

**Manuel de référence**

**Mandrake Linux 9.1**



<http://www.MandrakeSoft.com>

## Manuel de référence: Mandrake Linux 9.1

Publié 2003-03-24

Copyright © 2003 MandrakeSoft SA

par Camille Bégnis, Christian Roy, Fabian Mandelbaum, Joël Pomerleau, Vincent Danen, Roberto Rosselli del Turco, Stefan Siegel, Marco De Vitis, Alice Lafox, Kevin Lecouvey, Christian Georges, John Rye, Robert Kulagowski, Pascal Rigaux, Frédéric Crozat, Laurent Montel, Damien Chaumette, Till Kamppeter, Guillaume Cottenceau, Jonathan Gotti, Christian Belisle, Sylvestre Taburet, Thierry Vignaud, Juan Quintela, Pascal Lo Re, Kadjo N'Doua, Mark Walker, Roberto Patriarca, Patricia Pichardo Bégnis, Alexis Gilliot, Arnaud Desmons, Wolfgang Bornath, Alessandro Baretta, Aurélien Lemaire, Daouda Lo, Florent Villard, Gwenole Beauchesne, Giuseppe Ghibò, Joël Wardenski, et Debora Rejnharc Mandelbaum

## Notice légale

Ce manuel est la propriété intellectuelle de **MandrakeSoft**. Il peut être librement copié, distribué et/ou modifié selon les termes de la Licence GNU *Free Documentation License*, Version 1.1 ou ultérieure publiée par la Free Software Foundation ; avec la section inaltérable *Au sujet de Mandrake Linux*, page i ; les textes de couverture, cités ci-dessous, et sans texte de quatrième de couverture. Une copie complète de la licence se trouve sur le site de GNU (<http://www.gnu.org/licenses/fdl.html>).

Textes de couverture :

MandrakeSoft, mars 2003

<http://www.mandrakesoft.com/>

Copyright © 1999, 2000, 2001, 2002, 2003 MandrakeSoft S.A.  
et MandrakeSoft inc.

« Mandrake », « Mandrake Linux » et « MandrakeSoft » sont des marques déposées par **MandrakeSoft S.A.** ; Linux est une marque déposée de **Linus Torvalds** ; *UNIX* est une marque déposée de « *The Open Group* » aux États-Unis et dans d'autres pays. Toutes les autres marques déposées et copyrights appartiennent à leurs propriétaires respectifs.

## Outils utilisés dans la conception de ce manuel

Ce manuel a été rédigé avec la grammaire *DocBook XML*. *Borges* (<http://linux-mandrake.com/en/doc/project/Borges/>) a été utilisé pour gérer l'ensemble des fichiers. Les fichiers source XML ont été préformatés avec *openjade* et *jadetex* avec l'aide des feuilles de style personnalisées de Norman Walsh. Les images ont été prises avec *xwd* et *GIMP*, puis converties avec la commande *convert* du paquetage *ImageMagick*. Tous ces logiciels sont libres et disponibles sur votre distribution **Mandrake Linux**.

# Table des matières

Préface .....	i
1. Au sujet de Mandrake Linux .....	i
1.1. Contacter la communauté Mandrake .....	i
1.2. Encouragez Mandrake Linux .....	i
1.3. Contribuer à Mandrake Linux .....	ii
1.4. Acquisition de produits Mandrake .....	ii
2. Note des traducteurs .....	ii
3. Conventions utilisées dans ce manuel .....	ii
3.1. Conventions typographiques .....	ii
3.2. Conventions générales .....	iii
<b>I. Le Système Linux .....</b>	<b>1</b>
1. Concepts UNIX de base .....	1
1.1. Utilisateurs et groupes .....	1
1.2. Notions de base au sujet des fichiers .....	2
1.3. Les processus .....	4
1.4. Petite introduction à la ligne de commande .....	4
2. Disques et partitions .....	9
2.1. Structure d'un disque dur .....	9
2.2. Conventions pour nommer disques et partitions .....	11
3. Introduction à la ligne de commande .....	13
3.1. Utilitaires de manipulation de fichiers .....	13
3.2. Manipulation des attributs de fichiers .....	15
3.3. Motifs d'englobement du shell .....	17
3.4. Redirections et tubes .....	17
3.5. Le complètement ( <i>completion</i> ) dans les lignes de commande .....	19
3.6. Lancement et manipulation de processus en arrière-plan .....	20
3.7. Le mot de la fin .....	21
4. L'édition de texte : Emacs et VI .....	23
4.1. Emacs .....	23
4.2. Vi : l'ancêtre .....	26
4.3. Un dernier mot.....	30
5. Les utilitaires en ligne de commande .....	31
5.1. Opérations sur les fichiers et filtres .....	31
5.2. find : rechercher des fichiers selon certains critères .....	35
5.3. Programmation de démarrage de commandes .....	37
5.4. at : programmer une commande une seule fois .....	38
5.5. Archivage et compression de données .....	38
5.6. Et plus encore.....	41
6. Contrôle des processus .....	43
6.1. Un mot encore sur les processus .....	43
6.2. Obtenir des informations sur les processus : ps et pstree .....	43
6.3. Envoyer des signaux aux processus : kill, killall, top .....	44
6.4. Contrôler la priorité des processus : nice, renice .....	45
<b>II. Linux en profondeur .....</b>	<b>47</b>
7. Organisation de l'arborescence des fichiers .....	47
7.1. Données partagées et non partagées, statiques et dynamiques .....	47
7.2. Le répertoire racine : / .....	47
7.3. /usr : le gros morceau .....	48
7.4. /var : données modifiables en cours d'utilisation .....	48
7.5. /etc : les fichiers de configuration .....	49
8. Systèmes de fichiers et points de montage .....	51
8.1. Principes .....	51
8.2. Partitionner un disque dur, formater une partition .....	52
8.3. Les commandes mount et umount .....	52
8.4. Le fichier /etc/fstab .....	53
8.5. La fonction supermount .....	54
9. Le système de fichiers Linux .....	55
9.1. Comparatif de quelques systèmes de fichiers .....	55

9.2. Tout est fichier .....	56
9.3. Les liens .....	58
9.4. Tubes "anonymes" et tubes nommés .....	59
9.5. Les fichiers spéciaux : fichiers mode bloc et caractère .....	60
9.6. Les liens symboliques et la limitation des liens en dur .....	61
9.7. Les attributs des fichiers .....	62
10. Le système de fichiers /proc .....	63
10.1. Renseignements sur les processus .....	63
10.2. Informations sur le matériel .....	64
10.3. Le sous-répertoire /proc/sys .....	66
11. Les fichiers de démarrage : init sysv .....	67
11.1. Au commencement était init .....	67
11.2. Les niveaux d'exécution .....	67
<b>III. Utilisations avancées .....</b>	<b>69</b>
12. Installation d'un logiciel libre .....	69
12.1. Introduction .....	69
12.2. Décompactage .....	71
12.3. Configuration .....	73
12.4. Compilation .....	75
12.5. Installation .....	81
12.6. Assistance .....	81
12.7. Remerciements .....	83
13. Compilation et mise en place de nouveaux noyaux .....	85
13.1. Où trouver les sources du noyau .....	85
13.2. Décompactage des sources du noyau, correction éventuelle du noyau .....	86
13.3. Configuration du noyau .....	86
13.4. Sauvegarder et réutiliser vos fichiers de configuration du noyau .....	87
13.5. Compilation et installation des modules .....	88
13.6. Installation du nouveau noyau .....	89
<b>A. La licence Publique Générale GNU (GPL) .....</b>	<b>95</b>
A.1. Introduction .....	95
A.2. Préambule .....	95
A.3. Stipulations et conditions relatives à la copie, la distribution et la modification .....	96
<b>B. Licence de documentation libre GNU .....</b>	<b>99</b>
B.1. A propos de cette traduction française de la GFDL .....	99
B.2. Licence de documentation libre GNU .....	99
0. PRÉAMBULE .....	99
1. APPLICABILITÉ ET DÉFINITIONS .....	99
2. COPIES CONFORMES .....	100
3. COPIES EN NOMBRE .....	100
4. MODIFICATIONS .....	101
5. FUSION DE DOCUMENTS .....	102
6. REGROUPEMENTS DE DOCUMENTS .....	102
7. AGRÉGATION AVEC DES TRAVAUX INDÉPENDANTS .....	102
8. TRADUCTION .....	102
9. CADUCITÉ .....	103
10. RÉVISIONS FUTURES DE CETTE LICENCE .....	103
B.3. Comment utiliser cette Licence pour vos documents .....	103
<b>C. Glossaire .....</b>	<b>105</b>
.....	

## Liste des tableaux

9-1. Caractéristiques des systèmes de fichiers .....	56
--	----

## Liste des illustrations

1-1. Connexion en mode graphique .....	1
1-2. Connexion en mode console .....	2
1-3. L'icône de l'émulateur de terminal sur le tableau de bord de KDE .....	5
2-1. Premier exemple de noms de partitions sous <i>GNU/Linux</i> .....	11
2-2. Second exemple de noms de partitions sous <i>GNU/Linux</i> .....	11
4-1. Emacs; : édition simultanée de deux fichiers .....	23
4-2. <i>Emacs</i> , avant la copie du bloc de texte .....	24
4-3. Emacs, après la copie du bloc de texte .....	25
4-4. Situation de départ dans vim .....	26
4-5. vim, avant la copie du bloc de texte .....	28
4-6. vim, après la copie du bloc de texte .....	29
6-1. Exemple d'exécution de top .....	44
8-1. Avant le montage d'un système de fichiers .....	51
8-2. Après le montage du système de fichiers .....	51



# Préface

## 1. Au sujet de Mandrake Linux

**Mandrake Linux** est une distribution *GNU/Linux* développée par **MandrakeSoft** S.A. La société **MandrakeSoft** est née sur Internet en 1998 ; son ambition première demeure de fournir un système *GNU/Linux* convivial et facile à utiliser. Les deux piliers de **MandrakeSoft** sont le logiciel libre et le travail coopératif.

### 1.1. Contacter la communauté Mandrake

Nous présentons ci-dessous plusieurs liens Internet pointant vers de nombreuses ressources liées à **Mandrake Linux**. Si vous souhaitez en savoir plus sur la société **MandrakeSoft**, consultez son site Web (<http://www.mandrakesoft.com/>). Un site dédié à la distribution **Mandrake Linux** (<http://www.mandrakelinux.com/>) et tous ses dérivés existe également.

Premièrement, permettez-nous de vous présenter notre plate-forme d'aide libre. MandrakeExpert (<http://www.mandrakeexpert.com/>) n'est pas un autre site Web où les gens en aident d'autres pour régler leurs problèmes informatiques en échange d'un prix payable d'avance, quelle que soit la qualité du service rendu. Il propose une nouvelle expérience s'appuyant sur la confiance et le plaisir de récompenser son prochain pour sa contribution.

Vous êtes également invité à participer aux nombreuses listes de diffusion (<http://www.mandrake-inux.com/fr/flists.php3>), où la communauté **Mandrake Linux** déploie tout son enthousiasme et sa vivacité.

Finalement, n'oubliez pas de vous connecter sur MandrakeSecure (<http://www.mandrakesecure.net/>). Ce site rassemble tout ce qui traite de la sécurité des distributions **Mandrake Linux**. Notamment, vous y trouverez des avertissements de bogues et de sécurité, ainsi que des articles traitant de sécurité informatique et de privacité. Bref, voilà un site incontournable pour tout administrateur système, ou le simple utilisateur soucieux de sécurité.

### 1.2. Encouragez Mandrake Linux

À la demande générale, **MandrakeSoft** propose à ses utilisateurs de participer au financement du succès de la société. À travers le Club des utilisateurs de Mandrake Linux (<http://www.mandrakelinux.com/fr/club/>) et le Mandrake Corporate Club (<http://www.mandrakelinux.com/corporateclub>), vous pouvez :

- télécharger des logiciels commerciaux, qui ne sont normalement disponibles que dans les packs de détail, tels que des pilotes logiciel, des applications commerciales, des partagiciels et des versions démo ;
- voter et proposer des nouveaux logiciels à travers un système de vote RPM que des bénévoles opèrent ;
- obtenir des remises pour des produits et services sur le MandrakeStore (<http://www.mandrakestore.com>) ;
- accéder à des offres spéciales MandrakeOnline incluant des remises, des comptes supplémentaires gratuits pour les membres niveau Gold (et plus élevé), ainsi que l'élimination de la pub !
- obtenir une copie de StarOffice 6.0, disponible aux membres de niveau Silver et plus élevé ;
- accéder à une meilleure liste de miroirs, exclusive aux membres du Club (**expérimental**) ;
- lire des forums et articles multilingues.

Le MandrakeClub est l'endroit idéal pour vous faire entendre !

En finançant **MandrakeSoft** par l'entremise du MandrakeClub, vous améliorerez directement la distribution **Mandrake Linux** et vous nous permettrez de proposer le meilleur poste de travail *GNU/Linux* possible à nos utilisateurs.

### 1.3. Contribuer à Mandrake Linux

Quels que soient vos habiletés, vous êtes encouragé à participer à l'une des nombreuses tâches que requise à la construction du système **Mandrake Linux** :

- paquetages : un système *GNU/Linux* est principalement constitué de programmes rassemblés depuis Internet. Ils doivent être mis en forme de façon à ce qu'ils puissent fonctionner ensemble, si tout se passe bien ;
- programmation : une foule de projets sont directement développés par **MandrakeSoft** : trouvez celui qui vous intéresse le plus et proposez votre aide au développeur principal ;
- internationalisation : vous pouvez nous aider à traduire des pages de nos sites Web, des programmes et leur documentation respective.
- documentation : afin de suivre l'évolution rapide du système, nous ne comptons plus le temps et les efforts investis pour que le manuel que vous êtes en train de lire demeure à jour. Rédacteurs et traducteurs seront accueillis à bras ouverts !

Consultez aussi les pages des contributeurs (<http://www.mandrakesoft.com/labs/>) pour en savoir plus sur les différentes façons de contribuer à l'évolution de **Mandrake Linux**.

### 1.4. Acquisition de produits Mandrake

Pour les fans de **Mandrake Linux** désirant acheter en ligne, **MandrakeSoft** vend désormais ses produits dans le monde entier depuis le MandrakeStore (<http://www.mandrakestore.com>). Vous n'y trouverez pas **que** des logiciels **Mandrake Linux** – systèmes d'exploitation et outils réseau (pare-feu) (*Multi Network Firewall*), mais aussi des offres spéciales d'abonnement, de l'assistance, des logiciels tiers et des licences, des manuels et des livres *GNU/Linux*, ainsi que des tee-shirts, casquettes, et autres gadgets **MandrakeSoft**.

## 2. Note des traducteurs

En passant d'un chapitre à l'autre, vous remarquerez que ce livre est une œuvre composite. Même si la cohérence technique et lexicale a été notre priorité, vous remarquerez que le style spécifique de chaque auteur (anglophone pour la plupart) a été quelque peu préservé.

Garder cette documentation à jour est un véritable défi ! Choisir le bon mot n'est pas toujours facile puisque la plupart des termes informatiques utilisés par les professionnels de par le monde demeurent en fait des mots de la langue anglaise. Inventer un terme français plus clair pour le novice risque alors de dérouter l'expert ! Que faire ? On aura en général opté pour un compromis : donner les deux termes, anglais et français, lors de la première occurrence textuelle de la notion, puis un seul par la suite, afin de ne pas entraver la lecture. N'hésitez pas à nous donner votre avis à ce sujet.

Car enfin, dans le droit fil de l'esprit particulier de la communauté du libre (*open source*), nous accueillons les collaborations à bras ouverts ! Il vous est tout à fait possible d'apporter votre petite pierre à l'édification de ce projet de diverses façons. Vous disposez de temps libre ? Proposez-nous un nouveau chapitre ! Vous comprenez bien l'anglais ? Aidez-nous à traduire ! Vous avez des idées pour améliorer ce document ? Nous sommes tout ouïe !

Pour toute information au sujet du projet de documentation de **Mandrake Linux**, communiquez avec le responsable de la documentation ([documentation@mandrakesoft.com](mailto:documentation@mandrakesoft.com)) ou visitez notre site Web (<http://www.linux-mandrake.com/en/doc/project>) (en anglais seulement).

## 3. Conventions utilisées dans ce manuel

### 3.1. Conventions typographiques

Afin d'accentuer clairement certains mots ou groupes de mots, nous avons utilisé certains attributs typographiques. Le tableau suivant en donne la signification symbolique :



Exemple formaté	Signification
<i>inœud</i>	Signale un terme technique.
<code>ls -lta</code>	Types utilisés pour une commande et ses arguments, les options et les noms de fichier (voir la section <i>Synopsis d'une commande</i> , page iii).
<code>ls(1)</code>	Référence vers une page de manuel (aussi appelée page de man). Pour consulter la page correspondante dans un <i>shell</i> (ou ligne de commande), exécutez la commande <code>man 1 ls</code> .
<code>\$ ls *.pid</code>	Ce style est utilisé pour une copie d'écran texte. Signale une interaction utilisateur-ordinateur, le code source d'un programme, etc.
<code>localhost</code>	Données littérales qui ne correspondent généralement pas à une des catégories précédemment définies : citation d'un mot clé tiré d'un fichier de configuration, par exemple.
<i>Apache</i>	Nom des applications. Notre exemple (« Apache ») n'est pas un nom de commande. Toutefois, dans des contextes particuliers, l'application et le nom de la commande peuvent être les mêmes, mais formatés de façon différente.
<code>C onfigurer</code>	Entrée de menu ou label des interfaces graphiques, en général. La lettre soulignée indique le raccourci clavier éventuel.
<i>Bus SCSI</i>	Partie d'un ordinateur ou ordinateur lui-même.
<i>Once upon a time...</i>	Citation en langue étrangère.
<b>Attention !</b>	Types réservés pour les mots que nous voulons accentuer. Lisez-les à voix haute :-)



Cette icône introduit une note. Il s'agit généralement d'une remarque dans le contexte courant, pour donner une information supplémentaire.



Cette icône introduit une astuce. Il peut s'agir d'un conseil d'ordre général sur la meilleure façon d'arriver à un but spécifique, ou une fonctionnalité intéressante qui peut vous rendre la vie plus facile.



Soyez très attentif lorsque vous rencontrez cette icône. Il s'agit toujours d'informations très importantes sur le sujet en cours de discussion.

## 3.2. Conventions générales

### 3.2.1. Synopsis d'une commande

L'exemple ci-dessous présente les différents signes et symboles que vous rencontrerez lorsque nous décrivons les arguments d'une commande :

```
command <argument non littéral> [--option={arg1,arg2,arg3}]  
[argument optionnel...]
```

Ces conventions étant standardisées, vous les retrouverez en bien d'autres occasions (dans les pages de man, par exemple).

Les signes « < » (inférieur) et « > » (supérieur) indiquent un argument **obligatoire** qui ne doit pas être copié tel quel mais remplacé par votre texte spécifique. Par exemple : <fichier> désigne le nom d'un fichier ; si ce fichier est toto.txt, vous devrez taper toto.txt, et non <toto.txt> ou <fichier>.

Les crochets « [ ] » indiquent des arguments optionnels que vous déciderez ou non d'inclure dans la ligne de commande.

Les points de suspension (« ... ») signifient qu'un nombre illimité d'options peut être inséré à cet endroit.

Les accolades (« { } ») contiennent les arguments autorisés à cet endroit. Il faudra obligatoirement insérer un d'entre eux à cet endroit précis.

### 3.2.2. Notations particulières

De temps à autre, il vous sera demandé de presser les touches Ctrl+R. Cela signifie que vous devez taper et maintenir la touche Ctrl enfoncée pendant que vous appuyez sur la touche R. Le même chose se rencontre pour les touches Alt et Shift.

De même, à propos des menus, aller sur l'entrée de menu Fichier→Relire la configuration utilisateur (**Ctrl+R**) signifie : cliquez sur le label Fichier du menu (généralement en haut et à gauche de la fenêtre) puis sur le menu vertical qui apparaît, cliquez sur Relire la configuration utilisateur. De plus, vous pouvez également utiliser la combinaison de touches Ctrl+R, comme décrit ci-dessus pour arriver au même résultat.

### 3.2.3. Utilisateurs système génériques

À chaque fois que cela est possible, nous utiliserons deux utilisateurs génériques dans nos exemples :

Reine Pingusa	Cet utilisateur est créé pendant l'installation.
Pierre Pingus	Cet utilisateur est ensuite créé par l'administrateur système.

## Introduction

Bienvenue, et merci d'utiliser **Mandrake Linux** ! Ce manuel est destiné à ceux et celles qui désirent plonger dans les méandres de leur système *GNU/Linux*, et qui veulent bénéficier au maximum de son potentiel. Ce manuel est constitué de trois parties :

- Dans la partie intitulée *Le Système Linux*, nous traitons de l'utilisation de la ligne de commande et de ses nombreuses possibilités. Nous discutons également des bases de l'édition de texte, un concept essentiel sous *GNU/Linux*.

Le chapitre *Concepts UNIX de base*, page 1 présente les mondes *UNIX* et plus particulièrement, *GNU/Linux*. C'est une introduction aux outils standards utilisés pour manipuler les fichiers et certaines fonctionnalités utiles du *shell*. Il est nécessaire de bien comprendre les concepts qui y sont présentés avant de passer au chapitre *Introduction à la ligne de commande*, page 13. Puis, *Disques et partitions*, page 9 aborde le système de gestion des disques durs sous *GNU/Linux*, ainsi que le concept de partition.

Puis suit *L'édition de texte : Emacs et Vi*, page 23. Comme la plupart des fichiers de configuration sont en format texte sous *UNIX*, vous aurez sûrement besoin de les modifier avec un *éditeur de texte*. Vous apprendrez comment utiliser deux des plus célèbres éditeurs du monde *UNIX* : le puissant *Emacs* et le vénérable *Vi*.

Maintenant, vous devriez pouvoir mener à bien quelques tâches d'entretien de base sur votre système. Les deux chapitres qui suivent présentent des utilisations pratiques de la ligne de commande (*Les utilitaires en ligne de commande*, page 31) et le contrôle des processus en général (*Contrôle des processus*, page 43).

- Dans *Linux en profondeur*, nous discutons brièvement du noyau *Linux* et de l'architecture du système de fichiers.

Le chapitre *Organisation de l'arborescence des fichiers*, page 47 aborde l'organisation du système de fichiers. Les systèmes *UNIX* tendent à grossir énormément, mais chaque fichier a sa place dans un répertoire spécifique. Après avoir lu ce chapitre, vous saurez où chercher les fichiers en fonction de leur rôle à même le système.

Ensuite, nous traitons de deux sujets, soit le *système de fichiers* et les *points de montage* (*Systèmes de fichiers et points de montage*, page 51). Vous apprendrez alors ce que signifient ces deux termes et vous en verrez des exemples pratiques.

Nous poursuivons avec le chapitre *Le système de fichiers Linux*, page 55. Après une présentation des systèmes de fichiers disponibles, vous en apprendrez plus au sujet des types de fichiers et autres concepts, comme les *i-nœuds* et les *tubes*. *Le système de fichiers /proc*, page 63, introduit pour sa part un système de fichiers bien particulier sous *GNU/Linux* : */proc*.

Dans *Les fichiers de démarrage : init sysv*, page 67, nous présentons la procédure de démarrage de **Mandrake Linux** et comment l'utiliser efficacement.

- Finalement, la partie *Utilisations avancées* traite de sujets que seuls les téméraires ou les lecteurs expérimentés voudront mettre en pratique. *Installation d'un logiciel libre*, page 69, vous guidera à travers les étapes nécessaires pour construire et installer des logiciels libres depuis leur source. La lecture de ce chapitre devrait vous encourager à vous faire la main, même si, à première vue, ça peut paraître intimidant. Finalement, *Compilation et mise en place de nouveaux noyaux*, page 85, s'avère une des dernières étapes vers une autonomie totale sous *GNU/Linux*. Après avoir lu et appliqué la théorie expliquée dans ce chapitre, commencez à convertir des utilisateurs *Windows* (si ce n'est déjà fait !).

Le livre se termine par deux annexes, soit les deux licences principalement utilisées pour les applications *GNU/Linux* et la documentation, respectivement : *La licence Publique Générale GNU (GPL)*, page 95 et *Licence de documentation libre GNU*, page 99. Un *Glossaire*, page 105 et un *index* concluent cet ouvrage.



# Chapitre 1. Concepts UNIX de base

Le nom « *UNIX* » dira quelque chose à certains d'entre vous. Peut-être même utilisez-vous un système *UNIX* dans le cadre de votre travail, auquel cas la lecture de ce chapitre ne vous apprendra pas grand-chose.

Pour ceux et celles d'entre vous qui ne l'ont jamais utilisé, la lecture de ce chapitre est nécessaire ! La connaissance des concepts que nous allons présenter ici répondra à un nombre surprenant de questions que se posent les débutants dans le monde GNU/Linux . De même, il est fort probable que ces seuls concepts vous donnent des pistes de recherche sur les causes d'un problème que vous pourriez rencontrer.

## 1.1. Utilisateurs et groupes

Cette notion est sans doute la plus importante, car elle a une influence directe sur tous les autres concepts que nous allons introduire dans ce chapitre.

*Linux* est un véritable système *multiutilisateurs*, et pour pouvoir se connecter à un système *GNU/Linux*, il faut avoir un *compte* sur ce système. Quand vous avez créé des utilisateurs lors de l'installation, vous avez en fait ajouté des comptes utilisateur. Vous vous souvenez sans doute que la création d'un compte a exigé que vous entriez, entre autres, les éléments suivants :

- le « vrai nom » de l'utilisateur (en fait, ce que vous voulez) ;
- un *nom de connexion* ;
- un *mot de passe*.

Les deux paramètres importants ici sont le nom de connexion (très souvent appelé nom de login) et le mot de passe. Ce sont en effet eux que vous devrez utiliser pour vous connecter au système.

Une autre action effectuée parallèlement à l'ajout d'un utilisateur est la création d'un groupe. Comme nous le verrons plus loin, les groupes sont utiles dans le cadre de partage de fichiers entre différentes personnes. Un groupe peut contenir autant d'utilisateurs que vous le souhaitez, et il est très courant de voir une telle séparation dans les gros systèmes. Dans une université, par exemple, vous pouvez avoir un groupe par département, un autre pour les professeurs et ainsi de suite. L'inverse est également vrai : un utilisateur peut être membre d'un ou de plusieurs groupes, la limite étant de trente-deux groupes. Un professeur de mathématiques, par exemple, peut être membre du groupe des professeurs et également membre du groupe de ses étudiants.

Cela ne vous dit toujours pas comment vous connecter. On y arrive.

Si vous avez choisi d'avoir l'interface graphique au démarrage, votre fenêtre de connexion sera similaire à la figure 1-1.



Figure 1-1. Connexion en mode graphique

Pour vous connecter, vous devez d'abord sélectionner votre compte dans la liste. Un nouveau dialogue apparaît pour rentrer le mot de passe. Notez que vous devrez taper ce mot de passe à l'aveugle, car chaque caractère sera représenté par une étoile \* à l'écran. Vous pouvez aussi choisir votre type de session selon vos préférences. Pressez enfin sur le bouton Connexion.

Si vous êtes en mode console, vous obtiendrez une image similaire à celle de la figure 1-2.

```
Mandrake Linux release 9.1 (Cooker) for i586
Kernel 2.4.19-24mdk on an i686 / tty1
localhost login:
```

Figure 1-2. Connexion en mode console

Tapez votre nom de connexion à l'invite Login:, suivi d'une pression sur la touche **Entrée**. Ensuite, le programme de connexion (appelé, comme par hasard, *login*) vous présentera une invite Password:, et vous vous exécuterez en entrant le mot de passe de ce compte (toujours à l'aveugle !).

Notez que vous pouvez vous connecter plusieurs fois sous le même nom d'utilisateur, par exemple sur des *consoles* additionnelles et sous *X*. Chaque session que vous ouvrirez sera indépendante, et il est même possible d'ouvrir plusieurs sessions *X* simultanément. Par défaut, **Mandrake Linux** dispose de six *consoles virtuelles*, en plus de celle réservée à l'interface graphique. Vous pouvez basculer de l'une à l'autre en tapant la séquence de touches **Ctrl-Alt-F<n>**, où **<n>** représente le numéro de la console vers laquelle vous voulez vous diriger. En général, l'interface graphique est sur la console numéro 7. Ainsi pour vous rendre sur la seconde console vous presserez les touches Ctrl, Alt et F2 simultanément.

Lors de l'installation, *DrakX* vous a demandé d'entrer un mot de passe pour un utilisateur bien particulier : root. Cet utilisateur est spécial pour une raison bien simple : root est l'administrateur du système, donc très probablement vous. Pour la sécurité de votre système, il est très important que le compte root soit toujours protégé par un bon mot de passe !

Si vous vous connectez régulièrement en tant que root, il est très facile de faire une erreur qui pourrait rendre votre système inutilisable. Une seule mauvaise manipulation peut suffire. En particulier, si vous n'avez pas mis de mot de passe à ce compte, n'importe qui peut altérer votre système (y compris d'autres systèmes d'exploitation sur votre machine !). Ce qui, évidemment, peut s'avérer fort ennuyeux.

Enfin, il est bon de mentionner qu'en interne, le système ne vous identifie pas par votre nom de connexion, mais par un numéro unique associé à votre nom de connexion : un UID (*User ID*, soit un **identifiant utilisateur**). De même, chaque groupe est identifié par son **identifiant de groupe** ou GID (*Group ID*).

## 1.2. Notions de base au sujet des fichiers

Les fichiers sont un autre domaine où *GNU/Linux* diffère totalement de *Windows* et de la plupart des *systèmes d'exploitation*. Nous ne verrons ici que les différences visibles immédiatement. Si vous le souhaitez, vous pouvez lire le chapitre *Le système de fichiers Linux*, page 55, qui approfondit ce sujet.

La première différence est une conséquence directe du fait que *GNU/Linux* est un système multiutilisateurs : chaque fichier est la propriété exclusive d'un utilisateur et d'un groupe. Un peu plus tôt, nous avons parlé des utilisateurs, mais une chose que nous n'avons pas mentionné est que chaque utilisateur dispose de son propre répertoire (appelé son répertoire personnel, à savoir son *home directory* en anglais). Il est le propriétaire de ce répertoire, ainsi que de tous les fichiers qu'il y créera par la suite. Lui, et personne d'autre.

Cependant, la notion de propriété d'un fichier, prise seule, ne servirait pas à grand-chose. Mais il y a plus : en tant que propriétaire d'un fichier, un utilisateur peut établir des **droits** sur ce fichier. Ces droits distinguent trois catégories d'utilisateurs : le propriétaire du fichier, tout utilisateur qui est membre du groupe propriétaire associé au fichier (appelé le *groupe propriétaire*) mais n'est pas le propriétaire lui-même, et les autres, catégorie qui regroupe tout utilisateur qui n'est ni le propriétaire, ni membre du groupe propriétaire.

On distingue trois types de droits :

1. Droit de lecture (*r* pour *Read* ou lire). Sur un fichier, cela autorise la lecture de son contenu. Pour un répertoire, cela autorise son contenu (c'est-à-dire les fichiers qu'il contient) à être listé.
2. Droit d'écriture (*w* pour *Write* ou écrire). Pour un fichier, cela autorise la modification de son contenu. Pour un répertoire, l'accès en écriture autorise un utilisateur à ajouter et retirer des fichiers de ce répertoire, même s'il n'est pas le propriétaire des-dits fichiers.
3. Droit d'exécution (*x* pour *eXecute* ou exécuter). Pour un fichier, cela en autorise l'exécution (par conséquent, seuls les fichiers exécutables devraient normalement avoir ce droit positionné). Pour un répertoire,

cela autorise un utilisateur à le *traverser* (ce qui signifie entrer dans ce répertoire ou passer par celui-ci). Notez bien la séparation avec le droit en lecture : il se peut très bien que vous puissiez traverser un répertoire sans pouvoir lire son contenu !

Toutes les combinaisons de ces droits sont possibles : vous pouvez par exemple autoriser la lecture du fichier à vous seul et l'interdire à tous les autres. Vous pouvez même faire l'inverse, même si ce n'est pas très logique à première vue... En tant que propriétaire du fichier, vous pouvez en changer le groupe propriétaire (si et seulement si vous êtes aussi membre du nouveau groupe), et même vous déposséder du fichier (c'est-à-dire en changer le propriétaire). Bien entendu, si vous vous dépossédez d'un fichier, vous perdrez tous les droits sur celui-ci...

Prenons l'exemple d'un fichier et d'un répertoire. L'affichage ci-dessous correspond à la frappe de la commande `ls -l` depuis une *ligne de commande* :

```
$ ls -l
total 1
-rw-r----- 1 reine  users          0 Jul  8 14:11 un_fichier
drwxr-xr--  2 pierre users       1024 Jul  8 14:11 un_repertoire/
$
```

Les différents champs de sortie de la commande `ls -l` sont les suivants (de gauche à droite) :

- les dix premiers caractères désignent successivement le type du fichier et les droits qui lui sont associés ; le premier caractère désigne le type du fichier : c'est un tiret (-) s'il s'agit d'un fichier ordinaire, d si le fichier est un répertoire. Il existe d'autres types de fichiers dont nous parlerons dans ce manuel. Les neuf caractères qui suivent représentent les droits associés au fichier. On constatera ici la distinction faite entre les différents types d'utilisateur pour un même fichier : les trois premiers caractères représentent les droits octroyés à l'utilisateur propriétaire, les trois suivants s'appliquent à tout utilisateur du groupe qui n'est pas le propriétaire, et les trois derniers aux autres ; un tiret (-) signifie que le droit n'est pas octroyé ;
- vient ensuite le nombre de liens du fichier. Nous verrons plus loin que l'identifiant unique d'un fichier n'est pas son nom, mais un numéro (le *numéro d'inœud*), et qu'il est possible pour un fichier sur disque d'avoir plusieurs noms. Pour un répertoire, le nombre de liens a une signification spéciale, que nous aborderons également un peu plus loin ;
- viennent ensuite le nom de l'utilisateur propriétaire du fichier et le nom du groupe propriétaire ;
- enfin sont affichés la taille du fichier (en *octets*) ainsi que la date de sa dernière modification. Pour finir, vous trouverez également le nom du fichier ou du répertoire lui-même.

Observons maintenant en détails les droits associés à l'accès de chacun de ces fichiers : il faut tout d'abord enlever le premier caractère, qui désigne le type. Donc, pour le fichier `un_fichier`, les droits accordés sont : `rw-r-----`. Voici comment les interpréter :

- les trois premiers (`rw-`) sont les droits accordés à l'utilisateur propriétaire de ce fichier, en l'occurrence `reine`. L'utilisateur `reine` peut donc lire le fichier (`r`), le modifier (`w`) mais ne peut pas l'exécuter (`-`) ;
- les trois suivants (`r--`) sont concédés à tout utilisateur qui n'est pas `reine` mais qui appartient au groupe `users` : il pourra lire le fichier (`r`), mais ne pourra ni le modifier ni l'exécuter (`--`) ;
- les trois derniers (`---`) s'appliquent à tout utilisateur qui n'est pas `reine` et qui n'appartient pas au groupe `users` : un tel utilisateur n'a tout simplement aucun droit sur ce fichier.

Pour le répertoire `un_repertoire`, les droits sont `rxr-xr--`, et donc :

- `pierre`, en tant que propriétaire du répertoire, peut en lister le contenu (`r`), peut ajouter des fichiers dans ce répertoire ou en supprimer (`w`), et il peut traverser ce répertoire (`x`) ;
- tout utilisateur qui n'est pas `pierre` mais qui appartient au groupe `users` pourra lister le contenu de ce répertoire (`r`) mais ne pourra pas y ajouter des fichiers (`-`) ; par contre, il aura le droit de le traverser (`x`) ;
- tout autre utilisateur ne pourra que lister les fichiers de ce répertoire (`r`), mais c'est tout. Il sera incapable de le traverser.

