'est quoi cette horreur !* »

Les expressions régulières peuvent paraitre très compliquées de prime abord et c'est pour cette raison, quand on est débutant, qu'on a tendance à vouloir repousser leur apprentissage à plus tard.

3. Cette expression régulière ne tient pas compte des numéros courts.

Pourtant elles sont tellement indispensables qu'on ne peut pas y échapper, alors autant s'y mettre dès à présent.

Si l'on examine chaque élément d'une expression régulière séparément, elle devient plus facile à comprendre. Pour savoir si un fragment de texte correspond à une expression régulière, la machine procède à une comparaison caractère par caractère. Par exemple, si mon expression régulière contient le caractère a, un a devra être présent dans mon fragment de texte. Comme une correspondance exacte n'est pas très utile, les *méta-caractères* permettent de faire des comparaisons par type de caractères. Par exemple le méta-caractère . représente n'importe quel caractère, mais attention, là encore, le . ne correspond qu'à un seul caractère. Pour désigner un groupe de plusieurs caractères il faudra ensuite ajouter des quantificateurs. Par exemple le méta-caractère + indique la présence de 1 ou plusieurs caractères : l'expression régulière a+ indique donc la présente d'un ou plusieurs caractères a successifs, "a", "aa", "aaa", etc.

La liste suivante indique les différents métacaractères utilisés accompagnés d'exemples :

**^** indique le début d'une ligne.

- **^abc** abc en début de ligne.

**$** indique la fin de la ligne.

- **abc$** abc en fin de ligne.

**\<** indique le début d'un mot.

- **\<a** un mot commençant par a.

**\>** indique la fin d'un mot.

- **a\>** un mot finissant par a.

**[ ]** définit une liste de caractères autorisés. Cette expression ne capturera qu'un seul caractère correspondant à un des éléments de la liste.

- **[abc]** le caractère a, b ou c.
- **[a-c]** un caractère de a à c.
- **[a-cf]** un caractère de a à c ou f.
- **[a-cf-h]** un caractères de a à c ou de f à h.
- **[a-c3-7]** un caractère de a à c ou un chiffre de 3 à 7
- **[0-5-]** un chiffre de 0 à 5 ou -.

**[^ ]** définit une liste de caractères interdits. Cette expression ne capturera qu'un seul caractère ne correspondant à aucun des éléments de la liste. Si on veut capturer le caractère ^ alors il suffit de ne pas le mettre en début de liste (voir dernier exemple).

- **[^abc]** un caractère sauf a, b, ou c.
- **[^0-9]** pas un chiffre.
- **[^a-zA-Z]** pas une lettre minuscule ou majuscule sans accent.
- **[012^]** le caractère 0, 1, 2 ou ^.

**.** représente n'importe quel caractère

- **a.b** n'importe quel groupe de 3 caractères qui commence par a et fini par b.

***** l'élément qui précède peut être présent 0, 1 ou N fois.

- **ab*c** la chaîne « *ac* » ou « *abc* » ou « *abbc* » ou « *abbbc* », etc.
- **a[to]*** la chaîne « *a* » suivie de 0, 1 ou plusieurs lettres parmi t et o.

**?** l'élément qui précède est optionnel.

- **ab?c** la chaîne « *abc* » ou « *ac* ».
- **a[bcd]?e** le caractère a, suivi éventuellement de b,c ou d, et suivi de e.

**+** l'élément qui précède peut être présent 1 ou N fois.

- **a+** la chaîne « *a* » ou « *aa* » ou « *aaa* », etc.

**{ }** permet de définir un intervalle. Le premier chiffre indique le nombre minimal de caractères à capturer, le second, le nombre maximal. Si on omet le premier, 0 est sous-entendu, si on omet le second, il n'y a pas de limite, $+\infty$ est sous-entendu.

- **[a-z]{3,5}** un groupe de 3 à 5 lettres parmi les lettres de a à z.
- **[a-z]{6,}** un groupe d'au moins 6 lettres parmi les lettres de a à z.
- **[0-9]{,8}** rien ou un nombre jusqu'à 8 chiffres.
- **[0-9]{4}** un nombre de 4 chiffres.

Comme `*` et `+`, cette expression est un quantificateur qui permet d'indiquer une quantité de caractères à capturer. Il n'est pas possible de combiner plusieurs quantificateurs à la suite puisque leur sens est contradictoire. Par exemple `[a-z]*{1,3}` n'a aucun sens, puisque `*` indique n'importe quel nombre de caractères et `{1,3}` indique seulement entre 1 et 3 caractères.

**( )** délimite un morceau d'une expression. On peut ajouter un quantificateur après pour indiquer que toute une expression est répétée plusieurs fois. On peut aussi faire référence à un morceau entre parenthèses avec `\` suivi du numéro du morceau.

- **([0-9] ){4}** 1 chiffre suivi d'un espace, le tout répété 4 fois.
- **([-. ][0-9]{2}){4}** 2 chiffres précédés d'un tiret, point ou espace, le tout répété 4 fois.
- **(pé|pa)\1** capture la syllabe « *pa* » ou « *pé* » puis la réutilise grâce à `\1`, l'expression capturée est donc « *papa* » ou « *pépé* ».
- **(.+)(.+)\1\2** capture un groupe de 1 ou plusieurs lettres puis un second, répète le premier puis le second. L'expression capturée peut par exemple être « *coucou* » les deux groupes étant « *c* » et « *ou* » ou bien « *co* » et « *u* ».
- **<a href="mailto:(.*)">\1<\a>** capture une balise HTML[4] correspondant à un lien vers une adresse email avec la même adresse email à l'intérieur de cette balise.

**|** sépare deux possibilités. Si l'expression est complexes, ont peut délimiter ce qui fait partie des deux possibilités par des parenthèses.

- **a|b** le caractère a ou b.
- **un|une|le** le mot « *un* », « *une* » ou « *le* ».
- **^a|^g** une ligne commençant par un a ou par un g.
- **a.(c|d)** un groupe de 3 caractères commençant par a et finissant par c ou d.

**** permet d'utiliser un caractère spécial tel quel sans qu'il soit interprété, c'est un caractère d'*échappement*. Les caractères spéciaux entre crochets n'ont pas besoin d'être échappés ; pour éviter toute confusion avec la fin du groupe de caractères défini entre les crochets, le crochet fermant doit être placé en première position si on désire le mettre dans une liste de caractères à filtrer.

- **coucou\?** la chaîne « *coucou ?* ».
- **<\\.*>** une balise fermante dans le langage HTML.

4. Une balise HTML est un groupe de caractère commençant par le symbole < et finissant par le symbole >. Les balises donnent des informations de structure ou de mise en page. Par exemple <body> et </body> indiquent respectivement le début et la fin du corps d'une page.

**[*$]** un caractère * ou $.

**[][]** un caractère [ ou ].

Une dernière chose à savoir est que toutes les commandes n'utilisent pas les mêmes expressions régulières[5]. Certaines fonctionnalités que nous venons de voir font partie des expressions régulières complètes et ne sont pas disponibles avec toutes les commandes, notamment grep et sed. Ce tableau résume les fonctionnalités disponibles :

| E.R. réduites | | | | | E.R. complètes | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ^ | $ | [ ] | . | * | ( ) | ? | + | \| | { } |
| grep, sed | | | | | | | | | |
| egrep, sed -r, awk | | | | | | | | | |

Prenez le temps de tester les différents exemples donnés dans la longue liste précédente pour vous familiariser avec l'utilisation des expressions régulières. Vous pouvez tester ces expressions régulières avec grep ou egrep sur le fichier adresses.txt fourni avec ce cours. Par exemple :

```
$ grep '^G' adresses.txt              # cherche une ligne commençant par G
$ grep '00$' adresses.txt             # cherche une ligne finissant par 00
$ egrep '\<T|r\>' adresses.txt        # cherche un mot commençant par T
                                      # ou finissant par r
$ grep '[^a-zA-Z]' adresses.txt       # cherche tout ce qui n'est pas une lettre
$ egrep '([a-zA-Z])\1' adresses.txt   # cherche une lettre suivie d'une
                                      # lettre identique
```

Maintenant, reprenons les expressions proposées au début de cette sections et analysons les pas à pas. Commençons par le numéro de téléphone :

```
'^0'        # une ligne commençant par 0
'[1-9]'     # un chiffre entre 1 et 9
'[-. ]?'    # un caractère optionnel qui peut être un tiret, point ou espace
'[0-9]{2}' # deux chiffres entre 0 et 9
'[-. ]?[0-9]{2}'  # deux chiffres entre 0 et 9 précédés éventuellement
                  # d'un tiret, point ou espace
'([-. ]?[0-9]{2}){4}'  # la même expression que précédemment répétée 4 fois
'$'         # la fin de la ligne

'^0[1-9]([-. ]?[0-9]{2}){4}$' # un numéro de téléphone isolé sur une ligne
```

Maintenant l'adresse email :

```
'^'               # le début de la ligne
'[a-z0-9._-]+'    # un ou plusieurs caractères parmi les suivants :
                    # un caractère entre a et z, un chiffre entre 0 et 9,
                    # un point, un underscore ou un tiret
'@'               # le caractère arobase
```

5. Sous Linux (et MacOS) il est possible d'utiliser les expressions complètes avec grep (sans l'option -E) et sed, à condition de précéder chaque caractère spécial qui fait partie des expressions régulières complètes avec le caractère d'échappement \, cependant les commandes ainsi écrites ne sont pas garanties de fonctionner sur tous les systèmes Unix. Le manuel indique aussi que « *la version traditionnelle d'*egrep *ne connaît pas le méta-caractère* {*, et certaines implantations d'*egrep *utilisent* \{ *à la place, si bien que des scripts shell portables devraient éviter* { *dans les motifs de* grep -E *et utiliser* [{] *pour désigner un caractère* { ».

```
'[a-z0-9._-]{2,} # deux caractères ou plus parmi les suivants :
                 # un caractère entre a et z, un chiffre entre 0 et 9,
                 # un point, un underscore ou un tiret
'\.'             # le caractère .
'[a-z]{2,4}'     # un groupe de 2 à 4 lettres entre a et z

'^[a-z0-9._-]+@[a-z0-9._-]{2,}\.[a-z]{2,4}$'  # une adresse email
                                              # isolée sur une ligne
```

### 5.4.2 Utiliser les expressions régulières

Lors de la programmation d'applications traitant du texte ou simplement pour modifier rapidement un de vos fichiers, vous aurez fréquemment besoin des expressions régulières. Si par exemple, vous voulez récupérer le texte contenu dans un lot de pages web et supprimer toutes les balises HTML, vous pourrez utiliser un outil manipulant des expressions régulières.

Les vieux programmeurs font cela avec les commandes shell sed ou awk, les moins vieux utilisent le langage *Perl* pour créer un script (un petit programme) qui effectue le traitement du fichier. Quant à vous, vous allez apprendre à manipuler le langage *Python* dans un des cours de la licence et vous serez sans doute plus à l'aise avec ce langage pour créer un script utilisant les expressions régulières.

#### sed

La commande sed («*Stream EDitor*») lit un fichier ou l'entrée standard ligne par ligne. Un motif permet de sélectionner les lignes à traiter, un traitement est appliqué et le résultat est retourné ou non : le fichier d'origine n'est pas modifié sauf si on le demande explicitement.

Je ne vais pas détailler ici l'utilisation de sed. Si vous êtes curieux, vous pouvez aller jeter un œil au manuel, mais comme cette commande est vraiment très puissante et donc complexe à utiliser vous trouverez peut être plus éclairant les différents tutoriels disponibles sur internet pour apprendre à manipuler cette commande.

Voici quelques exemples d'utilisation de sed que vous pouvez tester sur le fichier adresses.txt fourni avec ce cours :

```
$ sed '3d' adresses.txt                 # supprime la troisième ligne
$ sed -n '3~5p' adresses.txt            # récupère les lignes de 5 en 5 en
                                        # commençant par la 3eme
$ sed -n '/Rafi/,+3p' adresses.txt      # récupère la première ligne contenant
                                        # la chaîne "Rafi" et les 3 suivantes
$ sed -n -e '1p' -e '6p' adresses.txt # récupère les lignes 1 et 6
$ sed 'y/aeio/4310/' adresses.txt       # remplace les voyelles a, e, i et o
                                        # par les chiffres 4, 3, 1 et 0
$ sed 's/01/06/g'   # remplace toutes les occurrences de 01 par 06
$ sed -rn '1~5s/([A-Z])([A-Z]*)/\1\L\2/pg' adresses.txt # remplace les noms
                    # tout en majuscules par des noms dont seule la première
                    # lettre est en majuscule puis affiche uniquement les
                    # lignes transformées
$ ls -d ~/* | sed -e 's|/home/.*/|/root/|' # liste les dossiers dans
                    # mon compte et remplace le chemin par celui du compte
```

```
                    # de l'administrateur (root)
```

### awk

La commande awk est elle aussi très puissante, c'est un langage de programmation à elle tout seule qui permet une recherche de chaînes d'après un motif donné et l'exécution d'actions sur les lignes sélectionnées. Elle est utile, par exemple, pour récupérer de l'information, générer des rapports, transformer des données.

Une grande partie de la syntaxe a été empruntée au langage C et les fonctionnalités de awk sont si nombreuses que je préfère vous renvoyer au manuel et aux différentes documentations disponibles sur Internet si vous désirez en savoir plus.

Voici quelques exemples très simples, tenant sur une ligne, d'utilisation de awk :

```
$ awk -F ":" '{ $2 = "" ; print $0 }' /etc/passwd   # imprime chaque ligne
            # du fichier /etc/passwd après avoir effacé le deuxième champs
$ who | awk '{print $1,$5}'       # imprime le login et le temps de connexion.
$ awk 'length($0)>20 {print $0}' adresses.txt # imprime les lignes de plus
                                              # de 20 caractères.
```

### exemple avec python

Avec un langage comme *Python*, vous pourrez écrire un script pour traiter votre texte avec des expressions régulières. Les commandes en langage python peuvent également être saisies une à une dans l'*interprète Python* qui les exécute au fur et à mesure. Pour lancer cet interprète, utilisez la commande python dans le shell :

```
$ python
Python 3.4.3 (default, Mar 10 2015, 14:53:35)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Après un message de bienvenue, l'interprète vous affiche son prompt >>>. A partir de là nous sommes en communication avec l'interprète et c'est lui qui va exécuter nos ordres. Nous devons donc taper des commandes en langage Python et non plus des commandes shell.

L'exemple suivant à juste pour but de vous montrer qu'il est possible d'utiliser des expressions régulières avec Python. Le langage Python ne vous étant pas famillier pour le moment, ne paniquez pas si vous ne comprenez pas comment ces instructions fonctionnent, ce sera l'objet d'un autre cours.

```
>>> import re        # cette commande demande à Python d'importer le
                     # necessaire pour utiliser des expressions régulières
>>> adresses = open("adresses.txt").read() # nous ouvrons le fichier
                     # adresses.txt et importons son contenu pour tester
                     # nos expressions régulières dessus
>>> re.findall('01.*', adresses) # avec findall, on recherche tous les éléments
                                 # correspondant à l'expression régulière '01.*'
['01 49 40 72 00', '01 49 40 72 00', '01 49 40 72 00', '01 49 40 72 10',
```

```
 '01 49 40 72 20']
>>> re.search("[A-Z][a-z]*", adresses).group(0) # avec search on recherche
                                    # le premier éléments (n°0) correspondent à
                                    # l'expression "[A-Z][a-z]*"
'Gilles'
>>> regex = re.compile("(([a-z]+).([a-z]+)@[a-z0-9._-]{2,}\.[a-z]{2,4})")
                # avec compile je prépare une expression régulière pour
                # capturer un email ainsi que le nom et le prénom dans le login
>>> resultat = regex.sub(r'<a href="mailto:\1">\2 \3</a>', adresses)
                # j'utilise cette expression pour remplacer l'email
                # capturé par un lien HTML qui permet d'envoyer un email
>>> print(resultat) # j'affiche le résultat obtenu
Gilles BERNARD
Directeur de l'IED
01 49 40 72 00
<a href="mailto:gilles.bernard@iedparis8.net">gilles bernard</a>

(...)
>>>
```

Quand vous commencerez à travailler sur les fichiers texte dans le cours de Python (*Méthodologie de la programmation*), vous verrez qu'il est possible d'effectuer certains traitement simples sans avoir recours aux expressions régulières. Cependant, à mesure que les traitements que vous souhaiterez effectuer se complexifieront, vous constaterez qu'il devient très difficile de s'en passer.

Quand vous en serez arrivés à ce point ou les expressions régulières vous seront indispensables, je vous invite à vous reporter à la page de la documentation de Python concernant l'utilisation des expressions régulières : https://docs.python.org/2/library/re.html.

# 6 Commandes avancées et commandes d'administration

Dans ce chapitre nous allons voir différentes commandes vous permettant de gérer les utilisateurs du système et les droits qui leurs sont accordés. Nous verrons comment lancer des commandes qui s'exécutent en arrière plan et comment créer des scripts : des petits programmes regroupant plusieurs commandes. Nous verrons enfin comment effectuer quelques tâches d'administration : surveiller le système ou installer un programme en lignes de commandes.

Une fois de plus, je vous conseille vivement de tester les commandes décrites et de consulter la documentation pour en savoir plus.

## 6.1 Super-utilisateur, utilisateurs, groupes et gestion des droits

Les systèmes Unix sont prévus pour être multi-utilisateurs et, comme vous avez pu le voir, chacun possède son compte personnel dans `/home/` où il organise ses affaires. Chaque utilisateur est identifié par un login et possède un mot de passe permettant de prouver qui il est. Quand un utilisateur se connecte avec ses identifiants, il est par défaut considéré comme un utilisateur lambda et son champ d'action est réduit. Même un administrateur utilise la machine comme un simple utilisateur : cela évite de commettre des erreurs, d'effacer un programme par mégarde, etc.

Quand un administrateur veut faire une action qui n'est pas permise à un simple utilisateur, il doit lancer une commande pour indiquer « *Je suis chef !* » et redonner son mot de passe pour obtenir le statut qui lui permet de faire ce qu'il veut. Un *administrateur* (ou *super-utilisateur*) a tous les droits, il peut créer, lire, effacer les fichiers qu'il souhaite, ajouter ou supprimer des programmes, etc. Quand il a fini, l'administrateur le signale et redevient simple utilisateur.

Les utilisateurs sous Unix sont répartis en groupes et chaque groupe dispose de droits particuliers. Par exemple, Martin fait partie du groupe des étudiants de L2, ce statut ne lui donne pas accès aux cours de L3, mais on l'a nommé tuteur et il peut accéder aux cours de L3 parce qu'il fait aussi partie du groupe des tuteurs. Ainsi un utilisateur peut faire partie de plusieurs groupes ce qui permet de définir avec précision qui peut accéder à quoi.

Il existe trois types de droits : le droits de lecture, d'écriture et d'exécution. Le droit de lecture indique si on a simplement le droit d'ouvrir un fichier pour en lire le contenu. Le droit d'écriture indique si l'on est autorisé à modifier un fichier ou le supprimer. Le droit d'exécution indique si on a le droit de le lancer au cas ou il s'agirait d'un programme.

Chaque fichier possède dans ses méta-données, dont nous avons déjà parlé (§. 5.1.5), des infor-

mations définissant quel utilisateur et quel groupe en est propriétaire ainsi que les droits de chacun. Inutile de préciser les droits d'un administrateur puisqu'il a tous les droits, par contre une information indique quels sont les droits des autres utilisateurs (qui ne sont ni le propriétaire, ni membres du groupe propriétaire du fichier). Il y a donc deux propriétaires (un utilisateur et un groupe) et trois informations sur les droits : droit de l'utilisateur, droit du groupe et droit des autres.

### 6.1.1 Visualiser le propriétaire et les droits d'un fichier

Nous allons maintenant revenir sur les informations affichées par `ls -al` et les examiner plus en détails. Voici quelques lignes extraites du listing de mon compte utilisateur :