Il existe **une** exception à ces règles : `root`. `root` peut changer les attributs (droits, propriétaire, groupe propriétaire) de tous les fichiers, même s'il n'en est pas le propriétaire. Cela veut dire qu'il peut aussi s'en attribuer la

propriété ! Il peut lire des fichiers sur lesquels il n'a pas le droit de lecture, traverser des répertoires auxquels il n'aurait normalement pas accès, et ainsi de suite. Et s'il lui manque un droit, il lui suffit simplement de le rajouter...

Pour conclure, il est utile de mentionner les différences entre les noms de fichiers dans le monde *UNIX* et le monde *Windows*. *UNIX* permet une flexibilité bien plus grande et a moins de limitation :

- un nom de fichier peut comporter n'importe quel caractère (à l'exception du caractère ASCII 0, qui dénote la fin d'une chaîne de caractères, et /, qui est le séparateur de répertoires), même des caractères non imprimables. De plus, *UNIX* est sensible à la casse : les fichiers *readme* et *Readme* sont différents, car *r* et *R* sont deux **caractères** distincts sous *UNIX* ;
- comme vous avez pu le remarquer, un nom de fichier ne comporte pas forcément une extension, à moins que vous le vouliez. Les extensions de fichier n'identifient pas le contenu des-dits fichiers sous *GNU/Linux*. D'ailleurs, il en va de même sur tous les systèmes d'exploitation. Cependant, ces « extensions » ainsi nommées sont toujours très pratiques. Le caractère point (.) sous *UNIX* n'est qu'un caractère comme les autres. Notons que les noms de fichier commençant avec un point sous *UNIX* sont des « fichiers cachés ».



Toutefois il est à signaler que certaines applications graphiques (gestionnaires de fichiers, applications bureautique, etc.) utilisent effectivement les extensions de noms de fichiers pour reconnaître facilement les formats de fichier. C'est donc une bonne idée d'utiliser ces extensions pour les applications qui en tirent parti.

### 1.3. Les processus

On désigne par le terme de *processus* une instance de programme en cours d'exécution et son *environnement*. Comme pour les fichiers, nous n'introduisons ici que les différences les plus importantes ; continuez la lecture de ce manuel pour une discussion plus en profondeur sur le sujet.

La différence la plus importante est, encore une fois, liée au concept d'utilisateurs : en effet, chaque processus s'exécute avec les droits de l'utilisateur qui l'a lancé. En interne, le système identifie les processus de façon unique grâce à un numéro. Ce numéro est appelé le PID (*Process ID*, donc identifiant de processus). Avec ce PID, le système sait, entre autres choses, qui (quel utilisateur) a lancé le processus. Il lui reste alors à vérifier que le processus demandé est « légal ». Reprenons l'exemple du fichier *un\_fichier* susmentionné. L'utilisateur pierre sera capable d'ouvrir ce fichier en *lecture seule*, mais pas en *lecture/écriture*, puisque les droits associés au fichier l'interdisent. Encore une fois, l'exception à la règle est *root*...

En conséquence, *GNU/Linux* est virtuellement immunisé contre les virus : pour opérer, les virus doivent infecter des fichiers exécutables du système. Mais avec le seul statut d'utilisateur, il leur est impossible de modifier ces fichiers, ce qui réduit d'autant plus les risques. Ajoutons que les virus sont, en général, très rares dans le monde *UNIX*. Jusqu'à présent, il existe moins d'une douzaine de virus connus sous *Linux*, et ils sont complètement inoffensifs lorsqu'ils sont lancés par un utilisateur normal. Un seul utilisateur peut vraiment endommager le système en activant ces virus, et, encore une fois, c'est *root*.

Toutefois, il existe bien des logiciels antivirus sous *GNU/Linux*, la plupart d'entre eux étant destinés aux fichiers *DOS / Windows*... En effet, de plus en plus de serveurs de fichiers *GNU/Linux* desservent des machines *windows* avec le paquetage logiciel *Samba*.

*Linux* permet également un contrôle aisé des processus, entre autres grâce aux signaux. Avec ceux-ci, vous pouvez, par exemple, suspendre un processus ou le tuer. Envoyez simplement le signal correspondant au processus et c'est fait. Toutefois, vous serez limité à l'envoi de signaux à vos propres processus, pas aux processus lancés par un autre utilisateur. L'exception à la règle est *root*. Oui, encore lui ! Dans le chapitre *Contrôle des processus*, page 43, vous apprendrez comment obtenir le PID d'un processus et lui envoyer des signaux.

### 1.4. Petite introduction à la ligne de commande

La ligne de commande est le moyen le plus direct pour donner des ordres à la machine. Si vous utilisez la ligne de commande de *GNU/Linux*, vous découvrirez vite qu'elle est bien plus puissante et polyvalente que d'autres lignes de commande que vous avez déjà pu utiliser. La raison en est que vous avez non seulement accès à toutes les applications de *X*, mais aussi à des milliers d'utilitaires en mode console (par opposition au



mode graphique) qui n'ont pas d'équivalents graphiques, ou dont les nombreuses options et combinaisons possibles seront difficilement accessibles sous la forme de boutons ou de menus.

Mais, il faut bien l'admettre, cette ligne de commande nécessite un peu d'aide pour débiter. Si vous n'êtes pas déjà en mode console et utilisez l'interface graphique, la première chose à faire est de lancer un émulateur de terminal. En accédant au menu *GNOME* ou *KDE*, vous trouverez ceux-ci dans le menu Terminaux. Ensuite, choisissez celui que vous voulez, par exemple, Konsole ou XTerm. Une icône qui identifie clairement l'émulateur de terminal se trouve également sur les tableaux de bord de *GNOME* et *KDE* (figure 1-3).



Figure 1-3. L'icône de l'émulateur de terminal sur le tableau de bord de KDE

Le *shell* est le nom du programme avec lequel vous entrez en relation. Vous êtes devant cette *invite* (*prompt* en anglais) :

```
[reine@localhost reine]$
```

Ceci suppose que votre nom d'utilisateur est reine et que votre nom de machine est localhost (ce qui est le cas si votre machine ne fait pas partie d'un réseau existant). L'espace après l'invite est disponible pour taper votre commande. Notez que quand vous êtes root, le \$ de l'invite devient un # (ceci est vrai dans la configuration par défaut, chacun de ces éléments pouvant être personnalisé). Lorsque vous avez lancé un *shell* en tant qu'utilisateur et que vous désirez « devenir » root, utilisez la commande *su* :

```
# Entrez le mot de passe root ; il n'apparaîtra pas à l'écran
[reine@localhost reine]$ su
Password:
# exit vous fera revenir à votre compte utilisateur normal
[root@localhost reine] # exit
[reine@localhost reine]$
```

Partout ailleurs dans cette documentation, l'invite sera représentée symboliquement par un \$, que vous soyez un utilisateur normal ou root. Il vous sera indiqué quand vous devez être root, alors n'oubliez pas *su*. Un # en début de ligne de code représentera un commentaire.

Quand vous *lancez* le *shell* pour la première fois, vous vous retrouverez normalement dans votre répertoire personnel. Pour savoir, à tout moment, dans quel répertoire vous vous situez, tapez la commande *pwd* (pour *Print Working Directory*, soit afficher le répertoire de travail) :

```
$ pwd
/home/reine
```

Nous allons maintenant examiner quelques commandes de base, et vous verrez bientôt que vous ne pourrez rien faire sans elles.

#### 1.4.1. cd : changer de répertoire

La commande *cd* est exactement la même que celle sous *DOS*, avec quelques fonctionnalités en plus. Elle fait exactement ce qu'énonce son acronyme : elle change le répertoire de travail. Vous pouvez utiliser *.* et *..*, qui sont respectivement le répertoire courant et son répertoire parent. Taper simplement *cd* vous ramènera à votre répertoire personnel. Taper *cd -* vous renverra dans le dernier répertoire visité. Et enfin, vous pouvez spécifier le répertoire de l'utilisateur pierre en tapant *~pierre* (*~* seul signifie votre propre répertoire personnel). Notez qu'en tant qu'utilisateur normal, vous ne pouvez, en général, pas accéder au répertoire d'un autre utilisateur (à moins qu'il ne l'ait explicitement autorisé ou que tel soit le réglage de la configuration par défaut du système), sauf si vous êtes root, donc soyons root et entraînons-nous :

```
$ pwd
/root
$ cd /usr/share/doc/HOWTO
$ pwd
/usr/share/doc/HOWTO
$ cd ../FAQ-Linux
$ pwd
/usr/share/doc/FAQ-Linux
$ cd ../../../../lib
$ pwd
/usr/lib
$ cd ~pierre
$ pwd
/home/pierre
$ cd
$ pwd
/root
```

Maintenant, redevenons un utilisateur ordinaire.

### 1.4.2. Quelques variables d'environnement et la commande echo

Tous les processus ont en fait leurs *variables d'environnement*. Le *shell* vous permet de les visualiser directement avec la commande `echo`. Voici quelques variables intéressantes :

1. HOME : cette variable d'environnement contient une chaîne de caractères désignant votre répertoire personnel.
2. PATH : elle contient la liste de tous les répertoires dans lesquels le *shell* doit chercher des exécutables quand vous tapez une commande (notez que, contrairement à *DOS*, par défaut, le *shell* n'ira **pas** chercher les commandes dans le répertoire courant !).
3. USERNAME : cette variable contient votre nom de login.
4. UID : elle contient votre identifiant utilisateur.
5. PS1 : cette variable abrite la définition de votre invite. C'est souvent une combinaison de séquences spécifiques, et vous pouvez lire `bash(1)` dans les *pages de manuel* pour plus de renseignements.

Pour que le *shell* affiche la valeur d'une variable, vous devez mettre un `$` devant son nom. Ici, `echo` va vous être utile :

```
$ echo Bonjour
Bonjour
$ echo $HOME
/home/pierre
$ echo $USERNAME
pierre
$ echo Bonjour $USERNAME
Bonjour pierre
$ cd /usr
$ pwd
/usr
$ cd $HOME
$ pwd
/home/pierre
```

Vous constaterez que le *shell* substitue la valeur de la variable avant d'exécuter la commande, sinon notre `cd $HOME` n'aurait pas fonctionné. En fait, le *shell* a en premier lieu remplacé `$HOME` par sa valeur, soit `/home/pierre` ; la ligne est donc devenue `cd /home/pierre`, ce que nous recherchions. Même chose pour `echo $USERNAME`, etc.

### 1.4.3. cat : afficher le contenu d'un ou de plusieurs fichiers à l'écran

Peu de choses à dire, si ce n'est que cette commande fait simplement et littéralement ce qu'elle énonce : afficher le contenu d'un ou de plusieurs fichiers sur la sortie standard, donc l'écran en temps normal :

```
$ cat /etc/fstab
/dev/hda5 / ext2 defaults 1 1
/dev/hda6 /home ext2 defaults 1 2
/dev/hda7 swap swap defaults 0 0
/dev/hda8 /usr ext2 defaults 1 2
/dev/fd0 /mnt/floppy auto sync,user,noauto,nosuid,nodev 0 0
none /proc proc defaults 0 0
none /dev/pts devpts mode=0620 0 0
/dev/cdrom /mnt/cdrom auto user,noauto,nosuid,exec,nodev,ro 0 0
$ cd /etc
$ cat modules.conf shells
alias parport_lowlevel parport_pc
pre-install plip modprobe parport_pc ; echo 7 > /proc/parport/0/irq
#pre-install pcmcia_core /etc/rc.d/init.d/pcmcia start
#alias char-major-14 sound
alias sound esssolo1
keep
/bin/zsh
/bin/bash
/bin/sh
/bin/tcsh
/bin/csh
/bin/ash
/bin/bsh
/usr/bin/zsh
```

### 1.4.4. less : un pager

Son nom est un jeu de mots sur le premier *pager* existant sous *UNIX*, qui se nommait *more*<sup>1</sup>. Un *pager* est un programme dont le but est d'autoriser la visualisation de longs fichiers page par page (plus précisément, écran par écran). Nous parlons de *less* plutôt que de *more* car son utilisation est beaucoup plus intuitive. Utilisez donc *less* pour voir des gros fichiers, qui sont trop gros pour l'écran. Par exemple :

```
less /etc/termcap
```

Pour naviguer dans le fichier, utilisez simplement les touches fléchées haut et bas, et **q** pour quitter. En fait, *less* peut faire bien plus : tapez simplement **h** pour avoir de l'aide (en anglais), et lisez. Mais de toute façon, l'objectif de cette section était de vous permettre de lire de longs fichiers et, nous l'espérons, cet objectif est maintenant atteint :-)

### 1.4.5. ls : faire une liste de fichiers

Cette commande est équivalente à celle nommée *dir* sous *DOS*, mais elle peut accomplir beaucoup plus de choses. Ceci est dû en grande partie au fait que les fichiers, eux-mêmes, font nettement plus ! La syntaxe de la commande *ls* est comme suit :

```
ls [options] [fichier|répertoire] [fichier|répertoire...]
```

Si aucun fichier ou répertoire n'est mentionné sur la ligne de commande, *ls* fera la liste des fichiers du répertoire courant. Ses options sont très nombreuses, et nous n'en citerons que quelques-unes :

1. *More* signifie « plus » et *less* signifie « moins »

1. `-a` : fait une liste de tous les fichiers, y compris les *fichiers cachés* (rappelons que sous *UNIX*, les fichiers cachés sont ceux dont le nom commence par un point (.) ; l'option `-A` fait une liste de « presque » tous les fichiers, à savoir tous les fichiers qu'afficherait l'option `-a` sauf « . » et « .. ».
2. `-R` : fait une liste récursivement, par exemple, tous les fichiers et sous-répertoires des répertoires mentionnés sur la ligne de commande.
3. `-s` : affiche la taille en kilo-octets à côté de chaque fichier.
4. `-l` : affiche des informations supplémentaires sur les fichiers.
5. `-i` : affiche le numéro d'inœud (le numéro unique du fichier sur un système de fichiers, voir le chapitre *Le système de fichiers Linux*, page 55) en face de chaque fichier.
6. `-d` : traite les répertoires comme des fichiers normaux au lieu de lister leur contenu.

Quelques exemples :

1. `ls -R` : fait une liste récursive des fichiers du répertoire courant.
2. `ls -is images/ ..` : fait une liste des fichiers du répertoire `images/` et du répertoire parent, avec pour chaque fichier son numéro d'inœud et sa taille en kilo-octets.
3. `ls -al images/*.png` : fait une liste de tous les fichiers (y compris les fichiers cachés) du répertoire `images/` dont le nom se termine par `.png`. Notez que cela comprend aussi le fichier `.png`, si celui-ci existe.

#### 1.4.6. Raccourcis clavier utiles

Beaucoup de séquences de touches sont disponibles, lesquelles peuvent vous faire gagner du temps précieux. Nous vous proposons une liste de celles qui sont essentielles. Nous supposons ici que vous utilisez le *shell* par défaut de **Mandrake Linux**, soit *bash*. Toutefois, ces séquences de touches devraient aussi fonctionner avec d'autres *shells*.

D'abord, les touches fléchées : *bash* maintient un historique des commandes que vous tapez, dans lequel vous pouvez vous déplacer avec les flèches haut et bas. Vous pouvez remonter jusqu'à un nombre de lignes définies dans la variable d'environnement `HISTSIZE`. De plus, l'historique est persistant d'une session à l'autre, donc vous ne perdrez pas les commandes que vous avez tapées lors d'une session précédente.

Les flèches gauche et droite déplacent le curseur dans le sens indiqué. Vous pouvez ainsi éditer vos lignes de cette façon. Mais il y a plus en matière d'édition : `Ctrl+a` et `Ctrl+e`, par exemple, vous amèneront respectivement au début et à la fin de la ligne courante. Les touches `<--` (*Backspace*) et `Suppr` fonctionneront comme on s'y attend. Un équivalent de *Backspace* est `Ctrl+h` et un équivalent de `Suppr` est `Ctrl+d`. `Ctrl+k` effacera toute la ligne depuis la position du curseur jusqu'à la fin de la ligne, et `Ctrl+w` effacera le mot qui précède la position du curseur.

Taper `Ctrl+d` sur une ligne vide fermera la session actuelle, ce qui est un vrai raccourci par rapport à la commande `exit`. `Ctrl+c` interrompra la commande en cours d'exécution, sauf si vous étiez en train d'éditer une ligne. Dans ce cas, ce sera l'édition en cours qui sera interrompue et vous serez ramené à l'invite. `Ctrl+l` nettoie l'écran.

Enfin, il parlons un peu de `Ctrl+s` et `Ctrl+q` : ces séquences de touches servent respectivement à suspendre et à restaurer le flux de caractères sur un *Terminal*. Elles sont très peu utilisées, mais il peut arriver que vous tapiez `Ctrl+s` par inadvertance (après tout, les touches `s` et `d` sont très proches l'une de l'autre sur un clavier...). Donc, si vous appuyez sur des touches mais ne voyez rien apparaître à l'écran, essayez `Ctrl+q` d'abord et faites attention : tous les caractères que vous avez tapés entre le `Ctrl+s` non désiré et le `Ctrl+q` apparaîtront alors sur l'écran.

## Chapitre 2. Disques et partitions

Si vous lisez ce chapitre, c'est que vous avez décidé de faire une installation en mode expert de **Mandrake Linux**, qui implique un repartitionnement. Ou alors, c'est que vous êtes curieux ou curieuse, et vous avez raison :-).

Ce chapitre donne une description complète du système de partitionnement du *PC*. Il n'est utile que si vous décidez de partitionner manuellement votre disque dur. Dans tous les cas de figures, vous pouvez sauter ce chapitre sans arrière-pensée, *DrakX* fera tout cela « automagiquement » pour vous.

### 2.1. Structure d'un disque dur

Pour simplifier, disons qu'un disque est physiquement divisé en petits secteurs. Une suite de secteurs peut former une partition. En fait, vous pouvez créer autant de partitions que vous le souhaitez, chacune d'entre elles étant considérées comme un disque dur séparé.

#### 2.1.1. Les Secteurs

Un disque dur n'est finalement rien d'autre qu'une suite de **secteurs**. Un secteur est la plus petite unité d'information sur un disque dur, et sa taille est en général de 512 octets. Les secteurs d'un disque dur de (n) secteurs sont numérotés de (0) à (n-1).

#### 2.1.2. Les Partitions

L'utilisation de plusieurs partitions vous permet de créer autant de disques durs virtuels sur votre disque dur réel. Ceci comporte plusieurs avantages:

- Des systèmes d'exploitation différents utilisent des structures de disque (systèmes de fichiers) différents; cela est notamment le cas entre *Windows* et *GNU/Linux*. Avoir plusieurs partitions sur un disque dur vous permet d'installer plusieurs systèmes d'exploitation sur le même disque matériel.
- Pour des raisons de performances, un système d'exploitation aura avantage à avoir plusieurs disques avec des systèmes de fichiers différents puisqu'ils seront utilisés pour des tâches complètement distinctes. C'est ainsi le cas pour *GNU/Linux* qui nécessite une deuxième partition appelée « *swap* » (échange), utilisée pour accueillir la mémoire virtuelle.
- Finalement, il peut s'avérer très utile de séparer les différentes parties de votre OS en autant de partitions, même si elles utilisent le même système de fichiers. Dans la configuration la plus simple, vous pouvez répartir vos fichiers sur deux partitions, une pour vos données personnelles, une autre pour le système lui-même. Cela vous permet de mettre à jour ce dernier en effaçant complètement la partition du système mais en gardant vos fichiers personnels intacts.
- Les erreurs physiques sur un disque dur sont généralement situées sur des secteurs adjacents et non dispersées sur tout le disque. Distribuer vos fichiers sur des partitions différentes limitera les pertes de données en cas de dommages physiques de votre disque dur.

Normalement, le type d'une partition spécifie le système de fichiers que la partition est censée héberger. Chaque système d'exploitation en reconnaît certains, mais pas d'autres.

#### 2.1.3. Définition de la structure du disque dur

##### 2.1.3.1. Le plus simple

Avoir seulement deux partitions ! une pour l'espace d'échange, l'autre pour les fichiers<sup>1</sup>.

---

1. Le système de fichiers actuellement utilisé par les fichiers *GNU/Linux* est appelé `ext3`



La règle générale pour la taille de la partition d'échange est de choisir le double de la taille de votre mémoire vive (RAM). Néanmoins, pour des configurations ayant beaucoup de mémoire (>512 Mo), cette règle ne s'applique plus, et des tailles plus petites sont alors préférables.

### 2.1.3.2. Une autre configuration courante

Opter pour la séparation des données et des programmes. Et pour être encore plus efficace, on ajoute généralement une troisième partition dite « racine » et étiquetée /. Elle accueillera les programmes nécessaires au démarrage du système et les programmes de maintenance de base.

Nous pouvons ainsi définir quatre partitions :

Échange :

Une partition de type swap, dont la taille est équivalente à celle de la mémoire vive

Racine : /

C'est la partition la plus importante. Non seulement elle contient les données et les programmes les plus importants pour le système, mais elle aura également à servir comme point d'accueil pour les autres partitions.

Les besoins en terme de taille de la partition racine sont très limités, 300Mo sont suffisants. Néanmoins, si vous envisagez d'installer des applications commerciales, résidant généralement dans /opt, vous devrez augmenter cette taille en conséquence. Une autre option est alors de créer une partition /opt séparée.

Données statiques : /usr

La plupart des paquetages installent presque tous leurs exécutables et fichiers de données dans /usr. L'avantage d'avoir l'ensemble sur une partition séparée est que l'on peut aisément la partager avec d'autres machines du réseau.

La taille dépendra des paquetages que vous souhaitez installer. Elle varie de 100Mo pour une installation minimum à plusieurs Go pour une installation complète. Un compromis de un ou de deux Go (selon la taille disponible sur votre disque) est généralement suffisant.

Répertoires personnels : /home

Sont stockés ici les répertoires personnels de tous les utilisateurs du système. Il accueille aussi généralement les répertoires servis par HTTP ou FTP (respectivement pour la navigation Web et les transferts de fichiers).

La taille de la partition est très variable: elle dépend du nombre d'utilisateurs (ou de services) hébergés et de leurs besoins.

Une variante de cette solution est de ne pas utiliser une partition séparée pour /usr : /usr sera alors un simple répertoire à l'intérieur de la partition /.

### 2.1.3.3. Configurations exotiques

Lorsque votre machine est configurée pour des utilisations bien particulières, pour être un serveur Web ou un pare-feu, ses besoins seront radicalement différents de ceux d'une machine de bureau standard. Par exemple, un serveur FTP aura probablement besoin d'une grosse partition séparée pour accueillir /home/ftp, alors que /usr sera plutôt limité. Dans de telles situations, il vaut mieux avoir soigneusement défini ses besoins avant même de commencer l'installation.



Si après un certain temps d'utilisation de votre machine, vous vous rendez compte que les partitions ou leur taille sont inadéquates, il vous sera possible de redimensionner la plupart des partitions sans avoir pour cela à réinstaller tout le système ; il n'y a en général pas de risque pour vos données. Consultez *Gérer ses partitions* du manuel *Guide de démarrage*.

Avec un peu d'expérience, vous pourrez même déplacer une partition comble vers un tout nouveau disque dur. Mais, ça, c'est une autre histoire...

## 2.2. Conventions pour nommer disques et partitions

*GNU/Linux* utilise une convention beaucoup plus logique pour nommer les partitions. D'une part, le type des partitions éventuellement présentes n'entre pas en ligne de compte ; d'autre part, les partitions sont nommées en fonction du disque où elles se situent. Tout d'abord, voici comment les disques sont nommés :

- les périphériques IDE (que ce soient des disques durs, lecteurs CD-ROM ou autres) primaires, maître et esclave, sont appelés respectivement `/dev/hda` et `/dev/hdb`
- sur l'interface secondaire, ce sont `/dev/hdc` et `/dev/hdd` pour maître et esclave respectivement
- si votre ordinateur contient d'autres interfaces IDE (par exemple l'interface IDE présente sur certaines cartes Soundblaster), les disques s'appelleront alors `/dev/hde`, `/dev/hdf`, etc.
- les disques SCSI sont appelés `/dev/sda`, `/dev/sdb`, etc. dans l'ordre de leur apparition sur la chaîne SCSI (en fonction des *ID* croissants). Les lecteurs de CD-ROM SCSI sont appelés `/dev/scd0`, `/dev/scd1`, toujours dans l'ordre d'apparition sur la chaîne SCSI.

Ainsi, *GNU/Linux* nommera les partitions de la façon suivante :

- Les partitions primaires (ou étendues) sont nommées `/dev/hda1` à `/dev/hda4` lorsqu'il y en a
- Les partitions logiques, s'il-y-en-a, sont nommées `/dev/hda5`, `/dev/hda6`, etc. dans leur ordre d'apparition dans la table des partitions logiques.

Ainsi *GNU/Linux* nommera les partitions de la façon suivante :

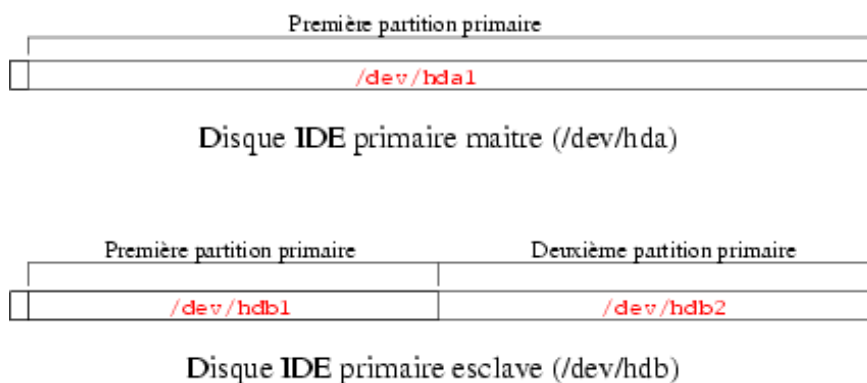


Figure 2-1. Premier exemple de noms de partitions sous GNU/Linux

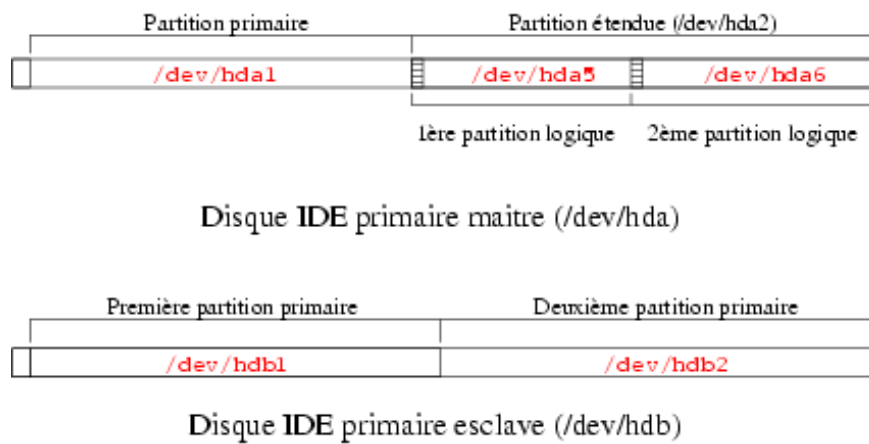


Figure 2-2. Second exemple de noms de partitions sous GNU/Linux

Vous voici maintenant à même de nommer les différentes partitions et disques durs quand vous en aurez besoin. Vous remarquerez également que *GNU/Linux* nomme les partitions même s'il ne sait pas les gérer d'entrée de jeu *a priori* (il ignore le fait que ce ne sont pas des partitions *GNU/Linux* autochtones).



Pour les noyaux récents 2.4, **Mandrake Linux**, utilise le Système de fichiers de périphériques Linux Devfs (Device File System) (<http://www.atnf.csiro.au/~rgooch/linux/docs/devfs.html>). Ce système assure une compatibilité totale avec les notations utilisées ci-dessus, mais cette compatibilité pourrait disparaître ultérieurement. En fait, chaque périphérique est ajouté dynamiquement au fur et à mesure de leur disponibilité ou nécessité.

Par exemple, le premier disque dur IDE devient désormais :

```
[root@localhost root]# ll /dev/hda
lr-xr-xr-x  1 root    root      32 Sep  2 17:14 /dev/hda
-> ide/host0/bus0/target0/lun0/disc
```



## Chapitre 3. Introduction à la ligne de commande

Dans le chapitre *Concepts UNIX de base*, page 1, nous avons vu comment lancer un *shell* et ses principes de base, mais nous ne l'avons pas fait fonctionner. C'est ce que nous nous proposons de faire dans ce chapitre.

Le principal avantage du *shell* est le nombre d'utilitaires existants : des milliers sont disponibles et chacun d'entre eux a une tâche bien définie. Nous n'en examinerons ici qu'un petit nombre. L'une des grandes forces d'*UNIX* est la possibilité de combiner ces utilitaires, comme nous le verrons plus loin.

### 3.1. Utilitaires de manipulation de fichiers

La manipulation de fichiers signifie ici « copier », « déplacer » et « effacer » des fichiers. Le changement de leurs attributs (propriétaire, permissions associées) sera examiné par la suite.

#### 3.1.1. mkdir, touch : création de répertoires et fichiers vides

`mkdir` (*MaKe DIRectory*) est utilisé pour créer des répertoires. Sa syntaxe est simple :

```
mkdir [options] <répertoire> [répertoire ...]
```

En fait, une seule option est vraiment intéressante : l'option `-p`. Si cette option est passée en argument, elle implique deux comportements :

1. `mkdir` créera les répertoires parents s'il n'existaient pas avant. Sans cette option, `mkdir` échouerait, se plaignant que les répertoires parents n'existent pas ;
2. `mkdir` terminera silencieusement si le répertoire que vous désirez créer existe déjà. De même, si vous ne spécifiez pas l'option `-p`, `mkdir` renverra un message d'erreur, se plaignant cette fois-ci que le répertoire à créer existe déjà.

Voici quelques exemples :

- `mkdir toto` crée un répertoire du nom de `toto` dans le répertoire courant.
- `mkdir -p images/divers docs` crée un répertoire `divers` dans le répertoire `images` après avoir créé ce dernier s'il n'existait pas (`-p`) ; il crée également un répertoire `docs` dans le répertoire courant.

Initialement, la commande `touch` n'a pas pour but de créer des fichiers mais de mettre à jour les dates d'accès et de modification<sup>1</sup>. Toutefois, l'un des effets de bord de `touch` est de créer les fichiers mentionnés comme des fichiers de taille 0 s'ils n'existaient pas déjà. La syntaxe est :

```
touch [options] fichier [fichier...]
```

Il faut donc lancer la commande :

```
touch fichier1 images/fichier2
```

ce qui créera un fichier vide appelé `fichier1` dans le répertoire courant et un fichier vide appelé `fichier2` dans le répertoire `images`, si ces fichiers n'existaient pas avant.

#### 3.1.2. rm : supprimer des fichiers ou des répertoires

`rm` (*ReMove*) remplace les commandes `del` et `deltree` de *DOS*, et rajoute des options supplémentaires. Sa syntaxe est :

```
rm [options] <file|directory> [file|directory...]
```

Parmi les options, on trouve :

---

1. Il y a trois mesures de temps distinctes pour chaque fichier sous *UNIX* : la date du dernier accès au fichier (`atime`), c'est-à-dire la date de la dernière ouverture du fichier en lecture ou en écriture ; la date de la dernière modification des attributs de l'inode (`mtime`) ; et enfin la date de la dernière modification du contenu du fichier (`ctime`).

- `-r` ou `-R` : supprime récursivement. Cette option est **obligatoire** pour supprimer un répertoire, même vide. Toutefois, pour effacer des répertoires vides, vous pouvez également utiliser la commande `rmdir`.
- `-i` : demande une confirmation avant chaque effacement. Prenez note que, par défaut et pour des raisons de sécurité, la commande `rm` dans **Mandrake Linux** est un *alias* de `rm -i` (comme le sont également les commandes `cp` et `mv`). Si vous désirez les effacer, vous pouvez éditer le fichier `~/ .bashrc` et ajouter la ligne suivante : `unalias rm cp mv`.
- `-f` : le contraire de `-i`, cette option force la suppression des fichiers ou répertoires, même si l'utilisateur n'a pas l'autorisation d'écriture sur les fichiers<sup>2</sup>.

Quelques exemples :

- `rm -i images/*.jpg fichier1` : suppression de tous les fichiers dont le nom se termine par `.jpg` dans le répertoire `images`, ainsi que le fichier `fichier1` dans le répertoire courant. Une confirmation est demandée pour chacun des fichiers. Répondez `o` ou `y` pour confirmer, `n` pour annuler.
- `rm -Rf images/divers/ fichier*` : suppression sans demande de confirmation de tout le répertoire `divers/` dans le répertoire `images/`. De plus, tous les fichiers du répertoire courant dont le nom commence par `fichier` sont également effacés.



Un fichier effacé avec `rm` l'est de façon **irréversible** : il n'y a alors aucun moyen de récupérer ce fichier<sup>3</sup>! N'hésitez donc pas à utiliser l'option `-i` afin d'éviter d'effacer des données par erreur.

### 3.1.3. `mv` : déplacer ou renommer des fichiers

La syntaxe de la commande `mv` est la suivante :

```
mv [options] <file|directory> [file|directory ...] <destination>
```

Quelques options :

- `-f` : force l'opération. Aucun avertissement en cas d'écrasement d'un fichier au cours de l'opération.
- `-i` : le contraire. Demande une confirmation à l'utilisateur avant d'écraser un fichier existant.
- `-v` : mode *verbeux* qui rapporte tous les changements.

Quelques exemples :

- `mv -i /tmp/pics/*.png .` : déplace tous les fichiers du répertoire `/tmp/pics/` dont le nom se termine par `.png` vers le répertoire courant (`.`). Une confirmation est demandée avant d'écraser un fichier existant.
- `mv toto titi` : renomme le fichier (ou le répertoire) `toto` en `titi`. Si un répertoire `titi` existait déjà, l'effet de cette commande serait de bouger tout le répertoire `toto` (le répertoire lui-même et tous ses fichiers et sous-répertoires) dans le répertoire `titi`.
- `mv -vf fichier* images/ trash/` : déplace, sans demander de confirmation, tous les fichiers dans le répertoire courant dont le nom commence par `fichier`, ainsi que tout le répertoire `images/` vers le répertoire `trash/`. Tous les changements effectués sont mentionnés.

---

2. Pour un utilisateur, il est suffisant de pouvoir écrire dans un répertoire pour en effacer des fichiers, même s'il n'en est pas le propriétaire.

### 3.1.4. cp : copier des fichiers et des répertoires

cp remplace les commandes copy et xcopy de *DOS* et contient d'autres options. Sa syntaxe est la suivante :

```
cp [options] <file|directory> [file|directory ...] <destination>
```

Il existe une myriade d'options. Voici les plus utilisées :

- -R : copie récursivement ; **obligatoire** pour copier un répertoire, même vide.
- -i : demande une confirmation avant d'écraser des fichiers.
- -f : le contraire de -i. Remplace tous les fichiers existants sans demander de confirmation.
- -v : mode verbeux qui mentionne toutes les actions effectuées par cp.

Quelques exemples :

- cp -i /tmp/images/\* images/ : copie tous les fichiers du répertoire /tmp/images dans le répertoire images/ du répertoire courant, en demandant une confirmation avant d'écraser un fichier.
- cp -vR docs/ /shared/mp3s/\* mestrucs/ : copie tout le répertoire docs du répertoire courant, en plus de tous les fichiers du répertoire /shared/mp3s dans le répertoire mestrucs, lequel est situé dans le répertoire courant.
- cp toto titi : copie le fichier toto sous le nom de titi dans le répertoire courant.

## 3.2. Manipulation des attributs de fichiers

La série de commandes présentée ici est utilisée pour changer le propriétaire ou groupe propriétaire d'un fichier ou ses droits d'accès. Les différents droits d'accès sont présentés dans le chapitre « Concepts de base des systèmes UNIX ».

### 3.2.1. chown, chgrp : changer l'utilisateur et le groupe propriétaire d'un ou plusieurs fichiers

La syntaxe de la commande chown est la suivante :

```
chown [options] <user[.group]> <file|directory> [file|directory...]
```

Entre autres options, vous trouverez celles-ci :

- -R : récursif. Change le propriétaire de tous les fichiers et sous-répertoires d'un répertoire donné ;
- -v : mode verbeux. Décrit toutes les actions effectuées par chown ; indique quels fichiers ont changé de propriétaire à la suite de la commande ainsi que ceux qui demeurent inchangés ;
- -c : comme -v, mais ne mentionne que les fichiers pour lesquels il y a eu un changement.

Quelques exemples :

- chown nobody /shared/book.tex : change le propriétaire du fichier /shared/book.tex pour nobody ;
- chown -Rc reine.musique \*.mid concerts/ : donne la propriété de tous les fichiers dans le répertoire courant se terminant par .mid et de tous les fichiers et sous-répertoires du répertoire concerts/ à reine et au groupe musique. Cette commande ne mentionne que les fichiers affectés par la commande.

La commande chgrp (*CHange GRouP*) ne vous laisse changer que le groupe propriétaire d'un fichier ou d'un groupe de fichiers. Sa syntaxe est très semblable à celle de la commande chown :

```
chgrp [options] <group> <file|directory> [file|directory...]
```

Les options de cette commande sont les mêmes que pour chown, et elle est utilisée de façon très similaire. Ainsi, la commande :

```
chgrp disk /dev/hd*
```

attribue au groupe disk tous les fichiers du répertoire /dev dont le nom commence par hd.

### 3.2.2. chmod : changer les permissions sur des fichiers et des répertoires

La commande chmod a une syntaxe bien particulière. Sa syntaxe générale est :

```
chmod [options] <change mode> <file|directory> [file|directory...]
```

mais ce sont les différentes formes que peut prendre le changement de mode qui la rendront plus spécifique. Ceci peut se produire de deux façons :

1. en octal. Les droits d'accès de l'utilisateur propriétaire correspondent alors à des chiffres de la forme  $\langle x \rangle 00$ , où  $\langle x \rangle$  correspond au droit assigné : 4 pour lecture, 2 pour écriture, 1 pour exécution. De même, les droits d'accès du groupe propriétaire sont de la forme  $\langle x \rangle 0$  et ceux des « autres » sont de la forme  $x$ . Pour obtenir le chiffre correct, il suffira d'additionner les droits d'accès assignés. Ainsi, les permissions `rw-r--r--` correspondent à  $400+200+100$  (droits d'accès de l'utilisateur propriétaire, `rw`)  $+40+10$  (droits d'accès du groupe, `r--`)  $+4$  (droits d'accès des autres, `r--`) = 754. Les droits d'accès sont ainsi exprimés de manière absolue : les droits d'accès précédents sont remplacés de façon inconditionnelle ;
2. à l'aide de certaines expressions. Les droits d'accès sont ici indiqués par une suite d'expressions séparées par des virgules, une expression étant de la forme `[catégorie]<+|-|=><droits d'accès>`.

La catégorie peut être une combinaison de :

- `u` (*User*, soit utilisateur, permission pour propriétaire) ;
- `g` (*Group*, soit groupe, permission pour le groupe propriétaire) ou ;
- `o` (*Others*, permission pour les « autres »).

Si aucune catégorie n'est spécifiée, le changement s'applique à toutes les catégories. Un `+` appose un droit d'accès, un `-` le retire et un `=` établit la permission. Pour finir, les droits d'accès sont définis par une ou plusieurs des lettres suivantes :

- `r` (*Read*, soit lecture) ;
- `w` (*Write*, soit écriture) ou ;
- `x` (*eXecute*, soit exécution).

Les options principales sont très similaires à celles de `chown` ou `chgrp` :

- `-R` : change les droits d'accès récursivement ;
- `-v` : mode verbeux. Il décrit les actions effectuées pour chaque fichier ;
- `-c` : comme `-v`, mais ne mentionne que les fichiers dont les droits d'accès ont changé.

Exemples :

- `chmod -R o-w /shared/docs` : enlève de façon récursive le droit d'écriture aux « autres » sur tous les fichiers et sous-répertoires du répertoire `/shared/docs/` ;
- `chmod -R og-w,o-x /privé/` : enlève de façon récursive le droit d'écriture pour le groupe et les autres sur tout le répertoire `/privé/`, et retire le droit d'exécution pour les autres ;
- `chmod -c 644 /divers/fichier*` : change les droits d'accès de tous les fichiers du répertoire `/divers/` dont les noms commencent par `fichiers` en `rw-r--r--` (droit de lecture pour tout le monde et droit d'écriture pour le propriétaire du fichier seulement). Cette commande ne mentionne que les fichiers affectés par l'opération.

### 3.3. Motifs d'englobement du shell

Il est probable que vous utilisiez déjà sans le savoir des caractères d'*englobement*. Quand vous enregistrez un fichier dans une application sous *Windows* ou lorsque vous recherchez un fichier, vous utilisez `*` pour désigner une suite de caractères quelconques. Par exemple, `*.txt` désigne l'ensemble des fichiers dont le nom se termine par `.txt`. Nous l'avons également utilisé fréquemment dans la section précédente, mais l'englobement va beaucoup plus loin que le seul `*`.

Quand vous tapez une commande comme `ls *.txt`, puis Entrée, la tâche de trouver quels fichiers correspondent au motif `*.txt` n'est pas du ressort de `ls`, mais doit passer par le *shell* lui-même. Cela requiert donc une petite explication sur la façon dont le *shell* interprète une ligne de commande. Lorsque vous tapez :

```
$ ls *.txt
readme.txt  recipes.txt
```

la ligne de commande est tout d'abord séparée en mots (`ls` et `*.txt` en l'occurrence). Quand le *shell* voit le `*` dans un des mots, il interprète le mot comme étant un motif englobant et le remplace dans la ligne de commande par les noms de tous les fichiers correspondant au motif. La ligne qui se trouve juste avant que ne s'enclenche l'exécution par le *shell* est donc devenue `ls readme.txt recettes.txt`, ce qui donnera le résultat recherché. Le *shell* réagit aussi à la vue d'autres caractères :

- `?` : correspond à un caractère unique, quel qu'il soit ;
- `[...]` : correspond à tout caractère écrit entre les crochets ; les caractères peuvent désigner soit des intervalles (par exemple, `1-9`), soit des valeurs *discrètes*, soit encore un mélange des deux. Exemple : `[a-zA-Z0-9]` correspond à tous les caractères de `a` à `z`, un `B`, un `E`, un `5`, un `6` ou un `7` ;
- `[^...]` : correspond à tous les caractères qui ne se trouvent **pas** entre les crochets ; `[^a-z]`, par exemple, correspond à tout caractère qui n'est pas une lettre minuscule<sup>4</sup>.
- `{c1,c2}` : correspond à `c1` ou `c2`, où `c1` et `c2` sont également des caractères d'englobement, ce qui signifie que vous pouvez écrire `{[0-9]*,[acr]}` par exemple.

Voici quelques exemples de motifs et leur signification :

- `/etc/*conf` : tous les fichiers du répertoire `/etc` dont le nom se termine par `conf`. Cela peut correspondre au fichier `/etc/inetd.conf`, mais aussi à `/etc/conf.linuxconf`, et à `/etc/conf` si un tel fichier existe. Souvenez-vous que `*` peut correspondre à une chaîne vide.
- `image/{cars,space[0-9]}/*.jpg` : tous les fichiers dont le nom se termine par `.jpg` dans les répertoires `image/cars`, `image/space0`, jusqu'à `image/space9`, s'ils existent.
- `/usr/share/doc/*/README` : tous les fichiers de nom `README` dans tous les sous-répertoires immédiats de `/usr/share/doc`. Cela correspondra à `/usr/share/doc/mandrake/README` par exemple, mais pas à `/usr/share/doc/myprog/doc/README`.
- `*[^a-z]` : tous les fichiers du répertoire courant dont le nom ne finit **pas** par une lettre minuscule.

### 3.4. Redirections et tubes

#### 3.4.1. Encore un mot au sujet des processus

Pour comprendre le principe des redirections et des tubes, ils nous faudra introduire ici une nouvelles notion qui concerne les processus. Chaque processus sous *UNIX* (y compris les applications graphiques) utilise un minimum de trois descripteurs de fichiers : l'entrée standard, la sortie standard et le canal d'erreur standard.

Leurs numéros respectifs sont 0, 1 et 2. En général, ces trois descripteurs sont associés au *Terminal* depuis lequel le processus a été lancé, l'entrée standard étant lue depuis le clavier. Le but des redirections et des tubes est de rediriger ces descripteurs. Les exemples de cette section vous aideront à mieux comprendre.

4. Attention ! Bien que cela soit vrai pour la plupart des langues, il est possible que cela ne fonctionne pas pour votre langue locale. Cela dépend de l'ordre de tri (*collating order*). Pour certaines configuration de langage, `[a-z]` correspondra à `a`, `A`, `b`, `B` (...), `Z`. Et cela, sans parler du fait que certaines langues contiennent des caractères accentués...

### 3.4.2. Redirections

Supposons, par exemple, que vous vouliez connaître la liste des fichiers se terminant par `.png`<sup>5</sup> dans le répertoire `images`, et que cette liste soit très longue : il serait donc pertinent de la stocker dans un fichier pour la consulter à loisir ensuite. Vous pouvez alors taper ceci :

```
$ ls images/*.png 1>liste_fichiers
```

Ce qui signifie que la sortie standard de cette commande (1) est redirigée (>) vers le fichier qui a pour nom `liste_fichiers`. Le signe > est l'opérateur de redirection de sortie. Dans le cas où le fichier de redirection n'existe pas, il est alors créé. Par contre, s'il existait précédemment, son ancien contenu est écrasé. Cependant, par défaut, le descripteur redirigé par cet opérateur est la sortie standard, il n'est donc pas nécessaire de le spécifier sur la ligne de commande. Vous pouvez donc écrire plus simplement :

```
$ ls images/*.png >liste_fichiers
```

et le résultat sera exactement le même. Vous pouvez ensuite consulter le fichier à l'aide d'un visualiseur de fichiers textes tel que `less`.

Supposons maintenant que vous vouliez connaître le nombre exact de ces fichiers. Au lieu de compter manuellement, vous pouvez utiliser le bien nommé `wc` (*Word Count*, soit comptage des mots) avec l'option `-l`, qui écrit sur la sortie standard le nombre de lignes du fichier. Pour obtenir le résultat désiré, une solution possible serait la suivante :

```
wc -l 0<liste_fichiers
```

Le signe < est l'opérateur de redirection d'entrée. Le descripteur redirigé par défaut est également celui de l'entrée standard, donc 0. La ligne s'écrit alors simplement :

```
wc -l <liste_fichiers
```

Supposons maintenant que vous vouliez retirer de cette liste toutes les extensions des fichiers puis sauvegarder le résultat dans un autre fichier. L'outil dont vous avez besoin est `sed`, pour *Stream EDitor* (soit éditeur de flux). Il suffit de rediriger l'entrée standard de `sed` vers le fichier `liste_fichiers` et de rediriger sa sortie vers le fichier résultat, par exemple `la_liste` :

```
sed -e 's/\.[a-z]*$//' <liste_fichiers >la_liste
```

Il vous sera également possible de consulter à loisir cette nouvelle liste avec un visualiseur.

Il pourrait aussi s'avérer utile de rediriger l'erreur standard. Par exemple, vous voulez savoir quels répertoires dans `/shared` ne vous sont pas accessibles : une solution est de lister récursivement ce répertoire et de rediriger les erreurs vers un fichier, tout en n'affichant pas le canal de sortie standard :

```
ls -R /shared >/dev/null 2>erreurs
```

Ceci signifie que la sortie standard sera redirigée (>) vers `/dev/null`, un fichier spécial dans lequel tout ce qu'on écrit est perdu (et aura comme effet secondaire que la sortie standard ne sera pas affichée) et que le canal d'erreur standard (2) est redirigé (>) vers le fichier `erreurs`.

### 3.4.3. Tubes

Les tubes (*pipes* en anglais) sont en quelque sorte une combinaison des redirections d'entrée et de sortie. Leur principe mime en effet celui d'un tube : un processus envoie des données dans le tube par un bout et un autre processus lit les données par l'autre bout. L'opérateur tube est `|`. Reprenons l'exemple de la liste des fichiers `.png` ci-dessus. Supposons que vous vouliez seulement connaître le nombre de fichiers en question sans avoir à stocker la liste dans un fichier temporaire : utilisez alors la commande suivante :

```
ls images/*.png | wc -l
```

---

5. Il peut vous paraître saugrenu de dire « les fichiers se terminant par `.png` » plutôt que « les images PNG ». Mais, encore une fois, les fichiers sous *UNIX* n'ont d'extension que par convention : une extension ne détermine en aucun cas le type d'un fichier. Un fichier dont le nom se termine par `.png` peut indifféremment être une image JPEG, un exécutable, un fichier texte ou tout autre chose. Il en est de même sous *Windows* !

ce qui signifie que la sortie standard de la commande `ls` (donc la liste des fichiers) est redirigée vers l'entrée standard de la commande `wc`. Vous obtenez donc le résultat désiré.

Vous pouvez de même construire directement la liste des fichiers « sans les extensions » avec la commande suivante :

```
ls images/*.png | sed -e 's/\.png//g' >la_liste
```

ou, si vous voulez simplement consulter la liste sans la stocker dans un fichier :

```
ls images/*.png | sed -e 's/\.png//g' | less
```

Les tubes et les redirections ne sont pas limités à du texte. Ainsi en est-il de la commande suivante, lancée à partir d'un *Terminal* :

```
xwd -root | convert - ~/mon_bureau.png
```

ce qui effectuera une capture d'écran de votre bureau dans le fichier intitulé `mon_bureau.png`<sup>6</sup> dans votre répertoire personnel.

### 3.5. Le complètement (completion) dans les lignes de commande

**Complètement** Le complètement est une fonctionnalité des plus pratiques et tous les *shells* modernes (dont *bash*) l'incluent désormais. Son but est d'aider l'utilisateur à en faire le moins possible. La meilleure façon d'illustrer ceci est de donner un exemple.

#### 3.5.1. Un exemple

Supposons que vous ayez dans votre répertoire personnel un fichier `fichier_au_nom_très_long_impossible_à_taper`, et que vous vouliez le consulter. Mais, vous avez également dans ce même répertoire un autre fichier appelé `fichier_texte`. Que faire ? Vous vous trouvez dans votre répertoire personnel et vous tapez alors la séquence suivante :

```
$ less fi<TAB>
```

(c'est-à-dire, tapez `less fi` suivi d'une pression sur la touche `TAB`). Le *shell* aura alors étendu la ligne de commande pour vous :

```
$ less fichier_
```

et aura également marqué la liste des choix possibles (dans sa configuration par défaut, qui peut être personnalisée). Tapez alors la séquence de touches suivante :

```
less fichier_a<TAB>
```

et le *shell* aura étendu la ligne de commande pour obtenir le résultat que vous souhaitiez :

```
less fichier_au_nom_très_long_impossible_à_taper
```

Il ne vous reste plus alors qu'à taper sur la touche `Entrée` pour valider et consulter le fichier.

#### 3.5.2. Autres méthodes de complètement

La touche `TAB` n'est pas l'unique moyen d'activer le complètement, bien qu'il soit le plus courant. En général, le mot à compléter sera un nom de commande pour le premier mot de la ligne de commande (`ns1<TAB>` donnera `nslookup`), et un nom de fichier pour tous les autres, à moins que le mot ne soit précédé d'un caractère « magique » parmi `~`, `@` ou `$`. Dans ce cas, le *shell* essaiera de compléter respectivement un nom d'utilisateur, un

6. Oui, ce sera bien une image PNG ! (Le paquetage *ImageMagick* devra néanmoins être installé...)

nom de machine ou une variable d'environnement<sup>7</sup>. Il existe également un caractère magique pour compléter un nom de commande (!) ou un nom de fichier (/).

Les deux autres façons d'activer le complètement sont les séquences `Esc-<x>` et `Ctrl+x <x>` (`Esc` pour la touche Échap, `Ctrl+x` voulant dire `Control+<x>`) où `<x>` est l'un des caractères magiques déjà mentionnés. Taper `Esc-<x>` réussira le complètement seulement s'il n'y a pas d'ambiguïté et, en cas d'échec, complétera le mot à la plus grande sous-chaine possible dans la liste des choix. Un *bip* signifie soit que le choix n'est pas univoque ou qu'il n'y a tout simplement pas de choix correspondant. La séquence `Ctrl+x <x>` affichera la liste des choix possibles sans tenter aucun complètement. La pression sur la touche `TAB` est équivalente à une pression successive de `Esc-<x>` et de `Ctrl+x <x>`, le caractère magique dépendant du contexte.

Ainsi, une des façons permettant de voir toutes les variables d'environnement définies est de taper sur une ligne vierge la séquence `Ctrl+x $`. Un autre cas : pour voir la page de manuel de la commande `nslookup`, il suffira de taper `man nslookup` puis `Esc-!`, et le *shell* complétera automatiquement la commande vers `man nslookup`.

### 3.6. Lancement et manipulation de processus en arrière-plan

Vous aurez remarqué que quand vous lancez une commande à partir d'un *Terminal*, vous devez normalement attendre que la commande soit terminée pour que le *shell* vous rende la contrôle : c'est que vous avez lancé la commande au *premier plan*. Il y a des situations, cependant, où cela n'est pas souhaitable.

Supposons, par exemple, que vous ayez entrepris de copier récursivement un gros répertoire vers un autre. Vous décidez également d'ignorer les erreurs, donc vous redirigez le canal d'erreur vers `/dev/null` :

```
cp -R images/ /shared/ 2>/dev/null
```

Une telle commande peut prendre plusieurs minutes avant de se terminer. Vous disposez alors de deux solutions : la première, brutale, est d'interrompre (de « tuer ») la commande pour la relancer plus tard, quand vous aurez le temps. Pour ce faire, tapez `Ctrl+c` : cela terminera le processus et vous retournerez alors à l'invite. Mais attendez, ne faites pas ça ! Lisez ce qui suit.

Supposons que vous vouliez exécuter une commande pendant que vous faites quelque chose d'autre. La solution est de placer le processus en *arrière-plan*. Pour ce faire, tapez d'abord `Ctrl+z` pour suspendre le processus :

```
$ cp -R images/ /shared/ 2>/dev/null
# Type C-z here
[1]+  Stopped                  cp -R images/ /shared/ 2>/dev/null
$
```

et vous voilà à nouveau devant l'invite. Le processus est alors suspendu, dans l'attente d'être relancé (comme l'indique le mot-clé `Stopped`, soit arrêté). Vous allez le relancer mais en le maintenant en arrière-plan. Tapez `bg` (pour *BackGround*, soit arrière-plan) provoquera exactement l'effet escompté :

```
$ bg
[1]+ cp -R images/ /shared/ 2>/dev/null &
$
```

Le processus aura alors repris son exécution en tâche de fond, ce qu'indique le signe `&` (esperluette) à la fin de la ligne. Vous vous retrouvez alors en face de l'invite et pouvez continuer à travailler. Un processus qui tourne en tâche de fond, ou arrière-plan, est appelé un *job*.

Vous pouvez bien sûr lancer directement des processus en tâche de fond, justement en ajoutant une `&` à la fin de la commande. Ainsi, vous pouvez lancer la copie du répertoire en arrière-plan en écrivant :

```
cp -R images/ /shared/ 2>/dev/null &
```

Si vous le souhaitez, vous pouvez également remettre ce processus au premier plan et attendre qu'il se termine en tapant `fg` (pour *ForeGround*, soit premier plan). Répétez alors la séquence `Ctrl+z`, `bg` pour le remettre à l'arrière-plan.

---

7. Souvenez-vous : *UNIX* différencie les majuscules des minuscules. La variable d'environnement `HOME` et la variable `home` ne sont pas les mêmes. Il existe aussi un caractère magique pour compléter un nom de fichier (/) et une commande pour rappeler une commande de l'historique (!).



Vous pouvez lancer plusieurs jobs de cette façon : chacune de ces commandes recevra alors un numéro de job. La commande `jobs` du *shell* indique la liste de tous les jobs associés au *shell* courant. Le job précédé d'un signe + désigne le dernier processus mis en tâche de fond. Pour remettre un job en particulier au premier plan, vous pourrez alors taper `fg <n>` où `<n>` désigne le numéro de job, par exemple `fg 5`.

Notez que vous pouvez également suspendre ou lancer de cette façon des applications *plein écran* (si elles sont correctement programmées), telles que `less` ou un éditeur de texte comme *Vi*, et les remettre au premier plan quand vous le voulez.

### 3.7. Le mot de la fin

Vous avez sans doute remarqué que le *shell* est très complet. L'utiliser efficacement sera avant tout une question de pratique. Cependant, ce chapitre (somme toute relativement long) n'aura fait mention que de quelques-unes des commandes disponibles : **Mandrake Linux** comporte des milliers d'utilitaires, et même les utilisateurs les plus expérimentés n'en utilisent qu'une centaine au grand maximum.

Il existe en effet des utilitaires pour tous les goûts et toutes les utilisations : vous avez des utilitaires de manipulation d'images (tels que `convert` susmentionné, mais aussi le mode *batch* de *GIMP* et tous les utilitaires de manipulation de *pixmap*s), de sons (encodeurs MP3, lecteurs de CD audio), de gravure de CD-ROM, des programmes de courrier électronique, des clients FTP et même des navigateurs Web (*lynx* ou *links*), sans oublier tous les outils d'administration.

Même s'il existe des applications graphiques aux fonctionnalités équivalentes, elles ne sont que des interfaces graphiques faites autour de ces mêmes utilitaires. De plus, les utilitaires en ligne de commande ont l'avantage de pouvoir fonctionner en mode non interactif : vous pouvez lancer une gravure de CD et ensuite vous déconnecter du système, tout en étant certain que la gravure s'effectuera (voir la page de manuel de la commande `nohup(1)` ou `screen(1)`).



## Chapitre 4. L'édition de texte : Emacs et VI

L'édition de texte<sup>1</sup> est un aspect incontournable de l'utilisation d'un système *UNIX*. Les deux éditeurs dont nous allons étudier (brièvement) l'utilisation paraîtront un peu difficiles au premier abord, mais dès que vous en aurez acquis les bases, vous en constaterez le formidable potentiel.

### 4.1. Emacs

*Emacs* est sans doute l'éditeur de texte le plus puissant actuellement. Il peut absolument tout faire et il est extensible à l'infini grâce à son langage de programmation inclus, s'appuyant sur *lisp*. Avec lui, vous pouvez vous promener sur le Web, lire votre courrier, faire un tour dans les forums, quasiment faire du café ! Toutefois, nous nous limiterons de vous donner les clés pour ouvrir *Emacs*, éditer un ou plusieurs fichiers, les sauvegarder, et quitter *Emacs*. Ce qui est déjà très bien !

#### 4.1.1. Brève présentation

Invoquer *Emacs* est relativement simple :

```
emacs [fichier] [fichier...]
```

*Emacs* ouvrira chaque fichier passé en argument dans un tampon jusqu'à un maximum de deux tampons visibles en même temps, et vous présentera le tampon *\*scratch\** si vous ne spécifiez pas de fichier. Si vous êtes sous *X*, des menus sont également à votre disposition, mais nous apprendrons ici à manipuler *Emacs* à l'aide des raccourcis clavier.

#### 4.1.2. Pour commencer

Il est temps de se jeter à l'eau ! Ouvrons par exemple deux fichiers, *fichier1* et *fichier2*. Si ces deux fichiers n'existent pas, ils seront alors créés (à condition que vous écriviez quelque chose dedans) :

```
$ emacs fichier1 fichier2
```

Vous obtenez la fenêtre suivante (figure 4-1) :

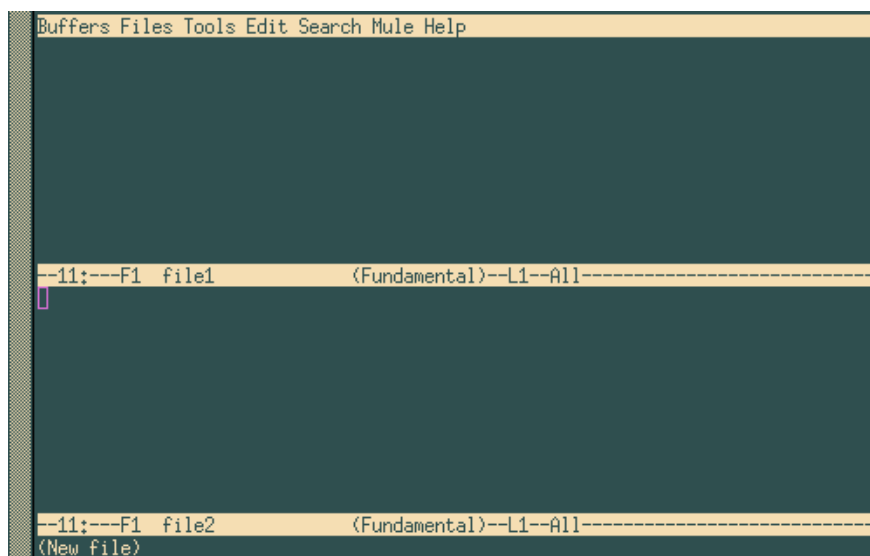


Figure 4-1. Emacs ; édition simultanée de deux fichiers

Vous pouvez constater que deux tampons ont été créés, c'est-à-dire un par fichier. Un troisième est également présent, au bas de l'écran (là où est écrit *(New file)*) : c'est le mini-tampon. Vous ne pouvez pas aller de vous-

---

1. « Éditer du texte » signifie modifier le contenu d'un fichier contenant uniquement des lettres, des chiffres et des signes de ponctuation ; de tels fichiers peuvent être des messages électroniques, du code source de programmes, des documents ou des fichiers de configuration.

même dans ce tampon, il faut qu'*Emacs* vous y invite lors de saisies interactives. Pour changer de tampon, tapez `Ctrl+x o`. Vous pouvez taper du texte soit comme dans un éditeur « normal », soit en effacer avec la touche `Suppr` ou bien la touche `Backspace`.

Pour vous déplacer, vous pouvez utiliser les touches fléchées, mais aussi d'autres combinaisons : `Ctrl+a` pour aller en début de ligne, `Ctrl+e` pour aller en fin de ligne, `Alt-<` pour aller au début du tampon et `Alt->` pour aller à la fin du tampon. Il existe de nombreuses autres combinaisons, même pour chacune des touches fléchées<sup>2</sup>.

Dès que vous voulez enregistrer les modifications faites sur un fichier, tapez `Ctrl+x Ctrl+s` ou, si vous voulez enregistrer le contenu du tampon dans un autre fichier, tapez `Ctrl+x Ctrl+w` et *Emacs* vous demandera le nom du fichier dans lequel écrire le contenu du tampon. Pour ce faire, vous disposez du *complément*.

### 4.1.3. Manipulation des tampons

Vous pouvez, si vous le voulez, ne montrer qu'un tampon à l'écran. Vous avez deux solutions :

- vous êtes dans le tampon que vous voulez cacher : tapez `Ctrl+x 0` ;
- vous êtes dans le tampon que vous voulez conserver à l'écran : tapez `Ctrl+x 1`.

Vous pouvez ensuite remettre le tampon que vous souhaitez à l'écran de deux manières :

- tapez `Ctrl+x b` et rentrez le nom du tampon que vous souhaitez voir ;
- tapez `Ctrl+x Ctrl+b`. Un nouveau tampon sera alors ouvert, appelé *\*Buffer List\** ; vous pouvez vous déplacer dans ce tampon à l'aide de la séquence `Ctrl+x o`, sélectionner le tampon que vous souhaitez et puis appuyer sur la touche `Entrée`, ou bien taper le nom dans le mini-tampon. Le tampon *\*Buffer List\** retournera en arrière-plan dès que votre choix est fait.

Si vous en avez fini avec un fichier et voulez vous débarrasser du tampon associé, tapez `Ctrl+x k`. *Emacs* vous demandera alors quel tampon il doit fermer. Par défaut, c'est le nom du tampon dans lequel vous êtes ; si vous voulez vous débarrasser d'un autre tampon que celui proposé, entrez directement son nom ou bien appuyez sur `TAB` : *Emacs* ouvrira alors à nouveau un autre tampon appelé *\*Completions\**, indiquant la liste des choix possibles. La touche `Entrée` valide le choix.

Vous pouvez également à tout moment remettre deux tampons visibles à l'écran ; pour cela, tapez `Ctrl+x 2`. Par défaut, le nouveau tampon créé sera une copie du tampon en cours (ce qui vous permet par exemple d'éditer un gros fichier en plusieurs endroits « en même temps ») et il vous suffit alors de procéder comme indiqué précédemment pour passer à un autre tampon.

Vous pouvez à tout moment ouvrir d'autres fichiers, avec `Ctrl+x Ctrl+f`. *Emacs* vous demandera alors le nom du fichier (vous disposez là aussi du complément).

### 4.1.4. Copier, couper, coller, rechercher

Supposons que nous sommes dans la situation de la figure 4-2.

---

2. *Emacs* a été conçu pour fonctionner sous un maximum d'environnements et il existe encore aujourd'hui des terminaux qui n'ont pas de touches fléchées. C'est encore plus vrai de *Vi*.

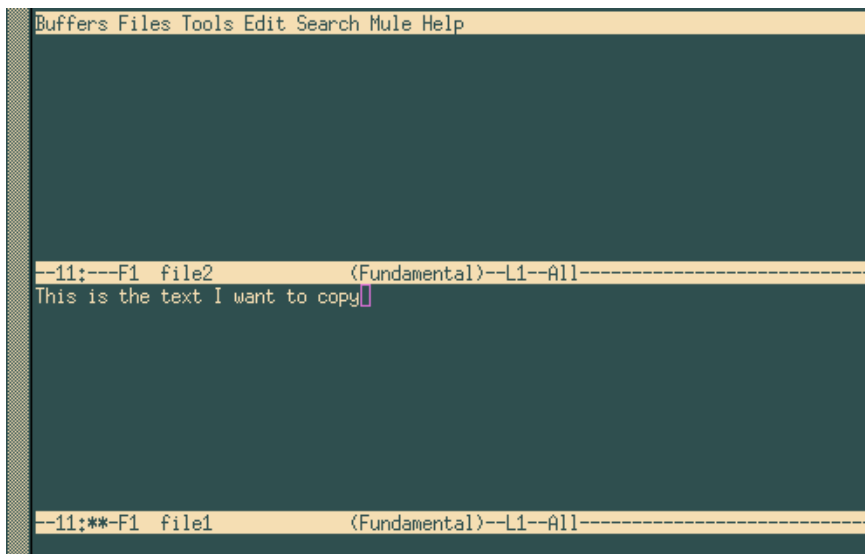


Figure 4-2. Emacs, avant la copie du bloc de texte

Il faut d'abord sélectionner le texte que nous voulons copier. Sous *X*, vous pouvez le faire à la souris et la région sélectionnée apparaîtra alors en surbrillance. Mais présentement, nous sommes en mode texte :-). Ici, nous voulons copier toute la phrase. Il faut marquer le début de la région. En supposant que le curseur soit à l'endroit où il est dans la figure ci-dessus, tapez d'abord `Ctrl+ESPACE` (`Control` + barre espace) : *Emacs* affichera alors le message `Mark set` dans le mini-tampon. Puis déplacez-vous en début de ligne avec `Ctrl+a` : la région sélectionnée pour copier ou couper est toute celle se situant entre la marque et la position actuelle du curseur, donc dans le cas présent, toute la ligne. Tapez ensuite `Alt-w` (pour copier) ou `Ctrl+w` (pour couper). Si vous copiez, *Emacs* reviendra alors brièvement à la position de la marque pour que vous visualisiez la région sélectionnée.

Puis, rendez-vous dans le tampon où vous voulez copier le texte, et tapez `Ctrl+y`, afin d'obtenir à l'écran ce que montre la figure 4-3.

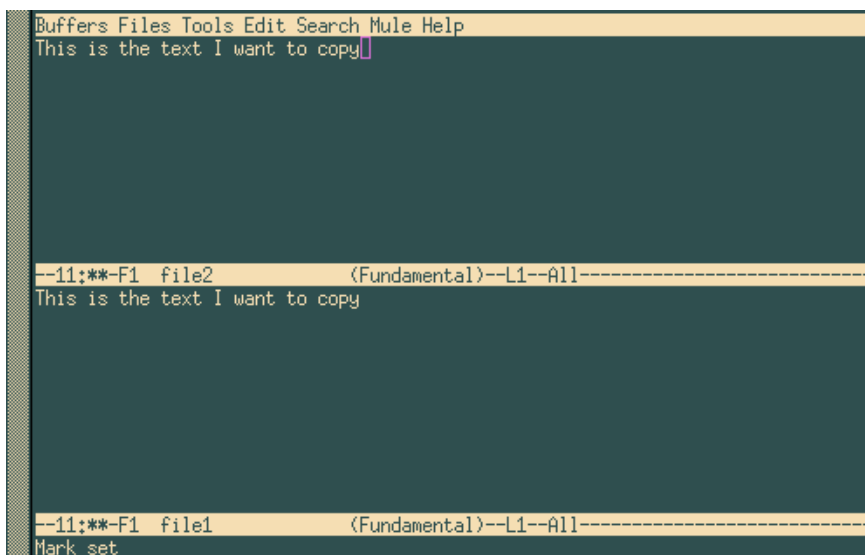


Figure 4-3. Emacs, après la copie du bloc de texte

En fait, vous venez de copier du texte dans le *kill ring* (soit « cercle des morts ») d'*Emacs* : ce *kill ring* contient toutes les régions copiées ou coupées depuis le lancement d'*Emacs*. **Toute** région qui vient d'être copiée ou coupée est mise en tête du *kill ring*. La séquence `Ctrl+y` ne fait que « coller » la région en tête : si vous voulez avoir accès aux autres régions, appuyez sur `Ctrl+y` puis sur `Alt-y` jusqu'à ce que vous tombiez sur le texte souhaité.

Pour rechercher du texte, placez-vous dans le tampon souhaité et tapez `Ctrl+s` : *Emacs* vous demande alors la chaîne à rechercher. Pour lancer une nouvelle recherche avec la même chaîne, toujours dans le tampon courant,

tapez `Ctrl+s` une nouvelle fois. Dès qu'*Emacs* arrive à la fin du tampon et ne trouve plus d'occurrence de la chaîne cherchée, vous pouvez de nouveau taper `Ctrl+s` pour recommencer la recherche depuis le début du tampon. Une pression sur la touche `Entrée` termine la recherche.

Pour rechercher et remplacer, tapez `Alt-%`. *Emacs* vous demande la chaîne à rechercher, par quoi elle doit être remplacée, et vous interroge pour chaque occurrence repérée.

Une dernière chose bien utile : `Ctrl+x` u permet d'annuler l'opération précédente. Vous pouvez annuler autant d'opérations que vous le souhaitez.

#### 4.1.5. Quitter Emacs

Pour ce faire, le raccourci est `Ctrl+x Ctrl+c`. *Emacs* vous demandera alors s'il faut enregistrer les modifications effectuées dans les tampons si vous ne les avez pas sauvegardées.

### 4.2. Vi : l'ancêtre

*Vi* a été le premier éditeur plein écran existant. Étrangement, il représente un des principaux arguments à la fois des détracteurs d'*UNIX* et de ses défenseurs : s'il est compliqué à appréhender, c'est aussi un outil extrêmement puissant une fois maîtrisé. En ne tapant que quelques touches, un utilisateur de *Vi* peut déplacer des montagnes ! Mis à part *Emacs*, peu d'éditeurs de texte peuvent se vanter de cela.

La version incluse dans **Mandrake Linux** est en fait *vim*, pour *VI iMproved* (*VI aMélioré*), mais nous le nommerons *Vi* tout au long de ce chapitre.