```
$ ls -la ~
drwxr-xr-x 25 alinehuf alinehuf  4096 août  13 19:44 .
drwxr-xr-x  3 root     root      4096 août   7 13:32 ..
-rw-rw-r--  1 alinehuf alinehuf    40 août  13 18:59 .bash_aliases
-rw-------  1 alinehuf alinehuf 27921 août  13 19:21 .bash_history
-rw-r--r--  1 alinehuf alinehuf  3637 août  13 19:22 .bashrc
drwxr-xr-x  2 alinehuf alinehuf  4096 août   7 18:37 Bureau
-rw-rw-r--  1 alinehuf alinehuf    53 août  14 12:09 image.jpg
-rw-r--r--  1 alinehuf alinehuf   675 août  13 19:22 .profile
lrwxrwxrwx  1 alinehuf alinehuf     7 août  14 12:07 tmp -> Web/tmp
```

Le terme `root` est utilisé pour désigner l'administrateur du système sous Linux.

Les informations affichées pour chaque fichier sont, de gauche à droite :

- le type,
- le *masque de protection*, c'est à dire les droits de l'utilisateur, du groupe et des autres,
- le nombre de liens,
- l'utilisateur propriétaire,
- le groupe,
- la taille,
- la date de dernière modification,
- le nom.

Le type de fichier est représenté par un seul caractère placé au tout début de la ligne. Voici les différents types existants ; les trois premiers sont ceux que vous rencontrerez le plus souvent :

| type | code |
|---|---|
| standard | - |
| répertoire | d |
| lien symbolique | l |
| fichier spécial mode block | b |
| fichier spécial mode caractère | c |
| fichier spécial mode réseau | n |
| pipe nommé | p |

Les droits sont représentés de la manière suivante avec `r` pour «*read*» (lecture), `w` pour «*write*» (écriture) et `x` pour «*execute*» (exécution). Le symbole `-` indique qu'un de ces droits est absent :

| user | | | group | | | other | | |
|---|---|---|---|---|---|---|---|---|
| r | w | x | r | w | x | r | w | x |

Si on examine le listing on voit que `.bash_history` ne peut être lu et modifié que par l'utilisateur propriétaire du fichier, tandis que tous les utilisateurs ont tous les droits sur le lien symbolique. Un lien symbolique ne fait que pointer sur un autre fichier, ces droits ne sont pas pris en compte et ne sont donc pas modifiables, seuls ceux du fichier cible sont pris en compte.

Vous pouvez également constater que les fichiers dans mon compte sont attribués par défaut à un groupe qui porte mon nom. Le fichier spécial `..` pointe sur le répertoire `/home/` qui est logiquement la propriété de l'administrateur du système, nommé `root`, et est attribué au groupe `root`.

Les droits d'accès (on parle aussi parfois de *permissions*) peuvent être également représentés par des nombres en octal[1]. Les trois permissions sont alors représentées de la manière suivante : `r=4`, `w=2` et `x=1`. Pour indiquer qu'un fichier a plusieurs permissions, on additionne les valeurs : `rw=6`, `rwx=7`. Il y a donc un chiffre entre 0 et 7 pour chacun (user, group et other). Avec ce codage, les permissions d'un lien symbolique sont donc `777` et celles du fichier `.bash_history` sont `600`.

### 6.1.2 Gestion des droits

Voici les différentes fonctions qui vous permettent de modifier le propriétaire, le groupe ou les droits d'un fichier.

#### sudo et su

Certaines opérations sont réservées à l'administrateur de l'ordinateur. Vous êtes peut-être l'unique utilisateur de votre machine, néanmoins, quand vous vous êtes identifié avec votre nom et mot de passe en début de session, vous avez démarré votre session en tant que simple utilisateur.

Pour indiquer que vous désirez faire une opération en tant qu'administrateur de la machine, il faut utiliser la commande sudo («*Substitute User DO*»). Cette commande se place en début de ligne, avant la commande que vous désirez lancer en tant qu'administrateur :

```
$ touch /home/test
touch: impossible de faire un touch «/home/test»: Permission non accordée
$ sudo touch /home/test
[sudo] password for alinehuf:
$ ls -l /home/
total 4
drwxr-xr-x 25 alinehuf alinehuf 4096 août  14 13:03 alinehuf
-rw-r--r--  1 root     root        0 août  14 14:12 test
$ rm /home/test
rm : supprimer fichier vide (protégé en écriture) «/home/test» ? y
rm: impossible de supprimer «/home/test»: Permission non accordée
$ sudo rm /home/test
$ ls -l /home/
total 4
drwxr-xr-x 25 alinehuf alinehuf 4096 août  14 13:03 alinehuf
```

1. Nous avons généralement l'habitude de compter en décimal (10 chiffres de 0 à 9). En octal, il n'y a que 8 chiffres de 0 à 7.

J'ai tenté de créer un fichier test dans le répertoire /home/ avec touch /home/test, or ce répertoire appartient à root, l'action a donc été refusée. J'ai alors déclaré que j'étais administrateur avant de taper ma commande avec sudo touch /home/test ; j'ai saisi mon mot de passe et l'action a été acceptée.

Vous constatez que le fichier a été attribué à root et au groupe root puisque j'étais identifiée comme administrateur à ce moment là. Pour effacer le fichier il a donc fallu que je m'identifie à nouveau comme root.

La commande sudo permet en fait de changer d'identité, il est donc possible de passer certaines options à la commande pour prendre l'identité de n'importe quel autre utilisateur pourvu qu'on en ait le droit. Pour connaître ces options consultez le manuel.

Si vous avez besoin de faire une suite d'opérations en tant qu'administrateur il peut être fastidieux de retaper sudo devant chaque commande. La commande su permet de devenir un autre utilisateur pour la durée d'une session. Par défaut, si l'on ne précise pas l'identité de l'utilisateur dont on désire prendre l'identité, c'est l'utilisateur *root* qui est utilisé. La commande su permet de lancer une autre session shell à l'intérieur du shell. Cette session prend fin si vous tapez exit ou utilisez le raccourci Ctrl+D.

La documentation de Ubuntu indique que « *par défaut, sous Ubuntu, l'accès direct au compte système (root) est désactivé. La logique du système est d'utiliser* sudo *pour effectuer toutes les tâches administratives. Il est totalement déconseillé d'activer l'accès et d'utiliser directement le compte* root *sous Ubuntu.* »[2]. Ainsi, sous Ubuntu vous pouvez utilisez la commande sudo -i pour acquérir momentanément les droits d'administration tout en utilisant vos préférences et votre mot de passe utilisateur. Sous Ubuntu, vous verrez également qu'il est possible d'utiliser sudo su qui vous connecte en tant que root (avec le dossier personnel et les préférences de root), mais d'après la documentation, c'est fortement déconseillé.

```
alinehuf@pupuce:~$ sudo -i     # je me connecte avec des droits d'administrateur
root@pupuce:~# > /home/test    # mais je suis toujours dans mon compte (~)
root@pupuce:~# ls -l /home/    # je vois que le fichier créé appartient à root
total 4
drwxr-xr-x 22 alinehuf alinehuf 4096 oct.   8 16:27 alinehuf
-rw-r--r--  1 root     root        0 oct.  10 16:07 test
root@pupuce:~# rm /home/test   # mais je peux l'effacer
root@pupuce:~# exit            # et je sors
déconnexion

alinehuf@pupuce:~$ sudo su     # je me connecte en tant qu'administrateur
root@pupuce:/home/alinehuf# > /home/test # ce compte n'est plus le mien
root@pupuce:/home/alinehuf# ls -l /home/
total 4
drwxr-xr-x 22 alinehuf alinehuf 4096 oct.   8 16:27 alinehuf
-rw-r--r--  1 root     root        0 oct.  10 16:08 test
root@pupuce:/home/alinehuf# rm /home/test
root@pupuce:/home/alinehuf# exit
exit
alinehuf@pupuce:~$             # je retrouve mon prompt utilisateur
```

J'ai laissé ici l'affichage complet du prompt pour que vous puissiez remarquer les différents changements visibles. Après le lancement d'une session avec les droits d'administrateur, je ne suis

2. Voir la page https://doc.ubuntu-fr.org/root de la documentation d'Ubuntu pour plus d'informations.

plus identifiée comme l'utilisateur alinehuf mais comme l'utilisateur root. Si j'utilise sudo su, vous pouvez remarquer également que dossier /home/alinehuf n'étant pas le compte personnel de root, le chemin est affiché à la place de ~. Vous pouvez voir aussi que $ du prompt a été changé en #.

### chown

La commande chown («*CHange OWNer*») vous permet de donner la propriété d'un fichier à un autre utilisateur. Vous pouvez préciser uniquement le nom du nouvel utilisateur propriétaire ou bien également son groupe. Dans le cas d'un dossier, l'utilisation de l'option -R permet de changer également le propriétaire des sous-dossiers et fichiers :

```
$ ls -l
total 0
-rw-rw-r-- 1 martin martin 0 août  14 16:11 texte
$ sudo chown dudley texte
[sudo] password for martin:
$ ls -l
total 0
-rw-rw-r-- 1 dudley martin 0 août  14 16:11 texte
$ sudo chown dudley:muggles texte
$ ls -l
total 0
-rw-rw-r-- 1 dudley muggles 0 août  14 16:11 texte
```

Pour ne changer que le groupe propriétaire, il suffit d'omettre le nom de l'utilisateur :

```
$ chown :muggles texte
```

### chmod

La commande chmod permet de modifier les droits d'un fichier. Vous pouvez utiliser le code en octal pour préciser les droits ou utiliser une notation par lettres :

```
$ chmod 640 document.txt

$ chmod u+x document.txt

$ chmod u=rwx,g=r,o=- document.txt
```

La notation en octal permet de définir d'un coup tous les droits (pour user, group et other). Avec la notation par lettre il est possible d'ajouter ou de supprimer un droit à la fois. Voici la composition de l'instruction à passer à chmod :

- Un groupe de lettres parmi u (user), g (group), o (other) et a (all) pour signaler qui est concerné.
- Un signe + (ajouter le droit), - (supprimer le droit), = (donner exactement ce droit).
- Un droit ou plusieurs droits parmi r (read), w (write), x (execute), - (aucun droit, utilisable uniquement seul avec =).

Par défaut, si le premier élément est omis, l'action affecte tout le monde. Il est possible de donner plusieurs instructions à la fois en les séparant par des virgules comme dans le dernier exemple ci-dessus.

### 6.1.3 Les droits spéciaux : setuid, setgid, sticky bit

Quand on exécute un programme, celui-ci est exécuté avec les droits de l'utilisateur qui l'a lancé. Ceci peut être très ennuyeux pour une commande comme `passwd` qui permet à un utilisateur de changer son mot de passe. Cette commande appartient à `root`, tous les utilisateurs peuvent l'utiliser, mais il faut que la commande soit lancée avec les droits de `root` pour avoir le droit d'écrire dans le fichier `/etc/passwd` (ou `/etc/shadow`) car il appartient lui aussi à `root`.

De la même manière un exécutable peut nécessiter d'être exécuté avec les droits du groupe qui en est propriétaire.

Pour régler ce problème il existe des permissions spéciales *setuid* (Set User ID) et *setgid* (Set Group ID) qui viennent modifier le droit d'exécution. Quand la permission *setuid* est ajoutée, le programme est lancé avec les droits de l'utilisateur propriétaire quelque soit l'utilisateur qui l'a lancé. La permission *setgid* permet d'obtenir la même chose pour le groupe.

Cette permission apparait sous la forme d'un `s` placé à la place du `x`. Si vous recherchez la commande `passwd` et que vous affichez ses droits, vous pouvez constater la présence de ce `s` à la place du droit d'exécution de l'utilisateur propriétaire :

```
$ which passwd
/usr/bin/passwd
$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 47032 juil. 15 21:29 /usr/bin/passwd
```

Pour ajouter ou retirer cette permission, on peut utiliser la commande `chmod` comme expliqué précédemment en utilisant la lettre `s` pour représenter le droit :

```
$ chmod u+s programme

$ chmod u-s programme
```

A noter : si le fichier ou le répertoire ne possède pas de permission "x", c'est un "S" majuscule qui est indiqué, la permission n'a dans ce cas aucun effet.

La permission setgid peut également être utilisée sur un dossier. Imaginons qu'un groupe d'utilisateurs veuille travailler sur un projet commun. L'un d'eux a créé un dossier qui lui a donc été attribué et pour que chacun puisse y placer ses documents, il a changé le groupe pour un groupe `projet` et il a donné les droits d'écriture à ce groupe :

```
$ ls -l
total 4
drwxrwxr-x 2 martin projet 4096 août  14 18:25 projet
$ ls -l projet/
total 0
-rw-rw-r-- 1 martin martin 0 août  14 18:16 file1
-rw-rw-r-- 1 dudley      dudley      0 août  14 18:25 file2
```

Cependant, quand les différents membres du groupe ont créé des fichiers dans ce dossier, ces

fichiers ont été créés avec le groupe portant leur nom : les autres membres du groupe ne peuvent alors pas y accéder. Pour que les fichiers soient créés avec le même groupe que celui qui possède le dossier, on peut ajouter la permission setgid au dossier :

```
$ chmod g+s projet/
$ ls -l
total 4
drwxrwsr-x 2 martin projet 4096 août  14 18:55 projet
martin@pupuce:~/Documents$ touch projet/file3
martin@pupuce:~/Documents$ ls -l projet/
total 0
-rw-rw-r-- 1 martin martin 0 août  14 18:16 file1
-rw-rw-r-- 1 dudley  dudley  0 août  14 18:25 file2
-rw-rw-r-- 1 martin projet  0 août  14 18:58 file3
```

Reste un problème : les différents membres du groupe peuvent accidentellement supprimer les fichiers des autres utilisateurs. Pour éviter cela il est possible d'utiliser le *sticky bit* : une permission spéciale qui indique qu'un utilisateur a le droit de lire et modifier les fichiers mais qu'il ne peut pas effacer les fichiers sauf s'il en est le propriétaire. Cela permet de dissocier le droit de modification du droit de suppression. Ce sticky bit est ajouté sur le dossier contenant les fichiers à protéger avec la commande `chmod` en utilisant cette fois la lettre `t`. La présence du sticky bit est visible à la place du droit `x` de `other`.

```
$ chmod +t projet/
$ ls -l
total 4
drwsrwsr-t 2 martin projet 4096 août  14 18:58 projet
```

Le sticky bit peut être également utilisé sur le dossier contenant un lien symbolique pour empêcher les utilisateurs de l'effacer (pour rappel, tous les utilisateurs ont tous les droits sur un lien symbolique).

Quand on écrit en octal, les permissions setuid, setgid et sticky bit sont représentées par un chiffre supplémentaire placé au début. Il correspond à l'addition des chiffres correspondant aux permissions : `setuid=4`, `setgid=2`, `sticky=1`. Par exemple les droits `rwsrwsr-t` (`rwxrwxr-x`, setuid, setgid, sticky bit) se traduisent par `7775`.

### 6.1.4 Gestion des utilisateurs et groupes

Différentes commandes permettent de gérer les utilisateurs et les groupes. Les listes des utilisateurs et des groupes existants sont placées dans les fichiers `/etc/passwd` et `/etc/groups`. Pour extraire et afficher les noms des utilisateurs ou des groupes, vous pouvez utiliser la commande `cat` pour afficher le contenu du fichier et `awk` pour extraire le premier champ :

```
$ cat /etc/passwd | awk -F: '{print $1}'

$ cat /etc/group | awk -F: '{print $1}'
```

### users et groups

La commande `users` permet de connaître les utilisateurs ayant une session en cours sur la machine.

La commande `groups` suivie du nom d'un utilisateur permet de connaître les groupes auxquels cet utilisateur appartient.

### addgroup et delgroup

Les commandes `addgroup` et `delgroup`, utilisables uniquement par un administrateur, permettent respectivement de créer un groupe ou de le supprimer.

Ces commandes sont des versions plus conviviales des commandes `groupadd` et `groupdel` héritées de Unix.

### adduser et deluser

La commande `adduser`, utilisable uniquement par un administrateur, permet de créer un nouvel utilisateur. Lors de la création, un mot de passe doit être attribué à l'utilisateur et des informations sur son identité sont collectées.

La commande `adduser` peut également être utilisée pour ajouter un utilisateur dans un groupe (recommandé car moins risqué que usermod). La commande suivante ajoute l'utilisateur `martin` dans le groupe `projet` :

```
$ sudo adduser martin projet
```

La commande `deluser` permet au contraire de supprimer un utilisateur. L'option `--remove-home` permet d'effacer le compte personnel de l'utilisateur ainsi que toutes les données qu'il contient et l'option `--remove-all-files` permet d'effacer tous les fichiers possédés par l'utilisateur.

Ces commandes sont des versions plus conviviales des commandes `useradd` et `userdel` héritées de Unix.

### passwd

La commande `passwd` permet à un utilisateur de changer son mot de passe. L'ancien mot de passe est demandé avant d'en choisir un nouveau. La commande exige un mot de passe suffisamment long et complexe [3].

### usermod

La commande `usermod` permet de renommer un utilisateur avec l'option `-l`, de définir le groupe principal d'un utilisateur avec l'option `-g`, de définir des groupes supplémentaires avec l'option `-G` :

3. Sur certaines distributions de Linux il parait qu'il est possible d'imposer un mot de passe faible en insistant plusieurs fois. Sur la version actuelle de Ubuntu, j'ai pu constater que ce n'est pas le cas.

l'option -a ajoutée à -G permet d'ajouter les groupes aux groupes déjà présents. La commande suivante ajoute le groupe projet aux autres groupes de l'utilisateur martin :

```
$ sudo usermod martin -aG projet
```

La commande usermod ne semble pas dangereuse à priori, mais si vous êtes l'unique administrateur de votre machine et que vous oubliez l'option -a, vous pouvez écraser les groupes existants auxquels vous appartenez : vous ne pourrez alors plus vous identifier comme administrateur avec la commande sudo. Il faut donc la manier avec prudence et utiliser de préférence la commande adduser pour ajouter un utilisateur dans un groupe supplémentaire.

#### who

La commande who permet d'afficher la liste des utilisateurs connectés avec différentes informations comme le type de console utilisée (tty : terminal, pts : console graphique), l'IP de connexion si c'est une connexion distante, l'heure de connexion, le temps d'inactivité et les processus en cours.

L'équivalent de la commande users peut être obtenu avec :

```
$ who | awk '{print $1}' | sort | uniq
```

## 6.2 Programmation en shell

Il est possible de réunir différentes commandes pour écrire un script shell. Dans la plupart des scripts, vous aurez besoin de stocker des informations dans des variables, faire des tests, créer des boucles pour répéter une action : nous allons voir cela maintenant.

### 6.2.1 Les variables

Pour créer une variable dans le shell et lui attribuer une valeur il suffit d'écrire son nom suivi du signe égal = et de la valeur à attribuer. Attention, il ne faut pas d'espace ni avant, ni après le signe =. Si cette valeur est une chaîne contenant des espaces, il faut l'entourer de guillemets. Pour demander au shell d'utiliser la valeur mémorisée précédemment dans une variable, il faut faire précéder le nom de la variable du signe $ sinon c'est le nom lui même qui est utilisé :

```
$ var=coucou
$ echo var
var
$ echo $var
coucou
$ var2="bonjour tout le monde"
$ echo $var2
bonjour tout le monde
```

Le caractère $ indique au shell qu'il doit prendre le contenu de la variable avant d'interpréter la commande tapée. Les quotes doubles (les guillemets) évitent l'interprétation des caractères spéciaux comme ?, mais ce n'est pas le cas du caractère $ : la variable est donc remplacée par son contenu même entre des guillemets. Par contre cela n'est plus vrai si la variable est placée entre des quotes

simples :

```
$ nom="Martin Miggs"
$ echo tu es $nom \?
tu es Martin Miggs ?
$ echo "tu es $nom ?"
tu es Martin Miggs ?
$ echo 'tu es $nom ?'
tu es $nom ?
```

Ces variables ne sont valables que dans le shell courant, on dit que ce sont des variables *locales*. Si vous lancez une commande, elle ne connait pas les variables que vous avez définies dans le shell. Pour qu'une commande lancée depuis le shell puisse voir une variable que vous avez définie, il faut la placer dans l'*environement* (§. 6.2.2).

Certaines variables ont un rôle prédéfini :

**$$** cette variable contient le numéro du shell courant

**$!** cette variable contient le numéro du dernier processus lancé en arrière plan.

**$?** cette variable contient le compte-rendu de la dernière commande lancée au premier plan (un numéro indiquant si tout c'est bien passé ou si il y a eu une erreur).

## 6.2.2 L'environnement

Pour passer une variable aux différentes commandes lancées depuis un shell, il faut la placer dans l'*environnement*. L'*environnement* est un ensemble de variables qui est transmis à un processus, par le processus qui l'a lancé.

Quand le shell bash a été lancé dans le terminal, il a reçu un *environnement*. Cet *environnement* il le transmettra à chaque processus que l'on lancera avec une commande. Il est possible de modifier les variables contenues dans l'environnement ou d'en ajouter depuis le shell en cours, mais cela n'affectera que ce shell et les processus qu'il lancera. L'environnement du processus qui a lancé le terminal ne sera pas affecté (Fig. 6.1).

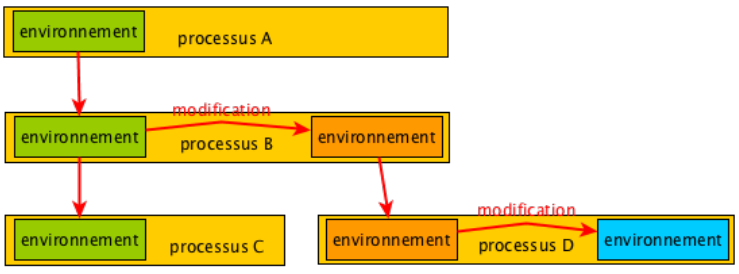


**Figure 6.1 – Transmission de l'environnement d'un processsus à l'autre.**

### export et printenv

Pour ajouter une variable à l'environnement, il faut utiliser la commande `export` et pour visualiser les variables présentes dans l'environnement il faut utiliser `printenv`. Pour vérifier la présence de `var` dans l'environnement, je peux filtrer les résultats retournés par `printenv` avec la commande `grep` ou bien donner directement le nom de la variable à printenv :

```
$ printenv var  # le shell ne retourne rien :
                        # var n'est pas dans l'environnement
$ var=coucou    # je crée une variable locale nommée var
$ export var    # je place cette variable dans l'environnement
$ printenv | grep '^var='   # je vérifie que var est dans l'environnement
var=coucou              # elle y est bien et vaut "coucou"
$ printenv var   # j'affiche directement le contenu de la variable
                 # d'environnement nommée var
coucou                  # elle vaut "coucou"
```

Vous pouvez créer cette variable var et la placer dans l'environnement avant de lancer un autre shell avec la commande sh : la variable est accessible dans le second shell. Si maintenant, depuis ce second shell, vous modifiez la variable d'environnement, quand vous sortez du second shell avec exit, vous constatez que l'environnement du premier shell n'a pas changé.

```
alinehuf@pupuce:~$ var=coucou    # je crée une variable locale nommée var
alinehuf@pupuce:~$ export var    # je place cette variable
                                 # dans l'environnement
alinehuf@pupuce:~$ sh            # je lance un autre shell
$ printenv var # j'affiche le contenu de la variable d'environnement var
coucou          # elle vaut "coucou"
$ var=bonjour
$ printenv var # j'affiche le contenu de la variable d'environnement var
bonjour          # dans le second shell elle vaut maintenant "bonjour"
$ exit         # je ferme le second shell et revient au premier
alinehuf@pupuce:~$ printenv var  # j'affiche le contenu de la
                                 # variable d'environnement var
coucou          # dans le premier shell elle vaut toujours "coucou"
```

Il est possible de placer une variable dans l'environnement transmis à une commande sans modifier l'environnement du shell, pour cela il suffit de préfixer la commande avec l'affectation de la variable :

```
alinehuf@pupuce:~$ chat=garfield   # je définis une variable locale
alinehuf@pupuce:~$ chat=rominet sh # je lance un autre shell en donnant
                                   # une autre valeur à la variable
$ printenv chat        # j'examine la valeur de la variable dans ce shell
rominet
$ exit                 # je reviens au premier shell
alinehuf@pupuce:~$ printenv chat   # la variable n'est pas dans l'environnement
alinehuf@pupuce:~$ echo $chat      # j'affiche la variable locale nommée chat
garfield                           # elle a gardé son ancienne valeur
```

### $HOME, $PATH, etc.

L'environnement contient des variables qui permettent de paramétrer le comportement de certaines commandes, par exemple LS_COLOR définit la manière dont la commande ls colorie les informations qu'elle affiche. D'autres variables d'environnement permettent de stocker des informations générales comme le nom de l'utilisateur (USER), le chemin vers son dossier personnel (HOME), la langue et l'encodage des caractères utilisés (LANG), le répertoire courant (PWD).

Une variable de l'environnement qui vous sera particulièrement utile est la variable PATH. Cette

variable contient la liste des répertoires où chercher les commandes que l'on saisit dans la console. Sans elle, il faudrait indiquer le chemin complet de chaque commande et se souvenir où elle se trouve pour l'appeler. Par exemple il faudrait appeler `/bin/pwd` pour afficher le répertoire courant ou `/usr/bin/passwd` pour changer son mot de passe. Dans `PATH`, le caractère `:` sépare le chemin de chaque répertoire à explorer :

```
$ echo $PATH
/home/alinehuf/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
/bin:/usr/games:/usr/local/games
```

Si, sur ma machine, je tape la commande `toto` (fictive), le shell va commencer par chercher dans le répertoire `/home/alinehuf/bin`. S'il ne trouve pas de commande `toto` à cet endroit, il va chercher dans `/usr/local/sbin`, puis dans `/usr/local/bin`, etc. jusqu'à ce qu'il ait trouvé la commande. S'il a exploré tous les répertoires de la liste sans trouver la commande, il affiche alors un message d'erreur.

### which

La commande `which` permet de savoir dans quel répertoire se situe la commande qui est utilisée. Les répertoires placés en premier dans le `PATH` sont généralement les dossiers personnels de l'utilisateur, ce qui peut permettre de personnaliser les commandes : si une commande est trouvée dans le répertoire `bin` de votre compte elle sera utilisée en premier.

```
$ which passwd              # où est la commande passwd ?
/usr/bin/passwd             # dans /usr/bin/
$ echo 'echo "non, non, moi je ne change rien !"' > ~/bin/passwd
            # j'utilise echo pour écrire une commande 'echo "..."'
            # qui est redirigée avec > dans le fichier ~/bin/passwd
$ chmod +x ~/bin/passwd   # j'ajoute les droits d'exécution au fichier
$ which passwd            # où est la commande passwd maintenant ?
~/bin/passwd              # le shell l'a trouvée dans mon compte !
$ passwd                  # je lance la commande
non, non, moi je ne change rien ! # et c'est le fichier passwd
                                  # de mon compte qui est exécuté
```

Quand vous créerez vos propres programmes, vous voudrez peut être les lancer depuis le répertoire où ils se trouvent. Malheureusement ce répertoire ne sera pas forcement dans `PATH`. Il faudra donc indiquer au shell où se situe votre programme avec le chemin absolu ou en utilisant le fichier spécial `.` qui signifie «*le dossier actuel*» :

```
$ ./monProgramme
```

Pour éviter d'avoir à le retaper à chaque fois, vous pouvez ajouter `.` au `PATH` ce qui indiquera au shell qu'il doit commencer par chercher le programme à exécuter dans le dossier courant :

```
$ PATH=.:$PATH
```

« *Mais dès que je ferme la console ma modification du* `PATH` *est perdue !* »

En effet, la modification de l'environnement n'est valable que dans le processus courant et les processus qu'il lancera. Nous verrons à la section 6.2.8 comment personnaliser votre console pour faire des réglages permanents, mais avant de le faire, vous devez être capable de comprendre un

script shell.

### 6.2.3 Tests et structures de contrôle

Pour lancer une commande uniquement si certaines conditions sont remplies il est nécessaire de faire des tests. Ensuite, il faut disposer de *structures de contrôle* qui vont permettre de lancer différentes commandes si la condition testée est remplie ou non.

#### test et [ ]

La commande `test` permet de tester si l'expression qui suit est vraie, dans ce cas elle retourne 0, sinon elle retourne 1. Pour tester la valeur de retour, il faut interroger la variable `$?`. Différents opérateurs permettent d'effectuer des tests, en voici quelque uns, consultez le manuel pour connaitre les autres :

| Sur des fichiers | |
|---|---|
| `-e foo` | *le fichier foo existe* |
| `-d foo` | *foo est un répertoire* |
| `-f foo` | *foo est un fichier standard* |
| `foo -nt bar` | *foo est plus récent que bar* |

| Sur des nombres | |
|---|---|
| `$A -lt $B` | `$A` *est strictement inférieur à* `$B` |
| `$A -le $B` | `$A` *est inférieur ou égal à* `$B` |
| `$A -eq $B` | `$A` *est égal à* `$B` |
| `$A -ne $B` | `$A` *n'est pas égal à* `$B` |

| Sur du texte | |
|---|---|
| `$A = $B` | `$A` *est identique à* `$B` |
| `-z $A` | *la chaine* `$A` *est vide* |

Voici quelques exemples. Le dernier exemple vous montre comment il est possible d'inverser un test avec l'opérateur `!` :

```
$ test "toto" = "toto" ; echo $?
0
$ A=3 ; B=5
$ test $A -eq $B ; echo $?
1
$ test $A -lt $B ; echo $?
0
$ test -f /home/ ; echo $?
1
$ test ! -d /etc/passwd ; echo $?
0
```

Il est possible d'utiliser des crochets à la place de `test`, mais attention : il faut laisser un espace entre les crochets et le test sinon le shell retourne une erreur. La variable `$?` contient alors le chiffre 2, code d'erreur indiquant qu'une commande du shell a été mal utilisée :

```
$ [ -f /etc/passwd ] ; echo $?
0
$ [ -f /etc/passwd] ; echo $?
bash: [: « ] » manquant
2
```

Il est possible d'utiliser les expressions suivantes pour enchaîner plusieurs tests :

| | |
|---|---|
| `[ $test1 ] && [ $test2 ]` | `$test1` *et* `$test2` *sont vrais tous les deux (si* `$test1` *est faux,* `$test2` *n'est même pas évalué).* |
| `[ $test1 ] \|\| [ $test2 ]` | *un des deux tests est vrai (si* `$test1` *est vrai,* `$test2` *n'est même pas évalué).* |

### if [ ... ] ; then ... ; fi

La structure de controle `if [ ... ] ; then ... ; fi` permet d'exécuter une commande seulement si un test à réussi. Par exemple :

```
$ if [ ! -e projet ]; then mkdir projet; fi # si projet n'existe pas, crée le
$ ls
projet
$ if [ -e projet ] && [ -d projet ]; then echo "projet existe !"; fi
            # si projet existe et est bien un dossier, affiche le message
projet existe !
```

### if ... elif ... else ... fi

La structure de contrôle `else` permet d'exécuter une commande dans le cas ou le test a échoué :

```
$ if [ -e projet ]; then echo "projet existe" ;
                                    else echo "projet n'existe pas" ; fi
projet existe
$ rm -r projet/
$ if [ -e projet ]; then echo "projet existe" ;
                                    else echo "projet n'existe pas" ; fi
projet n'existe pas
```

La structure de contrôle `elif` permet d'exécuter un autre test si le premier à échoué. Il est possible d'enchaîner autant de `elif` que nécessaire :

```
$ age=15
$ if [ $age -lt 12 ]; then
>     echo "tu est un enfant"
> elif [ $age -lt 18 ]; then
>     echo "tu es un ado"
> elif [ $age -lt 25 ]; then
>     echo "tu es un adulte"
> else
>     echo "tu es un vieillard"
> fi
```

```
tu es un ado
```

Comme vous le voyez, il est possible d'écrire une commande sur plusieurs lignes, dans ce cas, le shell détecte que la commande est incomplète et affiche un prompt différent (qui correspond au contenu de la variable PS2). Remarquez que lorsque l'on ajoute des sauts de ligne, les points virgules ne sont plus indispensables pour séparer les différentes parties de la structure de contrôle. Le décalage de certaines lignes avec des espaces (*indentation*) n'est là que pour améliorer la lisibilité.

### for ... in ... ; do ... ; done

La structure de contrôle `for ... in ... ; do ... ; done` permet d'exécuter une boucle : une commande est répétée plusieurs fois en donnant à chaque fois une valeur différente à une variable. Dans la boucle qui suit, la commande `echo $var` est exécutée 9 fois pour chaque valeur de `var` piochée dans la liste de 1 à 9. Les chiffres de 1 à 9 sont donc affichés dans la console :

```
$ for var in 1 2 3 4 5 6 7 8 9; do
>     echo $var
> done
```

Une boucle for peut être très utile, par exemple, pour appliquer un traitement sur chaque fichier d'un répertoire. La boucle suivante renomme chaque fichier `.txt` du répertoire courant en ajoutant `.old` à la fin :

```
for var in *.txt; do mv $var $var.old; done
```

Si l'on désire itérer 100 fois, il peut être fastidieux d'énumérer les nombres de 1 à 100. Pour éviter cela, il est possible d'utiliser une syntaxe particulière de `for` indiquant `((` un état initial `;` une condition d'arrêt `;` et la modification à faire à chaque itération `))` :

```
$ for (( i=1; i<=100; i++ ))
> do
>    echo $i
> done
```

Ici, la variable `i` est égale à 1 au départ. Tant que `i` est inférieur ou égal à 100, la boucle est exécutée et, à chaque tour, on incrémente `i` (on ajoute 1 à la valeur de `i`).

### while [ ... ] ; do ... ; done

Parfois, il n'est pas possible de savoir à l'avance combien de fois on désire itérer avec la boucle `for` . Dans ce cas, on peut utiliser une boucle `while`. Cette instruction signifie «*tant que*» et prend un test entre crochets. La boucle s'arrête quand la condition n'est plus remplie. Il faut cependant être prudent car si la condition donnée est toujours vraie, la boucle peut tourner indéfiniment.

La commande suivante (sans intérêt), permet de remonter jusqu'à la racine en affichant le répertoire courant à chaque pas. La condition donnée entre crochets permet de stopper la boucle quand la commande `pwd` indique la racine :

```
alinehuf@pupuce:~/Documents$ while [ $(pwd) != '/' ]; do cd .. ; pwd ; done
/home/alinehuf
/home
```

```
/
alinehuf@pupuce:/$
```

Dans ce deuxième exemple, je crée une variable name contenant le nom d'un fichier réduit à un seul caractère et un compteur count. Ma boucle a pour but d'augmenter la taille du nom et de voir si il est possible de nommer le fichier ainsi. Quand la création du fichier échoue, la boucle s'arrete et une commande suplémentaire me permet d'afficher un message avec la taille maximale du nom utilisé avec succès :

```
$ name=x                  # nom contenant 1 caractère
$ count=1                 # compteur initialisé à 1
$ > $name                 # je crée le fichier nommé 'x'
$ while [ -e $name ];     # tant que le fichier existe
> do
>     rm $name                    # je supprime ce fichier
>     name=x$name                 # j'ajoute un caractère au nom
>     count=$(expr $count + 1); # j'ajoute 1 au compteur
>     > $name                     # je tente de créer un fichier avec ce nom
> done
bash: xxxxxxxxxxxxxxxxx(...)xxxxxxxxxxxxxxx: Nom de fichier trop long
$ echo "taille maximum d'un nom de fichier = $count caractères"
taille maximum d'un nom de fichier = 256 caractères
```

La commande expr utilisée dans la boucle permet de réaliser une opération arithmétique : cette commande ajoute 1 au contenu de la variable $count qui sert de compteur. La commande est contenue dans $(...) ce qui permet de la remplacer par son résultat. Ce résultat est alors affecté à la variable count.

Dans certains cas, il peut être utile de faire tourner une boucle indéfiniment. Dans ce cas il est possible de donner le test true (vrai) comme condition à la commande while. Comme son nom l'indique, ce test est toujours vrai et la commande ne s'arrête jamais à moins de l'arrêter en force.

```
$ while true;
> do
>     ...
> done
```

Une boucle tournant indéfiniment peut permettre par exemple de placer des photos prises par une webcam en ligne à intervalles réguliers.

Il existe d'autres structures de contrôle (case, until, etc.) que nous ne détaillerons pas ici, je vous laisse les découvrir en vous documentant sur la programmation shell.

### 6.2.4 Créer un fichier script exécutable

Maintenant que nous avons les outils pour créer un vrai petit programme shell, nous pouvons écrire les commandes dans un fichier et le rendre exécutable pour les lancer simplement en appelant le nom du fichier.

Les différents shell (bourne, bash, C shell, Korn shell, etc.) peuvent interpréter certaines commandes de manières différentes. Pour éviter toute catastrophe dûe à une mauvaise interprétation des commandes, il faut commencer par préciser à quel type de shell les instructions sont destinées.

Pour le shell bash, il faut ajouter la ligne suivante au début du fichier :

```
#! /bin/bash
```

Les commandes retournent généralement la valeur `0` quand tout s'est bien passé. Pour que votre script retourne également la valeur `0`, vous pouvez ajouter l'instruction suivante à la fin de votre fichier de script shell.