#### 4.2.1. Mode d'insertion, mode commande, mode ex...

Tout d'abord, l'invocation : elle est exactement la même que pour *Emacs*. Reprenons donc nos deux fichiers et tapons :

```
$ vi fichier1 fichier2
```

À partir de là, vous vous retrouverez devant une fenêtre comme celle de la figure 4-4.

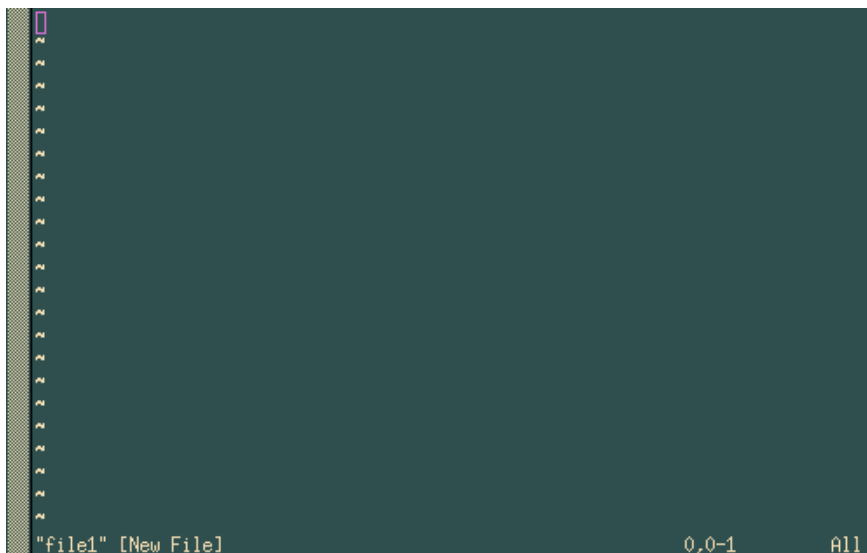


Figure 4-4. Situation de départ dans vim

Vous vous retrouvez alors en *mode commande* devant le premier fichier ouvert. Et là, ça se complique un tantinet :-> En mode commande, vous ne pouvez pas insérer de texte dans un fichier... Il vous faut pour cela passer en *mode insertion* et entrer l'une des commandes qui le permettent :

- **a** et **i** : pour insérer du texte respectivement derrière et devant le curseur (**A** et **I** insèrent du texte à la fin et au début de la ligne courante) ;

- **o** et **O** : pour insérer du texte respectivement au-dessous et au-dessus de la ligne courante.

En mode d'insertion, vous verrez la chaîne `--INSERT--` apparaître en bas de l'écran (de cette façon vous savez dans quel mode vous êtes). C'est dans ce mode et uniquement dans celui-ci que vous pouvez insérer du texte. Pour revenir en mode commande, appuyez sur la touche Échap.

En mode d'insertion, vous disposez des touches Backspace et Suppr pour effacer du texte à la volée. Pour vous déplacer dans le texte, aussi bien en mode commande qu'en mode insertion, vous disposez des touches fléchées. En mode commande, il existe également d'autres combinaisons de touches, que nous verrons plus loin.

Le mode `ex` est disponible en tapant le caractère : en mode commande : ce même : apparaîtra en bas de l'écran, le curseur s'y positionnera également et tout ce que vous tapez à la suite, suivi d'une pression sur Entrée, sera considéré par *Vi* comme une commande `ex`. Si vous effacez la commande jusqu'à « effacer » le :, vous revenez alors en mode commande et le curseur retrouvera sa place originelle.

Pour enregistrer les modifications faites dans un fichier vous taperez `:w` en mode commande. Si vous voulez enregistrer le contenu du tampon dans un autre fichier, tapez la séquence `:w <nom_du_fichier>`.

#### 4.2.2. Manipulation de tampons

Pour passer d'un fichier à l'autre dans un même tampon, parmi ceux ayant été passés sur la ligne de commande, il vous faut taper `:next` pour passer au fichier suivant et `:prev` pour retourner au fichier précédent. Vous pouvez aussi vous servir de `:e <nom_de_fichier>`, qui permet à la fois de changer vers le fichier désiré si celui-ci est déjà ouvert, ou bien d'ouvrir un autre fichier. Vous disposez là aussi du complément.

Comme avec *Emacs*, vous pouvez avoir plusieurs tampons visibles à l'écran. Pour cela, utilisez la commande `:split`.

Pour changer de tampon, tapez `Ctrl+w j` pour passer au tampon du dessous ou `Ctrl+w k` pour retourner au tampon du dessus. Vous pouvez utiliser également les touches fléchées vers le haut ou vers le bas en lieu et place de `j` ou `k`. La commande `:close` cachera un tampon, la commande `:q` le fermera.

Attention, *Vi* est tatillon : si vous tentez de cacher ou de fermer un tampon dont les changements n'ont pas été sauvegardés, la commande ne sera pas effectuée et vous aurez ce message :

No write since last change (use! to override)

Soit : pas de sauvegarde depuis le dernier changement (utilisez `!` pour forcer la commande). Dans ce cas, il n'y a qu'une solution : faire ce qui est indiqué :-). Tapez `:q!` ou `:close!`.

#### 4.2.3. Édition de texte et commandes de déplacement

Outre les touches Backspace et Suppr dans le mode d'édition, *Vi* dispose de beaucoup de commandes pour effacer, copier, coller, remplacer du texte en mode commande. Nous en verrons ici quelques-unes. Toutes les commandes présentées ici sont en fait séparées en deux parties : l'action à effectuer et sa portée. L'action peut être :

- **c** : pour remplacer (*Change*). L'éditeur efface le texte demandé et repasse en mode d'insertion après cette commande ;
- **d** : pour effacer (*Delete*) ;
- **y** : pour copier (*Yank*), nous verrons cela dans la section suivante ;
- **.** : reproduit la dernière action effectuée.

La portée désigne le groupe de caractères sur lequel la commande doit agir. Ces mêmes commandes de portée, entrées telles quelles en mode commande, correspondent à des déplacements :

- **h, j, k, l** : un caractère à gauche, en bas, en haut, à droite<sup>3</sup> ;
- **e, b, w** : jusqu'à la fin (resp. au début) du mot courant, jusqu'au début du mot suivant ;
- **^, 0, \$** : jusqu'au premier caractère non blanc de la ligne courante, jusqu'au début de la ligne courante, et jusqu'à la fin de la ligne courante ;

3. Un raccourci pour `d1` (effacer un caractère à droite) est `x` ; un raccourci pour `dh` est `X` ; `dd` efface la ligne courante.

- `f<x>` : jusqu'à la prochaine occurrence du caractère `<x>` ; par exemple `fe` déplace le curseur jusqu'à la prochaine occurrence du caractère `e` ;
- `/<chaîne>`, `?<chaîne>` : jusqu'à la prochaine occurrence de la chaîne ou expression régulière `<chaîne>`, et de même en remontant dans le fichier ; par exemple, `/toto` déplace le curseur jusqu'à la prochaine occurrence du mot `toto` ;
- `{ et }` : jusqu'au début, jusqu'à la fin, du paragraphe ;
- `G, H` : jusqu'à la fin du fichier, jusqu'au début de l'écran.

Chacun de ces caractères de portée ou commandes de déplacement peut être précédé d'un nombre de répétition quand cela a un sens. `G` référence le numéro de ligne dans le fichier. À partir de là, vous pouvez faire toutes sortes de combinaisons. Quelques exemples :

- `6b` : se déplace 6 mots en arrière ;
- `c8fk` : efface tout le texte jusqu'à la huitième occurrence du caractère `k` puis passe en mode insertion ;
- `91G` : va à la ligne 91 du fichier ;
- `d3$` : efface jusqu'à la fin de la ligne courante plus les deux lignes suivantes.

Ce ne sont certes pas des commandes très intuitives, mais comme toujours, le meilleur moyen est de se familiariser avec elles par la pratique. En tout cas, vous pouvez voir que l'expression « déplacer des montagnes en quelques touches » n'est pas si exagérée que ça :-)

#### 4.2.4. Couper, copier, coller

*Vi* dispose d'une commande que nous avons déjà vue pour copier du texte : la commande `y`. Pour couper du texte, utilisez tout simplement la commande `d`. Vous disposez de 27 mémoires pour `y` stocker du texte : une mémoire anonyme et 26 mémoires portant le nom des 26 lettres minuscules de l'alphabet.

Pour utiliser la mémoire anonyme, il suffira d'entrer la commande telle quelle. Ainsi, la commande `y12w` copie dans la mémoire anonyme les 12 mots après le curseur.<sup>4</sup> Utilisez `d12w` si vous voulez couper cette zone.

Pour utiliser l'une des 26 mémoires nommées, utilisez la séquence "`<x>`" avant la commande, où `<x>` désigne le nom de la mémoire. Ainsi, pour copier les mêmes 12 mots dans la mémoire `k`, on écrirait `"ky12w`, et `"kd12w` si on veut les couper.

Pour coller le contenu de la mémoire anonyme, vous disposez des commandes `p` ou `P` (pour *Paste*, soit coller), ce qui insérera le texte respectivement après le curseur ou devant le curseur. Pour coller le contenu d'une mémoire nommée, utilisez de la même façon "`<x>p` ou "`<x>P` (par exemple `"dp` collera le contenu de la mémoire `d` après le curseur).

Prenons l'exemple de la figure 4-5.

---

4. Si le curseur se trouvait au début du premier mot, évidemment !



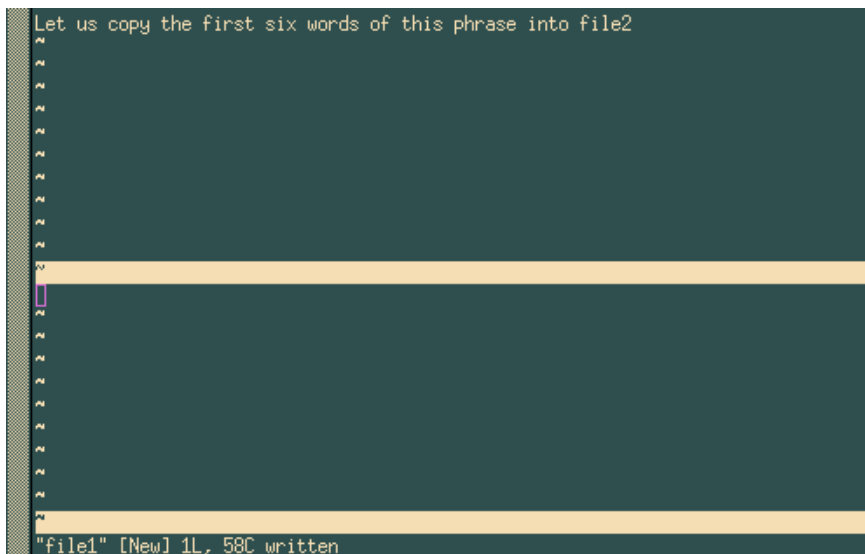


Figure 4-5. vim, avant la copie du bloc de texte

Pour effectuer cette action, on va donc :

- recopier les 6 premiers mots de la phrase dans la mémoire **r** (par exemple) : `"ry6w5` ;
- passer dans le tampon `fichier2`, qui se situe dessous : `Ctrl+w j` ;
- coller le contenu de la mémoire **r** devant le curseur : `"rp`.

Le résultat, présenté dans la figure 4-6, est bien celui qui est attendu.

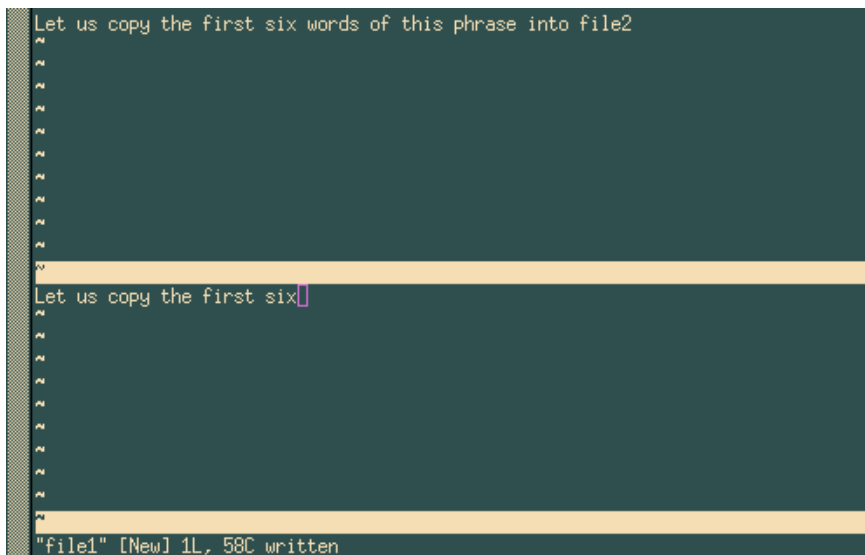


Figure 4-6. vim, après la copie du bloc de texte

Quant à la recherche de texte, elle s'avère très simple : en mode commande, il suffit de taper `/` suivi de la chaîne à rechercher et d'une pression sur la touche Entrée. Par exemple, `/partie` recherchera la chaîne `partie` à partir de la position courante. Un appui sur **n** conduit à la prochaine occurrence et si vous arrivez à la fin du fichier, la recherche recommencera depuis le début. Pour rechercher en remontant dans le fichier, il faut remplacer `/` par `?`.

5. En anglais, `y6w` donne littéralement : « *Yank 6 words* », soit extirper 6 mots, et donc copier 6 mots en français.

#### 4.2.5. Quitter Vi

Pour quitter, la commande est :q (en fait, cette commande ferme le tampon actif, comme nous l'avons déjà vu, mais si c'est le seul tampon présent, vous quittez *Vi*). Il existe un raccourci : la plupart du temps, on n'édite qu'un seul fichier. Pour quitter, vous avez deux solutions :

- :wq pour sauvegarder les modifications et quitter (solution plus rapide : ZZ) ;
- :q! pour quitter sans enregistrer.

Par extension, vous aurez deviné que si vous avez plusieurs tampons, :wq écrira le tampon actif puis le fermera.

#### 4.3. Un dernier mot...

Il vous a peut-être semblé que nous en disions beaucoup plus que nécessaire (après tout, le but premier était d'éditer un fichier texte). Il s'agissait aussi de vous faire découvrir certaines des possibilités de chacun de ces éditeurs. Il reste encore bien des choses à raconter, comme en témoignera le grand nombre de livres consacrés à l'un ou à l'autre de ces éditeurs.

Prenez le temps de bien digérer ces informations et jetez votre dévolu sur l'un ou l'autre ou encore, n'apprenez que ce que vous jugez nécessaire. Mais, le jour où vous voudrez explorer leur potentiel plus avant, vous saurez que c'est tout à fait faisable :-)

## Chapitre 5. Les utilitaires en ligne de commande

Ce chapitre présentera un petit nombre d'outils en ligne de commande qui peuvent s'avérer utiles au quotidien. Bien sûr, passez ce chapitre si vous pensez n'utiliser qu'un environnement graphique. Un rapide coup d'œil peut vous faire changer d'avis.

Chaque commande sera illustrée par un exemple, mais comme exercice, vous pourrez lui trouver des usages encore plus intéressants.

### 5.1. Opérations sur les fichiers et filtres

La plupart du travail en ligne de commande s'applique à des fichiers. Dans cette section nous allons voir comment surveiller et filtrer l'information dans les fichiers, extraire des informations spécifiques avec une simple commande, et trier des fichiers

#### 5.1.1. cat, tail, head, tee: Afficher des fichiers

Ces commandes ont pratiquement toute la même syntaxe :

```
commande [option(s)] [fichier(s)]
```

et peuvent être utilisées dans un tube. Toutes sont utilisées pour imprimer une partie d'un fichier selon certains critères.

L'utilitaire cat agglomère des fichiers et les imprime sur la sortie standard. c'est une des commandes les plus utilisées. Vous pouvez utiliser :

```
# cat /var/log/mail/info
```

pour afficher, par exemple, le contenu du journal (*log*) du démon de courrier sur la sortie standard<sup>1</sup>. la commande cat a une option très utile (-n) qui permet de numéroté les lignes affichées.

Certains fichiers, comme les fichiers journaux des démons (s'ils sont utilisés) sont souvent de taille énorme<sup>2</sup> et les afficher dans leur intégralité à l'écran n'a aucun intérêt. Vous voudrez souvent ne consulter que le début d'un fichier. Vous pouvez pour cela utiliser la commande head. Elle affiche les 10 première lignes par défaut. Ainsi, la commande :

```
# head /var/log/mail/info
```

affichera les dix premières lignes du fichier /var/log/mail/info. Si vous souhaitez n'afficher que les deux premières lignes vous pouvez utiliser la commande :

```
# head -n2 /var/log/mail/info
```

La commande tail est similaire à head, mais affiche les dernières lignes d'un fichier. Cette commande :

```
# tail /var/log/mail/info
```

affiche les 10 dernières lignes de /var/log/mail/info. Comme avec head vous pouvez imprimer les deux dernières lignes :

```
# tail -n2 /var/log/mail/info
```

Vous pouvez aussi utiliser ces commandes de concert. Par exemple, si vous ne souhaitez qu'afficher les lignes 9 et 10, vous pouvez lancer :

---

1. Certains exemples de cette section sont basés sur des tâches réelles avec des fichiers journaux de certains services. Assurez vous que le démon *syslogd* est lancé (il permet aux autres démons d'écrire leurs journaux), ainsi que le démon dont il est question (ici *Postfix*) et que vous travaillez en tant que root. Vous pouvez quoi qu'il en soit toujours appliquer les exemples à d'autres fichiers.

2. Par exemple, le fichier /var/log/mail/info enregistre toutes les informations à propos des messages envoyés, messages rapatriés par les utilisateurs avec le protocole *POP*, etc.

```
# head /var/log/mail/info | tail -n2
```

où la commande `head` sélectionne les 10 premières lignes du fichier, les passe à la commande `tail` au travers d'un tube (`|`) qui n'en retourne que les deux dernières lignes. De la même façon vous pouvez sélectionner toutes les lignes à partir de la ligne 20 :

```
# tail -n20 /var/log/mail/info |head -n1
```

Nous disons ici à `tail` de sélectionner les 20 dernières lignes du fichier, puis de les passer à `head`. Cette dernière en imprime alors la première ligne.

Supposons maintenant que nous souhaitions à la fois afficher et enregistrer le résultat de la commande précédente dans le fichier `resultats.txt`. L'utilitaire `tee` va nous y aider. Sa syntaxe est :

```
tee [option(s)] [file]
```

Nous changeons alors la commande précédente :

```
# tail -n20 /var/log/mail/info |head -n1|tee resultats.txt
```

Encore un exemple. Nous voulons sélectionner les 20 dernières lignes, les enregistrer dans `resultats.txt`, mais imprimer à l'écran seulement les premières de ces lignes. Nous écrivons alors :

```
# tail -n20 /var/log/mail/info |tee resultats.txt |head -n1
```

La commande `tee` a une option très utile (`-a`) qui permet d'ajouter les informations reçues à un fichier existant au lieu de l'écraser.

Revenons à la commande `tail`. Les fichiers comme les journaux changent souvent car les démons y rajoutent sans cesse de l'information sur leur activité. Si vous voulez alors surveiller de manière interactive des changements sur ces journaux, vous pouvez tirer avantage de l'une des options les plus utiles de `tail` : `-f` :

```
# tail -f /var/log/mail/info
```

Dans ce cas, tous les changements dans le fichier `/var/log/mail/info` sont affichés immédiatement à l'écran. Cette option est très utile pour surveiller l'activité de votre système. Par exemple, en regardant dans le journal `/var/log/messages`, vous pouvez surveiller les messages du système et plusieurs démons.

Dans la section qui suit nous allons voir comment nous pouvons utiliser `grep` comme filtre pour séparer les messages de *Postfix* des messages émanant des autres services.

### 5.1.2. `grep` : rechercher du texte dans un ou plusieurs fichier(s)

D'accord, le nom n'est pas intuitif mais son utilisation est très simple : il cherche un motif donné en argument dans un ou plusieurs fichiers. Sa syntaxe est :

```
grep [options] <motif> [un ou plusieurs fichier(s)]
```

Si on a précisé plusieurs noms de fichiers, ceux-ci apparaîtront en début des lignes et rempliront les critères du motif, à moins qu'on n'ait utilisé l'option `-h`. L'option `-l`, quant à elle, a pour effet de n'afficher que le nom des fichiers dont au moins une ligne remplit les conditions du motif. Le motif est une expression régulière, bien que la plupart du temps il ne consiste qu'en un simple mot. Les options les plus couramment utilisées sont les suivantes :

1. `-i` : rechercher en ignorant la casse (c'est-à-dire ignorer la différence entre majuscules et minuscules).
2. `-v` : inverser la recherche, donc trouver les lignes ne correspondant **pas** au motif.
3. `-n` : afficher le numéro de ligne pour chaque ligne trouvée.
4. `-w` : dit à `grep` que le motif doit correspondre à un mot entier. Attention, les caractères considérés comme pouvant faire partie d'un mot dépendent du réglage de la langue.

Reprenons notre exemple du fichier journal du démon de courrier. Nous souhaitons trouver toutes les lignes du fichier `/var/log/mail/info` qui contiennent la séquence « postfix » :

```
# grep postfix /var/log/mail/info
```

La commande *grep* peut être utilisée dans un tube. Nous pouvons ainsi obtenir le même résultat en faisant :

```
# cat /var/log/mail/info | grep postfix
```

Si nous voulons inverser la condition pour sélectionner les lignes qui ne contiennent pas la séquence « postfix », nous utilisons l'option *-v* :

```
# grep -v postfix /var/log/mail/info
```

Supposons que nous voulons trouver tous les messages signalant un message envoyé avec succès. Dans ce cas, nous devons filtrer les lignes du journal provenant du démon de courrier (contenant donc la séquence « postfix ») contenant la séquence indiquant un envoi réussi (« status=sent ») :

```
# grep postfix /var/log/mail/info |grep status=sent
```

Dans ce cas, *grep* est utilisé deux fois. C'est possible mais pas très élégant. Nous pouvons obtenir le même résultat en utilisant l'utilitaire *fgrep*. Pour cela nous devons créer le fichier *sequences.txt* contenant les séquences écrites sur une colonne. un tel fichier peut être obtenu de cette façon :

```
# echo -e 'status=sent\npostfix' >./sequences.txt
```

Nous appelons alors une commande utilisant le fichier de séquences *sequences.txt* et l'utilitaire *fgrep* au lieu du double appel à *grep* :

```
# fgrep -f ./sequences.txt /var/log/mail/info
```

le fichier *./sequences.txt* peut contenir autant de séquences de filtrage que vous souhaitez. Chacune d'elles doit se trouver sur une nouvelle ligne. Par exemple, pour sélectionner les messages correctement envoyés à *pierre@mandrakesoft.com*, il suffira de rajouter cette adresse dans notre fichier *./sequences.txt* :

```
# echo 'pierre@mandrakesoft.com' >>./sequences.txt
```

et relancer la commande précédente.

Il est évident que vous pouvez combiner *grep* avec *tail* et *head*. Si nous souhaitons trouver l'avant dernier message envoyé à Pierre nous tapons :

```
# fgrep -f ./sequences.txt /var/log/mail/info | tail -n2 | head -n1
```

Nous appliquons ici le filtre construit précédemment et dirigeons le résultat dans un tube pour les commandes *tail* et *head*. Elles se chargent de sélectionner l'avant dernière ligne des données reçues.

### 5.1.3. wc: Compter des éléments de fichier

la commande *wc* (*Word Count* : compteur de mots) est utilisée pour compter le nombre de lignes et de mots d'un fichier. Elle peut aussi servir à compter le nombre d'octets, caractères, et la longueur de la ligne la plus longue. Sa syntaxe est :

```
wc [option(s)] [fichier(s)]
```

Les options suivantes sont particulièrement utiles :

- *-l*: affiche le nombre de lignes ;
- *-w*: affiche le nombre de mots ;
- *-m*: affiche le nombre total de caractères;
- *-c*: affiche le nombre d'octets ;
- *-L*: affiche la longueur de la plus longue ligne du texte d'entrée.

la commande *wc* affiche par défaut le nombre de lignes, mots et caractères du fichier fourni. Voici quelques exemples :

Si nous souhaitons connaître le nombre d'utilisateurs sur notre système, nous pouvons taper :

```
$wc -l /etc/passwd
```

Si nous souhaitons connaître le nombre de processeurs sur notre système :

```
$grep "model name" /proc/cpuinfo |wc -l
```

Dans la section précédente nous avons obtenu une liste de messages correspondants aux séquences du fichier `./sequences.txt`. Si nous souhaitons connaître le nombre de ces messages, nous pouvons rediriger les résultats de notre filtre dans un tube pour la commande `wc` :

```
# fgrep -f ./sequences.txt /var/log/mail/info | wc -l
```

#### 5.1.4. sort: Trier

Voici la syntaxe de cet utilitaire de tri très puissant<sup>3</sup>:

```
sort [option(s)] [fichier(s)]
```

Considérons le tri du fichier `/etc/passwd`. Comme vous pouvez voir :

```
$ cat /etc/passwd
```

le fichier `/etc/passwd` n'est pas trié. Nous voulons le trier par champ `login` (nom d'utilisateur) :

```
$ sort /etc/passwd
```

La commande `sort` trie les données par ordre croissant sur le premier champs (dans notre cas, le `login`) par défaut. Si nous voulons trier les données par ordre décroissant il faut utiliser l'option `-r` :

```
$ sort -r /etc/passwd
```

Chaque utilisateur a son propre UID (identifiant numérique de l'utilisateur) écrit dans le fichier `/etc/passwd`. Trions donc les utilisateurs selon leur UID :

```
$ sort /etc/passwd -t":" -k3 -n
```

Nous avons utilisé ici les options suivantes de `sort` :

- `-t":"`: indique à `sort` que le séparateur de champs est le symbole `":"` ;
- `-k3`: indique que le tri doit s'effectuer sur la troisième colonne ;
- `-n`: indique que le tri doit s'effectuer sur des données numérique et non pas alphabétiques.

Et par ordre décroissant :

```
$ sort /etc/passwd -t":" -k3 -n -r
```

Notez bien ces deux options importante de `sort` :

- `-u`: fourni un ordre strict : les doublons sont écartés ;
- `-f`: ignore la casse (ne fait pas de différence entre minuscules et majuscules).

Finalement si nous voulons trouver l'utilisateur ayant l'identifiant UID le plus élevé :

```
$ sort /etc/passwd -t":" -k3 -n |tail -n1
```

Nous trions le fichier `/etc/passwd` par ordre d'UID ascendant, et redirigeons le résultat sur un tube pour la commande `tail` qui ne laisse passer que la première ligne.

---

3. Nous ne parlons que brièvement de `sort` ici car des livres entiers pourraient être écrit sur le sujet.

## 5.2. find : rechercher des fichiers selon certains critères

`find` est un utilitaire *UNIX* de longue date. Son but est de parcourir récursivement un ou plusieurs répertoires et d'y trouver des fichiers correspondant à un certain ensemble de critères. Bien qu'il soit très utile, sa syntaxe est vraiment complexe, et l'utiliser requiert une certaine habitude. La syntaxe générale est :

```
find [options] [répertoires] [critère] [action]
```

Si vous ne spécifiez aucun répertoire, `find` recherchera dans le répertoire courant. L'absence de critère équivaut à « vrai », et donc tous les fichiers seront trouvés. Les options, les critères et les actions possibles sont si nombreux que nous n'en mentionnerons que quelques-uns ici. Commençons par les options :

- `-xdev` : ne pas étendre la recherche aux répertoires se trouvant sur d'autres systèmes de fichiers.
- `-mindepth <n>` : descendre d'au moins `<n>` niveaux au dessous du répertoire de recherche avant de chercher des fichiers.
- `-maxdepth <n>` : rechercher les fichiers se trouvant au plus `n` niveaux au dessous du répertoire de recherche.
- `-follow` : suivre les liens symboliques s'il pointent vers des répertoires. Par défaut, `find` ne les suivra pas.
- `-daystart` : quand il est fait usage de tests relatifs à la date et à l'heure (voir ci-dessous), prendre le début de la journée courante comme repère au lieu de la marque par défaut (24 heures avant l'heure courante).

Un critère peut être un ou plusieurs tests *atomiques*; quelques tests utiles sont :

- `-type <type>` : rechercher un type de fichiers donné ; `<type>` peut être : `f` (fichier normal), `d` (répertoire), `l` (lien symbolique), `s` (*socket*), `b` (fichier en mode bloc), `c` (fichier en mode caractère) ou `p` (tube nommé).
- `-name <motif>` : trouver les fichiers dont les noms correspondent au `<motif>` donné. Avec cette option, `<motif>` est traité comme un *motif d'englobement* du *shell* (voir le chapitre *Motifs d'englobement du shell*, page 16).
- `-iname <motif>` : comme `-name`, mais ne tient pas compte de la casse.
- `-atime <n>`, `-amin <n>` : trouver les fichiers dont la dernière date d'accès remonte à `<n>` jours (`-atime`) ou `<n>` minutes (`-amin`). Vous pouvez aussi spécifier `+<n>` ou `-<n>`, auquel cas la recherche sera effectuée pour des fichiers dont la date d'accès remonte à au plus ou au moins `<n>` jours/minutes.
- `-anewer <fichier>` : trouver les fichiers dont la dernière date d'accès est plus récente que celle du fichier `<fichier>`
- `-ctime <n>`, `-cmin <n>`, `-cnewer <fichier>` : même chose que pour `-atime`, `-amin` et `-anewer`, mais s'applique à la dernière date de changement du contenu des fichiers.
- `-regex <motif>` : comme pour `-name`, mais `motif` est traité comme une *expression régulière*.
- `-iregex <motif>` : comme `-regex`, mais sans tenir compte de la casse.

Beaucoup d'autres tests existent, référez-vous à `find(1)` pour plus de détails. Pour combiner ces tests, vous pouvez utiliser :

- `<c1> -a <c2>` : vrai si `<c1>` et `<c2>` sont vrais tous les deux; `-a` est implicite, donc vous pouvez utiliser `<c1> <c2> <c3> ...` si vous voulez que tous les tests `<c1>`, `<c2>`, ... soient vérifiés.
- `<c1> -o <c2>` : vrai si l'un de `<c1>` ou `<c2>` est vrai, ou les deux. Notez que `-o` a une *priorité* moins grande que `-a`, donc si vous voulez, par exemple, que les fichiers correspondant aux critères `<c1>` ou `<c2>` et qu'ils vérifient obligatoirement le critère `<c3>`, vous devrez utiliser des parenthèses et écrire `( <c1> -o <c2> ) -a <c3>`. Vous devez *échapper* (désactiver) les parenthèses, sans quoi le *shell* les prendra en compte dans son interprétation !
- `-not <c1>` : inverse le test `<c1>`, donc `-not <c1>` est vrai si `<c1>` est faux.

Enfin, vous pouvez demander une action précise pour chaque fichier retrouvé. Les plus fréquentes sont :

1. `-print` : écrit seulement le nom de chaque fichier sur la sortie standard. C'est l'action par défaut si vous n'en spécifiez aucune.
2. `-ls` : affiche l'équivalent d'un `ls -l` sur chaque fichier trouvé sur la sortie standard.

3. `-exec <commande>` : exécute la commande `<commande>` sur chaque fichier trouvé. La ligne de commande `<commande>` doit se terminer par un `;`, que vous devez désactiver de telle sorte que le *shell* ne l'interprète pas la position du fichier dans la commande est repérée par `{}`. Regardez les exemples d'utilisation pour bien comprendre.
4. `-ok <commande>` : même chose que `-exec` mais demander confirmation pour chaque commande.

Toujours là ? Alors, maintenant entraînons-nous un peu, c'est après tout la meilleure façon de maîtriser ce monstre ! On supposera ceci : vous désirez trouver tous les répertoires dans `/usr/share`. Tapez alors :

```
find /usr/share -type d
```

Admettons que vous avez un serveur HTTP, que tous vos fichiers HTML sont dans `/var/www/html`, qui s'avère aussi être votre répertoire courant. Il s'agit de chercher tous les fichiers qui n'ont pas été modifiés depuis... un mois, par exemple. Les pages proviennent, incidemment, de différents auteurs : certains fichiers auront une extension `html` et d'autres, l'extension `htm`. Vous voulez lier ces fichiers dans le répertoire `/var/www/obsolete`. Vous taperez alors<sup>4</sup> :

```
find \( -name "*.htm" -o -name "*.html" \) -a -ctime -30 \  
-exec ln {} /var/www/obsolete \;
```

D'accord, cette exemple est un peu compliqué et requiert quelques explications. Le critère est :

```
\( -name "*.htm" -o -name "*.html" \) -a -ctime -30
```

qui accomplit ce qu'on lui demande : il recherche tous les fichiers dont le nom se termine soit par `.htm`, soit par `.html` (`\( -name "*.htm" -o -name "*.html" \)`), et (`-a`) qui n'ont pas été modifiés dans les derniers 30 derniers jours, ce qui représente en gros un mois (`-ctime -30`). Notez les parenthèses : elles sont nécessaires ici, parce que `-a` a une priorité plus grande. En leur absence, tous les fichiers se terminant par `.htm` auraient été sortis, plus tous les fichiers se terminant par `.html` n'ayant pas été modifiés depuis un mois, ce qui n'est pas ce que nous voulons. Notez également que les parenthèses sont désactivées par rapport au *shell* : si nous avions mis `( .. )` à la place de `\( .. \)`, le *shell* les aurait interprétés et aurait tenté d'exécuter `-name "*.htm" -o -name "*.html"` dans un sous-*shell*... Une autre solution aurait été de mettre les parenthèses entre doubles ou simples apostrophes, mais une barre oblique inverse (*backslash*) est préférable ici dans la mesure où nous ne devons isoler qu'un seul caractère.

Et enfin, la commande doit être exécutée pour chacun des fichiers :

```
-exec ln {} /home/httpd/obsolete \;
```

Ici aussi, vous devez désactiver le `;` par rapport au *shell*, car autrement le *shell* l'interprétera comme un séparateur de commandes. Si vous ne le faites pas, `find` se plaindra qu'il manque un argument à `-exec`.

Un dernier exemple : vous avez un gros répertoire nommé `/shared/images`, contenant toutes sortes d'images. Régulièrement, vous utilisez la commande `touch` pour mettre à jour les dates d'un fichier nommé `stamp` dans ce répertoire, de façon à avoir une référence dans le temps. Vous voulez trouver tous les fichiers *JPEG* dans ce répertoire qui sont plus récents que le fichier `stamp`, et comme vous avez des images de diverses sources, ces fichiers ont des extensions `jpg`, `jpeg`, `JPG` ou `JPEG`. Vous voulez aussi éviter de rechercher dans le répertoire `old`. Vous voulez vous faire envoyer la liste de ces fichiers par courrier électronique, et votre nom d'utilisateur est pierre :

```
find /shared/images -cnewer      \  
  /shared/images/stamp         \  
-a -iregex ".*\.jpg?g"          \  
-a -not -regex ".*\/old\/.*" \  
  | mail pierre -s "Nouvelles images"
```

---

4. Notez que cet exemple requiert que `/var/www` et `/var/www/obsolete` soient sur le même système de fichier!



Bien sûr, cette commande n'est pas très utile si vous devez l'exécuter régulièrement car vous devrez l'entrer à chaque fois. Il est possible de le faire ainsi :

### 5.3. Programmation de démarrage de commandes

#### 5.3.1. crontab : exécuter des commandes périodiques

`crontab` est une commande qui vous permet d'exécuter des commandes à des intervalles de temps réguliers, avec l'avantage supplémentaire que vous n'avez pas à être connecté au système et que la sortie de ces commandes vous est envoyée par courrier électronique. Vous pouvez spécifier les intervalles en minutes, en heures, en jours et même en mois. `crontab` agira différemment en fonction des options :

1. `-l` : affiche votre fichier `crontab` courant.
2. `-e` : édite votre fichier `crontab`.
3. `-r` : élimine votre fichier `crontab` courant.
4. `-u <utilisateur>` : applique les options ci-dessus à l'utilisateur `<utilisateur>`. Seul `root` est autorisé à faire cela.