```
exit 0
```

Pour exécuter votre script, vous devez vous assurer que vous avez les droits d'exécution. Si ce n'est pas le cas, utilisez la commande chmod (§. 6.1.2).

Vous allez créer votre premier script minimaliste en ajoutant entre ces deux instructions, juste un message écrit avec echo :

```
#! /bin/bash

echo "ce message est affiché par un script shell !"

exit 0
```

Enregistrez ces lignes dans un fichier msg.sh, donnez vous le droit de l'exécuter et lancez le. Vous devriez obtenir le résultat suivant :

```
$ chmod u+x msg.sh
$ ./msg.sh
ce message est affiché par un script shell !
```

### 6.2.5 Comprendre/Modifier un script shell

Voilà ! vous avez écrit votre premier script shell ! Maintenant, il est utile de savoir lire les scripts des autres et de comprendre ce qu'ils font, alors pour vous entrainer, le fichier premierScript.sh fourni avec le cours contient un premier script très simple. Voici son contenu :

```
#! /bin/bash

echo -n "Entrez votre pseudo: "
read nom

if [ -e "/home/$nom" ] && [ -d "/home/$nom" ]; then
        echo "Vous avez un compte sur cette machine!"
else
        echo "Vous n'avez pas de compte sur cette machine,
              du moins pas sous ce nom..."
fi

exit 0
```

L'instruction read nom permet d'attendre que l'utilisateur saisisse du texte au clavier et enregistre le texte tapé dans la variable nom quand l'utilisateur appuie sur la touche *Entrer*.

Testez ce script premierScript.sh et examinez le code source écrit dans ce fichier. Que fait ce

script ?

Prenez quelques minutes pour le tester et essayer de le comprendre avant de lire la réponse donnée juste après.

Bon, voyons si vous aviez bien compris, voici le même script avec des commentaires pour expliquer ce qu'il fait :

```
# je commence par indiquer que ce qui suit s'adresse au shell bash
#! /bin/bash

# j'affiche un message avec echo. -n permet d'éviter un saut de ligne.
echo -n "Entrez votre pseudo: "
# je récupère la saisie de l'utilisateur dans la variable nommée nom
read nom

# j'utilise le contenu de la variable nom dans une chaine "/home/$nom"
# cette chaine est le chemin vers le dossier personnel /home/[pseudo]
# de l'utilisateur
# je teste si le fichier /home/$nom existe (-e) et si c'est un dossier (-d)
if [ -e "/home/$nom" ] && [ -d "/home/$nom" ]; then
        # si oui j'affiche un message
        echo "Vous avez un compte sur cette machine!"
else
        # sinon j'affiche un autre message
        echo "Vous n'avez pas de compte sur cette machine,
              du moins pas sous ce nom..."
fi  # fin du test

exit 0  # je retourne 0 à la fin pour dire que tout s'est bien passé
```

Il est possible de transmettre des paramètres à un script shell comme nous l'avons fait jusqu'ici avec les commandes. Á l'intérieur de votre fichier script shell, la liste de tous les arguments peut être récupérée dans la variable spéciale `$*`. Le nombre d'arguments est stocké dans la variable `$#`. La variable `$0` contient le nom du script et les différents arguments sont placés dans des variables `$1`, `$2`, `$3`, etc.

Le script `deuxiemeScript.sh` montre comment récupérer les paramètres passés à un script sur la ligne de commande. Lancez `deuxiemeScript.sh` avec différents paramètres pour voir le résultat :

```
$ ./deuxiemeScript.sh un deux trois quatre
La commande se nomme ./deuxiemeScript.sh
Vous l'avez lancée avec 4 arguments
Ces arguments sont : un deux trois quatre
```

```
argument 1 = un
argument 2 = deux
argument 3 = trois
argument 4 = quatre
```

La commande expr permet de réaliser une opération arithmétique. Dans ce script ce sont des backslashs ` qui sont utilisés pour remplacer la commande par son résultat.

Dans le *shell Bash* il est possible de réaliser un calcul directement en l'entourant de doubles parenthèses :

```
$ count=2
$ ((count = $count + 3))
$ echo $count
5
```

Cette syntaxe est propre à *Bash* et ne marche pas avec le *shell Bourne*. L'utilisation de la commande expr en revanche marche toujours. Voici le message d'erreur que vous pouvez rencontrer en utilisant la syntaxe précédente avec le *shell Bourne* :

```
$ sh
$ count=3
$ echo $count
3
$ ((count = $count + 1))
sh: 3: count: not found
$
```

### 6.2.6 Exécuter un script à la "source"

Quand on exécute un script Shell, les commandes contenues dans le fichier ne sont pas exécutée par le Shell dans lequel on se trouve. Eh non ! Il procède exactement comme pour un script écrit dans un langage qu'il ne comprend pas, python par exemple. Il va donc examiner la première ligne du fichier, lancer l'interpréteur demandé et lui passer les instructions. Si la première ligne ne précise pas quelle interpréteur utiliser, il va lancer un Shell identique à lui-même et lui transmettre les instructions. Ceci peut être problématique quand le script à pour but de modifier le Shell courant en ajoutant de nouvelles variables dans l'environnement : les variables sont créées dans un Shell enfant et non dans le Shell courant. Une fois le script terminé, le Shell enfant s'arrête et rien a changé dans le Shell courant... Pour éviter cela, il existe une commande spéciale : source.

Si on indique source avant le nom du script à lancer, il sera exécuté par le Shell courant et non par un Shell enfant. Toutes les modifications effectuées dans l'environnement affecteront alors le Shell courant.

```
$ source ./script.sh
```

### 6.2.7 Créer un script à la volée avec sed et les expressions régulières

Vous pouvez utiliser la puissance de la commande sed que nous avons évoquée précédemment (§. 5.4.2) pour générer un script à la volée et demander au shell de l'interpréter. Cela est réalisable

également avec un script Python, mais pour une tache commune comme créer une copie de sauvegarde ou renommer tous les fichiers d'un dossier, l'utilisation de sed est bien plus rapide. Nous allons ici détailler un exemple :

```
$ ls | sed -e 's/.*/cp "&" "&.bak"/' | sh
```

La commande ls permet de lister le contenu du répertoire courant. La commande sed reçoit ce listing grâce au pipe |. On lui demande d'executer une expression avec -e. Cette expression décrit une substitution avec 's/.../.../'. Le s indique la substitution, le motif placé dans le premier espace indique le texte à modifier et le motif placé après indique le texte de remplacement. Le motif .* capture n'importe quelle suite de caractère, donc elle capture un nom de fichier entier. Ce nom de fichier est remplacé par l'expression cp "&" "&.bak" où & contient le nom de fichier capturé juste avant :

```
$ ls
text1.txt  text2.txt  text3.txt  text4.txt
$ ls | sed -e 's/.*/cp "&" "&.bak"/'
cp "text1.txt" "text1.txt.bak"
cp "text2.txt" "text2.txt.bak"
cp "text3.txt" "text3.txt.bak"
cp "text4.txt" "text4.txt.bak"
```

Notez que les noms de fichiers sont placés entre guillemets pour éviter qu'un caractère spécial ou un espace ne soit interprété et perturbe l'exécution des commandes générées. La commande sed construit ainsi une liste de commande de copie permettant de créer un fichier de sauvegarde pour chaque fichier trouvé. Le dernier pipe | permet de transmettre cette liste de commandes à la commande sh qui va créer un shell pour interpréter ces commandes. L'expression $ ls | sed -e 's/.*/cp "&" "&.bak"/' | sh permet donc de créer un script (un ensemble de commandes) à la volée et de le donner à sh pour l'exécuter.

### 6.2.8 Personnaliser sa console

Maintenant que vous êtes en mesure de comprendre un script shell, nous allons nous occuper du problème soulevé précédemment : comment modifier le paramétrage du terminal de manière durable et ajouter par exemple des variables dans le PATH, créer des alias ou modifier l'apparence de la console.

Plusieurs fichiers sont lus au démarrage du shell, qui permettent de configurer son apparence. Nous avons vu qu'il est possible de lancer une session sans interface graphique : dans ce cas, un système d'identification vous demande de vous identifier avant de démarrer le shell. Dans une session graphique le shell est lancé dans une fenêtre et il ne vous demande pas de vous identifier.

Si le shell demande une identification, le contenu des fichiers /etc/profile et ~/.profile est lu, ensuite le contenu des fichiers /etc/bash.bashrc et ~/.bashrc est chargé. Si le shell ne demande pas d'identification, seul le contenu des fichiers /etc/bash.bashrc et ~/.bashrc est chargé [4].

4. En réalité c'est un peu plus compliqué, mais ces fichiers sont les principaux scripts de configuration qui sont lus sous Ubuntu, du moins avec les réglages par défaut, et que vous aurez à modifier si vous souhaitez personnaliser ces réglages.

**/etc/profile**
: définit des réglages pour tous les utilisateurs et pour les shells en général, non spécifiques à bash donc.

**/etc/bash.bashrc**
: définit des réglages spécifiques à bash, pour tous les utilisateurs.

**~/.profile**
: définit vos réglages perso, pour les shells en général.

**~/.bashrc**
: définit vos réglages perso, pour le shell bash.

En fonction du type de réglage que vous souhaitez faire, vous ajouterez des informations dans un ficher ou l'autre. Pour le moment, je vous conseille néanmoins de ne toucher qu'à vos fichiers de configuration personnels. Avant de faire un quelconque réglage, prenez la peine de faire une copie des fichiers que vous souhaitez modifier, cela vous permettra de faire marche arrière en cas d'erreur.

### ~/.profile

Ouvrons le fichier `.profile`. Le contenu n'est pas très long et vous devriez être en mesure de le comprendre. Les lignes commençant par le caractère # sont des commentaires, elles ne sont pas interprétées :

```
# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi
```

Ce passage dit en gros que si la variable d'environnement `$BASH_VERSION` est définie c'est qu'on est en train d'utiliser un shell bash et donc on exécute le contenu du fichier `~/.bashrc` si il existe. La commande `.` à la 5ème ligne est une commande spéciale qui permet d'executer un script dans le shell courant (plutôt que de démarrer un autre shell pour l'exécuter) et de placer toute variable qui est définie dedans dans l'environnement. Cette commande executera le contenu du script même si le fichier en question n'est pas exécutable (il n'y a pas le bit x dans les droits d'exécution).

Voyons le passage suivant :

```
# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
```

Cette condition indique que si il existe un répertoire `~/bin/` dans votre compte personnel alors il faut ajouter ce répertoire dans le `PATH` pour utiliser vos script perso directement en tapant leur nom.

Nous voulons ajouter le répertoire courant dans le PATH. Nous allons donc ajouter l'instruction `PATH=.:$PATH` à la fin du fichier. Vous pouvez le faire en ouvrant le fichier avec *gedit* et en recopiant cette instruction après avoir sauté une ligne, ou vous pouvez faire ça avec la commande `echo` et une

redirection. Commencez par faire une copie de sauvegarde du fichier[5] :

```
$ cp ~/.profile ~/.profile.bak
```

Puis utilisez la commande echo avec la redirection >> pour ajouter l'expression à la fin du fichier et ne pas écraser son contenu. Mettez les instructions entre quotes simples pour que le $ de $PATH ne soit surtout pas interprété par le shell mais écrit tel quel dans le fichier de destination. Utilisez :

```
$ echo -e '\n# set PATH so it includes current directory' >> ~/.profile
$ echo 'PATH=.:$PATH' >> ~/.profile
```

Vous voyez que j'ai ajouté une première commande pour écrire un commentaire avant l'instruction (le \n indique un saut de ligne, l'option -e oblige echo à l'interpréter comme tel). Le commentaire permet de se souvenir à quoi sert une instruction[6]. Parfois on se contente de signer et dater pour indiquer qui a fait cette modification et quand. Le commentaire est en anglais, il va falloir vous y habituer, c'est la langue standard en informatique.

Maintenant votre fichier ~/.profile devrait se finir de cette manière (j'utilise la commande tail pour afficher les dernières lignes) :

```
$ tail -n8 ~/.profile

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi

# set PATH so it includes current directory
PATH=.:$PATH
```

Notez que ce fichier ~/.profile modifié ne sera prit en compte qu'à l'ouverture d'une nouvelle fenêtre du Terminal, il est donc normal de ne pas voir de changement dans votre Shell courant.

### ~/.bashrc

Le fichier ~/.bashrc contient beaucoup plus d'instructions. Vous devriez cependant être capables de les comprendre dans les grandes lignes. Les commentaires sont là pour expliquer l'utilité de chaque bloc de code.

Pour tester les modifications que vous allez apporter au fichier ~/.bashrc, pensez à démarrer un nouveau terminal ou bien utilisez la commande source ~/.bashrc pour exécuter une nouvelle fois le fichier dans le contexte de votre terminal afin qu'il soit au courant des changements.

Les instructions suivantes devraient peut être vous intéresser. Elles indiquent que vous pouvez forcer la mise en couleur du prompt. Si vous aimez la couleur, commentez les premières lignes en ajoutant des # et dé-commenter la ligne force_color_prompt=yes comme suit :

```
# set a fancy prompt (non-color, unless we know we "want" color)
#case "$TERM" in
```

5. Je vous recommande de toujours faire une copie de sauvegarde des fichiers de configuration que vous modifiez. Cela vous évitera bien des problèmes si vous faites une erreur de manipulation.

6. Il arrive, en relisant un fichier après quelques mois ou juste quelques semaines, que l'on ait du mal à se souvenir pourquoi on a ajouté une instruction ou bien à quoi elle sert. Le commentaire est très utile dans ce cas.

```
#    xterm-color) color_prompt=yes;;
#esac

# uncomment for a colored prompt, if the terminal has the capability; turned
# off by default to not distract the user: the focus in a terminal window
# should be on the output of commands, not on the prompt
force_color_prompt=yes
```

Que le prompt soit coloré ou pas, les instructions suivantes peuvent vous intéresser :

```
if [ "$color_prompt" = yes ]; then
    PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@
                          \h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ '
else
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
fi
```

Ces instructions indiquent à quoi doit ressembler le prompt : `\u` désigne le nom de l'utilisateur, `\h` le nom de la machine, `\w` le répertoire courant. Les instructions `\[\033[01;32m\]`, `\[\033[01;34m\]` et `\[\033[00m\]` permettent de régler les couleurs.

L'instruction `${debian_chroot:+($debian_chroot)}` tout au début signifie que si la variable `debian_chroot` est définie alors il faut l'ajouter entre parenthèses. `chroot` est une commande qui permet de faire croire momentanément au shell que le répertoire `root` à changé, ce qui l'empêche de remonter au delà du nouveau répertoire `root` : c'est un peu compliqué, mais tout ce que vous avez besoin de savoir c'est que cette procédure est utile notamment pour certaines opérations de maintenance. Dans ce cas, le prompt aura un aspect de ce type :

```
    (netboot)user@host:~$
```

En modifiant les expressions affectées à PS1, vous pouvez donc changer l'aspect de votre prompt[7].

Si vous n'aimez pas les couleurs, surtout dans les listings affichés par `ls` (voire même dans les résultats de `grep`), vous pouvez commenter les lignes suivantes qui permettent de mettre la sortie de ces commandes en couleur :

```
# enable color support of ls and also add handy aliases
if [ -x /usr/bin/dircolors ]; then
    test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolor
    alias ls='ls --color=auto'
    #alias dir='dir --color=auto'
    #alias vdir='vdir --color=auto'

    alias grep='grep --color=auto'
    alias fgrep='fgrep --color=auto'
    alias egrep='egrep --color=auto'
fi
```

Parmi les instructions ci-dessus vous voyez la commande `alias`. Cette commande est très pratique : elle permet de créer une sorte de raccourci pour lancer une commande avec certains para-

7. Si vous êtes fans de la couleur et voulez vous amuser à personnaliser votre prompt allez faire un tour sur http://bashrcgenerator.com/

mètres déjà définis. Par exemple, vous voyez que `grep` permet de lancer en fait la commande `grep --color=auto`.

A la section 6.4.1 vous aller installer *Common Lisp* car vous apprendrez ce langage dans un des cours de cette année. Il se trouve que `clisp`, par défaut, ne fait pas la différence entre les caractères majuscules et minuscules, de plus, il parle en anglais ce qui peut vous gêner si vous n'êtes pas à l'aise avec cette langue[8]. Ces deux inconvénients peuvent être facilement résolus en lui passant certains paramètres. Nous allons donc créer un alias pour lancer `clisp` directement avec ces paramètres.

Si on regarde un peu plus loin dans le fichier `~/.bashrc`, on voit ces lignes :

```
# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi
```

Ces commentaires nous indiquent qu'il est préférable de mettre tous nos alias personnalisés dans le fichier `~/.bash_aliases`. Les instructions qui suivent chargent le contenu de ce fichier. Nous allons donc ouvrir ce fichier `~/.bash_aliases` ; si il n'existe pas nous allons le créer. Nous ajouterons à l'intérieur l'alias suivant, l'option `-q` permet de supprimer l'affichage des messages de bienvenue et d'au revoir à l'intérieur de `clisp`, `-L` permet de changer la langue et `-modern` permet d'activer la sensibilité à la casse :

```
    alias lisp='clisp -q -modern -L french'
```

Une fois de plus, si vous voulez, vous pouvez utiliser `echo` pour écrire la commande et une redirection pour l'envoyer dans le fichier :

```
$ cp ~/.bashrc ~/.bashrc.bak
$ echo "# start clisp with french language, modern symbols
                                        and quiet option" >> ~/.bash_aliases
$ echo "alias lisp='clisp -q -modern -L french'" >> ~/.bash_aliases
$ cat ~/.bash_aliases
# start clisp with french language, modern symbols and quiet option
alias lisp='clisp -q -modern -L french'
```

Maintenant je vérifie le résultat en lançant les commandes `clisp` puis `lisp`. Avec `clisp`, l'interprète affiche un long message d'accueil. Si j'écris une liste `'(a B c)`, je constate que l'*interprète Lisp* me répond tout en majuscules. Quand je quitte, je vois le message d'au revoir :

```
$ clisp
  i i i i i i i       ooooo    o        ooooooo   ooooo   ooooo
  I I I I I I I      8     8   8           8     8     o  8    8
  I  \ `+' /  I      8         8           8     8        8    8
   \  `-+-'  /       8         8           8      ooooo   8oooo
    `-__|__-'        8         8           8           8  8
        |            8     o   8           8     o     8  8
  ------+------       ooooo    8ooooo   ooo8ooo   ooooo   8
```

8. Pourtant, il faudra bien vous y mettre à l'anglais si vous espérez travailler dans le domaine de l'informatique...

```
Welcome to GNU CLISP 2.49 (2010-07-07) <http://clisp.cons.org/>

Copyright (c) Bruno Haible, Michael Stoll 1992, 1993
Copyright (c) Bruno Haible, Marcus Daniels 1994-1997
Copyright (c) Bruno Haible, Pierpaolo Bernardi, Sam Steingold 1998
Copyright (c) Bruno Haible, Sam Steingold 1999-2000
Copyright (c) Sam Steingold, Bruno Haible 2001-2010

Type :h and hit Enter for context help.

[1]> '(a B c)
(A B C)
[2]> (quit)
Good bye!
```

Avec `lisp` je n'ai plus le long message d'accueil et il ne me répond plus tout en majuscules. Remarquez que je n'ai pas voulu remplacer totalement la commande `clisp` alors j'ai donné à mon alias un nom légèrement différent :

```
$ lisp
[1]> '(a B c)
(a B c)
[2]> (quit)
$
```

## 6.3 Surveiller le système

Maintenant que vous avez personnalisé votre console et pris vos marques, il est temps de découvrir quelques commandes qui vous permettrons de surveiller le système et d'intervenir quand un programme ne fonctionne pas bien.

### 6.3.1 Surveiller les ressources de la machine

Voici quelques commandes pour surveiller l'utilisation de la machine.

#### w

La commande `w` permet de savoir rapidement quels sont les utilisateurs en activité sur la machine et ce qu'ils font. Le *load average* indique la charge (de travail) des processeurs depuis 1, 5 et 15 minutes sous forme d'un chiffre à virgule $(0, ...)$. Une valeur de 1 indique une charge de 100%.

#### tload et xload

La commande `tload` permet de visualiser la charge des processeurs sous forme d'un graphe (ascii). La commande `xload` permet de faire la même chose dans une fenêtre graphique.

#### uptime

La commande `uptime` indique depuis combien de temps la machine est allumée et donne elle aussi des informations sur la charge des processeurs.

#### halt, reboot, poweroff, shutdown, pm-suspend, etc.

Les commandes `halt`, `poweroff`, `reboot` permettent d'arrêter ou redémarrer le système depuis le terminal. La commande `shutdown` prend en plus une information sur l'heure à laquelle le système doit s'arrêter ou redémarrer.

Les commandes `pm-suspend`, `pm-hybernate` et `pm-suspend-hybrid` permettent de mettre le système en sommeil plus ou moins profond avec une machine qui reste alimentée ou non et un réveil plus ou moins rapide.

### 6.3.2 Naissance, vie et mort des processus

Avant de découvrir les commandes permettant de surveiller et gérer les *processus* tournant sur la machine, nous allons expliquer ce qu'est un *processus*.

Un *processus*, aussi nommé *tâche* est l'exécution d'un programme ou script à un instant donné. Le programme ou le script, sont des ensembles inertes d'instructions sous forme de texte ou bien sous forme binaire. Ce programme ou script peut être exécuté plusieurs fois, en même temps ou à des moments différents. Chaque exécution donne lieu à la création d'un processus.

Il existe plusieurs types de processus : les processus système et les processus utilisateur. Les processus système ne sont attachés à aucun terminal. Ils sont soit créés au lancement du système d'exploitation et interrompus à son arrêt, soit lancés à des dates et heures fixés par l'administrateur. Parmi les processus lancés au démarrage du système, il existe des processus particuliers que l'on nomme *daemons*. Ce sont des processus qui tournent en arrière plan et effectuent différentes tâches comme, par exemple, la surveillance des connexions réseau. On peut citer par exemple *cron*, un daemon qui surveille l'horloge et permet de programmer et déclencher différentes tâches à des dates et heures précises (sauvegardes automatiques, mises à jour, etc.).

Les processus utilisateurs sont des programmes exécutés par les différents utilisateurs depuis une console ou bien programmés par les utilisateurs grâce à un outils comme *cron* ou déclenchés indirectement. Par exemple, le Shell est lancé quand l'utilisateur se logue sur un terminal.

Quand un programme est lancé, le code du programme est copié en mémoire et peut-être réutilisé si le programme est exécuté plusieurs fois en même temps (on dit que le code est *réentrant*). Un processus est alors créé, c'est un fichier qui indique l'état de l'exécution du programme et il possède plusieurs attributs permettant de décrire cette exécution :

- un identifiant unique pour le processus (PID, nombre entier)
- un propriétaire (UID : le User ID de celui qui a lancé le programme, et ses droits EUID)
- un groupe propriétaire (GID et ses droits : EGID)
- le terminal dans lequel il a été lancé (TTY)

- le processus parent (PPID) [9]
- autres attributs (priorité, répertoire de travail, etc.)

Le processus possède également un attribut indiquant son état (Fig. 6.2).

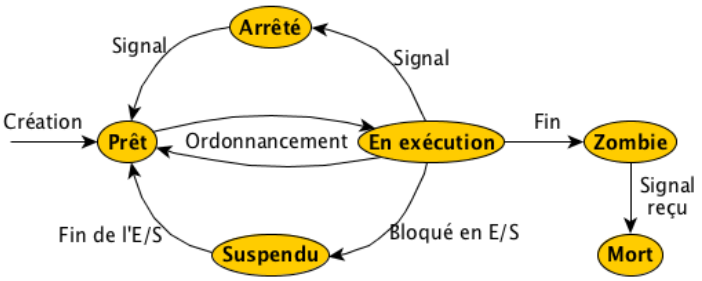


**Figure 6.2 – Vie et mort d'un processus**

- T : Stoppé ou tracé (Arrêté)
- R : En cours d'exécution ou prêt à l'être
- S : En sommeil (Suspendu)
- Z : Zombie
- D : En sommeil non interruptible (Planté)
- X : mort (jamais visible)

Un processus nouvellement créé est dans l'état R. Cependant, si un processus plus urgent (de priorité plus haute) est aussi prêt à être exécuté, c'est cet autre processus qui sera indiqué au processeur et le programme correspondant sera exécuté. Le système d'exploitation, ou plus exactement l'ordonnanceur du système d'exploitation, à pour rôle de veiller à ce que chaque processus puisse être exécuté à tour de rôle de manière équitable [10]. Chaque processus passera donc en exécution quand son tour sera venu.

Si l'utilisateur provoque un arrêt momentané du processus (il demande à le mettre en pause ou utilise un système de "trace" qui permet de stopper le processus pour examiner son fonctionnement dans le cadre d'un débogage par exemple), le processus a alors le statut T. Si le processus attend une information en entrée ou sortie (par exemple un logiciel de traitement de texte attend des données en provenance du scanner ou bien a envoyé des dpnnées à l'imprimante), il est alors suspendu (état S) le temps que l'information soit transmise (le processeur peut exécuter d'autre processus en attendant pour ne pas perdre de temps).

Enfin si le processus se plante, il peut passer dans l'état D. Un processus qui est en cours d'arrêt (fin du programme ou bien arrêt demandé par l'utilisateur) n'est pas immédiatement détruit. Le fichier contenant les informations décrivant le processus restent stockées en mémoire pour que le processus parent puisse savoir comment s'est déroulée l'exécution : le processus parent peut avoir besoin de savoir si le programme enfant a bien été exécuté ou si il a été stoppé avant la fin ou a planté en cours de route... Tant que le parent n'a pas récupéré les informations et envoyé un signal indiquant que ces informations peuvent être éliminées, le processus enfant est maintenu en mémoire : il n'est plus vivant, mais il n'est pas totalement mort non plus, c'est un *zombie* ! Enfin

9. Si un processus, par exemple un logiciel de courrier, provoque le lancement d'un autre processus, par exemple la connexion à internet, le premier est le parent du second.

10. La manière dont ce partage équitable de l'utilisation du processeur est effectuée sort du cadre de ce cours, mais pouvez vous renseigner sur internet...

quand le parent donne le signal, le processus enfant est supprimé et il est alors totalement mort. Il faut noter que le statut `X` n'est jamais visible quand on affiche une liste des processus et leur état puisque les processus morts sont éliminés de la mémoire et n'existent donc plus !.

Voici quelques commandes pour contrôler ces processus, les surveiller, les suspendre, les passer en arrière plan (tâche de fond) et les stopper si besoin.

### 6.3.3 Examiner les processus : ps et top

Si vous voulez examiner l'état d'un processus, trouver le processus qui consomme le plus de ressources sur votre machine, connaitre le parent d'un processus, son identifiant ou autre information, il existe deux commandes incontournables :

#### ps

La commande `ps` permet d'afficher la liste des processus lancés dans le shell. Les options suivantes sont fréquemment utilisées : `-l` permet d'avoir une liste détaillée, `-ef` permet d'afficher tous les processus lancés sur la machine avec entre autre les noms des utilisateurs qui les ont lancés, `-ejH` permet d'afficher les processus en arbre (pour voir quel processus à lancé quel autre) [11], `-u` suivie du nom d'un utilisateur permet d'afficher uniquement les processus qu'il a lancés, `-ax` permet d'afficher tous les processus de tous les utilisateurs, et `-axl` la même chose avec une liste détaillée. Remarquez la colonne `PID` qui affiche l'identifiant unique de chaque processus sur le système.

#### top

La commande `top` (Fig. 6.3) permet d'afficher une liste dynamique (qui se met à jour régulièrement) des processus actifs [12]. Remarquez le statut des processus indiqué dans la colonne `S`. Les colonnes `%CPU` et `%MEM` indiquent l'utilisation du processeur et de la mémoire. Les processus consommant le plus de ressources sont en haut de la liste ce qui est pratique pour détecter la source d'un problème quand, par exemple, votre ordinateur ralentit dangereusement ou que les ventillateurs de votre machine deviennent bruyants (signe que le processeur chauffe car il travaille intensément). Au dessus de la liste sont indiqués l'utilisation du processeur, l'état de la mémoire, le nombre de tâches en cours d'exécution, etc.

Pour quitter la commande `top` appuyez sur la touche `Q`.

### 6.3.4 Lancer, passer une tâche en arrière plan

Il peut être nécessaire parfois, quand une commande doit effectuer toute seule un travail long par exemple, de lancer une tâche en arrière plan.

11. La commande `pstree` permet aussi un affichage sous forme d'arbre.
12. Les commande `atop` et `htop` sont un peu plus conviviales (interface colorée et plus ergonomique), mais ces commandes ne sont pas installées par défaut.

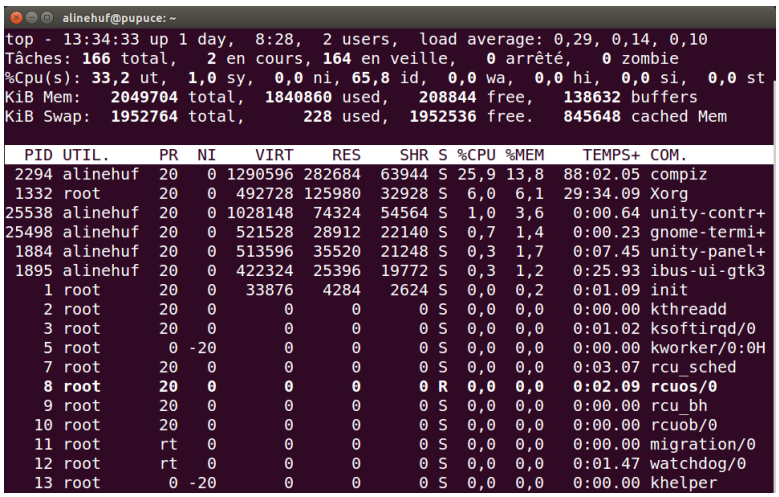


**Figure 6.3 – Exemple d'affichage obtenu en lançant la commande `top`**

## ... &

Pour lancer une tâche en arrière plan, il suffit de faire suivre n'importe quelle commande du symbole &. Dans ce cas, la commande est lancée et le prompt s'affiche à nouveau immédiatement. Vous pouvez continuer à travailler pendant que la commande s'exécute de son côté. Cependant, la commande reste attachée au shell qui l'a lancée : si le shell s'arrête (si vous fermez la fenêtre de la console) le processus en arrière plan sera également stoppé.

Pour tester les processus en arrière plan, utilisons la commande `sleep` qui permet de demander au shell d'attendre un certain nombre de secondes :

```
$ sleep 3
```

Si vous tapez cette commandes, vous verrez qu'il se passe 3 secondes pendant lesquelles le prompt ne réapparait pas, puis le prompt s'affiche de nouveau. Mettez maintenant un & à la suite de la commande. Vous constaterez que le prompt revient immédiatement.

```
$ sleep 3 &
[1] 9544
```

Quand la commande est passée en arrière plan, le shell a affiché deux numéros. Le numéro entre crochets est son *numéro de job* dans le shell. Il peut y avoir plusieurs processus en arrière plan, le shell les numérote donc pour les identifier. Le second numéro est le PID du processus, il permet d'identifier le processus de manière unique dans le système.

J'ai lancé plusieurs commandes `sleep` avec un long temps d'attente (1 heure) :

```
$ sleep 3600 &
[1] 9556
$ sleep 3600 &
[2] 9557
$ sleep 3600 &
[3] 9558
```

Vous voyez que chaque commande s'est vue attribuer un numéro de job.

### Ctrl+Z

Maintenant, j'ai lancé une quatrième commande en oubliant de la mettre en arrière plan. Je peux la suspendre en appuyant sur `Ctrl+Z`[13] :

```
$ sleep 3600
^Z
[4]+  Arrêté                 sleep 3600
```

Cette tâche est en attente, le programme n'est plus exécuté. Si je veux exécuter cette tâche en arrière plan il va falloir utiliser une instruction spéciale pour la redémarrer. En attendant, observons que cette tâche suspendue est placée en attente à l'arrière plan.

### jobs

Pour afficher la liste des tâches en arrière plan, utilisez la commande `jobs`

```
$ jobs
[1]   En cours d'exécution   sleep 3600 &
[2]   En cours d'exécution   sleep 3600 &
[3]-  En cours d'exécution   sleep 3600 &
[4]+  Arrêté                 sleep 3600
```

Vous constatez que la tâche est bien à l'arrière plan mais elle est stoppée.

### bg %$n$

Pour redémarrer une tâche en arrière plan, utilisez la commande `bg %`$n$ («*BackGround*») avec le numéro de la tâche à la place de $n$.

```
$ bg %4
[4]+ sleep 3600 &
$ jobs
[1]   En cours d'exécution   sleep 3600 &
[2]   En cours d'exécution   sleep 3600 &
[3]-  En cours d'exécution   sleep 3600 &
[4]+  En cours d'exécution   sleep 3600 &
```

La tâche est alors de nouveau exécutée sans quitter l'arrière plan. Ainsi, si vous avez lancé une tâche en oubliant de la placer à l'arrière plan, vous pouvez utiliser `Ctrl+Z` pour la suspendre puis `bg` pour redémarrer son exécution. Si vous ne précisez pas le numéro de la tache à reprendre après `bg`, par défaut, c'est la tache précédée d'un + qui sera redémarrée.

### fg %$n$

Pour ramener une tâche au premier plan, utilisez la commande `fg %`$n$ («*ForeGround*») avec le numéro de la tâche à la place de n. Si vous utilisez `fg` sans argument, c'est la dernière tâche lancée

13. Si vous êtes habitués à utiliser le raccourci `Ctrl+Z` pour annuler votre dernière action, faites attention : dans la console, la tâche sera juste suspendue, mais pas éliminée. Si votre but est d'annuler cette tâche, il faudra penser ensuite à la *tuer*. Sinon, la prochaine fois, pensez à utiliser `Ctrl+D` ou `Ctrl+C` pour stopper une tâche en force.

(signalée par un +) qui sera ramenée au premier plan.

```
$ fg 2
sleep 3600
```

Pour lancer deux commandes en arrière plan, pensez à les entourer de parenthèses pour que le symbole & influence les deux commandes et pas seulement la dernière :

```
$ (sleep 3600; echo "ça fait une heure !") &
```

Si la commande passée en arrière plan retourne des messages d'erreur ou des données, ces informations seront affichés dans la console. Pensez, donc, à rediriger les sorties pour ne pas être gênés (§. 5.2). Attention, si je veux lancer une commande utilisant des redirections en arrière plan, il faut placer le & tout à la fin pour que les redirections soient correctement prises en compte. Dans la commande suivante, il ne faut pas confondre les deux & : le premier fait partie de l'expression 2>&1 qui permet de rediriger les erreurs dans le même fichier que la sortie standard, le second demande que la commande soit passée en arrière plan :

```
$ backup.sh > backup.log 2>&1 &
[1] 7989
```

### 6.3.5 Détacher une tâche de la console

Comme je vous l'ai dit précédemment, une tâche passée en arrière plan reste attachée à la console qui l'a lancée. Pour fermer la console sans risquer d'interrompre la tâche, il faut utiliser la commande nohup («*no SIGHUP*», ne reçoit pas le signal SIGHUP (§. 6.3.6)). Cette commande est indispensable par exemple si vous vous connectez à une machine à distance (en ssh par exemple) et que vous désirez lancer une tâche qui se poursuivra après votre déconnexion. Pour lancer une tâche de fond détachée de la console, votre commande sera du type :

```
$ nohup commande &
```

Les sorties de la commande sont redirigées automatiquement dans un fichier nohup-out créé dans le répertoire courant. Si vous souhaitez récupérer les sorties dans un autre fichier, vous devez utiliser des redirections (§. 5.2).

### 6.3.6 Tuer un processus : kill

Il est parfois nécessaire d'arrêter un processus en force (de le *tuer*) : parce qu'il ne répond plus, parce qu'il consomme trop de mémoire et va faire planter l'ordinateur si on ne l'arrête pas fissa, parce qu'il est parti dans une boucle infinie suite à une erreur de programmation de votre part, etc.

Pour *tuer* un processus, on utiliser la commande kill. Cette commande a pour effet d'envoyer un signal à un processus, le signal par défaut est un signal d'arrêt (SIGTERM). La plupart du temps ce signal sera suffisant, mais parfois il faudra utiliser le signal SIGKILL qui force l'arrêt d'un processus. Aucun processus ne peut ignorer le signal SIGKILL, il est forcé d'obéir, ce n'est pas le cas pour les autres signaux : on peut ajouter des instructions dans un programme pour demander à ce que les signaux soient interprétés différemment que ce qui est prévu par défaut. Pour connaître la liste des signaux qui peuvent être envoyés par kill, vous pouvez utiliser la commande kill -l. L'envoi d'un signal peut être fait en utilisant son nom (-SIGKILL) ou son numéro (-9). Les signaux sont utilisés

entre les processus ou entre un utilisateur et les processus pour changer leur statut d'exécution.

Pour indiquer à `kill` à quel processus envoyer le signal, il faut lui donner son `PID`, le numéro unique qui identifie le processus sur le système. Vous pouvez voir le PID des processus avec les commandes `ps` ou `top`. Voici deux exemples d'emploi de la commande `kill` :

```
$ kill 1234

$ kill -9 1234
```

La commande killall permet de faire la même chose en utilisant le nom des processus à `tuer`. La commande suivante envoie le signal `SIGTERM` à tous les occurrences de la commande `sleep` :

```
$ killall sleep
```

Il est aussi possible de tuer un processus directement pendant l'utilisation de la commande `top`. Un appui sur la touche `K` permet alors de lancer un signal. Un message s'affiche au dessus de la liste des processus :

```
    PID to signal/kill [default pid = xxxx]
```

A la place des xxxx, le PID du premier processus de la liste (le plus gourmand) est indiqué. Si c'est lui qui pose problème, il suffit de confirmer en appuyant sur Entrer. Sinon, après cette invite de saisie, vous devez indiquez le `PID` du processus et valider. Ensuite, indiquez le type de signal à envoyer (-SIGKILL ou -9, -SIGTERM ou -15, etc.).

## 6.4 Installer des programmes depuis le terminal

Pour installer des programmes dans le chapitre précédent nous avons utilisé la logithèque. Nous allons voir maintenant qu'il est possible de le faire depuis la console.

Pour installer des programmes, on peut utiliser la commande `apt-get` qui va chercher des paquets dans les dépôts (comme la logithèque). `apt-get` est un gestionnaire de paquets utilisé sur les systèmes hérités de *Debian* et qui gère les dépendances entre paquets. Comme je l'ai déjà dit il existe d'autre types de paquets et donc d'autres gestionnaires sur les autres systèmes : rpm pour Red-Hat (ne gère pas les dépendances), yum sous Fedora (rpm + dépendances), rpmi sous Mandriva (rpm + dépendances), emerge sous Gentoo (compile toujours à partir des sources), etc.

Les paquets récupérés par `apt-get` sont déjà compilés et conçus pour fonctionner sur votre machine et correspondent à la toute dernière version du logiciel. Parfois, cela peut constituer un problème si la dernière version présente un problème de compatibilité. Cela arrive avec le programme `unison` pour lequel il existe des problèmes de numéro de version incompatibles qui font qu'on peut avoir envie d'installer une version particulière. Il arrive aussi qu'un programme ne soit pas proposé dans les dépôts. Dans ce cas, on peut avoir envie de compiler le programme à partir des sources, c'est à dire de taper une par une les commandes pour faire la procédure qui consiste à transformer des fichiers contenant du code lisible par un humain en un programme binaire compréhensible par votre machine. Nous allons donc voir ici les deux procédures.

### 6.4.1 Utiliser apt-get

La liste des dépôts utilisés par `apt-get` est dans le fichier `/etc/apt/sources.list`. Vous pouvez constater que chaque dépôt est indiqué sous forme d'un lien internet et est précédé de `deb` pour les paquets ou `deb-src` pour les sources. Si vous souhaitez ajouter des dépôts, il faut éditer ce fichier.

Comme `apt-get` permet d'installer des programmes et donc de faire des modifications majeures dans la machine, il faut être super-utilisateur pour utiliser les commandes qui suivent.

#### apt-get update

La commande `apt-get update` permet de mettre à jour la liste des paquets disponibles dans les dépôts (le cache). Il est bon de commencer par là si vous ne connaissez pas le nom exact du paquet que vous souhaitez installer et avez besoin de faire une recherche.

#### apt-add-repository

La commande `apt-add-repository` permet d'ajouter un nouveau dépôt dans la liste des dépôts existants. Il est possible également d'ajouter ce dépôt à la main dans le fichier `/etc/apt/sources.list`. La manière de déclarer un dépôt dans ce fichier est particulière. Je vous invite à examiner la documentation de Ubuntu pour avoir plus d'information à ce sujet.

#### apt-cache search ...

La commande `apt-cache search` suivie d'un mot clef permet de rechercher des paquets dans le cache. Si vous faites une recherche avec le mot clef « *unison* » vous verrez qu'il existe plusieurs paquets correspondant à cet outil avec ou sans interface GTK+.

#### apt-cache show ...

La commande `apt-cache show` suivie d'un nom de paquet, permet d'obtenir des informations sur le paquet en question : la version du programme, la taille qu'il occupera sur le disque, ses dépendances, un descriptif de ces fonctionnalités, etc. Recherchez par exemple la documentation de `unison` avec `apt-cache show unison`.

#### apt-get install ...

La commande `apt-get install` suivie du nom d'un paquet permet d'installer le paquet en question. Après téléchargement, la commande vous indique la taille qui sera occupée sur le disque, ce qui doit être installé et vous demande confirmation avant de procéder à l'installation.

#### apt-get remove ...

La commande `apt-get remove` suivie du nom d'un paquet permet de désinstaller le paquet en question mais ne supprime pas ses dépendances. Pour retirer également les dépendances utilisez la

commande `apt-get autoremove`

### apt-get upgrade

La commande `apt-get upgrade` permet de mettre à jour tous les paquets de la machine. Il est utile de lancer avant la commande `apt-get update` pour que la commande soit au courant des dernières versions existantes.

### Mise en pratique

Certains paquets dont vous aurez besoin pour suivre les cours, ne sont pas disponibles dans la logithèque utilisée par Ubuntu 16.04. Nous allons donc les installer avec `apt-get`, il s'agit de :

**clisp**
: Une implémentation de Common Lisp nécessaire pour le cours de *Programmation fonctionnelle*.

**python-tk**
: Module pour créer des interfaces graphiques avec le langage python. Nécessaire pour le cours de *Méthodologie de la programmation*.

**swi-prolog**
: Interpréteur Prolog nécessaire pour le cours de *Programmation logique*.

**manpages-dev manpages-fr-extra manpages-posix manpages-posix-dev**
: Des pages de manuel sur les fonctions posix et une version française du manuel (`manpages-fr-extras`).

**ubuntu-restricted-extras**
: Applications couramment utilisées restreintes par des droits d'auteur.

Pour commencer, vérifiez que `apt-get` est au courant des dernières versions de logiciels existantes dans les dépôts :