Commençons par éditer un fichier `crontab`. En tapant `crontab -e`, vous vous retrouverez en face de votre éditeur de texte préféré si vous avez initialisé la variable d'environnement `EDITOR` ou `VISUAL`, autrement c'est `Vi` qui sera utilisé. Une ligne dans un fichier `crontab` est composée de six champs. Les cinq premiers sont les intervalles de temps en minutes, heures, jours dans le mois, mois et jours dans la semaine. Le sixième champ est la commande à exécuter. Les lignes commençant par un `#` sont considérées comme des commentaires et seront ignorées par `crond` (le programme en charge d'exécution des fichiers `crontab`). Voici un exemple de fichier `crontab` :



afin de pouvoir imprimer l'extrait qui suit dans une police de caractères lisible, il nous a fallu ventiler les lignes les plus longues. C'est pourquoi certaines portions du texte doivent en réalité n'occuper qu'une seule ligne. Quand vous verrez le caractère `\` terminer une ligne, cela signifiera qu'il faut considérer que la ligne se poursuit au-delà. Cette convention fonctionne dans les fichiers de type `Makefile` et dans le `shell`, ainsi que dans d'autres cadres.

```
# Si vous ne voulez pas recevoir de courrier,
# "décommentez" la ligne suivante
#MAILTO=""
#
# Faire un rapport de toutes les nouvelles images
# à 14h tous les deux jours, en partant de
# l'exemple ci-dessus --- après ceci,
# "retoucher" le fichier "stamp". Le "%" est
# traité comme un retour chariot, cela vous
# permet de mettre plusieurs commandes sur la
# même ligne.
0 14 */2 * * find /shared/images          \
-cnewer /shared/images/stamp              \
-a -iregex ".*\.jpe?g"                    \
-a -not -regex                             \
  ".*old/.*"%touch /shared/images/stamp
#
# Jouer une mélodie tous les ans à Noël :)
0 0 25 12 * mpg123 $HOME/musiques/joyeux_noel.mp3
#
# Imprimer la liste des courses tous les mardis
# à 17 heures...
0 17 * * 2 lpr $HOME/liste-courses.txt
```

Il y a plusieurs autres moyens de spécifier des intervalles que ceux mentionnés dans l'exemple. Par exemple, vous pouvez spécifier un ensemble de valeurs *discrètes* séparées par des virgules (1, 14, 23) ou un intervalle (1-15), ou même combiner les deux (1-10, 12-20), éventuellement avec un pas (1-12, 20-27/2). Maintenant, il vous reste à trouver des commandes utiles à y mettre!

## 5.4. at : programmer une commande une seule fois

Vous pouvez aussi vouloir exécuter une commande à un jour donné, mais pas régulièrement. Par exemple, vous voulez vous rappeler un rendez-vous, aujourd'hui à 18 heures. Vous aimeriez que l'on vous le rappelle vers 17h30, par exemple. Vous employez *X*. *at* est ce qu'il vous faut ici :

```
$ at 5:30pm
# Vous vous retrouvez en face de l'invite de at
at> xmessage "C'est l'heure! rendez-vous à 18h"
# Tapez C-d pour sortir
at> <EOT>
$
```

Vous pouvez spécifier la date de différentes manières :

- *now* +<intervalle> : signifie « maintenant », plus un intervalle optionnel. Aucun intervalle signifie « maintenant ». La syntaxe pour l'intervalle est <n> (minutes|hours|days|weeks|months) (*minutes, heures, jours, semaines, mois*). Par exemple, vous pouvez spécifier *now* + 1 hour (dans une heure), *now* + 3 days (dans trois jours) et ainsi de suite.
- <heure> <jour> : spécifier la date en entier. Le paramètre <heure> est obligatoire. *at* est très libéral dans ce qu'il accepte : vous pouvez par exemple taper 0100, 04:20, 2am, 0530pm, 1800, ou une des trois valeurs spéciales : *noon* (*midi*), *teatime* (*l'heure du thé, 16h*) ou *midnight* (*minuit*). Le paramètre <jour> est optionnel. Vous pouvez aussi le spécifier de différentes manières : 12/20/2001 par exemple, notation américaine pour le 20 décembre 2001, ou à l'européenne, 20.12.2001. Vous pouvez omettre l'année mais, dans ce cas, seule la notation européenne est acceptée : 20.12. Vous pouvez aussi spécifier le mois par son abréviation en anglais : Dec 20 ou 20 Dec sont tous les deux valides.

*at* accepte aussi différentes options :

- -l : affiche la liste des commandes déjà programmées ; le premier champ est le numéro de la commande. C'est équivalent à la commande *atq*.
- -d <n> : enlever la commande numéro <n> de la liste. Vous pouvez obtenir les numéros avec *atq*. C'est équivalent à la commande *atrm* <n>.

Comme d'habitude, voyez la page de manuel *at*(1) pour plus d'options.

## 5.5. Archivage et compression de données

### 5.5.1. tar : Tape ARchiver (archivageur sur bandes)

Bien que nous ayons déjà vu une utilisation de *tar* dans le chapitre *Installation d'un logiciel libre*, page 69, nous n'avons pas expliqué son fonctionnement. C'est maintenant ce que nous nous proposons de faire. De même que *find*, *tar* est un utilitaire *UNIX* de longue date, et sa syntaxe est un peu spéciale :

```
tar [options] [fichiers...]
```

Voici maintenant une liste d'options. Notez que toutes celles-ci ont une option longue équivalente, mais vous devrez vous référer à la page de manuel pour cela car elles ne sont pas listées ici. Et bien sûr, il y a encore bien d'autres options non listées ici.



le tiret initial (-) des options courtes est maintenant désuet pour la commande `tar`, sauf après une option longue.

- `c` : à utiliser pour créer de nouvelles archives.
- `x` : à utiliser pour extraire des fichiers depuis une archive existante.
- `t` : affiche la liste des fichiers d'une archive existante.
- `v` : affichera simplement la liste des fichiers au fur et à mesure qu'ils sont ajoutés à une archive ou extraits d'une archive ; ou, en conjonction avec l'option `t` ci-dessus, produira un affichage long des fichiers au lieu d'un affichage court.
- `f <fichier>` : pour créer une archive de nom `<fichier>`, extraire depuis l'archive `<fichier>` ou faire une liste des fichiers de l'archive `<fichier>`; si cette option n'est pas disponible, le fichier par défaut sera `/dev/rmt0`, qui est généralement le fichier spécial associé à un *streamer*. Si le paramètre `fichier` est un tiret (-), l'entrée ou la sortie (selon que vous extrayez depuis une archive ou en créez une), sera associée à l'entrée standard ou la sortie standard.
- `z` : énonce à `tar` que l'archive à créer devra être compressée avec `gzip`, ou que l'archive depuis laquelle on fait l'extraction est compressée avec `gzip`.
- `j` : même chose que pour `z`, mais le programme utilisé pour la compression est `bzip2`;
- `p` : lors de l'extraction des fichiers d'une archive, préserve tous les attributs de fichiers, y compris la propriété, la dernière date d'accès et ainsi de suite ; très utile pour les sauvegardes de systèmes de fichiers.
- `r` : ajoute la liste des fichiers donnée sur la liste de commande à une archive existante ; notez que l'archive à laquelle vous voulez ajouter des fichiers ne doit **pas** être compressée !
- `A` : fusionne les archives données sur la ligne de commande à celle mentionnée avec l'option `f`. De même que pour l'option `r`, les archives ne doivent pas être compressées pour que cela fonctionne.

Il y a en fait beaucoup, beaucoup d'autres options. Référez-vous à la page de manuel `detar(1)` pour en obtenir une liste complète (entre autres l'option `d`, par exemple). Maintenant, passons à la pratique ! Supposons que vous vouliez créer une archive de toutes les images dans le répertoire `/shared/images`, compressée avec `bzip2`, de nom `images.tar.bz2` et située dans votre répertoire personnel. Vous taperez alors :

```
#
# Note: vous devez être dans le répertoire
#   contenant les fichiers de l'archive!
#
$ cd /shared
$ tar cjf ~/images.tar.bz2 images/
```

Comme vous le constatez, nous avons utilisé trois options ici : `c` a indiqué à `tar` de créer une archive ; `j` lui énonce que nous voulons qu'elle soit compressée avec `bzip2`, et `f ~/images.tar.bz2` lui signale que l'archive devait être créée dans notre répertoire personnel, avec le nom `images.tar.bz2`. Maintenant, il faut peut-être vérifier si l'archive est valide. Ceci se fera simplement en affichant la liste de ses fichiers :

```
#
# Retour à notre répertoire personnel
#
$ cd
$ tar tjvf images.tar.bz2
```

Ici, nous avons demandé à `tar` d'afficher la liste (`t`) des fichiers de l'archive `images.tar.bz2` (`f images.tar.bz2`), en ayant averti que cette archive était compressée avec `bzip2` (`j`), et que nous voulions un format d'affichage long (`v`). Maintenant, supposons que vous ayez effacé le répertoire des images. Heureusement, votre archive est intacte ! Vous voulez l'extraire de sa place originelle dans `/shared`. Mais comme vous ne voulez pas casser votre commande `find` pour trouver les nouvelles images, vous devez préserver les attributs de tous les fichiers :

```
#  
# Rendez-vous dans le répertoire où vous voulez  
# extraire  
#  
$ cd /shared  
$ tar jxpf ~/images.tar.bz2
```

Et voilà le travail !

Maintenant, supposons que vous vouliez seulement extraire le répertoire `images/cars` de l'archive. Vous pouvez alors taper ceci :

```
$ tar jxf ~/images.tar.bz2 images/cars
```

Au cas où cela vous inquiéterait, il n'y a pas de quoi ! Si vous essayez d'archiver des fichiers spéciaux, `tar` les prendra tels qu'ils sont, des fichiers spéciaux, et n'ira pas chercher leur contenu. Donc oui, vous pouvez mettre sans risque `/dev/mem` dans une archive :-). La gestion des liens, d'autre part, s'accomplit aussi correctement ; donc là non plus, pas d'inquiétude à avoir. Pour les liens symboliques, regardez également l'option `h` dans la page de manuel.

### 5.5.2. bzip2 et gzip : compression de données

Vous vous souvenez que nous avons déjà parlé de ces deux programmes quand nous avons évoqué `tar`. Contrairement à `winzip` sous *windows*, l'archivage et la compression sont faits en utilisant deux programmes séparés (`tar` pour l'archivage, et les deux programmes que nous allons maintenant présenter, `bzip2` et `gzip`, pour compresser). D'autres outils de compression existent sous *GNU/Linux*, comme `zip`, `arj` ou `rar` mais sont rarement utilisés.

À l'origine, `bzip2` a été écrit en tant que remplacement pour `gzip`. Ses possibilités de compression sont en général meilleures. Mais, d'un autre côté, il consomme plus de mémoire. La raison pour laquelle `gzip` est encore là est qu'il est encore beaucoup plus répandu que `bzip2`. Peut-être qu'un jour `bzip2` remplacera complètement `gzip` ou peut-être pas ?

Les deux commandes ont une syntaxe similaire :

```
gzip [options] [fichier(s)]
```

Si aucun nom de fichier n'est donné, `gzip` comme `bzip2` attendra des données sur l'entrée standard et enverra le résultat sur la sortie standard. Les deux programmes sont donc en fait utilisables avec des tubes. Les deux commandes ont aussi un ensemble d'options similaires :

- `-1, ..., -9` : règle le degré de compression ; plus le nombre est haut, plus la compression sera élevée, mais mieux signifie aussi plus lent : on n'a rien sans rien.
- `-d` : décompresse un ou des fichier(s) ; c'est l'équivalent de la commande `gunzip` ou de `bunzip2`.
- `-c` : envoie le résultat de la compression/décompression des fichiers donnés en paramètre sur la sortie standard.

Attention ! Par défaut, si vous n'utilisez pas l'option `-c`, `gzip` et `bzip2` effaceront le ou les fichier(s) qu'ils ont compressés (ou décompressés). Vous pouvez éviter cela avec `bzip2` en utilisant l'option `-k`. `gzip`, cependant, ne possède pas une telle option !

Quelques exemples seront utiles. Supposons que vous vouliez compresser tous les fichiers se terminant par `.txt` dans le répertoire courant avec `bzip2`, utilisez alors :

```
$ bzip2 -9 *.txt
```

Supposons que vous vouliez partager votre archive d'images avec quelqu'un, mais qu'il ne dispose pas de `bzip2`, mais uniquement de `gzip`. Nul besoin de décompresser l'archive et de la recompresser : décompressez-

la sur la sortie standard, utilisez un tube, compressez depuis l'entrée standard et redirigez le résultat vers la nouvelle archive :

```
bzip2 -dc images.tar.bz2 | gzip -9 >images.tar.gz
```

Et voilà le travail ! Vous pourriez d'ailleurs avoir tapé `bzcat` au lieu de `bzip2 -dc`. Il existe un équivalent pour `gzip` mais son nom est `zcat`, et non pas `gzcat`. Vous disposez aussi de `bzless` (resp. `zless`) si vous voulez directement visualiser des fichiers compressés sans avoir à les décompresser avant. Comme exercice, essayez de trouver la commande que vous auriez à taper pour voir des fichiers sans les décompresser, et sans utiliser `bzless` or `zless` ... :-)

## 5.6. Et plus encore...

Il existe tellement de commandes qu'un livre qui les présenterait toutes serait de la taille d'une véritable encyclopédie. Ce chapitre n'a touché qu'un dixième du sujet. Vous avez sans doute compris que vous êtes à même d'accomplir énormément avec celles que nous avons choisies. Nous pouvons vous conseiller de parcourir ces quelques pages de manuel : `sort(1)`, `sed(1)` et `zip(1)` (oui, c'est exactement ce que vous pensez : vous pouvez extraire/fabriquer des archives `.zip` sous *GNU/Linux*), `convert(1)`, et ainsi de suite. L'expérimentation reste le meilleur moyen de se familiariser avec ces outils, et vous leur trouverez probablement beaucoup d'autres utilisations que celles qu'elles présentent au premier abord. Amusez-vous bien !



## Chapitre 6. Contrôle des processus

### 6.1. Un mot encore sur les processus

Nous avons mentionné dans *Les processus*, page 4 qu'il était possible d'avoir le contrôle des processus : c'est ce que nous allons aborder dans ce chapitre. Mais, pour bien saisir de quoi il retourne dans les actions que nous allons effectuer, il faut en savoir un peu plus sur l'organisation même de ces processus.

#### 6.1.1. L'arborescence des processus

De même que pour les fichiers, tous les processus en cours d'exécution sur un système *GNU/Linux* sont organisés sous forme d'arborescence, et chaque processus possède un numéro (son PID, *Process ID*, soit identifiant du processus), ainsi que le numéro de son processus parent (PPID, *Parent Process ID*, soit identifiant du processus parent). Le PID de *init* est 1, de même que son PPID: *init* est son propre père.

#### 6.1.2. Les signaux

Chaque processus sous *UNIX* est susceptible de réagir à des signaux qui lui sont envoyés. Il existe 64 signaux différents que l'on identifie soit par leur numéro (en partant de 1), soit par leur nom symbolique. Les 32 signaux de rang le plus élevé (33 à 64) sont des signaux temps réel, et sont hors de portée de ce chapitre. Pour chacun de ces signaux, le processus peut redéfinir son propre comportement par défaut, sauf deux : le signal numéro 9 (KILL), et le signal numéro 19 (STOP).

Le signal 9 tue un processus de façon irrémédiable, sans lui laisser le temps de se terminer correctement. C'est ce signal qu'il faut envoyer à des processus dont vous voulez vous débarrasser. Une liste complète des signaux est disponible en utilisant la commande `kill -l`.

### 6.2. Obtenir des informations sur les processus : `ps` et `pstree`

Ces deux commandes affichent une liste des processus existants sur le système, selon les critères que vous voulez.

#### 6.2.1. `ps`

Lancer cette commande sans argument montrera uniquement les processus dont vous êtes l'initiateur et qui sont rattachés au *Terminal* que vous utilisez :

```
$ ps
  PID TTY          TIME CMD
 5162 ttya1    00:00:00 zsh
 7452 ttya1    00:00:00 ps
```

Les options sont nombreuses, nous ne citerons que les plus courantes :

- `a` : affiche aussi les processus lancés par les autres utilisateurs ;
- `x` : affiche aussi les processus n'ayant pas de terminal de contrôle (c'est le cas de pratiquement tous les serveurs) ou un terminal de contrôle différent de celui que vous êtes en train d'utiliser ;
- `u` : affiche pour chaque processus le nom de l'utilisateur qui l'a lancé et l'heure de son lancement.

Il existe beaucoup d'autres options. Reportez-vous à la page de manuel `ps(1)` pour plus de renseignements.

La sortie de cette commande est divisée en champs : celui qui vous intéressera le plus est le champ PID, qui contient l'identificateur du processus. Le champ CMD contient, quant à lui, le nom de la commande exécutée. Une façon très courante d'invoquer `ps` est la suivante :

```
$ ps ax | less
```

Vous obtenez ainsi une liste de tous les processus en cours d'exécution. Ceci permet de repérer le ou les processus problématiques avant de les éliminer.

### 6.2.2. pstree

La commande `pstree` affiche les processus sous forme d'arborescence et permet de les visualiser par leurs liens de parenté. Ainsi, pour tuer une série de processus de la même famille, il suffira d'en découvrir l'ancêtre commun. Vous aurez avantage à utiliser l'option `-p`, qui affiche le PID de chaque processus, ainsi que l'option `-u`, laquelle donne le nom de l'utilisateur ayant lancé le processus. L'arborescence étant généralement longue, il est plus facile d'invoquer `pstree` de cette façon :

```
$ pstree -up | less
```

pour en avoir une vue d'ensemble.

## 6.3. Envoyer des signaux aux processus : kill, killall, top

### 6.3.1. kill, killall

Ces deux commandes permettent d'envoyer des signaux à des processus. La commande `kill` attend un numéro de processus en argument, tandis que `killall` attend un nom de commande.

Les deux commandes peuvent, de façon optionnelle, recevoir un numéro de signal en argument. Par défaut, elles envoient toutes deux le signal 15 (TERM) à ou aux processus désigné(s). Par exemple, si vous voulez tuer le processus de PID 785, vous entrerez la commande :

```
$ kill 785
```

Si vous voulez lui envoyer le signal 9, vous entrerez alors :

```
$ kill -9 785
```

Supposons que vous vouliez tuer un processus pour lequel vous connaissez le nom de la commande. Au lieu de repérer le numéro du processus à l'aide de `ps`, vous pouvez tuer le processus directement à partir de son nom :

```
$ killall -9 netscape
```

Quoi qu'il arrive, vous ne tuerez que vos propres processus (sauf si vous êtes `root`), donc ne vous inquiétez pas des processus « du voisin » portant le même nom, ils ne seront pas affectés.

### 6.3.2. top

`top` est un programme tout en un : il remplit à la fois les fonctions de `ps` et `kill`. C'est un programme en mode console, vous le lancerez donc depuis un terminal (voir la figure 6-1) :



```

12:48pm up 8:05, 2 users, load average: 1.40, 1.40, 1.31
97 processes: 93 sleeping, 4 running, 0 zombie, 0 stopped
CPU states: 1.9% user, 1.9% system, 96.0% nice, 0.0% idle
Mem: 192072K av, 181432K used, 10640K free, 0K shrd, 5124K buff
Swap: 249472K av, 52872K used, 196600K free

  PID USER   PRI  NI  SIZE  RSS SHARE STAT  %CPU %MEM    TIME COMMAND
 1618 fabman   16    1 14140  13M  296 R  N   96.9  7.0  436:41 setiathome
20340 root      9    0 21400  12M 2428 S             1.1  6.7    0:04 X
20576 peter     9    0 11660  11M 10620 S            0.5  6.0    0:00 kdeinit
20632 peter    12    0 1052   820 R            0.5  0.5    0:00 top
20633 peter     9    0 13336  13M 11196 S            0.3  6.9    0:01 ksnapshot
20579 peter     9    0 16716  16M 14584 S            0.1  8.7    0:01 kdeinit
   1 root      8    0  124    76  64 S            0.0  0.0    0:03 init
   2 root      9    0    0    0    0 SW            0.0  0.0    0:01 keventd
   3 root      9    0    0    0    0 SW            0.0  0.0    0:00 kpmdd
   4 root     19  19    0    0    0 SWN            0.0  0.0    0:00 kseftirqd_CPU0
   5 root      9    0    0    0    0 SW            0.0  0.0    0:03 kswapd
   6 root      9    0    0    0    0 SW            0.0  0.0    0:00 bdfldsh
   7 root      9    0    0    0    0 SW            0.0  0.0    0:00 kupdated
   8 root     -1 -20    0    0    0 SW<            0.0  0.0    0:00 mdrecoveryd
  12 root      9    0    0    0    0 SW            0.0  0.0    0:00 kjournald
  57 root      9    0   584   488 S            0.0  0.2    0:01 devfsd
 200 root      9    0    0    0    0 SW            0.0  0.0    0:00 kjournald

```

Figure 6-1. Exemple d'exécution de top

Le programme se contrôle entièrement au clavier. Une aide est disponible en tapant sur **h**, mais elle est en anglais. Voici donc quelques-unes des commandes disponibles :

- **k** : sert à envoyer un signal à un processus. top vous demandera alors le PID du processus suivi du numéro du signal à envoyer (15 par défaut) ;
- **M** : trie par taux d'occupation mémoire (champ %MEM)
- **P** : sert à trier les processus selon le temps CPU qu'ils consomment (champ %CPU : c'est le tri par défaut) ;
- **u** : cette commande sert à afficher les processus d'un utilisateur donné, top vous demandera lequel. Vous devez entrer le **nom** de l'utilisateur, pas son UID. Si vous n'entrez aucun nom, tous les processus seront affichés ;
- **i** : cette commande fonctionne en tant que bascule. Par défaut, tous les processus, même endormis, sont affichés. Cette commande fera que seuls les processus en cours d'exécution seront affichés (processus dont le champ STAT indique R, *running*, soit en cours d'exécution) mais pas les autres. Un nouvel appel à cette commande permettra de revenir à l'état antérieur.

## 6.4. Contrôler la priorité des processus : nice, renice

Chaque processus actif sur le système s'exécute avec une priorité donnée (aussi appelée valeur amicale « *nice value* »). Cette valeur peut varier de -20 à +20. La priorité maximale pour un processus est -20. Si elle n'est pas définie, tous les processus s'exécutent avec une priorité de 0 par défaut (la priorité de planification de « base »). Les processus ayant la priorité maximale utilisent plus de ressources système que les autres. Les processus ayant une priorité inférieure s'exécuteront lorsque le système sera libéré par les tâches de priorité plus élevée. Les utilisateurs autres que le super-utilisateur root ne pourront que diminuer la priorité de leurs processus dans la fourchette 0 à 20. Le super-utilisateur pourra quand à lui définir de manière arbitraire la priorité de n'importe quel processus.

### 6.4.1. renice

Si un ou plusieurs processus prennent d'assaut toutes les ressources du système, vous pouvez changer leur priorité au lieu de les tuer. On utilise alors la commande **renice**. Sa syntaxe est la suivante :

```
renice priorité [[-p] pid ...] [[-g] pgrp ...] [[-u] utilisateur ...]
```

avec **priorité** étant la valeur numérique de la priorité, **PID** (précédé de **-p** pour une liste de PID) l'ID du processus, **pgrp** (**-g** pour plusieurs) l'ID du groupe du processus, et **utilisateur** (ou **-u** pour plusieurs utilisateurs) le nom d'utilisateur du propriétaire du processus.

Supposons que vous venez de lancer un processus auquel a été attribué le numéro PID 785, lequel lance une longue opération scientifique, et vous souhaitez jouer à un jeu pendant que ce processus tourne. Vous tapez alors :

```
$ renice +15 785
```

Avec ça le processus durera sans doute plus longtemps, mais il ne vous empêchera pas de profiter d'une animation fluide pour votre jeu en laissant la priorité au jeu lorsqu'il en a besoin.

Si vous êtes l'administrateur d'un système partagé et constatez qu'un des utilisateurs lance beaucoup de processus accaparant les ressources du système, Vous pouvez changer la priorité de tous les processus de cet utilisateur avec la commande :

```
# renice +20 -u pierre
```

Après cela, tous les processus de pierre auront une priorité plus basse et ne bloqueront pas les processus des autres utilisateurs.

#### 6.4.2. nice

Maintenant que vous savez que vous pouvez changer la priorité des processus, vous voudrez peut-être lancer une commande avec une priorité prédéfinie. Cela est accompli grâce à la commande `nice`.

Il suffit de placer votre commande en argument de la commande `nice`. Par défaut `nice` assigne une priorité de 10. Vous pouvez utiliser l'option `-n` pour assigner une priorité quelconque dans la fourchette -20 (priorité maximale) à 19 (minimale).

Par exemple, si vous souhaitez créer une image ISO à partir du CD-ROM d'installation de **Mandrake Linux**, vous utilisez d'habitude :

```
$ dd if=/dev/cdrom of=~/mdk1.iso
```

Mais sur certains systèmes avec un lecteur CD-ROM IDE standard, le fait de copier de grands volumes de données peut utiliser énormément de ressources. Afin que cela n'empêche pas les autres personnes ou processus de fonctionner correctement, on peut démarrer le processus de copie avec une priorité affaiblie :

```
$ nice -n 19 dd if=/dev/cdrom of=~/mdk1.iso
```

et continuer avec d'autres tâches courantes.

Pour modifier la priorité d'un processus vous pouvez aussi utiliser l'utilitaire `top` décrit ci-dessus. Utilisez la commande `r` depuis l'interface de `top` pour changer la priorité du processus sélectionné.

## Chapitre 7. Organisation de l'arborescence des fichiers

Aujourd'hui, un système *UNIX* est gros, très gros, et c'est particulièrement vrai avec *GNU/Linux*. La profusion de logiciels disponibles en ferait un système incompréhensible s'il n'y avait pas de lignes de conduite à suivre précisément quant au placement des fichiers dans l'arborescence.

Le standard reconnu en la matière est le FHS (*Filesystem Hierarchy Standard*, soit la norme pour les hiérarchies de systèmes de fichiers), et il en est à sa version 2.2 à l'heure où nous mettons sous presse. Le document décrivant la norme est disponible sur Internet en différents formats sur le site de pathname (<http://www.pathname.com/fhs/>). Ce chapitre n'en est qu'un résumé succinct, mais il devrait vous suffire pour savoir dans quel répertoire rechercher (ou placer) un fichier donné.

### 7.1. Données partagées et non partagées, statiques et dynamiques

Les données sur un système *UNIX* peuvent être classées selon les deux critères susmentionnés. La signification de ces critères est la suivante : des données partagées peuvent être communes à plusieurs machines sur un réseau, tandis que des données non partagées ne le peuvent pas. Les données statiques n'ont pas à être modifiées dans le cadre d'une utilisation normale du système, tandis que les données dynamiques peuvent l'être. Au fur et à mesure que nous explorerons l'arborescence, nous classerons les différents répertoires dans chacune de ces catégories.

Les classements proposés ici ne sont évidemment que des recommandations, mais celles-ci vous aideront grandement à gérer votre système. Gardez aussi à l'esprit que la distinction statique/dynamique ne s'applique qu'à l'utilisation courante du système, et non pas à sa configuration. Si vous installez un programme, vous aurez inmanquablement à modifier des répertoires normalement statiques (ex. : `/usr/`).

### 7.2. Le répertoire racine : /

Le répertoire racine contient toute la hiérarchie du système. Il est inclassable puisque ses sous-répertoires peuvent être statiques ou dynamiques, partagés ou non. Voici une liste des principaux répertoires et sous-répertoires :

- `/bin` : binaires essentiels au système. Ce répertoire renferme les commandes de base qui sont susceptibles d'être partagées par tous les utilisateurs et nécessaires pour utiliser le système : `ls`, `cp`, `login`, etc. Statique, non partagé ;
- `/boot` : contient les fichiers nécessaires au gestionnaire de démarrage de *GNU/Linux* (*grub* ou *LILO*). Cela peut ou non comprendre le noyau : s'il n'est pas ici, il doit être situé dans la racine. Statique, non partagé ;
- `/dev` : fichiers périphériques du système (`dev` pour *DEVices*, périphériques). Statique, non partagé ;
- `/etc` : ce répertoire contient tous les fichiers de configuration spécifiques à la machine, logiquement statique, non partagé ;
- `/home` : contient tous les répertoires personnels des utilisateurs du système. Ce répertoire peut ou non être partagé (les grands réseaux le rendent généralement partagé par NFS), et il est évidemment dynamique ;
- `/lib` : ce répertoire contient les bibliothèques essentielles au système ainsi que les modules du noyau, dans `/lib/modules`. Toutes les bibliothèques nécessaires aux binaires présents dans les répertoires du système `/bin` et `/sbin` doivent s'y trouver, ainsi que l'éditeur de liens `ld`. so. Statique, non partagé ;
- `/mnt` : répertoire contenant les points de montage pour les systèmes de fichiers temporaires tels que les CD-ROM ou les disquettes. Dynamique, non partagé ;
- `/opt` : ce répertoire contient des paquets non nécessaires au fonctionnement du système. Il est recommandé que les fichiers statiques (binaires, bibliothèques, pages de manuel, etc.) de tels paquets aient leur place dans le fichier `/opt/nom_du_paquetage` et que les fichiers de configuration spécifiques à la machine soient placés dans le répertoire `/etc/opt` ;
- `/root` : répertoire personnel de root. Dynamique, non partagé ;
- `/usr` : ce répertoire est décrit en détail plus bas. Statique, partagé ;
- `/sbin` : contient les binaires système essentiels au démarrage du système, exploitables uniquement par root. L'utilisateur pourrait également les exécuter, mais il n'en obtiendrait pas grand-chose en général. Statique, non partagé ;

- /tmp : répertoire destiné à contenir les fichiers temporaires que peuvent créer certains programmes. Dynamique, non partagé ;
- /var : emplacement pour les données souvent modifiées par des programmes (par exemple, le serveur de courrier électronique, les programmes d'audit, le serveur d'impression, etc.). Tout /var est dynamique, mais ses différents sous-répertoires peuvent être partagés ou non.

### 7.3. /usr : le gros morceau

Le répertoire /usr est le principal répertoire de stockage des applications. Tous les binaires dans ce répertoire ne doivent pas être nécessaires au démarrage ou à l'entretien du système, puisque la plupart du temps, la hiérarchie /usr est souvent sur un système de fichiers séparé. Étant donnée sa taille importante, /usr possède sa propre hiérarchie de sous-répertoires. Nous n'en citerons que quelques-uns :

- /usr/X11R6 : toute la hiérarchie de *X Window System*. Tous les binaires nécessaires au fonctionnement de *X* (cela comprend bien sûr les serveurs *X*) et toutes les bibliothèques nécessaires doivent s'y trouver. Le répertoire portant le nom /usr/X11R6/lib/X11 contient les aspects de la configuration de *X* qui ne varient pas de machine à machine. Nous avons déjà vu que la configuration spécifique pour chaque machine est dans /etc/X11 ;
- /usr/bin : ce répertoire contient la grande majorité des programmes binaires du système. **Tout** programme binaire qui n'est pas nécessaire à l'entretien du système et n'est pas un programme d'administration du système doit se trouver dans ce répertoire, à l'exception des programmes que vous installez vous-mêmes, qui doivent se trouver dans /usr/local ;
- /usr/lib : ce répertoire contient toutes les bibliothèques nécessaires pour employer les programmes situés dans le répertoire /usr/bin et /usr/sbin. Il existe également un lien symbolique, /usr/lib/X11, qui pointe vers le répertoire renfermant les bibliothèques de *X Window System*, c'est-à-dire /usr/X11R6/lib (si *X* est installé, bien sûr) ;
- /usr/local : c'est dans ce répertoire que vous devrez installer vos applications personnelles. Le programme d'installation y aura créé toute la hiérarchie nécessaire : lib/, bin/, etc. ;
- /usr/share/ : ce répertoire contient toutes les données nécessaires aux applications dans /usr, ainsi que toutes les données indépendantes de la plate-forme. Vous y trouverez notamment les informations de zone et de localisation (zoneinfo et locale).

Il existe également des répertoires /usr/share/doc et /usr/share/man, qui contiennent respectivement la documentation des applications et les pages de manuel du système.

### 7.4. /var : données modifiables en cours d'utilisation

Le répertoire /var contient toutes les données de fonctionnement des programmes qui tournent sur le système. Contrairement aux données de travail dans /tmp, ces données doivent être conservées intactes lors d'un redémarrage. Il existe de nombreux sous-répertoires dont quelques-uns sont très utiles :

- /var/log : contient les fichiers d'audit du système ;
- /var/spool : contient les fichiers de travail des démons du système. Par exemple, /var/spool/cups contient les fichiers de travail du serveur d'impression et le répertoire /var/spool/mail contient les fichiers de travail du serveur de courrier électronique (donc, tout le courrier qui arrive et sort de votre système) ;
- /var/run : ce répertoire sert à conserver une trace de tous les processus en cours d'utilisation par le système, ce qui vous permet d'agir sur ces processus lors d'un changement de *niveau d'exécution* du système (voir *Les fichiers de démarrage : init sysv*, page 67).

## 7.5. /etc : les fichiers de configuration

Le répertoire `/etc` est essentiel à tout système *UNIX*. Il contient tous les fichiers de configuration de base du système. Ne l'effacez **surtout pas** pour gagner de la place ! De même, si vous voulez étaler votre arborescence sur plusieurs partitions, sachez que `/etc` ne doit pas être mis sur une partition séparée : il est nécessaire à l'initialisation du système.

Quelques fichiers importants :

- `passwd` et `shadow` : ces deux fichiers contiennent tous les utilisateurs du système ainsi que leurs mots de passe (cryptés). `shadow` n'est là que si vous utilisez les mots de passe *shadow*, mais c'est l'option par défaut de l'installation ;
- `inittab` : c'est le fichier de configuration du programme `init`, qui joue un rôle fondamental lors du démarrage du système ainsi que nous le verrons plus tard ;
- `services` : ce fichier contient une énumération des services réseau existants ;
- `profile` : il s'agit du fichier de configuration du *shell*, bien que certains *shells* en utilisent d'autres. Par exemple, *bash* utilise `bashrc` ;
- `crontab` : fichier de configuration de `cron`, programme chargé de l'exécution périodique de commandes.

Il existe également certains sous-répertoires pour les programmes dont la configuration requiert un grand nombre de fichiers. C'est le cas de *X Window System*, par exemple, qui dispose de tout le répertoire `/etc/X11`.



## Chapitre 8. Systèmes de fichiers et points de montage

Pour connaître aisément le fonctionnement de ces systèmes, prenons un cas pratique. Supposons que vous veniez juste d'acheter un disque dur flambant neuf, encore vierge de toute partition. Votre partition consacrée à **Mandrake Linux** est pleine à ras bord, et plutôt que de tout refaire à partir de zéro, vous décidez de déplacer toute une partie de l'arborescence sur votre nouveau disque dur. Votre nouveau disque dur étant très gros, vous décidez d'y déplacer votre répertoire le plus encombrant : `/usr`. Avant d'expliquer la solution, faisons un pas en arrière pour comprendre les principes de base.

### 8.1. Principes

Comme nous l'avons déjà mentionné dans le *Guide d'installation*, chaque disque dur est divisé en plusieurs partitions et chacune de ces partitions contient un système de fichiers. Tandis que *windows* associe une lettre à chacun de ces systèmes de fichiers (enfin, seulement à ceux qu'il reconnaît), *GNU/Linux* possède une arborescence unique, et chacun des systèmes de fichiers est *monté* à un endroit de l'arborescence.