```
$ sudo apt-get update # je dois être administrateur
                      # pour avoir le droit d'installer des choses...
```

Ensuite vous pouvez utiliser `install` pour chacun des paquets. On peut lancer une seule commande pour installer plusieurs paquets à la fois ou bien installer chaque paquet l'un après l'autre, ce qui permet de savoir plus facilement quel paquet est en cause en cas de problème.

Si une interface graphique en console apparait pour la configuration d'un des paquets, utilisez la touche *Tab* (tabulation) pour sélectionner un bouton et *Entrer* pour valider un choix. Voici les commandes à saisir, appuyez sur « *o* » suivi de *Entrer* quand un message vous demande de confirmer l'installation :

```
$ sudo apt-get install clisp

$ sudo apt-get install python-tk

$ sudo apt-get install swi-prolog

$ sudo apt-get install ubuntu-restricted-extras

$ sudo apt-get install manpages-dev manpages-fr-extra # là on installe
```

```
                                    # plusieurs paquets en même temps
$ sudo apt-get install manpages-posix manpages-posix-dev # idem
```

### 6.4.2 Compiler un programme depuis les sources

Comme je l'ai dit plus haut, si le paquet proposé ne vous convient pas ou si il n'est pas proposé du tout, il est toujours possible de le compiler soi-même avant de l'installer. Comme le monde de Linux est un monde *open-source*, voire *libre*, les créateurs d'un programme proposeront souvent les sources de leur programme.

Pour nous faciliter quand même la vie, les créateurs de programmes, accompagnent leurs sources d'outils pour faciliter le processus de compilation. Afin de vous montrer ce processus avec un exemple concret que vous pourrez tester vous même, je vous propose de télécharger les sources d'un programme. Il est disponible à cette adresse : http://sourceforge.net/projects/chromium-bsu/, cliquez sur le bouton vert pour télécharger le fichier `chromium-bsu-0.9.15.1.tar.gz`.

Le fichier va certainement être enregistré dans votre dossier *Téléchargement*. Rendez-vous dans ce dossier pour décompresser et dés-archiver le fichier (`man tar` pour comprendre les options que j'ai utilisées) :

```
$ cd Téléchargements/
$ tar -xzf chromium-bsu-0.9.15.1.tar.gz
$ cd chromium-bsu-0.9.15.1/
```

#### configure

La première commande à utiliser est `configure`. Elle permet de vérifier que toutes les dépendances sont présentes et ensuite elle écrit un fichier `Makefile` qui contiendra les commandes de compilation. Si le créateur du logiciel est sympa, il aura indiqué sur son site les dépendances à installer avant ; dans le cas contraire, on lance `configure` pour voir :

```
$ configure
(...)   # ceci indique que je n'ai pas tout recopié ici
configure: error: in `/home/martin/Downloads/chromium-bsu-0.9.15.1':
configure: error: C++ compiler cannot create executables
See `config.log' for more details
```

Il est fort possible que la commande retourne des erreurs de dépendances. Dans ce cas, il faut installer les dépendances manquantes. Pour les trouver, il suffit de faire une recherche avec `apt-cache search ...`. Si la recherche retourne trop de résultats et que vous ne savez pas quoi installer, une petite recherche sur Internet ou dans la documentation Ubuntu devrait vous aider. Après une petite recherche, vous découvrirez que pour compiler du C++, il vous faut `g++` :

```
$ sudo apt-get install g++
(...)
$ configure
(...)
checking GL/glc.h usability... no
checking GL/glc.h presence... no
```

```
checking for GL/glc.h... no
(...)
checking for glpng/glpng.h... no
checking for pngBind in -lglpng... no
configure: error: cannot find OpenGL (drawing system)
```

Après avoir installé `g++` et relancé `configure`, je vois qu'il y a une autre erreur. Je remonte dans la liste des éléments vérifiés par `configure` (checking ...) et je repère des premiers éléments qui n'ont pas été trouvés : apparemment il manque une bibliothèque `glc`. Les bibliothèques commencent généralement par le préfixe `lib`, la bibliothèque `glc` se nomme donc `libglc`. Il y a habituellement deux types de paquets pour une bibliothèque. Celui dont nous aurons généralement besoin pour compiler est celui se terminant par `-dev` et qui contient les fichiers d'entêtes nécessaires à la compilation (vous comprendrez mieux cette notion après le cours de C) :

```
$ apt-cache search libglc
libglc-dev - An implementation of SGI's OpenGL Character Renderer (GLC)
libglc0 - QuesoGLC GLC implementation
$ sudo apt-get install libglc-dev
(...)
$ configure
(...)
configure: error: cannot find SDL >= 1.1.6 or GLUT (window system)
```

Cette fois, `configure` se plaint de ne pas avoir trouvé `SDL`. Un petit tour dans la documentation d'Ubuntu vous apprend que pour installer `SDL`, il faut installer les paquets `libsdl1.2debian` et `libsdl1.2-dev`. Après l'avoir installé et relancé `configure`, vous réalisez qu'il manque encore un truc appelé `SDL Mixer` et vous vous dites que si vous aviez lu la documentation jusqu'au bout, vous auriez remarqué qu'en effet il est conseillé d'installer tous les paquets suivants : `libsdl-image1.2`, `libsdl-image1.2-dev`, `libsdl-ttf2.0-0`, `libsdl-ttf2.0-dev`, `libsdl-mixer1.2` et `libsdl-mixer1.2-dev`.

```
$ sudo apt-get install libsdl1.2debian libsdl1.2-dev
(...)
$ configure
(...)
configure: error: cannot find OpenAL & ALUT or SDL Mixer (sound)
$ sudo apt-get install libsdl-image1.2 libsdl-image1.2-dev libsdl-ttf2.0-0
                       libsdl-ttf2.0-dev libsdl-mixer1.2 libsdl-mixer1.2-dev
(...)
$ configure
(...)
Chromium B.S.U. ready for building!

  Window type: SDL
  Window icon: yes
  Image type:  SDL_image
  Audio type:  SDL mixer
  Text type:   GLC
  Font search: default (bold Gothic Uralic)

Type 'make' to build.
```

A ce stade, `configure` vous affiche enfin le message tant attendu : « *ready for building ! [...] Type*

*'make' to build* ». Cet exemple vous permet de mesurer à quel point l'identification et l'installation des dépendances n'est pas évidente. Désormais, vous apprécierez d'autant plus que `apt-get` le fasse pour vous la plupart du temps.

La commande `configure` peut prendre différents paramètres pour réaliser la création du fichier `Makefile`. La plus courante est `--prefix=` qui permet de préciser le répertoire dans lequel on veut faire l'installation. Par défaut c'est `/usr/local`, ce qui est très bien lorsqu'on est dans une distribution standard de Linux.

#### make

La commande `make` permet ensuite de réaliser la compilation. Elle va lire le fichier `Makefile` et afficher chaque commande exécutée. Le processus est plus ou moins long selon le programme installé. Parfois un fichier `Makefile` sera déjà présent avec les sources, vous pouvez alors tenter de lancer la commande `make` pour compiler directement.

C'est le moment où l'on croise les doigts en espérant que tout va bien se passer. En cas d'erreur, il vous faudra aller modifier vous même le code source, ce qui demande de connaître le langage de programmation utilisé. Parfois vous pourrez passer de longues heures sur les forums ou à envoyer des mails au développeur du logiciel pour savoir comment résoudre le problème (bonne chance !). Le programme que j'ai choisi comme exemple ne devrait pas vous poser ce genre de problème et la compilation se termine sans que le terrible mot « *error* » ne se soit affiché (ouf !).

#### make install

Cette commande effectue l'installation. Elle demande à avoir les droits d'administrateur si vous faites cette installation dans un autre répertoire que votre `home`.

Maintenant, savourez votre première utilisation réussie de `configure && make && make install` en lançant le programme installé et en vous octroyant une petite récréation :

```
$ chromium-bsu
```

## 6.5 Quand ça ne marche pas...

Quand votre code ne fonctionne pas comme vous aimeriez, le premier réflexe à avoir n'est pas de foncer demander de l'aide sur un forum. Comme je vous l'ai dit, il est bon de prendre la peine de lire le manuel (ou la documentation dans le cas des langages de programmation que vous allez apprendre cette année), mais ce n'est pas la seule source d'information dont vous disposez. Vous avez peut-être pris l'habitude d'ignorer ces messages incompréhensibles que Windows peut parfois vous cracher à la figure (Fig. 6.4), mais sous Linux les messages d'erreurs peuvent être riches en informations et vous indiquent la position et la nature de votre erreur. Ces messages sont bien sûr en anglais. Si vous ne comprenez pas bien l'anglais, il est temps de vous y mettre, en attendant vous pouvez utiliser un traducteur automatique pour traduire les mots clefs présents dans les messages d'erreur.

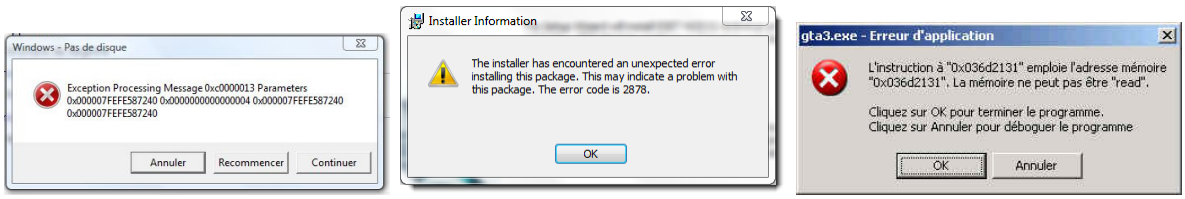


**Figure 6.4 – Messages d'erreur sibyllins.**

### 6.5.1 Comprendre les messages d'erreur

Comme je viens de le dire, Un message d'erreur vous informe premièrement sur la position de l'erreur, ensuite sur sa nature. Voici un message obtenu en tentant d'exécuter un script shell qui contient une erreur :

```
$ ./prog.sh
./prog.sh: line 5: syntax error near unexpected token `echo'
./prog.sh: line 5: `    echo "argument $count = $arg"'
```

Vous pouvez voir que le message commence par le nom du script contenant l'erreur et tout de suite après l'information `line 5` qui nous indique qu'il a stoppé l'exécution du script à cette ligne. Ensuite il nous informe sur la nature du problème `syntax error near unexpected token 'echo'` : il y a une erreur de syntaxe juste avant le mot `'echo'` et il précise que ce mot est `unexpected` (=inattendu) donc il manque quelque chose juste avant, il s'attendait à rencontrer une autre instruction. La suite du message d'erreur est une copie de la ligne 5 pour indiquer son contenu.

En examinant le script, je peux me focaliser sur l'endroit indiqué et voir rapidement ce qu'il manque :

```
#! /bin/bash

count=1
for arg in $*;  # ici on voit qu'il manque l'instruction "do"
    echo "argument $count = $arg"
    count=`expr $count + 1`
done

exit 0
```

Il n'y a pas que les commandes shell qui retournent des messages d'erreur. Il en est de même pour les autres langages que vous allez étudier cette année. Les explications qui suivent seront surement plus compréhensibles quand vous aurez commencé à étudier ces langages. Si le reste de cette section n'est pas clair pour vous pour le moment, je vous suggère d'y revenir quand vous aurez démarré les autres cours.

Voici par exemple une erreur obtenue en lançant un script python :

```
$ ./prog.py
Traceback (most recent call last):
  File "./prog.py", line 6, in <module>
    mafonction()
  File "./prog.py", line 4, in mafonction
    var = "truc" + 54
TypeError: cannot concatenate 'str' and 'int' objects
```

En soumettant à un traducteur automatique la première ligne, le premier mot est traduit par « *retraçage* » ou « *enquête de traçage* ». Le *Traceback* indique en fait les différents commandes qui ont été appelées (une commande peut en appeler une autre) jusqu'à ce que l'erreur se produise. Entre parenthèses, « *most recent call last* » nous indique que la commande appelée en dernier (la plus récente) se trouve à la fin.

Si on regarde les lignes suivantes, on voit que dans le fichier « *./prog.py* », à la ligne 6, dans « *<module>* » (c'est à dire le corps du programme), l'instruction « *mafonction()* » a été exécutée. Cette instruction a provoqué l'exécution d'une autre instruction située aussi dans le fichier « *./prog.py* », à la ligne 4 cette fois, dans « *mafonction* » (la fonction déclenchée par l'instruction précédente). Cette instruction « *var = "truc" + 54* » étant la dernière, l'erreur s'est produite au moment de l'exécuter.

Le message suivant explique le type d'erreur qui s'est produit : il s'agit d'une erreur de type « *TypeError* ». L'explication qui suit indique qu'il est impossible de concaténer (=juxtaposer) un élément de type 'str' (une chaîne de caractère) avec un élément de type 'int' (un entier). Si on regarde l'instruction précédente, on voit qu'on a en effet tenté d'additionner un mot avec un nombre (avant de mettre le résultat dans une variable nommée « *var* »), ce qui n'a pas vraiment de sens...

Maintenant examinons une erreur obtenue pendant la compilation d'un programme en langage C[14] :

```
$ gcc -Wall prog.c -o prog
prog.c:7:2: error: non-void function 'main' should return a value
      [-Wreturn-type]
        return;
        ^
1 error generated.
```

Nous voyons que le message commence par le nom du fichier .c contenant l'erreur, suivi de deux nombres « *7 :2* » qui indiquent le numéro de la ligne et de la colonne : l'erreur est donc identifiée au 2ème caractère de la ligne 7. La description de l'erreur indique ensuite qu'une fonction « *non-void* » a été définie et qu'elle doit donc retourner une valeur, ce qui n'a sans doute pas été fait puisque l'erreur se produit. L'expression « *[-Wreturn-type]* » rappelle l'option qui par défaut indique que ce type d'erreur doit être détecté à la compilation et qu'il faut désactiver si on ne veut plus d'alertes à ce propos : on ne le fera pas car c'est bien d'être prévenu quand on fait une erreur.

La ligne suivante, montre une copie de la ligne de code qui pose problème « *return ;* ». Nous constatons que le mot clef `return` n'est suivit d'aucune valeur à retourner, le problème vient donc bien de là. Le petit symbole « ^ » dessous pointe sur le caractère où l'erreur a été identifiée : c'est bien pratique pour localiser l'erreur quand l'instruction est très longue. Le message d'erreur se conclut par le nombre d'erreurs signalées : il peut y en avoir plusieurs dans un même message.

Pour terminer, nous allons voir comment se présente un message d'erreur pour Common Lisp. J'ai définis une fonction Lisp dans l'interprète et il m'a répondu « *filtre-o* » ce qui signifie que jusque là tout s'est bien passé. Mais quand j'ai tapé la commande suivante « *(filtre-o '(o a a b o o d o z o e))* » pour exécuter cette fonction, un message d'erreur s'est affiché :

```
> (defun filtre-o (liste)
    (cond
      ((atom liste) nil)
```

14. La compilation consiste à transformer un programme écrit dans un langage informatique compréhensible par un humain en langage machine compréhensible par votre ordinateur. On peut ensuite l'exécuter. Certaines erreurs peuvent être détectées pendant cette étape de transformation.

```
      ((equal (car liste) 'o) (cons (car liste) (filtre-o (cdr liste))))
      ((filtre-o (cdr liste nil))) ) )
filtre-o
> (filtre-o '(o a a b o o d o z o e))
*** - EVAL: too many arguments given to cdr: (cdr liste nil)
The following restarts are available:
ABORT          :R1      Abort debug loop
ABORT          :R2      Abort debug loop
ABORT          :R3      Abort debug loop
ABORT          :R4      Abort main loop
```

Ce message donne directement une description de l'erreur détectée : « *EVAL : too many arguments given to cdr* ». Ce message signifie que lors de l'évaluation de la commande, trop d'arguments ont été trouvés pour la fonction `cdr`. Ensuite une copie de l'instruction est donnée « *(cdr liste nil)* ». La fonction cdr ne prend qu'un seul argument et nous constatons qu'il y en a 2 : « *liste* » et « *nil* ». Les lignes suivantes indiquent les différentes options proposées pour résoudre l'erreur et continuer l'exécution du programme : ici on a pas d'autre choix que de tout arrêter (`ABORT`) et corriger l'erreur avant de recommencer.

« *Ok, là tu m'expliques ce que signifient les messages, mais moi qui débute comment je fais pour comprendre ?* »

Au départ, vos premières commandes seront très simples et vos premiers programmes ne feront que 3-4 lignes. C'est le moment d'en profiter pour tester plein de choses, tout et n'importe quoi, et prendre note des messages d'erreurs correspondants :

- « *Ça fait quoi si je tente d'additionner un nombre avec un mot ?* »
- « *Ça fait quoi si j'utilise une variable qui n'existe pas ?* »
- « *Ça fait quoi si je tente d'écrire dans la case 5 d'un tableau qui n'a que 4 cases ?* »
- etc...

Quand votre professeur de *Méthodologie de la programmation* vous encourage à tester, expérimenter, c'est aussi à cela qu'il fait référence. Quand vous recevez un message d'erreur, vous pouvez le noter et indiquer à quoi il correspond. De cette manière quand vous rencontrerez à nouveau ce message pour un programme de 3000 lignes, vous aurez immédiatement une idée de l'origine de l'erreur et déboguer votre code sera bien plus simple.

Bien sûr au départ ce travail est assez fastidieux. Si vous ne comprenez pas l'anglais, il faudra utiliser le traducteur pour comprendre chaque mot et rechercher sur internet quand le traducteur n'aide pas vraiment. Il peut être nécessaire d'interroger vos professeurs, les tuteurs ou les autres étudiants pour qu'ils vous expliquent ce que signifie un message d'erreur. Cependant, ils seront bien plus enclins à vous répondre que si vous demandez directement pourquoi votre code ne marche pas sans prendre la peine d'essayer de comprendre les messages d'erreur.

Maintenant, voici une liste partielle de messages d'erreurs que vous pourrez rencontrer en programmant dans différents langages, à vous de compléter la liste au fur et à mesure de vos expériences :

### Shell :

**`syntax error near unexpected token `****'`**

Vous avez oublié quelque chose ou ajouté une instruction inattendue juste avant le mot "****".

**expr: not a decimal number: 'val'**
: La commande 'expr' attend un nombre et vous lui avez donné le mot "val". Peut être avez vous oublié le `$` qui permet de récupérer le contenu d'une variable ?

******: command not found**
: Vous avez fait une erreur dans le nom de la commande ou bien vous avez oublié de placer le nom d'une commande en début de ligne.

******: Operation not permitted**
: Il vous manque sans doute les droits d'administrateur pour exécuter cette commande. Pensez à utiliser sudo en début de ligne pour exécuter cette commande en tant qu'administrateur.

Python :

**SyntaxError: invalid syntax**
: En gros l'interprète vous dit que vous écrivez des instructions qui ne sont pas en python ! L'interprète ne sais pas quelle langage vous parlez mais pas le sien en tout cas !

**IndentationError: expected an indented block**
: Vous avez fait une erreur d'indentation, l'interprète s'attendait à trouver une ligne décalée.

**IndentationError: unindent does not match any outer indentation level**
: L'indentation est fantaisiste. Vérifiez que vous n'avez pas mélangé les tabulations et les espaces.

**NameError: name '****' is not defined**
: Vous utilisez une variable ou une fonction dont le nom est inconnu.

**NameError: global name '****' is not defined**
: Vous utilisez une variable ou une fonction dont le nom est inconnu à l'intérieur d'une fonction.

**TypeError: Can't convert 'int' object to str implicitly**
: Vous utilisez une valeur ou une variable du mauvais type. Ici l'interprète attend une chaîne et vous lui donnez un entier : il ne sait pas comment faire pour faire une conversion de l'un en l'autre.

**TypeError: unsupported operand type(s) for /: 'str' and 'str'**
: Vous tenter de faire une opération avec des valeurs non supportées. Ici, vous tenter de diviser deux chaînes de caractères.

**TypeError: 'str' object is not callable**
: L'interprète vous signale que vous tentez « *d'appeler* » une chaîne de caractère : en gros vous voulez l'exécuter. On ne peut exécuter qu'une fonction, les chaînes et les nombres sont juste des valeurs, elles ne sont pas exécutables (=callable).

**TypeError: 'int' object has no attribute '__getitem__'**
: Vous tentez d'accéder à une case dans un entier... Il y a des cases dans les tableaux mais pas dans les nombres entiers !

**TypeError: ****() takes exactly 1 argument (2 given)**
: Vous n'avez pas donné le bon nombre d'arguments à une fonction.

**ValueError: invalid literal for int() with base 10: 'a'**
: La valeur fournie en argument à la fonction, n'est pas valide : ici int() attend un nombre et vous lui avez transmis une lettre.

**ZeroDivisionError: integer division or modulo by zero**
: Vous tentez de faire une division par 0 ou une opération modulo.

**ImportError: No module named ******
: Vous tentez d'importer un module inconnu avec l'instruction `import ****`.

**ImportError: cannot import name truc**
: Vous tentez d'importer un morceau inconnu d'un module connu avec une instruction du type `from connu import ****`.

C :

**warning: implicit declaration of function '****'**
: Vous utilisez une fonction qui n'est pas définie, le prototype de la fonction est absent ou après la ligne où vous l'utilisez.

**error: redefinition of '****'**
: Vous avez déclaré 2 fois une fonction du même nom.

**undefined reference to '****'**
: Vous faites référence à un élément (une variable ?) inconnu.

**error: '****' undeclared (first use in this function)**
: Vous utilisez pour la première fois un élément inconnu jusqu'ici.

**warning: comparison between signed and unsigned integer expressions**
: Vous comparez un entier naturel avec un entier relatif : le codage de ses nombres étant différent (voir cours d'architecture de L1) vous risquez un gros bugg.

**warning: comparison of unsigned expression >= 0 is always true**
: Vous faites un test « *>= 0* » avec un entier naturel qui est toujours supérieur à 0.

**warning: unused variable '****'**
: Vous avez déclaré une variable et vous ne l'utilisez pas.

**warning: '****' is used uninitialized in this function**
: Vous utilisez une variable sans lui avoir donné de valeur, elle peut contenir n'importe quoi en fonction de ce que contenait la mémoire avant d'être utilisée pour stocker cette variable.

**error: expected ';', identifier or '(' before '*'**
: Il manque un point-virgule.

**error: expected declaration or statement at end of input**
: Il manque une accolade fermante quelque part. Ne vous fiez pas au numéro de ligne de cette erreur, l'erreur peut être plusieurs lignes après en fonction des autres accolades présentes dans le code.

**warning : missing braces around initializer**
: Vous initialisez sans doute un tableau à deux dimensions comme un tableau à une dimension.

**warning: suggest braces around empty body in an 'if' statement**
: Vous avez un point-virgule en trop immédiatement après un bloc `if`, `while`, `for`...

**error: invalid conversion from 'void*' to 'int*'**
: Vous utilisez un pointeur du mauvais type.

**warning: passing argument N of '****' discards 'const' qualifier from pointer target type**
: Vous transmettez a une fonction une variable qui est déclarée "constante", si la fonction la modifie, cela peut provoquer un bug à l'exécution.

**warning: expected 'char *' but argument is of type 'const char *'**
: Vous tentez de transmettre une chaîne constante (stockée dans une zone spécifique de la mémoire) à une fonction qui veux une chaîne modifiable.

**error: lvalue required as left operand of assignment**
: Vous tentez d'assigner une valeur à quelque chose qui n'est pas une variable (ou qui n'est pas modifiable) : peux-être avez vous confondu `=` et `==` ?

**Segmentation fault: 11**
: Vous tentez (à l'exécution) d'écrire ou lire une zone mémoire qui n'appartient pas à votre programme. L'erreur est souvent due au fait que vous tentez d'écrire au delà de la taille d'un tableau.

### Common Lisp :

**+: **** is not a number**
: Lisp attend un nombre et la valeur fournie (ici à l'opération +) n'en est pas un.

**system::read-eval-print: variable **** has no value**
: Vous utilisez une variable qui n'est pas encore définie.

**EVAL: too many arguments given to ****: ...**
: Vous donnez trop d'arguments à la fonction.

**EVAL: too few arguments given to ****: ...**
: Vous donnez trop peu d'arguments à la fonction.

**eval: **** is not a function name; try using a symbol instead**
: Vous avez placé une valeur au début d'une liste et l'interprète essaye de l'interpréter comme un nom de fonction... Vous avez peut être oublié le ' qui permet de ne pas évaluer la liste.

**eval: undefined function ****
: Vous utilisez une fonction inconnue.

**read from ... : an object cannot start with #\)**
: Vous avez une parenthèse fermante en trop.

**...: syntax error after **** in ...**
: Vous n'écrivez pas du Lisp...

## 6.5.2 Savoir formuler une demande d'aide

Si vous avez épluché les messages d'erreur sans succès, que vous ne les comprenez pas ou que vous n'en recevez pas alors que votre code ne fait pas ce que vous espérez, que vous avez examiné votre code et que vous n'identifiez toujours pas l'origine du problème, il est temps d'aller chercher de l'aide.

La première chose à faire est d'éplucher le forum du cours et les forums en ligne sur internet pour voir si une question similaire n'a pas déjà été posée. Tous les débutants font plus ou moins les mêmes erreurs et il y a de fortes chances que vous trouviez un forum où la question a déjà été posée et le problème résolu.

Si malgré tout, vous ne trouvez pas de solution correspondant à votre problème, voici quelques conseils pour bien formuler votre question, augmenter vos chances d'obtenir une réponse et éviter toute sorte de remarques désobligeantes.

### Bien formuler le sujet

Tout d'abord pour avoir une chance d'obtenir une réponse et pour que votre expérience puisse servir à d'autres par la suite, il est bon de bien choisir le sujet. Il faut que le sujet ou la nature de votre problème soit immédiatement identifiable. Des sujets comme « *Problème en Lisp* » ou « *Besoin d'aide pour une fonction* » sont bien trop vagues et généraux. Utiliser le message d'erreur

comme sujet peut être une solution à condition qu'il ne soit pas trop long ni trop vague : un « *syntax error* » par exemple n'apportera aucune information.

Si vous écrivez sur le forum d'un cours de la licence, vous pouvez indiquer le numéro de l'exercice concerné, un rappel de son sujet et une brève description du type de problème rencontré, par exemple : « *px32-1, jeu de pendu, problème d'encodage* ». Sur un forum général sur internet, il faudra peut-être préciser le langage concerné et le sujet en plus du problème, par exemple : « *C, fonction récursive, problème de fuite de mémoire* ».

De manière générale, si vous ne savez pas quel sujet donner à votre question, demandez-vous quels mots clefs vous avez utilisés pour chercher si une question correspondant à un problème similaire au votre avait déjà été posée sur un forum et utilisez ces mots clefs dans le sujet de votre message.

### Exprimer politesse et reconnaissance

Bien entendu, dans votre message, il est bon d'être poli. Inutile de pester contre votre code qui ne marche pas, la source du bugg est presque toujours entre votre siège et votre clavier ! Dites « *bonjour* » ou « *salut* » au cas ou certains se formaliseraient que vous ne le fassiez pas et pensez aussi à ajouter à la fin un petit « *merci d'avance de l'aide que vous pourrez m'apporter* » ou autre formule de politesse encourageant vos lecteurs à vous répondre.

Si vous devez re-poster un nouveau message par la suite parce que la réponse apportée n'est pas totalement claire ou que l'on vous demande des précisions pour pouvoir vous aider, surtout commencez par exprimer votre reconnaissance à la personne qui a prit la peine de vous répondre. Évitez de râler si la réponse donnée semble hors sujet, faites le remarquer avec diplomatie et expliquer en quoi la réponse ne vous aide pas : on sait jamais, c'est peut-être vous qui n'avez pas compris la réponse (§. 6.5.2), dans ce cas être agressif ou désobligeant risque de vous faire passer pour un imbécile.

Quand le problème est résolu, prenez la peine de le signaler (c'est la règle sur la plupart des forums) et remerciez une dernière fois ceux qui on pris la peine de vous aider. Si vous avez finalement trouvé la solution par vous-même, indiquez quelle est cette solution, ça aidera ceux qui rencontreront le même problème après vous.

### Bien expliquer le contexte

Revenons à l'écriture de votre premier message. Évitez de plonger dans le vif du sujet sans prévenir. Vous avez passé beaucoup de temps sur votre code et êtes en plein dans le sujet, mais les autres débarquent et ne savent pas du tout sur quoi vous travaillez et les choix que vous avez fait dans la mise au point de votre programme, alors prenez le temps de leur indiquer ce que vous essayez de faire et à quel moment vous rencontrez un problème.

Par exemple si je rencontre un message qui commence ainsi : « *J'ai un problème dans mon programme quand je fais print(resultat) ça plante* », je vais me poser plein de questions : pourquoi il fait `print(resultat)` ? Qu'est-ce qu'il essaye de faire ? C'est quoi `resultat` ? Une variable contenant un nombre ? un tableau ? Ça plante comment ? Il y a un message d'erreur ? Le programme s'arrête sans prévenir ? La machine plante totalement et il faut la redémarrer ? Est-ce qu'il y a des signes avant coureurs (la machine rame) ou bien se fige brutalement ?

Il est généralement exaspérant de devoir demander à la personne qui pose une question de devoir

préciser toutes ces choses, et exaspérer dès le départ les personnes qui sont susceptibles de vous aider ce n'est pas une bonne chose. Prenez donc le temps de rappeler ce que vous essayer de faire : que doit faire votre programme/fonction/commande ? Si vous montrez un bout de code : expliquez à quoi servent les variables, tableaux, fonctions que vous avez définies si ce n'est pas évident à la lecture du code.

### Montrer que l'on a bien réfléchi au problème

Une fois le contexte rappelé, expliquez votre problème en décrivant ce que vous vous attendiez à avoir comme résultat et ce que vous avez effectivement obtenu : un message d'erreur ou un résultat différent et lequel.

Montrez à votre lecteur ce que vous avez tenté même si cela n'a rien donné. Quand je rencontre une question de ce genre : « *J'ai essayé de concaténer deux chaines de caractères sans utiliser aucune fonction prédéfinie. Mon problème est que ma solution n'a pas marché sans les fonctions prédéfinies.* », je me demande immédiatement si la personne a réellement cherché ou si elle compte sur les autres pour résoudre le problème à sa place : si c'est le cas, cela ne lui apportera rien et elle ne fera aucun progrès.

Si vous voulez avoir une réponse, il faut montrer que vous avez vraiment cherché. En voyant ce que vous avez tenté, il sera plus facile de vous apporter de l'aide. Si votre code est un peu n'importe quoi on pourra vous apporter des conseils de méthodologie sur la manière d'aborder le problème. Si vous avez fait une erreur localisée, on pourra vous l'indiquer, vous expliquer sa nature et comment la corriger.

### Montrer que l'on a cherché les réponses existantes

Comme je vous l'ai indiqué, la première chose à faire avant de poser une question, est de vérifier qu'il n'existe pas déjà une réponse à une question similaire sur internet. Si les réponses trouvées ont incompréhensibles pour vous ou si elles ne correspondent pas vraiment à votre problème et ne vous ont pas aidé, prenez la peine de le préciser dans votre message : les autres pourront vous expliquer en termes plus simples les réponses données ou éviter de vous donner le même genre de réponse incompréhensible ou inadaptée. Cela vous évitera aussi de recevoir un message du genre « *C'est si difficile de chercher avant de poster ?* http://lmgtfy.com/?q=concaneter ».

### Comprendre et accepter les solutions suggérées

Ça y est, vous avez obtenu une réponse ! Avec de la chance, la réponse sera compréhensible et éclairante et votre problème sera vite résolu. Il sera peut être nécessaire de demander des précisions, mais le problème est en bonne voie de résolution.

Cependant, il peut arriver que la réponse obtenue vous semble totalement « *à côté de la plaque* ». Attention, dans ce cas, relisez bien la réponse et vérifiez consciencieusement si vous avez bien compris. Demandez-vous aussi si ce n'est pas votre question qui est à côté de la plaque. Dans tous les cas, évitez de faire remarquer à votre interlocuteur son erreur de manière agressive et indiquez avec diplomatie que la réponse ne vous semble pas appropriée.