De même que *windows* a besoin d'un « lecteur C: », *GNU/Linux* a besoin de pouvoir monter la racine de son arborescence (`/`) quelque part, en l'occurrence sur une partition qui contient le *système de fichiers racine*. Une fois la racine montée, on peut monter d'autres systèmes de fichiers sur différents *points de montage* qui existent dans l'arborescence. N'importe quel répertoire en-dessous de la racine peut faire office de point de montage.

Cela autorise une grande souplesse dans la configuration. Dans le cas d'un serveur Web, par exemple, il est courant de consacrer une partition entière au répertoire hébergeant les données du serveur. Le répertoire qui les contient est en règle générale `/var/www` ; ce dernier fera donc office de point de montage pour la partition. Vous pouvez voir dans les figure 8-1 et figure 8-2 la situation du système avant et après le montage des fichiers.

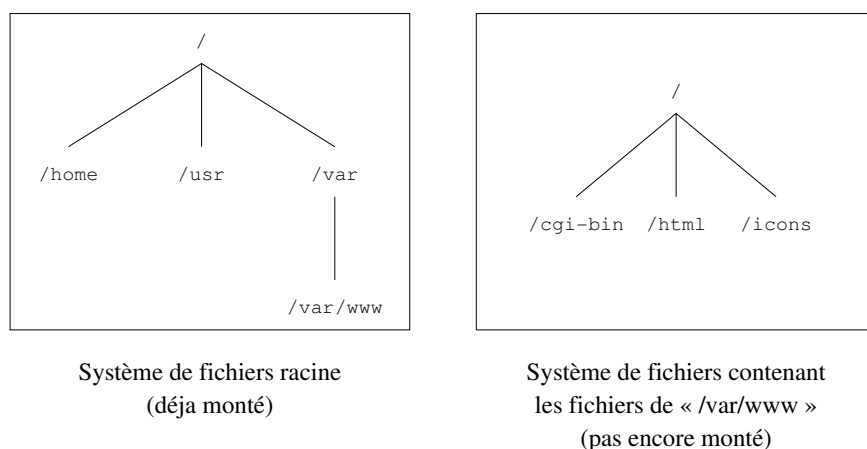


Figure 8-1. Avant le montage d'un système de fichiers

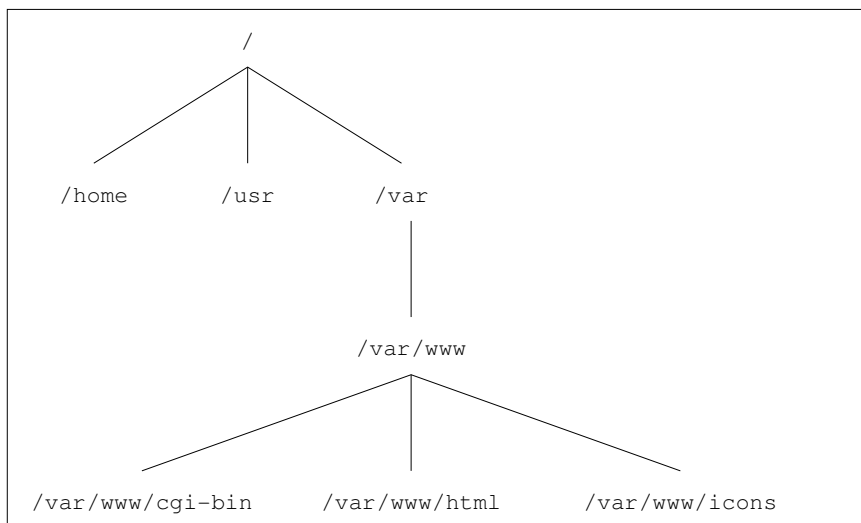


Figure 8-2. Après le montage du système de fichiers

Comme vous pouvez l’imaginer, cela présente de nombreux avantages : l’arborescence sera toujours la même, qu’elle s’étende sur un seul système de fichiers ou plusieurs dizaines<sup>1</sup>, et il est toujours possible de déplacer physiquement une partie encombrante de l’arborescence sur une autre partition quand la place vient à manquer, ce que nous allons faire ici.

Il faut savoir deux choses sur les points de montage :

- le répertoire faisant office de point de montage doit exister,
- ce même répertoire **devrait être vide** : si un répertoire choisi comme point de montage contient déjà des fichiers et sous-répertoires, ceux-ci seront tout simplement « cachés » par le système de fichiers nouvellement monté, mais ils ne seront pas effacés. Ils seront tout simplement inaccessibles jusqu’à ce que vous libériez ce point de montage.

## 8.2. Partitionner un disque dur, formater une partition

Des principes évoqués ci-dessus et pour ce qui nous intéresse dans cette section, il faut retenir deux choses : un disque dur fonctionnel est divisé en partitions et chacune de ces partitions héberge un système de fichiers. Or, pour l’instant, votre disque dur tout neuf ne possède ni l’un ni l’autre, c’est donc par là qu’il faut commencer, avec en premier lieu le partitionnement. Pour cela, vous devez être root.

Tout d’abord, il faut savoir quel est le « nom » de votre disque dur, c’est-à-dire quel fichier le matérialise. Supposons que vous l’ayez mis en tant qu’esclave sur votre interface IDE primaire, ce sera donc `/dev/hdb`<sup>2</sup>. Reportez-vous à la section *Gérer ses partitions* du *Guide de démarrage* qui explique comment partitionner un disque. Notez que *DiskDrake* créera aussi les systèmes de fichiers.

## 8.3. Les commandes mount et umount

Maintenant que le système de fichiers est créé, on peut monter la partition. Elle sera vide dans un premier temps, bien sûr. La commande pour monter des systèmes de fichiers est la commande `mount`, et sa syntaxe est la suivante :

```
mount [options] <-t type> [-o options de montage] <périphérique>
    <point de montage>
```

En l’occurrence, on souhaite monter notre partition sur `/mnt` ou tout autre point de montage que vous aurez choisi (n’oubliez pas qu’il doit exister) ; la commande pour monter notre partition nouvellement créée est la suivante :

1. *GNU/Linux* peut gérer jusqu’à 64 systèmes de fichiers montés en même temps.  
 2. La façon de trouver le nom d’un disque est expliquée dans le *Guide d’installation*.



```
$ mount -t ext2 /dev/hdb1 /mnt
```

L'option `-t` sert à spécifier quel type de système de fichiers la partition est censée héberger. Parmi les systèmes de fichiers que vous rencontrerez le plus souvent, vous trouverez ext2FS (le système de fichiers de *GNU/Linux*), VFAT (pour toutes les partitions *DOS/windows* : FAT 12, 16 ou 32) et ISO9660 (système de fichiers des CD-ROM). Si vous ne spécifiez aucun type, `mount` essaiera et trouvera quel système de fichier est hébergé par cette partition en lisant le *superblock*. Il y échoue rarement.

L'option `-o` sert à spécifier une ou plusieurs options de montage. Ces options dépendent du système de fichiers utilisé. Reportez-vous à la page de manuel de `mount(8)` pour plus de détails.

Maintenant que vous avez monté votre nouvelle partition, il s'agit de recopier tout le répertoire `/usr` dedans :

```
$ (cd /usr && tar cf - .) | (cd /mnt && tar xpvf -)
```

Maintenant que les fichiers sont recopiés, on peut démonter notre partition. Pour cela, la commande est `umount`. Sa syntaxe est simple :

```
umount <point de montage|périphérique>
```

Donc, pour démonter notre nouvelle partition, on peut taper :

```
$ umount /mnt
```

ou bien :

```
$ umount /dev/hdb1
```

Cette partition étant appelée à « devenir » notre répertoire `/usr`, il nous faut le dire au système. Pour cela, il nous faut remplir :

## 8.4. Le fichier `/etc/fstab`

Le fichier `/etc/fstab` permet d'automatiser le montage de certains systèmes de fichiers, en particulier au démarrage du système. Il contient une série de lignes décrivant les systèmes de fichiers, leur point de montage et d'autres options. Voici un exemple de fichier `/etc/fstab` :

```
/dev/hda1  /          ext2    defaults    1 1
/dev/hda5  /home      ext2    defaults    1 2
/dev/hda6  swap       swap    defaults    0 0
/dev/fd0   /mnt/floppy auto    sync,user,noauto,nosuid,nodev,unhide 0 0
/dev/cdrom /mnt/cdrom auto    user,noauto,nosuid,exec,nodev,ro 0 0
none      /proc      proc    defaults    0 0
none      /dev/pts   devpts  mode=0622   0 0
```

Une ligne contient, dans l'ordre :

- le périphérique hébergeant le système de fichiers,
- le point de montage,
- le type du système de fichiers,
- les options de montage,
- le *drapeau* de sauvegarde par l'utilitaire `dump`,
- l'ordre de la vérification par `fsck` (*FileSystem Check*, *vérification des systèmes de fichiers*).

Comme de juste, il y a toujours une entrée pour la racine. Les partitions de *swap* sont particulières puisqu'elles ne sont pas visibles dans l'arborescence, et le champ « point de montage » pour ces partitions contient le mot-clé *swap*. Nous reviendrons plus en détail sur `/proc` dans *Le système de fichiers /proc*, page 63. Un autre système de fichier particulier (que nous ne détaillerons pas) est `/dev/pts`.

Revenons à nos moutons. Vous avez bougé toute la hiérarchie `/usr` sur `/dev/hdb1` et donc vous voudriez que cette partition soit montée en tant que `/usr/` au démarrage. Dans ce cas il vous faudra ajouter une entrée dans le fichier :

```
/dev/hdb1      /usr      ext2      defaults  1 2
```

Ainsi à chaque démarrage la partition sera montée. Elle sera également vérifiée si besoin est.

Il existe deux options particulières : `noauto` et `user`. L'option `noauto` indique que le système de fichiers ne doit pas être monté au démarrage mais doit être monté explicitement. L'option `user` indique que n'importe quel utilisateur peut monter et démonter le système de fichiers. Ces deux options sont logiquement utilisées pour le lecteur CD-ROM et le lecteur de disquettes. Il existe d'autres options, et `/etc/fstab` dispose de sa propre page de manuel (`fstab(5)`).

Enfin, l'un des avantages (et non des moindres) de ce fichier est qu'il simplifie la syntaxe de la commande `mount`. Pour monter un système de fichiers qui y est référencé, on peut au choix référencer le point de montage ou le périphérique. Ainsi, pour monter une disquette, on peut taper :

```
$ mount /mnt/floppy
```

ou bien :

```
$ mount /dev/fd0
```

Terminons-en avec notre exemple de déplacement de partitions : nous avons recopié la hiérarchie `/usr` et rempli `/etc/fstab` pour que la nouvelle partition soit montée au démarrage. Mais pour l'instant les anciens fichiers de `/usr` sont toujours là! Il faut donc les effacer pour libérer de la place (ce qui, après tout, était notre objectif premier). Pour cela, donc, il vous faut tout d'abord mettre la machine en mode « `single user` » en lançant la commande `telinit 1`; puis :

- effacer tous les fichiers du répertoire `/usr` (l'« ancien », donc, puisque le « nouveau » n'est pas monté pour l'instant) : `rm -Rf /usr/*`;
- monter le « nouveau » `/usr` : `mount /usr/`

Et voilà ! Revenez maintenant en mode multiutilisateurs (`telinit 3` ou `telinit 5`), et si vous n'avez plus de tâche d'administration à accomplir sur votre machine, il est temps de mettre fin à la session de l'utilisateur privilégié `root`.

## 8.5. La fonction `supermount`

Les noyaux les plus récents tels que fournis par **Mandrake Linux** comportent une fonctionnalité intéressante pour les utilisateurs utilisant fréquemment des disquettes ou des CD. Installée ou pas suivant le niveau de sécurité choisi, elle monte et démonte automatiquement les supports lorsqu'ils sont insérés ou enlevés. Ceci est très utile car il n'y a plus besoin de lancer les commandes `mount` ou `umount` à chaque fois.

## Chapitre 9. Le système de fichiers Linux

Votre système *GNU/Linux* est naturellement contenu sur votre disque, dans un système de fichiers. Voici une présentation des différents aspects de ce système de fichiers, et des possibilités qui vous sont offertes.

### 9.1. Comparatif de quelques systèmes de fichiers

Lors de l'installation, vous pouvez choisir différents **systèmes de fichier** pour vos partitions, c'est-à-dire, de formater vos partitions selon différents algorithmes.

À moins d'être un spécialiste, le choix n'est pas forcément évident. Nous vous proposons ici une rapide présentation des trois systèmes de fichiers les plus courants, tous disponibles dans **Mandrake Linux**.

#### 9.1.1. Les différents systèmes de fichiers utilisables

##### 9.1.1.1. Ext2FS

Le **Second Extended File System**, en abrégé **Ext2fs** ou simplement **ext2**, est le système de fichier par défaut de *GNU/Linux* depuis de nombreuses années. Il est le successeur de **Extended File System** (d'où le « Second »), dont il corrige certains problèmes et certaines limitations.

Ext2FS respecte les standards usuels des systèmes de fichiers pour systèmes de type Unix. Dès sa conception, il était destiné à évoluer, tout en offrant une grande robustesse et de bonnes performances.

##### 9.1.1.2. Ext3

Comme le nom le laisse supposer, le **Third Extended File System** est appelé à devenir le successeur de Ext2FS. Il conserve une compatibilité avec celui-ci, mais ajoute une fonctionnalité très intéressante : la **journalisation**.

Un des problèmes majeurs avec les systèmes de fichiers « traditionnels » comme Ext2FS, est leur faible tolérance aux pannes, telles qu'un arrêt brutal du système (coupure de courant ou plantage logiciel). En général, de tels événements se soldent par un examen très long de la structure du système de fichiers, des tentatives de corrections d'erreurs, parfois pour aboutir à une corruption étendue du système de fichiers. Donc, une perte partielle ou totale des données enregistrées.

La journalisation est une réponse à ce problème. Pour simplifier, disons que le principe consiste à enregistrer les actions à effectuer dans un journal **avant** de les effectuer réellement, un peu comme un capitaine de bateau note dans son journal de bord les événements de la journée. Le résultat est un système de fichiers qui reste toujours cohérent. En cas de problème, l'examen du système de fichiers consiste à regarder le journal et effectuer les actions qui n'ont pas eu le temps d'être effectuées avant le crash. Le temps de vérification d'un système de fichiers n'est donc plus proportionnel à la taille de celui-ci, mais à son degré d'utilisation.

Ext3FS propose donc cette technologie, tout en conservant une structure interne basée sur Ext2FS, ce qui assure une excellente compatibilité.

##### 9.1.1.3. ReiserFS

Au contraire de Ext3FS, **ReiserFS** est un système de fichiers recréé en partant de zéro. Il est également journalisé comme Ext3FS, mais sa structure interne est radicalement différente. En particuliers, il utilise des concepts d'arbres binaires inspirés des logiciels de base de données.

##### 9.1.1.4. JFS

JFS est le système de fichiers journalisé développé et utilisé par IBM. Initialement propriétaire et fermé, IBM a récemment décidé d'ouvrir l'accès au monde du Logiciel Libre à ce système de fichiers. Sa structure interne est proche de celle de ReiserFS.

### 9.1.2. Différences entre ces systèmes de fichiers

	Ext2FS	Ext3FS	ReiserFS	JFS
Stabilité	Excellente	Bonne	Bonne	Moyenne
Outils pour récupérer un fichier effacé	Oui (complexe)	Oui (complexe)	Non	Non
Temps de redémarrage après un crash	Long, voire très long	Rapide	Très rapide	Très rapide
Intégrité des données en cas de crash	En générale, bonne, mais risques de pertes partielles ou totales non négligeables	Très bonne	Très bonne	Très bonne

**Tableau 9-1. Caractéristiques des systèmes de fichiers**

À propos des tailles maximales de fichiers, cela dépend d'un grand nombre de paramètres (comme la taille des blocs pour ext2/ext3), et est susceptible d'évoluer suivant la version du noyau et l'architecture du système. Ceci étant, le minimum disponible est actuellement généralement proche ou supérieur à 2Tb (1Tb=1024 Gb) et peut atteindre 4Pb (1Pb=1024 Tb) pour JFS. Cependant ces valeurs sont aussi limitées par la taille des périphériques bloc, qui, pour les noyaux 2.4.X actuels est limitée (sur les architectures X86) à 2To<sup>1</sup> même en mode RAID. Pour plus d'information, consulter *Adding Support for Arbitrary File Sizes to the Single UNIX Specification* ([http://ftp.sas.com/standards/large.file/x\\_open.20Mar96.html](http://ftp.sas.com/standards/large.file/x_open.20Mar96.html)).

### 9.1.3. Et question performances ?

Il est toujours très délicat de réaliser un comparatif de performances. Tous les tests que l'on peut effectuer présentent diverses limitations, et les résultats doivent toujours être interprétés avec précautions. De plus, si Ext2FS est aujourd'hui très mature et évolue fort peu, les systèmes journalisés Ext3FS et ReiserFS évoluent très rapidement. Des tests effectués il y a quelques mois ou quelques semaines sont déjà trop anciens. Par ailleurs, les performances physiques des matériels actuels (notamment des disques durs) estompent les différences. Néanmoins, JFS est actuellement celui montrant les meilleures performances.

Chaque système présente ses avantages et ses inconvénients, et en fait tout dépend de l'utilisation que vous comptez faire de votre ordinateur. Une simple machine de bureau pourra se contenter de Ext2FS. Pour un serveur, on préférera sans doute un système de fichier journalisé comme Ext3FS. ReiserFS, peut-être du fait de ce qui l'a inspiré, est plutôt recommandé pour un serveur de base de données. JFS sera préféré dans les cas où l'exigence principale est la rapidité du système de fichiers.

Pour une utilisation « normale », les quatre systèmes de fichiers présentent à peu près les mêmes performances moyennes. ReiserFS est plus rapide pour l'accès aux fichiers de petites tailles, mais sensiblement plus lent pour la manipulation de gros fichiers (plusieurs mégaoctets). Dans la plupart des cas, les avantages de la journalisation de ReiserFS l'emportent sur ces inconvénients.

## 9.2. Tout est fichier

Le *Guide de démarrage* a introduit les concepts de propriété des fichiers et les droits d'accès, mais pour vraiment comprendre le *système de fichiers* de *GNU/Linux* (incluant *ext2fs*), il faut redéfinir cette notion même de fichier.

Ici, « tout » veut **vraiment** tout : un disque dur, une partition sur ce disque dur, un port parallèle, une connexion à un site Web, une carte *Ethernet*, même les répertoires sont des fichiers pour *GNU/Linux*. Ainsi, mis à part les fichiers ordinaires et ces répertoires, d'autres types de fichiers sont donc reconnus. Mais ces types ne sont pas de l'ordre d'une différence de **contenu** puisque, pour *GNU/Linux* et pour tout système *UNIX*, un fichier,

1. Vous pouvez vous demander comment atteindre de telles capacités avec des disques durs qui atteignent difficilement les 180Go. En fait, en utilisant 3 cartes RAID hébergeant chacune 8 disques de 128Gb, on atteint les 3To...

qu'il soit une image GIF, un fichier binaire ou autre, n'est fondamentalement qu'une quantité d'octets. La différenciation des fichiers en fonction de leur contenu est du ressort des applications.

### 9.2.1. Les différents types de fichier

Vous vous souviendrez sans doute que dans la sortie de la commande `ls -l`, le caractère qui apparaît avant l'énoncé des droits d'accès représente le type du fichier. Deux types de fichiers ont déjà été présentés : les fichiers ordinaires (-) et les répertoires (d). Il est possible que vous rencontriez également ceux qui suivent si vous explorez l'arborescence et listez le contenu des répertoires :

1. **Fichiers en mode caractère** : ces fichiers sont soit des fichiers spéciaux du système (tel `/dev/null`, déjà mentionné) ou des périphériques (ports série ou parallèle) ; leur particularité commune étant en fait que leur contenu (s'il existe) n'est pas *conservé en mémoire*<sup>2</sup>. L'identification de ces fichiers se fait par la lettre 'c'.
2. **Fichiers en mode bloc** : ces fichiers sont des périphériques, et contrairement aux fichiers en mode caractère, leur contenu est, quant à lui, **sauvegardé** en mémoire. Les fichiers qui entrent dans cette catégorie peuvent être les disques durs, les partitions sur ces disques durs, les lecteurs de disquettes et de CD-ROM, etc. Les fichiers `/dev/hda`, `/dev/sda5` sont d'autres exemples de ces fichiers en mode bloc. Sur la sortie d'un `ls -l`, ces fichiers seront identifiés par la lettre 'b'.
3. **Liens symboliques** : ces fichiers sont très courants et très utilisés dans la procédure de démarrage de **Mandrake Linux** (voir le chapitre *Les fichiers de démarrage : init sysv*, page 67). Comme leur nom l'indique, leur raison d'être est de lier symboliquement d'autres fichiers, c'est-à-dire qu'ils pointent (ou non) sur un fichier existant (ce point précis sera explicité plus loin dans le présent chapitre). Ils sont très souvent appelés *soft links* en anglais, ce qui est une erreur. Ils sont identifiés par la lettre 'l'.
4. **Tubes nommés** : au cas où vous vous posez la question : ils sont effectivement très similaires aux tubes utilisés par le *shell*, mais avec la particularité que ceux-ci ont un nom. Ils sont en fait très rares, et il est peu probable que vous en voyiez un durant votre voyage dans l'arborescence. Au cas où, cependant, leur identification se fait par la lettre 'p' (pour en apprendre plus, lisez *Tubes "anonymes" et tubes nommés*, page 59).
5. **Sockets** : ce type de fichier est celui de toutes les connexions réseau. Mais seules quelques-unes<sup>3</sup> d'entre elles portent des noms. Pour compliquer les choses, il existe plusieurs types de *sockets*, mais ces subtilités de classement dépassent de loin les objectifs de ce manuel. Des tels fichiers seront identifiés par la lettre 's'.

Voici un exemple pour chacun de ces types de fichiers :

```
$ ls -l /dev/null /dev/sda /etc/rc.d/rc3.d/S20random /proc/554/maps \
/tmp/ssh-reine/ssh-510-agent
crw-rw-rw- 1 root root      1,  3 May  5 1998 /dev/null
brw-rw---- 1 root disk     8,  0 May  5 1998 /dev/sda
lrwxrwxrwx 1 root root      16 Dec  9 19:12 /etc/rc.d/rc3.d/S20random
-> ../init.d/random*
pr--r--r-- 1 reine  reine    0 Dec 10 20:23 /proc/554/maps|
srwx----- 1 reine  reine    0 Dec 10 20:08 /tmp/ssh-reine/ssh-510-agent=
$
```

### 9.2.2. I-nœuds

Les >I-nœuds sont, avec le paradigme « Tout est fichier », les fondements de tout système de fichier *UNIX*. Le mot « *inode* » est l'abréviation de *Nœud d'Index*.

Les i-nœuds sont stockés sur disque dans une **table des i-nœuds**. Les i-nœuds existent pour n'importe quel type de fichier susceptible d'être stocké sur un système de fichier, et cela inclut les répertoires, tubes nommés, fichiers en mode caractère, et ainsi de suite. Cela conduit à la fameuse phrase : « L'I-nœud est le fichier. » Les I-nœuds sont aussi le moyen grâce auquel *UNIX* identifie un fichier de manière unique.

2. On emploie parfois le néologisme barbare « *bufferisé* », qui vient de l'anglais *buffered*, c'est-à-dire gardé en mémoire dans des tampons (*buffers*).

3. On emploiera « socket » au féminin en français.

Oui, vous avez bien lu : sous *UNIX*, vous **n'identifiez pas un fichier par son nom**, mais par son numéro d'i-nœud <sup>4</sup>. La raison en est qu'un même fichier peut avoir plusieurs noms, ou même pas de nom du tout. Un nom de fichier, sous *UNIX*, n'est qu'une entrée dans un i-nœud répertoire. Une telle entrée est appelée **lien**. Voyons ces liens d'un peu plus près.

### 9.3. Les liens

Pour mieux comprendre ce que cache cette notion de liens, passons par une illustration. Créons un fichier (ordinaire) :

```
$ pwd
/home/reine/exemple
$ ls
$ touch a
$ ls -il a
32555 -rw-rw-r-- 1 reine      reine      0 Dec 10 08:12 a
```

L'option `-i` de la commande `ls` affiche le numéro d'i-nœud, qui constitue le premier champ dans la sortie ; comme on le constate, avant que ne soit créé le fichier `a`, le répertoire était vide de tout autre fichier. Le troisième champ présente également un intérêt : il est le compteur de liens pour le fichier (pour l'i-nœud en fait...).

On peut séparer la commande `touch a` en deux actions distinctes :

- création d'un i-nœud, auquel le système a attribué le numéro 32555, dont le type est celui d'un fichier ordinaire,
- création d'un lien vers cet i-nœud, nommé `a`, dans le répertoire courant, `/home/reine/exemple`. Par conséquent, le « fichier » appelé `/home/reine/exemple/a` est un lien vers l'i-nœud de numéro 32555. Il est pour l'instant le seul et le compteur de liens indique donc 1.

Et si, maintenant, nous entrons :

```
$ ln a b
$ ls -il a b
32555 -rw-rw-r-- 2 reine      reine      0 Dec 10 08:12 a
32555 -rw-rw-r-- 2 reine      reine      0 Dec 10 08:12 b
$
```

nous avons créé un autre lien vers le même i-nœud. Comme on peut le constater, aucun fichier nommé `b` n'a été créé, mais ce qui a été ajouté est en fait un autre lien vers l'i-nœud de numéro 32555 dans le même répertoire nommé `b`. La deuxième sortie de `ls -l` nous indique ainsi que le compteur de liens est maintenant à 2 et non plus à 1.

Et alors, si nous faisons ce qui suit :

```
$ rm a
$ ls -il b
32555 -rw-rw-r-- 1 reine      reine      0 Dec 10 08:12 b
$
```

nous voyons que même si nous avons effacé le « fichier original », l'i-nœud existe encore. Mais maintenant le seul lien vers cet i-nœud est `/home/reine/exemple/b`.

Ainsi, un fichier sous *UNIX* n'a pas de nom. À la place, il a un ou plusieurs **liens**, dans un ou plusieurs répertoires.

Les répertoires eux-mêmes sont aussi stockés dans des i-nœuds, mais leur compteur de liens, contrairement à tous les autres types de fichiers, dépend du nombre de leurs sous-répertoires. Il se trouvera au moins deux liens par répertoire : le répertoire lui-même (`.`) et son répertoire parent (`..`).

4. Important : notez que les numéros d'i-nœud sont uniques **par système de fichier**, ce qui signifie qu'un i-nœud avec le même numéro peut exister sur un autre système de fichier. Cela conduit à la différence entre les i-nœuds sur disque et les i-nœuds en mémoire. Alors que deux i-nœuds peuvent avoir le même numéro sur disque s'ils sont sur deux systèmes de fichiers différents, les i-nœuds en mémoire ont un numéro unique sur tout le système. Une solution pour obtenir l'unicité est par exemple de combiner le numéro sur disque avec l'identifiant du périphérique qui héberge le système de fichiers.

Des exemples typiques de fichiers qui ne sont pas liés (ils n'ont pas de noms) sont les connexions réseau : il vous sera impossible de voir le fichier correspondant à votre connexion à Mandrake Linux ([www.mandrakelinux.com](http://www.mandrakelinux.com)) dans votre arborescence, quel que soit le répertoire que vous essayiez. De même, quand vous utilisez un *tube* dans le *shell*, le fichier correspondant au tube existe bien, mais il n'est pas lié.

## 9.4. Tubes "anonymes" et tubes nommés

Revenons à l'exemple des tubes, car il est très intéressant et il constitue en soi une bonne illustration de la notion de liens. Lors de l'utilisation d'un tube dans une ligne de commande, voici ce qui se produit : le *shell* crée le tube et va faire en sorte que la commande située avant le tube écrive dans celui-ci, et que la commande située après lise les données du tube. Tous les tubes, qu'ils soient anonymes (comme ceux utilisés par le *shell*) ou nommés (voir ci-dessous), fonctionnent selon le principe FIFO (*First In, First Out, premier arrivé, premier servi*). L'utilisation des tubes avec le *shell* a déjà été illustrée, mais il est intéressant de le faire à nouveau :

```
$ ls -d /proc/[0-9] | head -5
/proc/1/
/proc/2/
/proc/3/
/proc/4/
/proc/5/
```

Ce qui ne peut se voir dans cet exemple (parce que cela se passe trop vite) est la chose suivante : les écritures sur le tube sont bloquantes. Cela veut dire que quand la commande `ls` écrit dans le tube, elle est bloquée jusqu'à ce qu'un processus à l'autre bout lise à partir du tube. Pour visualiser cet effet, on pourra créer des tubes nommés (et qui, donc, seront liés, par opposition aux tubes utilisés par le *shell* qui ne le sont pas)<sup>5</sup>. La commande pour créer de tels tubes est `mkfifo` :

```
$ mkfifo un_tube
$ ls -il
total 0
 169 prw-rw-r--  1 reine   reine           0 déc 10 14:12 un_tube|
#
# On constate que le compteur de liens indique 1, et la sortie
# que le fichier est un tube ('p').
#
# On pourra aussi utiliser ln ici :
#
$ ln un_tube le_même_tube
$ ls -il
total 0
 169 prw-rw-r--  2 reine   reine           0 déc 10 15:37 un_tube|
 169 prw-rw-r--  2 reine   reine           0 déc 10 15:37 le_même_tube|
$ ls -d /proc/[0-9] >un_tube
#
# Le processus est bloqué puisqu'il n'y a pas de lecteurs à l'autre
# bout. Tapez C-z pour suspendre le processus...
#
zsh: 3452 suspended  ls -d /proc/[0-9] > un_tube
#
# ...Puis placez-le en arrière-plan :
#
$ bg
[1] + continued  ls -d /proc/[0-9] > un_tube
#
# Maintenant lisez depuis le tube...
#
$ head -5 <le_même_tube
#
# ...le processus écrivain se termine :
#
[1] + 3452 done      ls -d /proc/[0-9] > un_tube
/proc/1/
/proc/2/
/proc/3/
/proc/4/
/proc/5/
$
```

5. D'autres différences existent entre les deux type de tubes, mais cela sort du cadre de ce chapitre.

De même, les lectures sont bloquantes. Si les commandes ci-dessus sont exécutées dans l'ordre inverse, nous observons que `head` se bloque en attendant qu'un processus lui envoie quelque chose à lire :

```
$ head -5 <un_tube
#
# Le processus est bloqué, suspendez-le : C-z
#
zsh: 741 suspended head -5 < un_tube
#
# Mettez-le en tâche de fond...
#
$ bg
[1] + continued head -5 < un_tube
#
# ...Et donnez-lui à manger :)
#
$ ls -d /proc/[0-9] >le_même_tube
$ /proc/1/
/proc/2/
/proc/3/
/proc/4/
/proc/5/
[1] + 741 done head -6 < un_tube
$
```

On constatera également un effet indésirable dans cet exemple : la commande `ls` a fini son exécution avant que la commande `head` ne prenne le relais. La conséquence est que vous êtes renvoyé immédiatement au prompt, et `head` n'est exécutée qu'après. Elle effectue en fait sa sortie seulement une fois le prompt récupéré.

## 9.5. Les fichiers spéciaux : fichiers mode bloc et caractère

Nous avons déjà mentionné que ces fichiers peuvent être soit des fichiers créés par le système, soit des périphériques de votre machine, et que le contenu des fichiers en mode bloc était gardé en mémoire alors que tel n'était pas le cas des fichiers en mode caractère. Pour illustrer ceci, insérez une disquette quelconque dans le lecteur et tapez la commande suivante deux fois de suite :

```
$ dd if=/dev/fd0 of=/dev/null
```

Vous pouvez observer la chose suivante : le premier lancement de la commande enclenche la lecture du contenu complet de la disquette alors qu'aucun accès au lecteur de disquette n'est effectué lors de la deuxième entrée de la commande. La résolution du mystère est simple : le contenu de la disquette a été gardé en mémoire lors de la première exécution de la commande (à la condition évidente que vous n'ayez pas changé de disquette entre temps !).

Et si, maintenant, vous avez envie de passer à l'impression d'un fichier important de cette façon :

```
$ cat /un/gros/fichier/imprimable/quelque/part >/dev/lp0
```

Que vous la tapiez une, deux ou cinquante fois, la commande prendra en fait autant de temps. Tout simplement parce que `/dev/lp0` est un fichier en mode caractère et que son contenu n'est donc pas gardé en mémoire.

Le fait que le contenu des fichiers en mode bloc soit gardé en mémoire comporte un effet de bord agréable : non seulement les lectures sont gardées en mémoire, mais également les écritures. Cela permet aux écritures sur disque d'être asynchrones : quand vous écrivez un fichier sur disque, l'opération d'écriture elle-même ne sera pas accomplie immédiatement. Elle n'aura lieu que quand *Linux* décidera de lancer l'écriture physique.

Enfin, chacun de ces fichiers spéciaux possède un numéro *majeur* et un numéro *mineur*. Sur la sortie d'un `ls` -l, ils apparaissent en lieu et place de la taille, puisque celle-ci n'entre pas en ligne de compte pour de tels fichiers :

```
ls -l /dev/hda /dev/lp0
brw-rw---- 1 root disk 3, 0 May 5 1998 /dev/hda
crw-rw---- 1 root daemon 6, 0 May 5 1998 /dev/lp0
```

Les numéros majeur et mineur de `/dev/hda` sont, respectivement 3 et 0, ici, et 6 et 0 pour `/dev/lp0`. Notez que ces chiffres correspondent chacun à un type de fichier : il peut ainsi exister un fichier en mode caractère ayant 3 pour majeur et 0 pour mineur (un tel fichier existe : c'est `/dev/tty0`), et un fichier en mode bloc ayant 6 pour



majeur et 0 pour mineur. Ces numéros majeurs et mineurs existent pour une raison bien simple : ils permettent au noyau d'associer les bonnes opérations aux bons fichiers (et donc aux périphériques auxquels ces fichiers se réfèrent). En effet, on ne contrôle pas un lecteur de disquettes de la même façon que, par exemple, un disque dur SCSI.

## 9.6. Les liens symboliques et la limitation des liens en dur

Il nous faut nous confronter à un malentendu très courant, même chez les utilisateurs d'*UNIX*. Celui-ci est, essentiellement, dû au fait qu'on attache la notion de liens (d'ailleurs faussement appelés « liens en dur ») aux fichiers ordinaires uniquement. Nous avons vu que ce n'est aucunement le cas puisque même les liens symboliques sont « liés ». Il nous faudra tout d'abord clarifier ce que sont ces liens symboliques<sup>6</sup>.

Les liens symboliques sont des fichiers d'un type particulier dont le seul contenu est une chaîne de caractères arbitraire, qui peut ou non pointer sur un vrai nom de fichier. Quand vous mentionnez un lien symbolique sur la ligne de commande ou dans un programme, vous accédez en fait au fichier sur lequel pointe le lien, s'il existe. Par exemple :

```
$ echo Bonjour >monfichier
$ ln -s monfichier monlien
$ ls -il
total 4
    169 -rw-rw-r--    1 reine    reine          6 déc 10 21:30 monfichier
    416 lrwxrwxrwx    1 reine    reine          6 déc 10 21:30 monlien
-> monfichier
$ cat monfichier
Bonjour
$ cat monlien
Bonjour
```

On constatera, d'une part, que le type du fichier monlien est 'l' (les droits d'accès à un lien symbolique n'ont aucune signification : ils seront toujours `lrwxrwxrwx`) ; et d'autre part, qu'il s'agit d'un fichier distinct de monfichier, parce que son numéro d'i-nœud est différent. Mais il se réfère au fichier monfichier de façon symbolique. Ainsi, lorsque vous tapez `cat monlien`, vous affichez en fait le contenu du fichier monfichier. Pour démontrer qu'un lien symbolique contient une chaîne arbitraire, nous pouvons faire ceci :

```
$ ln -s "Je n'existe pas" unautrelien
$ ls -il unautrelien
    747 lrwxrwxrwx    1 reine    reine        15 déc 15 18:01 unautrelien
-> Je n'existe pas
$ cat unautrelien
cat: unautrelien: Aucun fichier ou répertoire de ce type
$
```

Si les liens symboliques existent, c'est parce qu'ils peuvent se libérer de certaines limites imposées aux liens normaux (« en dur ») :

- deux fichiers qui existent sur des systèmes de fichiers distincts ne peuvent être liés ; et ceci pour une raison bien simple : le compteur de liens est stocké dans l'i-nœud lui-même, et les i-nœuds ne peuvent être partagés par plusieurs systèmes de fichiers. Les liens symboliques, quant à eux, le permettent ;
- deux répertoires ne peuvent être liés car, comme nous l'avons vu, le compteur de liens pour un répertoire possède un usage particulier. Cependant, on pourra faire pointer un lien symbolique sur un répertoire et l'utiliser comme s'il s'agissait vraiment d'un répertoire.