Pour vous faire comprendre ce qui peut se produire, voici une conversation fictive entre mon vieux père et moi même :

« *Aline, tu peux m'aider, je n'arrive pas à installer mon jeu que j'adore sur le portable avec Windows 10 que je viens d'acheter, dis moi comment faire ?* »

« *Il faut que tu installes VirtualBox. Ensuite tu installeras Windows 95 dessus...* »

« *Mais tu comprends rien ! C'est mon jeu que je veux installer ! et l'ordinateur il est sous Windows 10 !* »

La question de départ n'était pas la bonne. Il aurait plutôt dû se demander si c'était seulement faisable. La réponse est non car son jeu est trop ancien. Ma réponse suggère l'usage d'une machine virtuelle et d'un système d'exploitation plus ancien pour pouvoir installer le jeu et le faire tourner. Cependant, mon père ne comprenant pas ce que je lui suggère, s'emporte et pense que je n'ai pas compris sa question...

Ce genre de situation n'est pas si rare quand un professeur ou un étudiant expérimenté essaye d'aider un étudiant totalement débutant. Pour éviter de devoir finalement faire des excuses à la personne qui tente de vous aider ou vous retrouver seul sans réponse persuadé que personne ne vous écoute/ne vous comprend, faites toujours preuve de la plus grande diplomatie quand vous demandez de l'aide sur un forum ou par email. Prenez vraiment le temps de comprendre la réponse qui vous est donnée et demandez poliment des explications supplémentaires si vous ne voyez pas en quoi cette réponse peut vous aider.

# 7 Utiliser une machine virtuelle

Dans ce chapitre, nous allons découvrir ce qu'est une machine virtuelle à travers l'utilisation du logiciel VirtualBox. Je prendrai comme exemple l'installation de Ubuntu dans une VirtualBox sous MacOS. VirtualBox est également disponible sur Windows et sous Linux, les explications données s'appliqueront donc également si vous avez un OS de ce type.

## 7.1 C'est quoi une VM ?

En informatique, une machine virtuelle, *Virtual Machine* (VM) en anglais, est un logiciel qui va simuler la présence d'une machine avec des caractéristiques (mémoire, processeur, disque dur) qui peuvent être totalement différentes de celles votre ordinateur. L'intérêt d'une machine virtuelle est de faire fonctionner des programmes ou un système d'exploitation sur une machine qui n'a pas les caractéristiques pour le faire.

Par exemple, je peux utiliser une machine virtuelle pour simuler un vieil ordinateur des années 80 et faire tourner un très vieux logiciel qui serait inutilisable sur les ordinateurs actuels. Je peux aussi simuler un smart-phone pour tester une application que je suis en train de mettre tranquillement au point sur mon ordinateur. Je peux aussi simplement utiliser une machine virtuelle pour tester la nouvelle version de mon système d'exploitation avant de me décider à l'installer définitivement sur mon ordinateur.

On peut aussi utiliser une machine virtuelle pour des raisons de sécurité : pour isoler un serveur du reste de la machine et éviter que les erreurs n'endommagent l'OS principal. On peut aussi faire tourner plusieurs machines virtuelles en même temps.

Un inconvénient de la virtualisation est que la machine ainsi simulée sera forcément plus lente que l'ordinateur réel. En effet, il faut que l'ordinateur fasse fonctionner son propre système plus le logiciel de virtualisation avant de simuler une autre machine...

## 7.2 Pourquoi installer Ubuntu sous Virtualbox sur un Mac

Les utilisateurs de Mac devraient ici se poser la question suivante : « *Mais pourquoi installer Ubuntu dans une machine virtuelle si c'est plus lent ? Ne pourrait-on pas faire un double-boot comme sur un PC ?* ».

En réalité il est possible de le faire, mais il existe pour certains Mac des problèmes de prise en charge du matériel. L'installation de Ubuntu en double-boot sur un Mac est une opération délicate et peu de documentation est disponible sur Internet, je ne conseille donc pas à un utilisateur novice

de se lancer dans cette aventure.

Une autre raison pour laquelle la virtualisation est satisfaisante sous MacOS est qu'il vous sera possible de suivre une bonne partie des cours sur votre Mac lui-même. Comme je vous l'ai dit, MacOS est lui-aussi fondé sur Unix. Les commandes du terminal de MacOS sont quasiment identiques, Python est déjà installé à la base et vous pouvez installer facilement *Common Lisp* (clisp) et *Prolog* pour les utiliser dans le terminal de votre Mac [1]. Vous pouvez aussi compiler un programme C dans votre console Mac et donc réaliser les exercices de C directement sur votre Mac.

Pour tester Ubuntu, les utilisateurs de Mac peuvent donc de contenter de l'utiliser via une machine virtuelle.

Attention cependant à toujours vérifier que vos programmes fonctionnent correctement sous Ubuntu avant de les envoyer au professeur. Il arrive par exemple que des programmes en C se compilent et fonctionnent normalement sous Mac mais pas sous Ubuntu (le compilateur n'est pas exactement le même et les bibliothèques à importer ne sont pas placées au même endroit, le chemin d'accès peut varier...).

## 7.3 Installation de VirtualBox

La première chose à faire est de télécharger le logiciel VirtualBox (gratuit) sur ce site : `https://www.virtualbox.org/`. Allez dans la rubrique *download* et choisissez en haut de page le *VirtualBox packages* qui correspond à votre système.

Installez le logiciel sur votre ordinateur en suivant l'assistant d'installation puis démarrez l'application.

## 7.4 Création d'une machine virtuelle

Quand VirtualBox démarre pour la première fois, il affiche un message d'accueil vous invitant à créer votre première machine virtuelle (Fig. 7.1).

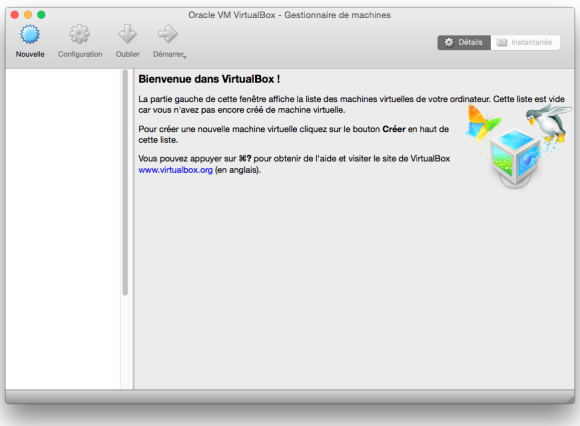


**Figure 7.1 – Premier démarrage du logiciel VirtualBox.**

1. Pour installer `clisp` et `prolog` sous MacOS, je vous recommande d'utiliser l'utilitaire `brew`. Vous entendrez peut-etre parler de `macport`, mais l'installation est plus compliquée avec cet outil.

Cliquez donc sur *Nouvelle* pour commencer. Vous devez nommer votre machine virtuelle (Fig. 7.2) : je conseille d'utiliser le nom du système d'exploitation avec le numéro de version, ça vous permettra par la suite de distinguer les différentes versions d'un même système si vous en installez plusieurs. Précisez bien le type du système installé (ici Linux) et sa version (ici un Ubuntu 64-bit).

Ensuite on vous demande la taille de la mémoire que la machine devra simuler. Ma machine dispose de 8Go de mémoire ce qui me permet d'en confier 2 à VirtualBox pour simuler la machine virtuelle (ce qui correspond à la mémoire recommandée pour faire fonctionner au mieux la version d'Ubuntu que j'ai choisie). Sur votre machine, vous ne disposez peut-être pas d'autant de mémoire. Notez la zone orangée puis rouge sous le curseur : elles correspondent à une quantité de mémoire qui ne permettrait plus au système hôte (votre système d'exploitation principal) de fonctionner correctement. Si vous confiez une telle quantité de mémoire à VirtualBox, votre ordinateur risque de ralentir dangereusement voire de planter. Restez donc dans la zone verte.

Enfin, indiquez comme sur la figure (Fig. 7.2) que vous souhaitez créer un disque dur virtuel. Cliquez sur *créer* pour passer à l'étape suivante.

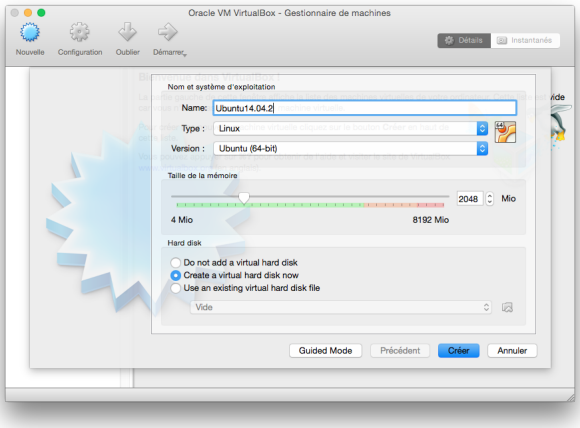


**Figure 7.2 – Création d'une machine virtuelle et choix de la taille de mémoire.**

Maintenant, VirtualBox va créer un disque dur virtuel (Fig. 7.3). Pour cela il va en fait créer un fichier qui sera placé sur votre vrai disque dur et qui contiendra le contenu du disque simulé. Vous devez choisir sa taille : sur la figure (Fig. 7.3) j'ai choisi environ 15Go.

Notez que j'ai sélectionné *Dynamiquement alloué*. Cela signifie que le fichier simulant le disque dur sera agrandi à mesure que j'ajouterai des données sur ce disque virtuel. Donc mon fichier atteindra au maximum une taille de 15Go et la machine virtuelle l'utilisant croira avoir affaire à un disque de 15Go mais le fichier n'occupera pas cette place sur mon disque si les informations réellement contenues dedans n'atteignent pas cette taille.

Attention, pour travailler confortablement sur votre Ubuntu (installer des programmes, stocker des données) prévoyez un espace supplémentaire par rapport à celui recommandé pour le fonctionnement de votre version d'Ubuntu. Par exemple, avec la version 16.04 de Ubuntu, je conseille de créer un disque de 40Go pour travailler confortablement.

Une fois cette étape passée, vous vous retrouvez de nouveau face à la fenêtre de départ mais cette fois, votre nouvelle machine virtuelle apparait dans la colonne de gauche.

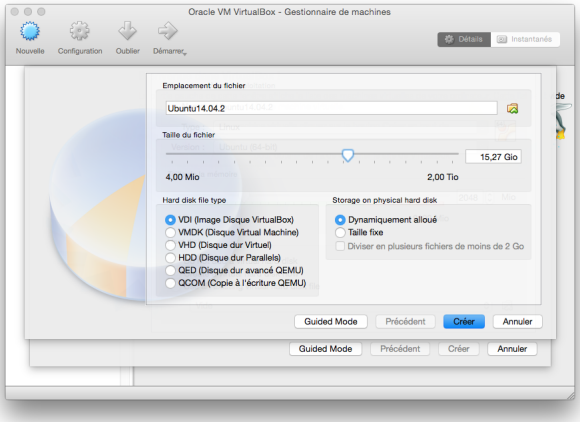


**Figure 7.3 – Création d'un disque dur virtuel.**

## 7.5 Installation d'un système sur votre machine virtuelle

Maintenant, nous avons une machine virtuelle prévue pour faire fonctionner le système d'exploitation que nous avons choisi, mais pour le moment cette machine est vide. Il va falloir procéder à l'installation du système.

Pour installer un système d'exploitation, il vous faut une copie de ce système. Si ce système est un système propriétaire comme Windows sachez qu'il vous faut un CD ou DVD d'installation et une licence légalement acquise pour être en accord avec la loi.

Dans notre cas, nous allons installer une version d'Ubuntu. Vous devriez avoir téléchargé une image ISO de Ubuntu (§. 2.2), si ce n'est pas le cas, faites le avant de passer à la suite. Nous allons demander à notre machine virtuelle d'utiliser directement cette image ISO pour simuler la présence d'un disque d'installation. Si vous avez gravé l'image sur un CD, ce qui n'était pas nécessaire, vous pourrez utiliser ce CD à la place.

Notre machine virtuelle dispose d'un lecteur de CD virtuel. En cliquant sur le titre du cadre *Stockage* nous pouvons accéder à une fenêtre de configuration (Fig. 7.4). Dans la fenêtre de gauche, cliquez en dessous de *Contrôleur IDE* pour choisir votre lecteur optique. Pour le moment il est surement vide. Cliquez sur le petit CD à droite et sur *Choose Virtual Optical Disk File...*. Vous pouvez alors sélectionner votre fichier ISO là où il se trouve. Si vous avez gravé le CD, placez-le dans le lecteur de votre machine (la vraie !) et sélectionner *Lecteur de l'hôte*.

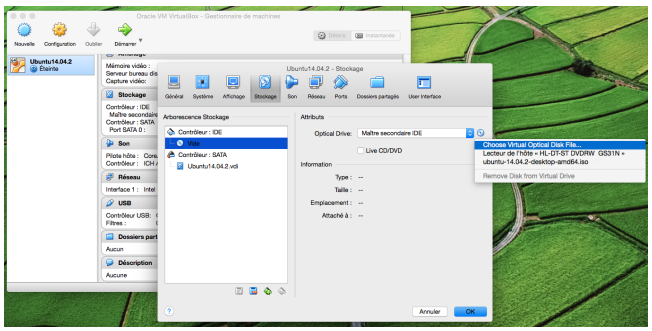


**Figure 7.4 – Utiliser une image ISO dans un lecteur CD virtuel.**

Maintenant votre machine virtuelle pense avoir un CD d'installation dans son lecteur virtuel. Dans la fenêtre principale vous devez constater que le contenu du cade *stockage* a changé et correspond

bien au réglage souhaité 7.5.

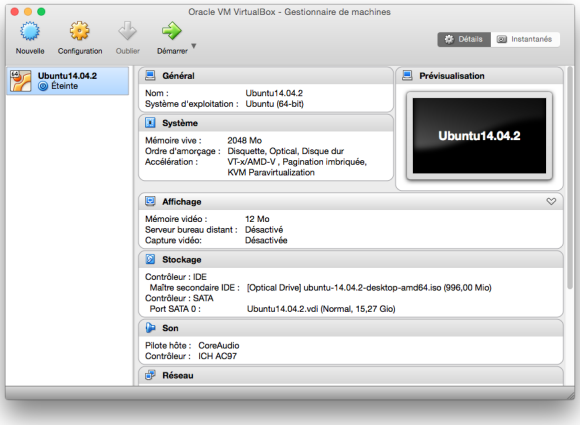


**Figure 7.5 – Après la configuration du lecteur optique virtuel. Le système est prêt à être installé.**

Il ne vous reste qu'à démarrer la machine virtuelle en cliquant dessus et à installer Ubuntu. Vous pouvez suivre les informations données dans le chapitre 2.5 durant l'installation.

## 7.6 Installation des Guest Additions

Vous avez installé Ubuntu sur votre machine virtuelle et redémarré (la machine virtuelle) comme demandé à la fin de l'installation pour utiliser le système installé et non le liveCD.

Vous constatez sans doute avec déception que Ubuntu (ou un autre système que vous avez installé) a démarré dans une toute petite fenêtre qui est loin d'occuper tout l'écran et si vous tentez d'agrandir cette fenêtre l'image reste désespérément fixée à une taille réduite. Pas d'inquiétude, c'est simplement parce que VirtualBox n'a pas encore installé tout ce qui est nécessaire (drivers) pour faire le lien entre votre machine réelle et la machine virtuelle. Les caractéristiques de votre écran notamment ne sont pas encore totalement reconnues.

Pour installer les éléments nécessaires à la prise en charge de votre matériel, allez dans le menu *Devices* et choisissez *Insert Guest Additions CD Image...*. Une image ISO d'un CD va alors être utilisée pour faire croire à votre machine virtuelle qu'un CD a été inséré dans son lecteur.

Si la boite de dialogue qui s'affiche dépasse la taille de la fenêtre (Fig. 7.6), déplacer la pour cliquer sur le bouton *Lancer*. Donner votre mot de passe (celui que vous avez choisi lors de l'installation) et attendez quelques instants que l'installation se fasse.

A la fin, un message vous demande d'appuyer sur la touche *entrer* pour fermer la fenêtre et achever l'installation (Fig. 7.7). Pour être sur que les nouveaux drivers soient pris en compte, redémarrez Ubuntu.

## 7.7 Fonctions utiles

Maintenant que les *Guest Additions* sont installés vous allez pouvoir redimensionner la fenêtre dans laquelle Ubuntu apparaît et voir l'affichage s'ajuster en fonction. Pour cela, assurez vous que

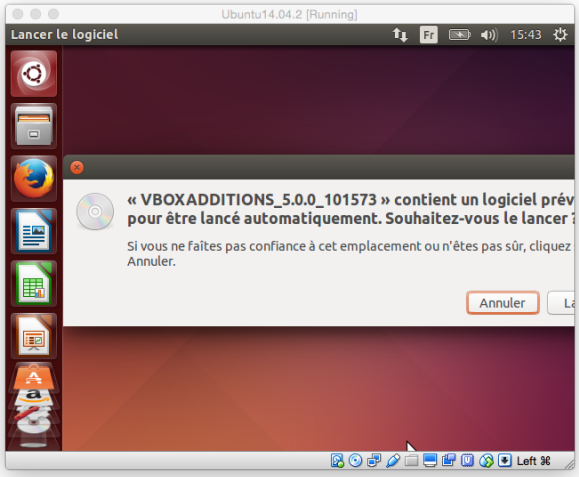


**Figure 7.6 – Une fenêtre s'affiche lors de la détection du média virtuel. Exemple d'un cas où la boite de dialogue dépasse la taille de la fenêtre.**

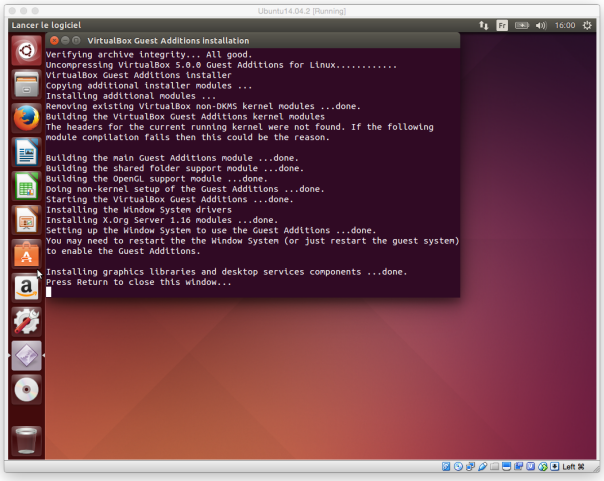


**Figure 7.7 – Fin de l'installation des Guest Addition.**

dans le menu *View*, l'option *Auto-Resize Guest Display* soit bien cochée (rien d'autre ne doit être coché pour le moment).

Pour copier ou glisser facilement des éléments de votre système hôte à Ubuntu et inversement, vous pouvez activer les options *Shared Clipboard/bidirectional* et *Drag and Drop/bidirectional* dans le menu *Device*.

Pour afficher Ubuntu en plein écran vous pouvez choisir *Full-screen Mode* dans le menu *View*. Pour sortir de ce mode il suffit d'amener la souris dans le haut de l'écran pour faire réapparaitre le menu et décocher la case.

## 7.8 Rendre un dossier de votre machine hôte accessible dans la VM

Une chose que vous pourriez avoir envie de faire est d'accéder directement au contenu des dossiers de votre machine hôte (votre Mac) sur votre machine virtuelle. Pour cela commencez par éteindre votre machine virtuelle. Dans la fenêtre d'accueil de Virtualbox, cliquez sur le titre du cadre *Dossiers Partagés* de votre machine virtuelle.

Dans la fenêtre qui s'affiche (Fig. 7.8), cliquez sur la petite icône représentant un dossier à droite du cadre. Dans la fenêtre qui s'affiche, indiquer le chemin du dossier sur votre machine hôte (ou utiliser le bouton contre le cadre pour aller le rechercher sur votre machine). Indiquer le nom que vous souhaitez donner à ce dossier sur votre machine virtuelle et cocher la case *montage automatique* pour qu'il apparaisse automatiquement sur votre machine virtuelle. Recommencer autant de fois que nécessaire pour ajouter tous les dossiers auxquels vous souhaitez accéder.

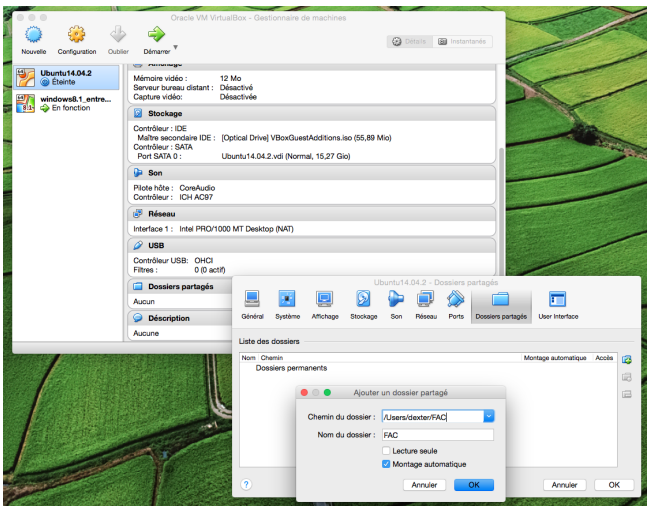


**Figure 7.8 – Ajout d'un dossier partagé entre la machine hote et la machine virtuelle.**

Démarrez à nouveau votre machine virtuelle. Ouvrez le terminal et allez dans le dossier `/media` (§. 5.1.1). Lister le contenu du dossier : vous voyez le nom de votre dossier avec le prefixe `sf_`. Cependant, si vous tenter d'aller dans ce dossier vous vous heurtez à un très désobligeant « *Permission non accordée* ».

En effet, vous n'avez pas les droits d'accès pour ce dossier puisqu'il n'est pas dans votre *home* dans Ubuntu... Regardez le groupe auquel appartient le dossier (§. 6.1.1) et ajouter vous dans ce groupe. Il peut être nécessaire de redémarrer la machine virtuelle pour que ce changement soit pris en compte. Maintenant vous pouvez accéder à votre dossier. Il ne vous reste qu'à créer un lien vers ce dossier là où vous le souhaitez pour y accéder rapidement (§. 5.1.4).

# 8 Conclusion

Arrivé à la fin de ce cours, vous devez avoir installé Ubuntu que vous soyez équipé d'un PC ou d'un Macintosh. Vous devez avoir trouvé vos repères sous Ubuntu autant en utilisant l'interface graphique que la console. Vous devez être en mesure de lire et comprendre un script shell et d'utiliser des commandes dans la console pour réaliser les tâches les plus courantes.

Ce cours vous a donné un aperçu des commandes les plus fréquentes, mais ne fait qu'effleurer l'étendue de ce qu'il est possible de faire avec. Pour une utilisation plus spécifique ou plus experte de ces commandes et pour en découvrir d'autres, je vous recommande une fois de plus de consulter les pages du manuel.

Certaines tâches comme renommer un lot de fichiers, sont bien plus rapides à faire en lignes de commandes. Cependant, quand on débute, on peut-être tenté de faire cette tâche répétitive en utilisant l'interface graphique plutôt que de faire l'effort d'une recherche dans le manuel pour savoir quelle commande utiliser. Résistez à ce penchant et prenez le temps de rechercher les commandes et de les utiliser. Copiez cette commande dans un carnet, sur un post-it ou tout autre support qui vous convient pour vous en souvenir plus rapidement la fois suivante.

Ce n'est qu'en faisant cet effort que vous pourrez maitriser toute la puissance des commandes textes et vous serez à l'aise sur n'importe quel système Unix.

# 9 GNU Free Documentation License

Version 1.3, 3 November 2008



Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom : to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation : a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals ; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties : any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License

applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts : Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version :

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another ; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History" ; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must

delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate. 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http ://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.