Les liens symboliques sont souvent d'une très grande utilité dans de nombreuses situations. Cependant, il est fréquent de les voir utiliser pour lier des fichiers qui pourraient l'être par un lien normal. Pourtant, l'avantage de ce dernier est qu'il empêche la perte du fichier lorsque l'« original » est effacé.

Et puis, pour finir, vous aurez remarqué que la taille d'un lien symbolique correspond tout simplement à... la taille de la chaîne de caractères.

6. En anglais, les liens symboliques sont aussi appelés « soft links », tandis que les liens normaux sont appelés (toujours faussement) « hard links », ce qui donne l'expression française (toujours aussi fausse) de liens « en dur ».

## 9.7. Les attributs des fichiers

Si des attributs de fichiers (archive, fichier système, mode invisible) existe pour la FAT, de même *ext2fs* possédera les siens en propre, même s'ils ne sont que peu utilisés. poursuivez la lecture, si vous désirez un système vraiment sécurisé !

La manipulation des attributs se fait par l'intermédiaire de deux commandes : `lsattr` et `chattr`. Vous l'aurez deviné, `lsattr` donne une *LiSte* des attributs, et `chattr` les *CHange*. Ces attributs s'appliquent seulement aux répertoires et aux fichiers ordinaires. Ce sont les suivants :

1. **A** (*no Access time, pas de date d'accès*) : lorsque cette propriété est attribuée à un fichier ou à un répertoire, la mise à jour de la dernière date d'accès (lecture et écriture) ne se fera pas. Cela peut être utile dans le cas de lectures répétées de fichiers ou de répertoires, ce paramètre étant le seul à changer sur un i-nœud lorsque celui-ci est ouvert en lecture seule.
2. **a** (*append only, uniquement pour ajout*) : lorsque cette propriété est attribuée à un fichier, la seule opération possible lors de son ouverture en écriture sera l'ajout de données en fin de fichier. Dans le cas d'un répertoire, on ne pourra qu'y ajouter des fichiers ; il sera alors impossible de renommer des fichiers déjà existants ou d'en effacer. Précisons que seul `root` pourra apposer ou enlever cet attribut.
3. **d** (*no dump, pas de sauvegarde*) : `dump` est l'utilitaire *UNIX* standard pour faire des sauvegardes. Il sauvegardera tout système de fichiers pour lequel le compteur de sauvegarde est à 1 dans `/etc/fstab` (voir le chapitre *Systèmes de fichiers et points de montage*, page 51). Apposer cet attribut à un fichier ou à un répertoire, c'est demander que ces derniers ne soient pas pris en compte, contrairement aux autres, lors d'une sauvegarde ; et, dans le cas d'un répertoire, bien sûr, cela impliquera tous les sous-répertoires et fichiers qu'il contient.
4. **i** (*immutable, immuable*) : un fichier ou répertoire avec cet attribut ne peut tout simplement pas être modifié : on ne pourra ni le renommer, ni y ajouter un lien <sup>7</sup>. Il sera également impossible de l'effacer. Seul `root` peut apposer ou enlever cet attribut. Notez qu'il empêche également les changements de la date de dernier accès ; donc, nul besoin d'ajouter l'attribut **A** quand **i** est là.
5. **s** (*secure deletion, effacement sécurisé*) : lorsqu'un fichier ou un répertoire ayant cet attribut est effacé, les blocs du disque qu'il occupait précédemment sont remplis de zéros.
6. **S** (*Synchronous mode, mode synchrone*) : toutes les modifications apportées sur un fichier ou un répertoire possédant cet attribut sont synchrones et donc écrites immédiatement sur disque.

Ainsi, placer l'attribut '**i**' sur des fichiers système essentiels permettra d'éviter maintes mésaventures ! L'attribut '**A**' apposé aux longs fichiers, quant à lui, diminuera grandement l'activité disque. Et donc prolongera sensiblement la vie de vos batteries de portables.

---

7. Assurez-vous de bien comprendre ce que signifie « ajouter un lien » pour un fichier et un répertoire :-)

## Chapitre 10. Le système de fichiers /proc

Le système de fichiers /proc est une spécificité de *GNU/Linux*. C'est un système de fichiers virtuel, et en tant que tel il ne prend aucun espace disque. C'est un moyen très pratique d'obtenir des informations sur le système, d'autant plus que la plupart des fichiers dans ce répertoire sont lisibles (enfin, avec un peu d'habitude). En fait, beaucoup de programmes collectent des informations depuis des fichiers de /proc, les formatent à leur façon puis les affichent. C'est le cas pour tous les programmes qui affichent des informations sur les processus, et nous en avons déjà vu quelques-uns (top, ps et autres). /proc est aussi une bonne source de renseignements sur votre matériel et, de même, bon nombre de programmes sont en fait des interfaces pour les informations contenues dans /proc.

Il existe également un sous-répertoire spécial, /proc/sys. Il permet de changer ou de consulter certains paramètres du noyau en temps réel.

### 10.1. Renseignements sur les processus

Si vous listez le contenu du répertoire /proc, vous verrez beaucoup de répertoires dont le nom est un nombre. Ce sont les répertoires contenant les informations sur tous les processus fonctionnant à un instant donné sur le système :

```
$ ls -d /proc/[0-9]*
/proc/1/      /proc/302/    /proc/451/    /proc/496/    /proc/556/    /proc/633/
/proc/127/    /proc/317/    /proc/452/    /proc/497/    /proc/557/    /proc/718/
/proc/2/      /proc/339/    /proc/453/    /proc/5/      /proc/558/    /proc/755/
/proc/250/    /proc/385/    /proc/454/    /proc/501/    /proc/559/    /proc/760/
/proc/260/    /proc/4/      /proc/455/    /proc/504/    /proc/565/    /proc/761/
/proc/275/    /proc/402/    /proc/463/    /proc/505/    /proc/569/    /proc/769/
/proc/290/    /proc/433/    /proc/487/    /proc/509/    /proc/594/    /proc/774/
/proc/3/      /proc/450/    /proc/491/    /proc/554/    /proc/595/
```

Notez qu'en tant qu'utilisateur, vous pouvez seulement afficher les informations relatives à vos propres processus (et ceci est tout à fait compréhensible). Donc, connectons-nous en tant que root et voyons quelle information devient disponible depuis le processus 127 :

```
$ su
Password:
$ cd /proc/127
$ ls -l
total 0
-r--r--r-- 1 root root 0 Dec 14 19:53 cmdline
lrwx----- 1 root root 0 Dec 14 19:53 cwd -> //
-r----- 1 root root 0 Dec 14 19:53 environ
lrwx----- 1 root root 0 Dec 14 19:53 exe -> /usr/sbin/apmd*
dr-x----- 2 root root 0 Dec 14 19:53 fd/
pr--r--r-- 1 root root 0 Dec 14 19:53 maps|
-rw----- 1 root root 0 Dec 14 19:53 mem
lrwx----- 1 root root 0 Dec 14 19:53 root -> //
-r--r--r-- 1 root root 0 Dec 14 19:53 stat
-r--r--r-- 1 root root 0 Dec 14 19:53 statm
-r--r--r-- 1 root root 0 Dec 14 19:53 status
$
```

Chaque répertoire contient les mêmes entrées. Voici une brève description de quelques-unes de ces entrées :

1. **cmdline** : ce (pseudo) fichier contient l'intégralité de la ligne de commande utilisée pour invoquer le processus. Elle n'est pas formatée : il n'y a aucun espace entre le programme et ses arguments, ni de saut de ligne à la fin du fichier. Pour le rendre lisible, vous pouvez utiliser : `perl -ple 's,\00, ,g' cmdline`.
2. **cwd** : ce lien symbolique pointe vers le répertoire de travail en cours (*Current Working Directory*) du processus.
3. **environ** : ce fichier contient toutes les variables d'environnement pour le processus, sous la forme `VARIABLE=valeur`. De même que pour **cmdline**, la sortie n'est pas du tout formatée : pas de saut de ligne pour séparer les différentes variables, ni de saut de ligne à la fin non plus. Une solution pour le consulter : `perl -ple 's,\00,\n,g' environ`.
4. **exe** : c'est un lien symbolique pointant sur le fichier exécutable correspondant au processus en cours d'exécution.

5. `fd` : ce sous-répertoire contient la liste de tous les descripteurs de fichiers actuellement ouverts par le processus. Voyez ci-dessous.
6. `maps` : lorsque vous affichez le contenu de ce tube nommé (avec `cat` par exemple), vous voyez toutes les parties de l'espace d'adressage du processus qui sont actuellement des projections en mémoire de fichiers. Les champs, de gauche à droite, sont : la plage d'adresses de la projection mémoire, les permissions associées à cette projection, le décalage depuis le début du fichier où commence la projection, les numéros majeur et mineur (en hexadécimal) du périphérique sur lequel le fichier projeté se trouve, le numéro d'inœud du fichier et enfin, le nom du fichier lui-même. Lorsque le périphérique est 0 et qu'il n'y a ni numéro d'inœud, ni nom de fichier, ce sont des projections anonymes. Voyez `mmap(2)`.
7. `root` : c'est un lien symbolique qui pointe vers le répertoire racine utilisé par le processus. Habituellement, ce sera `/`, mais voyez `chroot(2)`.
8. `status` : ce fichier contient diverses informations sur le processus : le nom de l'exécutable, son état actuel, son PID et son PPID, ses UID et GID réels et effectifs, son occupation mémoire, etc.

Si nous listons le contenu du répertoire `fd` pour le processus 127, nous obtenons ceci :

```
$ ls -l fd
total 0
lrwx----- 1 root    root      64 déc 16 22:04 0 -> /dev/console
l-wx----- 1 root    root      64 déc 16 22:04 1 -> pipe:[128]
l-wx----- 1 root    root      64 déc 16 22:04 2 -> pipe:[129]
l-wx----- 1 root    root      64 déc 16 22:04 21 -> pipe:[130]
lrwx----- 1 root    root      64 déc 16 22:04 3 -> /dev/apm_bios
lr-x----- 1 root    root      64 déc 16 22:04 7 -> pipe:[130]
lrwx----- 1 root    root      64 déc 16 22:04 9 -> /dev/console
$
```

Ceci représente en fait la liste des descripteurs de fichiers ouverts par le processus. Chaque descripteur ouvert est matérialisé par un lien symbolique (dont le nom est le numéro du descripteur) : ce lien pointe vers le fichier ouvert par le biais de ce descripteur<sup>1</sup>. Vous pouvez également remarquer les permissions des liens symboliques : c'est le seul endroit où elles ont un sens pour les liens symboliques, puisqu'elles sont ici le reflet des droits avec lesquels le fichier correspondant au descripteur a été ouvert.

## 10.2. Informations sur le matériel

Outre les répertoires des différents processus, `/proc` contient aussi une foule de renseignements sur le matériel présent dans votre machine. Une liste des fichiers du répertoire `/proc` donne ceci :

```
$ ls -d [a-z]*
apm      dma      interrupts  loadavg  mounts    rtc      swaps
bus/     fb        ioports    locks    mtrr      scsi/    sys/
cmdline  filesystems kcore      meminfo  net/      self/    tty/
cpuinfo  fs/       kmsg       misc     partitions slabinfo uptime
devices  ide/      ksyms      modules  pci       stat     version
$
```

Par exemple, en ce qui concerne le contenu de `interrupts`, on constate qu'il contient la liste des interruptions actuellement utilisées par le système, ainsi que le périphérique qui les utilise. De même, `ioports` contiendra la liste des plages d'entrée/sortie actuellement activées, et enfin `dma` fera de même avec les canaux DMA. Ainsi, pour repérer un conflit, il suffira de vérifier le contenu de ces trois fichiers :

```
$ cat interrupts
CPU0
0:      127648      XT-PIC  timer
1:       5191      XT-PIC  keyboard
2:         0      XT-PIC  cascade
5:      1402      XT-PIC  xirc2ps_cs
8:         1      XT-PIC  rtc
10:        0      XT-PIC  ESS Solo1
12:     2631      XT-PIC  PS/2 Mouse
13:         1      XT-PIC  fpu
14:    73434      XT-PIC  ide0
15:    80234      XT-PIC  ide1
NMI:         0
```

1. Pour un rappel de ce que sont ces descripteurs 0, 1 et 2, voyez la section *Redirections et tubes*, page 17.

```
$ cat ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
0300-030f : xirc2ps_cs
0376-0376 : ide1
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial(auto)
1050-1057 : ide0
1058-105f : ide1
1080-108f : ESS Solo1
10c0-10cf : ESS Solo1
10d4-10df : ESS Solo1
10ec-10ef : ESS Solo1
$ cat dma
4: cascade
$
```

Ou, pour aller vraiment plus vite, utilisez la commande `lsdev`, qui regroupe les informations de ces trois fichiers et classe les informations par périphérique<sup>2</sup> :

```
$ lsdev
Device          DMA  IRQ  I/O Ports
-----
cascade         4    2
dma              0080-008f
dma1             0000-001f
dma2             00c0-00df
ESS              1080-108f 10c0-10cf 10d4-10df 10ec-10ef
fpu              13  00f0-00ff
ide0             14  01f0-01f7 03f6-03f6 1050-1057
ide1             15  0170-0177 0376-0376 1058-105f
keyboard         1  0060-006f
Mouse            12
pic1             0020-003f
pic2             00a0-00bf
rtc              8  0070-007f
serial           03f8-03ff
Solo1            10
timer            0  0040-005f
vga+             03c0-03df
xirc2ps_cs       5  0300-030f
$
```

Une énumération complète des fichiers présents serait trop longue. Néanmoins, voici la description de quelques-uns d'entre eux :

- `cpuinfo` : informe sur le ou les processeur(s) présent(s) dans votre machine.
- `modules` : liste les modules actuellement utilisés dans le noyau ainsi que leurs compteurs d'utilisation. En fait, il s'agit de la même information que celle reportée par la commande `lsmod`.
- `meminfo` : contient des informations sur l'état de la mémoire à l'instant où vous affichez son contenu. Une sortie plus clairement formatée est disponible avec la commande `free`.
- `apm` : si vous avez un ordinateur portable, afficher le contenu de ce fichier vous permet de voir l'état de votre batterie. Vous pouvez savoir si l'alimentation externe est branchée, connaître la charge courante de votre batterie, et si le *BIOS* APM de votre portable le permet (malheureusement, ce n'est pas le cas pour tous les ordinateurs portables), la durée de vie restante, en minutes. Le fichier n'est pas très lisible en tant que tel. Il est donc conseillé d'utiliser la commande `apm` à la place, qui donne les mêmes informations dans un format plus lisible (si on comprend l'anglais...).

2. `lsdev` fait partie du paquetage `procinfo`.

- bus : ce sous-répertoire contient des renseignements sur tous les périphériques trouvés sur les différents bus de votre machine. En général, ces renseignements sont peu lisibles, et sont pour la plupart traités et remis en forme par des utilitaires externes : `lspci`, `lspnp`, etc.

### 10.3. Le sous-répertoire /proc/sys

Le rôle de ce sous-répertoire est de reporter différents paramètres du noyau, et de permettre la modification de certains d'entre eux en temps réel. À la différence de tous les autres fichiers de /proc, certains fichiers de ce répertoire sont accessibles en écriture, mais seulement par root.

Une liste des répertoires et fichiers présents serait trop longue. D'une part, ceux-ci dépendent en grande partie de votre système ; d'autre part, la plupart des fichiers ne sont utiles que pour des programmes hautement spécialisés. Voici toutefois trois utilisations courantes de ce sous-répertoire :

1. autoriser le routage : même si le noyau par défaut de **Mandrake Linux** est capable de router, il faut l'y autoriser explicitement. Pour cela, il suffit de taper la commande suivante en tant que root :

```
$ echo 1 >/proc/sys/net/ipv4/ip_forward
```

Remplacez le 1 par un 0 si vous voulez interdire le routage.

2. empêcher l'usurpation d'adresse IP : (*IP Spoofing* en anglais) elle consiste à faire croire qu'un paquet provenant de l'extérieur vient de l'interface même par laquelle il arrive. C'est une technique couramment employée par les *crackers*<sup>3</sup>, mais il est possible de faire en sorte que le noyau empêche ce genre d'intrusion pour vous. Il vous suffit de taper :

```
$ echo 1 >/proc/sys/net/ipv4/conf/all/rp_filter
```

Ce type d'attaque devient alors impossible.

3. augmentation de la capacité des tables de fichiers et d'inœuds : la taille de la table des fichiers ouverts et de la table des inœuds en mémoire est dynamique sous *GNU/Linux*. Les valeurs par défaut sont très largement suffisantes pour une utilisation normale, mais elles peuvent ne pas suffire si votre machine fait office de serveur important (un serveur de bases de données par exemple). Le premier obstacle est justement le fait que les processus ne peuvent plus ouvrir de fichiers parce que la table est pleine, il faut donc en augmenter la taille. Parallèlement, il faut aussi augmenter la taille de la table des inœuds. Ces deux lignes résoudront le problème :

```
$ echo 8192 >/proc/sys/fs/file-max
$ echo 16384 >/proc/sys/fs/inode-max
```

Pour que ces paramètres soient appliqués dès le démarrage du système, il est conseillé de rajouter ces deux lignes dans `/etc/sysctl.conf` afin de ne pas devoir les retaper à chaque fois. Voir `sysctl.conf(5)`.

---

3. Et non pas les *hackers* !

## Chapitre 11. Les fichiers de démarrage : init sysv

Dans la tradition *UNIX*, il existe deux schémas de démarrage du système : le schéma BSD et le schéma « *System V* », tous les deux nommés d'après le système *UNIX* qui les a produits pour la première fois (resp. *Berkeley Software Distribution*, *distribution logicielle de Berkeley* et *AT&T UNIX System V*). Le schéma BSD est le plus simple, mais le schéma *System V* est nettement plus souple d'utilisation même s'il est moins facile à comprendre. Et ce sera précisément le rôle de ce chapitre de vous en rendre l'accès plus aisé.

### 11.1. Au commencement était init

Lorsque le système démarre, après que le noyau a tout configuré et monté la racine du système de fichiers, il exécute le programme `/sbin/init`<sup>1</sup>. `init` est le père de tous les processus du système et il est chargé d'amener le système au *niveau d'exécution* (*runlevel*) voulu. Nous reviendrons sur les niveaux d'exécution dans la section suivante.

Le fichier de configuration d'`init` est `/etc/inittab`. Ce fichier possède sa propre page de manuel (`inittab(5)`), mais nous ne décrirons ici que quelques-unes des directives.

La ligne qui doit d'abord attirer votre attention est celle-ci :

```
si::sysinit:/etc/rc.d/rc.sysinit
```

Cette directive dit à `init` que `/etc/rc.d/rc.sysinit` doit être exécuté à l'initialisation du système (si pour *System Init*) avant tout autre chose. Pour déterminer le niveau d'exécution par défaut, `init` recherche ensuite la ligne contenant l'action `initdefault` :

```
id:5:initdefault:
```

En l'occurrence, `init` sait que le niveau d'exécution par défaut est 5. Il sait également que pour entrer dans le niveau 5, il lui faudra exécuter la commande suivante :

```
15:5:wait:/etc/rc.d/rc 5
```

Vous constatez que la syntaxe pour chacun des niveaux d'exécution est similaire.

`init` est également chargé de relancer (*respawn*) certains programmes que lui seul est en mesure de relancer. C'est le cas, par exemple, de tous les programmes de connexion qui tournent sur chacun des 6 terminaux virtuels<sup>2</sup>. Pour la deuxième console virtuelle, cela donnera :

```
2:2345:respawn:/sbin/mingetty tty2
```

### 11.2. Les niveaux d'exécution

Tous les fichiers qui ressortissent au démarrage du système se trouvent dans le répertoire `/etc/rc.d`. En voici la liste :

```
$ ls /etc/rc.d
init.d/  rc.local*  rc0.d/  rc2.d/  rc4.d/  rc6.d/
rc*      rc.sysinit* rc1.d/  rc3.d/  rc5.d/
```

En premier lieu, comme nous l'avons vu, le fichier `rc.sysinit` est exécuté. C'est lui qui est chargé de mettre en place la configuration de base de la machine : type du clavier, configuration de certains périphériques, vérification des systèmes de fichiers, etc.

Puis le script `rc` est exécuté avec en argument le niveau d'exécution souhaité. Nous avons vu que le niveau d'exécution est un simple entier et que pour chaque niveau d'exécution `<x>` défini, il doit exister un répertoire correspondant `rc<x>.d`. Dans une installation typique de **Mandrake Linux**, vous pouvez donc voir que 6 niveaux d'exécution sont ainsi définis :

1. Vous comprenez donc pourquoi placer `/sbin` sur un autre système de fichiers que le système de fichiers racine est une très mauvaise idée :-)

2. En modifiant ce fichier, il vous est donc possible d'ajouter des consoles virtuelles (pour un maximum de 64) en suivant la syntaxe. Mais n'oubliez pas que `X` occupe également une console virtuelle ! Laissez-lui en au moins une...

- 0 : arrêt complet de la machine ;
- 1 : mode *mono-utilisateur* ; à utiliser en cas de gros pépin !
- 2 : mode *multiutilisateur*, sans réseau ;
- 3 : mode multiutilisateur, avec réseau ;
- 4 : inutilisé ;
- 5 : comme 3, mais avec le lancement de l'interface de login graphique ;
- 6 : redémarrage.

Intéressons-nous de plus près au contenu du répertoire `rc5.d`, par exemple :

```
$ ls rc5.d
K15postgresql@  K60atd@      S15netfs@      S60lpd@        S90xfs@
K20nfs@         K96pcmcia@    S20random@    S60nfs@        S99linuxconf@
K20rstatd@      S05apmd@     S30syslog@    S66ypasswdd@   S99local@
K20rusersd@     S10network@  S40crond@     S75keytable@
K20rwhod@       S11portmap@  S50inet@      S85gpm@
K30sendmail@    S12ypserv@   S55named@     S85httpd@
K35smb@         S13ypbind@   S55routed@    S85sound@
```

Comme vous pouvez le voir, tous les fichiers de ce répertoire sont des liens symboliques et ils ont tous une forme bien particulière. Leur forme générale est

```
<S|K><ordre><nom_du_service>
```

Le *S* signifie *Start* (soit démarrer) le service et *K* signifie *Kill* (soit arrêter) le service. Les scripts sont exécutés dans l'ordre des numéros croissants et si deux scripts ont le même numéro, c'est l'ordre alphabétique qui prévaut. On peut voir également que chacun de ces liens symboliques pointe vers des scripts situés dans `/etc/rc.d/init.d` (à l'exception de `local`), des scripts qui sont chargés de contrôler un service bien particulier.

Quand le système entre dans un niveau d'exécution donné, il exécute d'abord les liens *K* dans l'ordre : `rc` vérifie où pointe le lien, puis appelle le script correspondant avec le seul argument `stop`. Puis il exécute les scripts *S*, toujours selon la même méthode, mis à part que le script est appelé avec l'argument `start`.

Ainsi, sans citer tous les scripts, on peut voir que lorsque le système entre dans le niveau d'exécution 5, il exécute d'abord `K15postgresql`, c'est-à-dire `/etc/rc.d/init.d/postgresql stop`. Ensuite, il exécute `K20nfs`, `K20rstatd`, jusqu'au dernier ; enfin, il exécute tous les scripts *S* : `S05apmd` en premier lieu ; ce qui invoque donc `/etc/rc.d/init.d/apmd start` et ainsi de suite.

Armé de tout ceci, vous pouvez à votre guise créer un niveau d'exécution entier en quelques minutes, ou empêcher le démarrage ou l'arrêt d'un service en effaçant le lien symbolique correspondant (il existe cependant des programmes d'interface pour ce faire, en particulier *drakxservices* et *chkconfig* ; le premier étant un programme graphique).



## Chapitre 12. Installation d'un logiciel libre

On nous demande souvent comment installer un logiciel libre à partir des sources. Compiler soi-même un logiciel est en fait très simple, car la marche à suivre est la même, quel que soit le logiciel à installer. Le but de ce chapitre sera de guider pas à pas le débutant en lui expliquant sommairement la signification de chacune des manipulations pour ne pas qu'elles lui apparaissent comme le simple effet de formules magiques ! Nous supposons quand même que le lecteur possède un minimum de connaissances *UNIX* (de type *ls* ou *mkdir*).

Ce chapitre est un simple guide (probablement perfectible) et ne se prétend aucunement un manuel de référence comme tel. C'est pourquoi nous donnons à la fin un certain nombre de liens qui, nous l'espérons, répondront aux questions qui seraient demeurées sans réponse. Nous en appelons à vous, lecteur ou lectrice, pour nous aider à en améliorer le contenu.

### 12.1. Introduction

Ce qui différencie un logiciel libre d'un logiciel propriétaire, c'est l'accès au code source du logiciel<sup>1</sup>. Cela implique que les logiciels libres soient généralement distribués sous forme d'archives de fichiers sources. C'est assez déroutant pour le débutant, car l'utilisateur du logiciel doit compiler lui-même les sources du logiciel avant de pouvoir utiliser celui-ci.

Aujourd'hui, il existe des versions pré-compilées de la plupart des logiciels libres existants. L'utilisateur pressé n'a plus qu'à installer le binaire. Cependant, certains logiciels ou des versions plus récentes de ceux-ci n'existent pas (ou pas encore) sous cette forme. De plus, si vous utilisez un couple système d'exploitation / architecture complexe, beaucoup des logiciels libres qui vous intéresseraient ne sont pas encore précompilés. Par ailleurs, compiler soi-même ses logiciels permet de n'en conserver que des options particulières ou d'en étendre les fonctionnalités par le biais d'extensions ciblées, répondant parfaitement à ses besoins.

#### 12.1.1. Pré-requis

Pour installer un logiciel libre, vous aurez besoin :

- d'un ordinateur allumé, pourvu d'un système d'exploitation,
- d'un peu d'espace disque,
- d'un compilateur (généralement pour le langage *C*), d'un programme d'archivage (*tar*),
- d'une connaissance généraliste du système d'exploitation que vous utilisez,
- de quoi manger (dans le pire des cas, cela peut effectivement durer longtemps ; au fait, un vrai *hacker* mange des steaks et pas des petits fours !),
- de quoi boire (pour les mêmes raisons ; et un vrai informaticien boit des sodas... bourrés de caféine ; pas de pub clandestine !).
- du numéro de téléphone de votre copain bidouilleur qui recompile son noyau toutes les semaines,
- mais surtout de patience (beaucoup).

Compiler un logiciel libre ne présente généralement pas trop de problèmes, mais si vous n'êtes pas habitué, la moindre anicroche peut vous plonger dans la confusion. Nous allons donc ici vous montrer comment vous sortir, sans trop de bleus, de toutes les situations difficiles !

#### 12.1.2. Compilation

##### 12.1.2.1. Principe

Pour passer d'une forme source à une forme binaire, il est nécessaire d'effectuer une *compilation*. Cette compilation est généralement effectuée sur des programmes écrits en langage *C* ou *C++* (qui sont les plus répandus dans la communauté du logiciel libre, notamment dans le monde *UNIX*). Certains logiciels libres sont écrits dans des langages ayant nul besoin de compilation (par exemple *perl* ou le *shell*), mais ils doivent quand même être configurés.

---

1. Ce n'est pas tout à fait vrai car certains logiciels propriétaires fournissent également leur code source. Mais à la différence d'un logiciel libre, l'utilisateur, en bout de chaîne, lui n'a pas le droit d'en faire ce qu'il veut.

La compilation *C* est assurée, très logiquement, par un compilateur qui est généralement *gcc*, le compilateur libre écrit par le projet GNU (<http://www.gnu.org/>). La compilation d'un logiciel entier est une tâche complexe, qui passe par la compilation successive de multiples fichiers sources (il est plus facile pour le programmeur d'isoler les différentes parties de son travail dans des fichiers distincts, pour diverses raisons). Afin de rendre cette tâche plus aisée, ces opérations répétitives sont effectuées par un utilitaire du nom de *make*.

### 12.1.2.2. Les quatre phases de la compilation

Pour bien comprendre le mécanisme de la compilation (et donc être à même de résoudre des problèmes éventuels), il faut savoir qu'elle s'effectue en quatre phases qui convertissent progressivement un texte écrit en un langage compréhensible par un humain entraîné (le langage *C* par exemple) vers un langage compréhensible par une machine (ou un humain **très** entraîné, et encore, dans de rares cas). *gcc* exécutera l'un après l'autre quatre programmes qui se chargeront chacun d'une étape particulière :

1. *cpp* : la première étape consiste à remplacer des directives (**pré-processeur**) par des instructions *C*. Typiquement, il s'agit d'insérer un fichier d'en-têtes (`#include`) ou de définir une macro fonction (`#define`). À la fin de cette phase, un code purement *C* est engendré.
2. *cc1* : cette étape consiste à convertir du *C* en **langage d'assemblage**. Le code généré est dépendant de l'architecture cible.
3. *as* : cette étape consiste à générer du code **objet** (ou **binnaire**) à partir du langage d'assemblage. À la fin de cette phase, un fichier se terminant par `.o` est généré.
4. *ld* : cette dernière étape (**l'édition de liens**, en anglais « *linkage* ») assemble (ou *lie*) tous les fichiers objets (`.o`) et les bibliothèques associées, et génère un exécutable.

### 12.1.3. Structure d'une distribution

Une distribution de logiciel libre correctement structurée est généralement organisée d'une manière bien précise :

- un fichier `INSTALL`, qui décrit la procédure d'installation du logiciel,
- un fichier `README` qui contient toutes les informations générales relatives au programme (courte description, auteur, adresse où le télécharger, documentation relative, pointeurs utiles, ...). Si le fichier `INSTALL` est absent, le fichier `README` contient généralement une procédure succincte d'installation;
- un fichier `COPYING` qui contient la licence ou décrit les conditions de distribution du logiciel ; parfois, c'est un fichier appelé `LICENCE` qui le remplace;
- un fichier `CONTRIB` ou `CREDITS` qui contient une liste de personnes ayant un rapport avec le logiciel (participation active, remarques pertinentes, logiciel tiers, etc.);
- un fichier `CHANGES` (ou, plus rarement, `NEWS`), qui contient les nouveautés de la version actuelle par rapport à la version précédente et les corrections de bogues;
- un fichier `Makefile` (voir la section *make*, page 75), qui permet de compiler le logiciel (c'est un fichier nécessaire à *make*). Parfois, ce fichier n'existe pas encore et sera généré lors du processus de configuration;
- assez souvent, un fichier `configure` ou `Imakefile`, qui permettra de générer un nouveau fichier `Makefile`;
- un répertoire contenant les sources, qui sera généralement celui où le binaire sera stocké une fois la compilation terminée; son nom est généralement `src`;
- un répertoire contenant la documentation relative au programme (généralement au format `man` ou *Texinfo*), dont le nom est généralement `doc`;
- éventuellement, un répertoire contenant des données propres au logiciel (typiquement, des fichiers de configuration, des exemples de données produites, ou des fichiers de ressources).

## 12.2. Décompactage

### 12.2.1. Archive tar.gz

La norme<sup>2</sup> pour la compression sous *UNIX* est le format *gzip*, développé par le projet GNU, et considéré comme un des meilleurs outils de compression généralistes.

*gzip* est souvent associé à un utilitaire nommé *tar*. Celui-ci est un rescapé des temps préhistoriques où les informaticiens stockaient leurs informations sur des bandes magnétiques. Aujourd'hui, les disquettes et les CD-ROM ont remplacé les bandes magnétiques, mais *tar* est toujours utilisé pour créer des archives. Il est par exemple possible de concaténer (mettre à la suite les uns des autres) tous les fichiers d'un répertoire dans un seul fichier grâce à *tar*. Ce fichier peut ensuite être facilement compressé à l'aide de *gzip*.

C'est pourquoi de nombreux logiciels libres sont disponibles sous la forme d'archives *tar*, compressées avec *gzip*. Leur extension est donc *.tar.gz* (ou encore, sous forme abrégée, *.tgz*).

### 12.2.2. Utilisation de GNU Tar

Pour décompresser cette archive, on peut utiliser *gzip*, puis *tar* ensuite. Mais la version GNU de *tar* (*tar*) permet d'utiliser *gzip* « *à la volée* », et ainsi de décompresser une archive de manière transparente, sans nécessiter d'espace disque supplémentaire.

L'utilisation de *tar* est d'une grande simplicité :

```
tar <options de fichier> <fichier en .tar.gz> [fichiers]
```

L'option <fichiers> est facultative. Dans le cas où elle est omise, le traitement s'effectuera sur toute l'archive. Si vous voulez extraire le contenu d'une archive *.tar.gz*, alors vous n'avez certainement pas besoin de spécifier cet argument.

Par exemple :

```
$ tar xvfz guile-1.3.tar.gz
-rw-r--r-- 442/1002      10555 1998-10-20 07:31 guile-1.3/Makefile.in
-rw-rw-rw- 442/1002      6668 1998-10-20 06:59 guile-1.3/README
-rw-rw-rw- 442/1002      2283 1998-02-01 22:05 guile-1.3/AUTHORS
-rw-rw-rw- 442/1002     17989 1997-05-27 00:36 guile-1.3/COPYING
-rw-rw-rw- 442/1002     28545 1998-10-20 07:05 guile-1.3/ChangeLog
-rw-rw-rw- 442/1002      9364 1997-10-25 08:34 guile-1.3/INSTALL
-rw-rw-rw- 442/1002      1223 1998-10-20 06:34 guile-1.3/Makefile.am
-rw-rw-rw- 442/1002     98432 1998-10-20 07:30 guile-1.3/NEWS
-rw-rw-rw- 442/1002      1388 1998-10-20 06:19 guile-1.3/THANKS
-rw-rw-rw- 442/1002      1151 1998-08-16 21:45 guile-1.3/TODO
...
```

Parmi les options à passer à *tar* :

- *v* permet de rendre *tar* « verbeux » : c'est-à-dire qu'il affichera à l'écran tous les fichiers qu'il trouve dans l'archive ; si cette option est omise, alors le traitement sera silencieux;
- *f* est une option obligatoire. Sans elle, *tar* essaiera d'utiliser une bande magnétique à la place d'un fichier d'archive (c'est-à-dire le périphérique */dev/rmt0*);
- *z* permet de manipuler une archive compressée par *gzip* (avec suffixe de nom de fichier en *.gz*). Si vous oubliez cette option lors du décompactage d'une archive compressée, alors *tar* produira une erreur. Inversement, si vous êtes en face d'une archive non compressée, n'utilisez pas cette option.

*tar* permet d'effectuer plusieurs manipulations différentes sur une archive (extraction, lecture, création, ajout...). Une option permet de spécifier l'usage qui en est fait :

- *x* : permet d'extraire des fichiers de l'archive;
- *t* : liste le contenu de l'archive;

2. De plus en plus, on utilise un nouveau programme, appelé *bzip2*, plus efficace sur les fichiers texte, et plus gourmand en mémoire et en puissance de calcul. Voir plus loin, la section *bzip2*, page 72 à ce sujet.

- `c` : permet de créer une archive, ce qui implique de détruire son contenu actuel. Vous n'utiliserez probablement cette option que dans le cas de votre usage personnel (vos sauvegardes, par exemple);
- `r` : permet d'ajouter des fichiers à la fin de l'archive. Elle ne fonctionne pas dans le cas d'une archive compressée.

### 12.2.3. bzip2

Un format de compression nommé bzip2 tend en ce moment à remplacer gzip. bzip2 produit des archives de taille plus petite que gzip, mais n'est pas encore une norme de fait. On trouve donc depuis peu des archives à l'extension `.tar.bz2`.

bzip2 s'utilise de la même manière que gzip par le biais de la commande tar. Il suffit de remplacer la lettre z par la lettre j. Par exemple :

```
$ tar xvjf toto.tar.bz2
```

A la place, certaines distributions utilisent (ou ont utilisé) l'option `I` :

```
$ tar xvfI toto.tar.bz2
```

Une autre possibilité (qui semble être plus portable, mais plus longue à taper !) :

```
$ tar --use-compress-program=bzip2 -xvf toto.tar.bz2
```

Précisons qu'il faut avoir installé bzip2 et l'avoir inclus dans la variable PATH avant d'exécuter tar.

### 12.2.4. C'est tout bête !

#### 12.2.4.1. Le plus simple

Maintenant que vous êtes prêt(e) à décompresser l'archive, n'oubliez pas de le faire en tant qu'administrateur (root). En effet, vous allez avoir besoin de faire des manipulations qu'un simple utilisateur ne peut faire, et même si ce n'est pas le cas de toutes ces manipulations, il est plus facile d'agir en tant que root pendant toute la durée de l'opération.

Commencez par vous rendre dans le répertoire `/usr/local/src`, et copiez l'archive dans ce répertoire. Cela vous permet de retrouver à tout moment l'archive si vous perdez le logiciel installé. Si vous n'avez pas beaucoup d'espace disque, alors sauvegardez l'archive sur disquette après avoir installé le logiciel, ou effacez-la, mais assurez-vous de pouvoir la retrouver sur le réseau à tout moment.

Normalement, le décompactage d'une archive tar devrait créer un nouveau répertoire (détail dont vous pouvez vous être assuré(e) grâce à l'option `t`). Rendez-vous maintenant dans ce répertoire, vous êtes prêt(e) à continuer.

#### 12.2.4.2. Le plus sûr

Le système *UNIX* (dont font partie *GNU/Linux* et *FreeBSD*) est un système sécurisé. Cela signifie que les utilisateurs normaux ne peuvent pas effectuer des opérations qui mettraient le système en danger (comme par exemple formater un disque), ni altérer les fichiers des autres utilisateurs. Dans la pratique et en particulier, cela immunise aussi le système contre les virus.

En revanche, l'utilisateur root a le droit de tout faire, y compris d'exécuter un programme malveillant (tel virus). Disposer du code source est une garantie de sécurité face aux virus, mais vous avez tout à fait le droit d'être paranoïaque<sup>3</sup>.

L'idée consiste à créer un utilisateur dédié à l'administration (*free* ou *admin* par exemple) par le biais de la commande *adduser*. Ce compte devra avoir le droit d'écrire dans le répertoire `/usr/local/src` ainsi que dans les répertoires `/usr/local/bin`, `/usr/local/lib` et toute l'arborescence de `/usr/man` (il se peut qu'il

---

3. Un proverbe venant du monde BSD dit : « *never trust a software you don't have the sources for* » (ne faites jamais confiance en un paquetage dont vous n'avez pas les sources.)

ait également besoin de copier des fichiers ailleurs). Pour cela, je vous recommande soit de rendre cet utilisateur propriétaire des répertoires nécessaires, soit de créer un groupe pour lui, et de rendre ces répertoires accessibles en écriture pour ce groupe.

Une fois que ces précautions sont prises, vous pouvez effectuer les manipulations décrites dans la section *Le plus simple*, page 72.

## 12.3. Configuration

Un intérêt purement technique du fait de disposer des sources est le *portage* du logiciel. Un logiciel libre développé pour un *UNIX* est utilisable sur tous les *UNIX* existants (libres ou propriétaires), avec cependant quelques modifications. Ceci implique une configuration du logiciel juste avant la compilation.

Il existe plusieurs systèmes de configuration, il va falloir utiliser celui que l'auteur du logiciel a prévu (parfois, plusieurs sont au programme). Généralement, vous pouvez :

- utiliser *AutoConf* (voir la section *AutoConf*, page 73) au cas où un fichier nommé *configure* existe dans le répertoire parent de la distribution;
- utiliser *imake* (voir la section *imake*, page 75) dans le cas où un fichier nommé *Imakefile* existe dans le répertoire parent de la distribution;
- exécuter un *script shell* (par exemple *install.sh*) selon ce que dit le fichier *INSTALL* (ou le fichier *README*).

### 12.3.1. AutoConf

#### 12.3.1.1. Principe

*AutoConf* permet de configurer correctement un logiciel. Il crée les fichiers nécessaires à la compilation (par exemple, *Makefile*), et modifie parfois directement les sources (comme par le biais d'un fichier *config.h.in*).

Le principe d'*AutoConf* est simple :

- l'auteur du logiciel sait quels tests sont nécessaires pour configurer son logiciel (ex : « quelle version de cette *bibliothèque* est utilisée ? »). Il les écrit dans un fichier nommé *configure.in*, en suivant une syntaxe précise;
- il exécute *AutoConf*. Ce dernier génère, à partir du fichier *configure.in*, un script de configuration appelé *configure*. Ce script est celui qui effectuera les tests nécessaires à la configuration du programme;
- l'utilisateur final exécute ce script, et *AutoConf* se charge de configurer tout ce qui est nécessaire à la compilation.

#### 12.3.1.2. Exemple

Un exemple d'utilisation d'*AutoConf* :

```
$ ./configure
loading cache ./config.cache
checking for gcc... gcc
checking whether the C compiler (gcc ) works... yes
checking whether the C compiler (gcc ) is a cross-compiler... no
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking for main in -lX11... yes
checking for main in -lXpm... yes
checking for main in -lguile... yes
checking for main in -lm... yes
checking for main in -lncurses... yes
checking how to run the C preprocessor... gcc -E
checking for X... libraries /usr/X11R6/lib, headers /usr/X11R6/include
checking for ANSI C header files... yes
checking for unistd.h... yes
checking for working const... yes
```

```
updating cache ./config.cache
creating ./config.status
creating lib/Makefile
creating src/Makefile
creating Makefile
```

Dans le cas où on désirerait avoir un peu plus de contrôle sur ce qu'engendre la commande `configure`, on peut lui passer des options via la ligne de commande ou des variables d'environnement. Exemple :

```
$ ./configure --with-gcc --prefix=/opt/GNU
```

ou encore (sous *bash*) :

```
$ export CC='which gcc'
$ export CFLAGS=-O2
$ ./configure --with-gcc
```

ou :

```
$ CC=gcc CFLAGS=-O2 ./configure
```

### 12.3.1.3. Et si... cela ne fonctionne pas ?

Typiquement, il s'agit d'une erreur de type `configure: error: Cannot find library guile (configure: erreur: je n'arrive pas à trouver la bibliothèque guile)` (cela couvre la plupart des erreurs du script `configure`).

Clairement, le script `configure` n'a pas réussi à trouver une bibliothèque (dans notre exemple, la bibliothèque `guile`). Le principe est que le script `configure` compile un petit programme de test qui utilise cette bibliothèque. S'il n'arrive pas à le compiler, c'est qu'il n'arrivera pas à compiler le logiciel. D'où l'erreur !

- Regardez d'où vient l'erreur rencontrée en examinant la fin du fichier `config.log`, qui contient une trace de toutes les étapes de la configuration. Le compilateur de langage *C* est généralement assez explicite dans ses messages d'erreur. Cela vous aidera à résoudre le problème.
- Vérifiez que la bibliothèque en question est bien installée. Si ce n'est pas le cas, installez-la (à partir des sources ou d'un binaire précompilé) et exécutez à nouveau le script `configure`. Une méthode efficace pour vérifier l'installation est de rechercher le fichier matérialisant la bibliothèque, qui porte invariablement le nom `lib<nom>.so`. Par exemple,

```
$ find / -name 'libguile*'
```

ou encore :

```
$ locate libguile
```

- Vérifiez qu'elle est accessible au compilateur, c'est-à-dire que la bibliothèque est bien située dans un répertoire parmi : `/usr/lib`, `/lib`, `/usr/X11R6/lib`; ou parmi ceux spécifiés par la variable d'environnement `LD_LIBRARY_PATH` (expliquée dans *Et si ça ne fonctionne pas ?*, page 77 numéro 5.b). Vérifiez que ce fichier est bien une bibliothèque en tapant `file libguile.so`.
- Vérifiez que les fichiers d'en-têtes correspondant à la bibliothèque sont bien installés à la bonne place (généralement, `/usr/include`, `/usr/local/include`, ou `/usr/X11R6/include`). Si vous ne savez pas de quels fichiers d'en-têtes vous avez besoin, vérifiez que vous avez bien installé la version de développement de la bibliothèque nécessaire (par exemple, `gtk+-devel` au lieu de `libgtk`). La version de développement de la bibliothèque est livrée avec les fichiers de type `include` (à *inclure*) nécessaires à la compilation d'un logiciel utilisant cette bibliothèque.
- Vérifiez que vous avez assez de place disque (en effet le script `configure` a besoin d'un peu de place pour des fichiers temporaires). Utilisez la commande `df -k` pour visualiser les partitions de votre système, et ne vous préoccupez que des partitions pleines ou presque.

Si vous ne comprenez pas le message d'erreur stocké dans le fichier `config.log`, n'hésitez pas à demander aide et assistance à la communauté du logiciel libre (voir la section *Assistance technique*, page 82).

De plus, méfiez-vous si `configure` répond 100% de No ou s'il répond No alors que vous êtes certain qu'une bibliothèque existe (par exemple il serait très étrange qu'il n'existe pas sur votre système au moins une bibliothèque curses). Dans de tels cas, on est probablement en présence d'une variable `LD_LIBRARY_PATH` mal positionnée !

### 12.3.2. imake

*imake* permet de configurer un logiciel libre en créant un fichier *Makefile* à partir de règles simples. Ces règles déterminent quels fichiers ont besoin d'être compilés pour construire le binaire, et *imake* engendre le *Makefile* correspondant. Ces règles sont spécifiées dans un fichier appelé *Imakefile*.

Là où *imake* prend tout son intérêt, c'est qu'il utilise des informations dépendantes du *site* (de l'architecture de la machine). C'est assez pratique dans le cas d'applications utilisant *X Window System*. Mais *imake* est utilisé dans le cas de nombreuses autres applications.

L'utilisation la plus simple de *imake* est de se rendre dans le répertoire principal de l'archive décompactée, et ensuite d'exécuter le script *xmkmf*, qui fera appel au programme *imake* :

```
$ xmkmf -a
imake -DUseInstalled -I/usr/X11R6/lib/X11/config
make Makefiles
```

Si le site est mal installé, recompiler et installer *X11R6* !

### 12.3.3. Scripts shell divers

Lisez le fichier *INSTALL* ou *README* pour de plus amples informations. Généralement, il va vous falloir exécuter un fichier de type *install.sh* ou *configure.sh*. Ensuite, soit le script d'installation sera silencieux (et déterminera tout seul ce dont il a besoin), soit il vous demandera des informations sur votre système (par exemple, des chemins).

Si vous n'arrivez pas à déterminer le fichier que vous devez exécuter, vous pouvez (sous *bash*) taper *./*, et ensuite taper deux fois la touche TAB (touche de tabulation). *bash* complétera automatiquement (dans sa configuration par défaut) par un éventuel fichier exécutable du répertoire (donc, un éventuel script de configuration). Dans le cas où plusieurs fichiers sont exécutables, il vous en donnera une liste. Il ne vous reste plus qu'à choisir le bon.

Un cas particulier est l'installation de modules *perl* (mais pas seulement). L'installation de tels modules se fait par l'exécution d'un script de configuration lui-même écrit en *perl*. La commande à effectuer est généralement :

```
$ perl Makefile.PL
```

### 12.3.4. Autres possibilités

Certaines distributions de logiciels libres sont mal organisées, surtout lors des premières phases de développement (et un homme prévenu en vaut deux !). Elles nécessitent parfois de retoucher « à la main » les fichiers de configuration. Généralement, ces fichiers sont un fichier *Makefile* (voir la section *make*, page 75) et un fichier *config.h* (mais ce nom n'est qu'une convention).

Je ne recommande ces manipulations qu'à des utilisateurs sachant ce qu'ils font. C'est un travail qui nécessite des connaissances réelles et une motivation nécessaire pour réussir. Mais c'est en forgeant qu'on devient forgeron.

## 12.4. Compilation

Maintenant que le logiciel est correctement configuré, il ne reste plus qu'à le compiler. C'est une étape qui est généralement très simple à effectuer, et qui ne pose pas de problèmes majeurs.

### 12.4.1. make

L'outil préféré de la communauté du logiciel libre pour compiler des sources est *make*. L'intérêt de *make* est double :

- il permet au développeur de gagner du temps, car il présente des avantages permettant de gérer la compilation de son projet de manière efficace,

- il permet à l'utilisateur final de compiler et d'installer le logiciel en quelques lignes de commande, et ceci sans connaissance préalable du développement.

Les actions à exécuter pour arriver à une version compilée des sources sont stockées dans un fichier nommé habituellement `Makefile`, ou `GNUMakefile`. En fait, lorsque `make` est invoqué, il lit ce fichier, s'il existe, dans le répertoire courant. Si ce n'est pas le cas, il est possible de spécifier ce fichier en passant l'option `-f` à `make`.

### 12.4.2. Règles

`make` fonctionne selon un système de *dépendances*. C'est-à-dire que pour qu'un binaire soit compilé (« *cible* »), un certain nombre d'étapes doivent être accomplies (« dépendances »). Par exemple, pour créer le binaire (imaginaire) `glloq`, on a besoin de compiler les fichiers objets (fichiers intermédiaires de la compilation) `main.o` et `init.o`, puis de les lier. Ces fichiers objets sont eux aussi des cibles, dont les dépendances sont les fichiers sources.

Ceci n'est qu'une introduction sommaire pour survivre dans le monde impitoyable de `make`. Si vous voulez en savoir plus, nous vous conseillons de vous rendre sur le site d'APRIL (<http://www.april.org/groupe/doc/>) pour une documentation un peu plus détaillée sur `make`. Pour une documentation exhaustive, voir *Managing Projects with Make* (*La gestion de projets avec make* (seconde édition) d'Andrew Oram et Steve Talbott chez O'Reilly).

### 12.4.3. C'est parti !

Généralement, l'utilisation de `make` obéit à plusieurs conventions. Par exemple :

- `make` sans argument exécute la compilation seule du programme, sans l'installer;
- `make install` compile le programme (mais pas toujours) et assure l'installation des fichiers nécessaires à la bonne place sur le système de fichiers. Certains fichiers ne sont pas toujours installés correctement (`man`, `info`), il faut alors les copier à la main. Dans certains cas, il faut effectuer une nouvelle fois un `make install` dans des sous-répertoires. Généralement, il s'agit de modules développés par des tiers.
- `make clean` efface tous les fichiers temporaires créés par la compilation, y compris le fichier exécutable dans la majorité des cas.

La première étape est de compiler le programme, et donc de taper (exemple fictif) :

```
$ make
gcc -c glloq.c -o glloq.o
gcc -c init.c -o init.o
gcc -c main.c -o main.o
gcc -lgtk -lgdk -lglib -lXext -lX11 -lm glloq.o init.o main.o -o glloq
```

Parfait ! le binaire est compilé correctement. Nous sommes prêts à passer à l'étape suivante, qui est l'installation des fichiers de la distribution (binaires, fichiers de données, etc...); (voir la section *Installation*, page 80).

### 12.4.4. Explications

Si vous avez la curiosité de regarder ce qu'il y a dans le fichier `Makefile`, vous y trouverez des commandes connues (`rm`, `mv`, `cp`, ...), mais aussi des chaînes de caractères étranges, de la forme `$(CFLAGS)`.

Il s'agit de *variables*, c'est-à-dire de chaînes qui sont fixées généralement au début du fichier `Makefile`, et qui seront ensuite remplacées par la valeur qui leur a été associée. C'est assez pratique pour utiliser plusieurs fois de suite les mêmes options de compilation.

Par exemple, pour afficher la chaîne « `toto` » à l'écran en tapant un `make all` :

```
TEST = toto
all:
    echo $(TEST)
```

La plupart du temps, les variables suivantes sont définies :



1. CC : il s'agit du compilateur que l'on va utiliser. Généralement, il s'agit de gcc, mais sur la plupart des systèmes libres, le compilateur par défaut utilisé par make (soit cc) est un synonyme de cc. Dans le doute, n'hésitez pas à mettre ici gcc;
2. LD : il s'agit du programme utilisé parfois pour assurer la phase finale de la compilation (voir la section *Les quatre phases de la compilation*, page 70); par défaut, la valeur est ld;
3. CFLAGS : ce sont les arguments supplémentaires qu'on passera au compilateur lors des premières phases de la compilation. Parmi ceux-ci :
  - -I<chemin> : spécifie au compilateur où chercher des fichiers d'en-têtes supplémentaires (ex : -I/usr/X11R6/include permet d'inclure les fichiers d'en-têtes se situant dans /usr/X11R6/include);
  - -D<symbole> : définit un symbole supplémentaire, utile dans certains programmes qui se compilent différemment selon les symboles définis (ex : utilise le fichier string.h si HAVE\_STRING\_H est défini).

On trouve souvent des lignes de compilation de la forme :

```
$(CC) $(CFLAGS) -c toto.c -o toto.o
```

4. LDFLAGS (ou LFLAGS) : ce sont les arguments passés lors de la dernière phase de la compilation. Parmi ceux-ci :
  - -L<chemin> : spécifie un chemin supplémentaire où chercher des bibliothèques (ex : -L/usr/X11R6/lib);
  - -l<bibliothèque> : spécifie une bibliothèque supplémentaire à utiliser lors de la dernière phase de compilation.

#### 12.4.5. Et si ça ne fonctionne pas ?

Pas de panique, cela arrive à tout le monde. Parmi les causes les plus communes :

1. glloq.c:16: decl.h: No such file or directory ( glloq.c:16 :decl.h: aucun fichier ou répertoire ne porte ce nom)

Le compilateur n'a pas réussi à trouver le fichier d'en-têtes correspondant. Pourtant, l'étape de configuration du logiciel aurait dû anticiper cette erreur. Comment résoudre ce problème :

- vérifiez que l'en-tête existe vraiment sur le disque dans un des répertoires suivants : /usr/include, /usr/local/include, /usr/X11R6/include ou un de leurs sous-répertoires. Si ce n'est pas le cas, recherchez-le sur tout le disque (avec find ou locate), et si vous ne le trouvez toujours pas, alors vérifiez à deux fois que vous avez installé la bibliothèque correspondant à cette en-tête. Vous trouverez des exemples des commandes find et locate dans leurs pages de manuel respectives;
- vérifiez que l'en-tête est bien accessible en lecture (tapez less <chemin>/<fichier>.h pour tester cela);
- s'il se trouve dans un répertoire comme /usr/local/include ou /usr/X11R6/include, il est parfois nécessaire de passer un argument supplémentaire au compilateur. Ouvrez le fichier Makefile correspondant (prenez garde à ouvrir le bon, celui qui se trouve dans le répertoire où la compilation échoue<sup>4</sup>) avec votre éditeur de textes favori (Emacs, Vi, ...). Recherchez la ligne fautive, et ajoutez la chaîne de caractères -I<path> (où <chemin> est le chemin où se trouve l'en-tête en question juste après l'appel du compilateur (gcc, ou parfois \$(CC)). Si vous ne savez pas où rajouter cette option, rajoutez-la au début du fichier, à la fin de la ligne CFLAGS=<quelquechose> ou de la ligne CC=<quelquechose>;
- exécutez make de nouveau, et si cela ne fonctionne toujours pas, vérifiez que cette option (cf point précédent) est bien ajoutée à la compilation sur la ligne fautive;
- si cela ne fonctionne toujours pas, demandez à votre gourou local ou faites appel à la communauté du logiciel libre pour résoudre votre problème (voir la section *Assistance technique*, page 82).

4. Analysez le message d'erreur renvoyé par make. Normalement, les dernières lignes devraient contenir un répertoire (un message de la forme make[1] : Leaving directory '/home/benj/Projet/toto'). Repérez celle dont le numéro est le plus grand. Pour vérifier qu'il s'agit bien du bon répertoire, rendez-vous dans ce répertoire, et exécutez make à nouveau pour obtenir la même erreur.

2. `glloq.c:28: 'struct toto' undeclared (first use this function) (glloq.c:28: « struct toto » n'est pas déclarée (ceci est la première utilisation de cette fonction))`

Les structures sont des types de données spéciaux, que tous les programmes utilisent. Beaucoup sont définies par le système dans les fichiers d'en-têtes. Ce qui signifie que le problème vient certainement d'une en-tête manquante ou mal utilisée. La marche à suivre pour résoudre le problème est la suivante :

- tenter de vérifier si la structure en question est une structure définie dans le programme ou bien par le système. Une solution est d'utiliser la commande `grep` afin de vérifier si la structure est définie dans un des fichiers d'en-têtes.

Par exemple, après vous être rendu dans la racine de la distribution :

```
$ find . -name '*.h' | xargs grep 'struct toto' | less
```

il est possible que plusieurs dizaines de lignes apparaissent à l'écran (à chaque fois qu'une fonction utilisant ce type de structure est définie, par exemple). Si elle existe, repérez la ligne où la structure est définie en regardant le fichier d'en-têtes obtenu par l'utilisation de `grep`.

La définition d'une structure est :

```
struct toto {
    <contenu de la structure>
};
```

Vérifiez si cela correspond à ce que vous avez. Si ce n'est pas le cas, c'est que ce fichier d'en-têtes n'est pas inclus dans le fichier `.c` fautif. Deux solutions s'offrent alors à vous :

- ajouter la ligne `#include "<nom_du_fichier>.h"` au début du fichier `.c` fautif.
  - ou bien copier-coller la définition de la structure au début de ce fichier (ce qui n'est pas très propre, mais ça a le mérite de fonctionner, en général).
- 
- si ce n'est pas le cas, faites la même chose sur les fichiers d'en-têtes du système (qui se trouvent sur `/usr/include`, `/usr/X11R6/include`, ou `/usr/local/include` en général). Mais cette fois-ci, utilisez la ligne `#include <<nom_du_fichier>.h>`.
  - si cette structure n'existe toujours pas, essayez de trouver dans quelle bibliothèque (au sens d'ensemble de fonctions regroupées dans un seul paquetage), elle devrait être définie (regardez dans le fichier `INSTALL` ou `README` quelles sont les bibliothèques utilisées par le programme et la version nécessaire). Si la version que le programme nécessite n'est pas celle installée sur votre système, alors effectuez une mise à jour de cette bibliothèque ;
  - si cela ne fonctionne toujours pas, vérifiez que le programme fonctionne correctement sur votre architecture (certains programmes n'ont pas encore été portés sur tous les systèmes *UNIX*). Vérifiez aussi que vous avez bien configuré le programme (au moment du configure, par exemple) pour votre architecture.

### 3. `parse error` (erreur d'analyse syntaxique)

C'est un problème assez compliqué à résoudre, car généralement il s'agit d'une erreur que le compilateur rencontre plus haut, mais qui ne se manifeste qu'à une certaine ligne. Parfois, il s'agit simplement d'un type de donnée qui n'est pas défini. Si vous rencontrez un message d'erreur de type :

```
main.c:1: parse error before 'glloq_t'
main.c:1: warning: data definition has no type or storage class
```

alors, le problème est que le type `glloq_t` n'est pas défini. La solution pour résoudre ce problème est environ la même que pour le problème précédent.



il peut y avoir une erreur de type `parse error` dans les vieilles bibliothèques curses si ma mémoire est bonne.

### 4. `no space left on device` (plus de place disponible sur le périphérique)

Le problème est assez simple à régler : la place sur le disque est insuffisante pour générer un binaire à partir du fichier source. La solution consiste à libérer de la place dans la partition abritant le répertoire d'installation : supprimez les fichiers temporaires ou les sources, désinstallez les programmes dont vous ne vous servez pas. Si vous l'avez décompacté dans `/tmp`, faites-le plutôt dans `/usr/local/src` ce qui évite de saturer inutilement la partition `/tmp`. Vérifiez de plus que vous n'avez pas de *fichiers* core <sup>5</sup> sur le disque. Si oui, effacez-les ou faites-les effacer s'ils appartiennent à un autre utilisateur.

5. `/usr/bin/ld: cannot open -lgllloq: No such file or directory (/usr/bin/ld: je ne peux pas ouvrir -lgllloq : aucun fichier ou répertoire ne porte ce nom)`

Clairement, le programme `ld` (utilisé par `gcc` lors de la dernière phase de la compilation) n'a pas réussi à trouver une bibliothèque. Il faut savoir que pour inclure une bibliothèque, `ld` va chercher un fichier dont le nom est passé par l'argument `-l<bibliothèque>`. Ce fichier porte le nom de `lib<bibliothèque>.so`. Si `ld` n'arrive pas à le trouver, alors il produit ce message d'erreur. Pour résoudre ce problème, procédons par étapes :

- a. Vérifions que ce fichier existe bien sur le disque dur, en utilisant la commande `locate`. Généralement, les bibliothèques graphiques se trouvent dans `/usr/X11R6/lib`. Par exemple :

```
$ locate libgllloq
```

Si cela ne produit rien, vous pouvez faire une recherche avec la commande `find` (ex : `find /usr -name libgllloq.so*`). Si vous ne trouvez toujours pas la bibliothèque, alors il vous reste plus qu'à l'installer.

- b. Une fois la bibliothèque localisée, vérifiez que cette bibliothèque est bien accessible pour la commande `ld` : le fichier `/etc/ld.so.conf` spécifie où trouver ces bibliothèques. Rajoutez le répertoire incriminé à la fin (il est possible que vous ayez à réinitialiser la machine pour que cela soit pris en compte). Il est aussi possible de rajouter ce répertoire en modifiant le contenu de la variable d'environnement `LD_LIBRARY_PATH`. Par exemple, en imaginant que le répertoire à ajouter est `/usr/X11R6/lib`, tapez :

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/X11R6/lib
```

(si votre *shell* est `bash`).

- c. Si cela ne fonctionne toujours pas, vérifiez que la bibliothèque incriminée est bien au format ELF (avec la commande `file`). Si c'est un lien symbolique, vérifiez que ce lien est correct et ne pointe pas vers un fichier inexistant (par exemple, en tapant `nm libgllloq.so`). Les permissions peuvent de plus être incorrectes (si vous utilisez un compte autre que `root` et que la bibliothèque est protégée en lecture par exemple).

6. `gllloq.c(.text+0x34): undefined reference to 'gllloq_init' ( gllloq.c(.text+0x34) référence inconnue au symbole « gllloq_init »)`

Aïe ! Aïe ! Aïe ! Pourquoi cette noirceur ? Il s'agit d'un symbole non résolu lors de la dernière phase de la compilation. Généralement, il s'agit d'un problème de bibliothèque. A priori, plusieurs causes possibles :

- la première chose à faire est de savoir si le symbole est **censé** être présent dans une bibliothèque. Par exemple, s'il s'agit d'un symbole commençant par `gtk`, il appartient très certainement à la bibliothèque `gtk`. Si le nom de la bibliothèque est difficilement identifiable (comme par exemple `zorglub_gloubiboulga`), il est possible de lister les symboles d'une bibliothèque avec la commande `nm`. Par exemple,

```
$ nm libgllloq.so
000000000109df0 d gllloq_message_func
00000000010a984 b gllloq_msg
000000000008a58 t gllloq_nearest_pow
000000000109dd8 d gllloq_free_list
000000000109cf8 d gllloq_mem_chunk
```

5. Fichiers exécutés par le système quand un processus tente d'accéder à une partie de la mémoire qui lui est interdite, et qui servent à analyser la raison de ce comportement pour corriger le programme fautif. Littéralement, *trognon* !

Rajouter l'option `-o` à `nm` permet de plus d'afficher le nom de la bibliothèque sur chaque ligne, ce qui simplifie les recherches. Imaginons que nous cherchions le symbole `bulgroz_max`, une solution de barbare est d'effectuer une recherche de ce genre :

```
$ nm /usr/lib/lib*.so | grep bulgroz_max
$ nm /usr/X11R6/lib/lib*.so | grep bulgroz_max
$ nm /usr/local/lib/lib*.so | grep bulgroz_max
/usr/local/lib/libzorglub.so:000000000004d848 T bulgroz_max
```

Formidable ! Le symbole `bulgroz_max` est défini dans la bibliothèque `zorglub` (la lettre majuscule `T` se trouve devant son nom). Il suffit de rajouter la chaîne `-lzorglub` dans la ligne de compilation en éditant le fichier `Makefile` : rajoutez-la à la fin de la ligne où `LD_FLAGS` ou `LFGLAGS` (ou au pire `CC`) sont définis, ou alors sur la ligne responsable de la création du fichier binaire final.

- la compilation se fait avec une version de la bibliothèque qui n'est pas la même que celle prévue pour le logiciel. Lisez le fichier `README` ou `INSTALL` de la distribution pour savoir quelle version de la bibliothèque doit être utilisée.
- tous les fichiers objets de la distribution ne sont pas correctement liés. Il manque le fichier dans lequel cette fonction est définie. Tapez `nm -o *.o` pour connaître son nom et ajoutez le fichier `.o` correspondant sur la ligne de compilation s'il est manquant.
- la fonction ou variable incriminée est peut-être fantaisiste. Essayez de la supprimer : éditez le fichier source incriminé (son nom est spécifié au début du message d'erreur). C'est une solution désespérée, qui va avoir pour conséquence un fonctionnement très probablement anarchique du programme (*erreur de segmentation* au démarrage, etc.)

#### 7. Segmentation fault (core dumped) (erreur de segmentation, fichier core produit)

Parfois, le compilateur échoue lamentablement et produit ce message d'erreur. Nous n'avons pas d'autre conseil que celui-ci : installez une version plus récente de votre compilateur !

#### 8. plus de place sur /tmp

La compilation a besoin d'espace temporaire de travail lors de ses différentes étapes; si elle ne l'a pas, elle échoue. Il faut donc faire du ménage. Mais attention ! la suppression de certains fichiers risque de faire échouer des programmes en cours d'exécution (serveur `X`, tubes...). Il faut maîtriser parfaitement ce que l'on fait ! Si `/tmp` fait partie d'une partition qui ne contient pas que lui (par exemple, la racine), recherchez et supprimez d'éventuels fichiers `core`.

#### 9. make/configure en boucle

Il s'agit généralement d'un problème d'heure sur votre système. `make` a en effet besoin de connaître la date et l'heure ainsi que celles des fichiers qu'il vérifie. Il compare les dates des fichiers et utilise le résultat pour savoir si la cible est plus récente que la dépendance.

Il se peut que des problèmes de date amènent `make` à se reconstruire sans fin (ou de construire et reconstruire un arborescence en boucle). Dans ce cas-là, l'utilisation de la commande `touch` (qui a pour conséquence de mettre à l'heure courante les fichiers passés en argument) permet de résoudre le problème dans la plupart des cas.

Par exemple :

```
$ touch *
```

Ou encore plus barbare (mais efficace) :

```
$ find . | xargs touch
```

## 12.5. Installation

### 12.5.1. Avec `make`

Maintenant que tout est compilé, il vous reste à copier les fichiers produits dans un endroit adéquat (généralement, dans un des sous-répertoires de `/usr/local`).

`make` permet généralement d'assurer ce travail. Une cible spéciale est la cible `install`. Très logiquement, utiliser `make install` permet d'installer les fichiers nécessaires.

Généralement, la procédure est décrite dans les fichiers `INSTALL` ou `README`. Mais parfois, l'auteur a oublié d'en prévoir une. Dans ce cas, il va falloir tout installer à la main.

Copiez alors :

- les exécutables (programmes) dans le répertoire portant le nom `/usr/local/bin`
- les bibliothèques (fichiers `lib*.so`) dans le répertoire portant le nom `/usr/local/lib`
- les fichiers d'en-têtes (fichiers `*.h`) dans le répertoire portant le nom `/usr/local/include` (attention à ne pas écraser les fichiers originaux);
- les fichiers de données vont généralement dans le répertoire `/usr/local/share`. Si vous ne connaissez pas la procédure d'installation, vous pouvez essayer de démarrer le programme sans copier les fichiers de données, et les mettre au bon endroit lorsqu'il vous les demande (dans un message d'erreur du type `Cannot open /usr/local/share/glloq/data.db`);
- la documentation est un peu à part :
  - les fichiers `man` se placent dans un des sous-répertoires de `/usr/local/man`. Généralement, ces fichiers sont au format `troff` (ou `groff`), et ont pour extension un chiffre. Leur nom est celui d'une commande (par exemple, `echo.1`). Si le chiffre est `n`, copiez ce fichier dans le sous-répertoire `/usr/local/man/man<n>`.
  - les fichiers `info` se placent dans le répertoire `/usr/info` ou `/usr/local/info`

Et voilà, c'est fini ! Félicitations ! Vous êtes maintenant fin prêt(e) à recompiler votre système d'exploitation tout entier.

### 12.5.2. Problèmes

Si vous venez d'installer un programme libre, par exemple *GNU tar* et si, lors de son exécution, ce n'est pas lui qui est appelé; ou bien si le fonctionnement de ce programme est différent de celui qu'il avait lorsque vous le testiez directement à partir du répertoire `src` : il s'agit là d'un problème de `PATH` qui trouve le programme dans un répertoire situé avant celui où vous avez installé le nouveau logiciel. Vérifiez ceci en exécutant `type -a <programme>`.

La solution est de mettre le répertoire d'installation plus haut dans le `PATH`, et/ou de supprimer/renommer les fichiers qui s'exécutent sans qu'on le désire, et/ou de renommer votre nouveau programme (en `gtar` dans cet exemple), de sorte qu'il n'y ait plus de confusion.

Vous pouvez aussi mettre en place un alias, si le *shell* le permet (par exemple dire que `tar`, est `/usr/local/bin/gtar`).

## 12.6. Assistance

### 12.6.1. Documentation

Il existe plusieurs sources de documentation :

- les *HOWTO*, petites documentations sur un sujet précis (généralement, pas trop proche de ce qui nous intéresse ici, mais parfois utiles). Voir sur votre disque dans le répertoire `/usr/doc/HOWTO` (pas toujours, parfois ils sont mis ailleurs, c'est la commande `locate HOWTO` qui vous donnera la réponse en cas de doute), ou alors, pour une version française des *HOWTO*, sur *freenix* (<http://www.freenix.fr/>).

- les pages de manuel. Tapez `man <commande>` pour obtenir de la documentation sur la commande `<commande>`,
- la littérature spécialisée. Plusieurs grands éditeurs commencent à publier des livres sur les systèmes libres (plus spécialement sur *GNU/Linux*). C'est souvent utile si vous débutez et si vous ne comprenez pas tous les termes de la présente documentation.

### 12.6.2. Assistance technique

Si vous avez acheté une distribution « officielle » de **Mandrake Linux**, vous pouvez demander à l'assistance technique des informations sur votre système. Celle-ci ne peut sans doute pas assister tous les utilisateurs dans l'installation de logiciels supplémentaires, mais une aide est disponible pour l'installation de x jours. Peut-être les techniciens pourront-ils passer quelque temps sur des problèmes de compilation.

Sinon vous pouvez aussi compter sur l'aide de la communauté du logiciel libre :

- les *newsgroups* (forums de discussions sur l'*Usenet*) dont le nom vérifie `fr.comp.os.linux.*` (`news:fr.comp.os.linux.*`) permettent de répondre à toutes les questions concernant *GNU/Linux*. Le forum intitulé `fr.comp.os.bsd` (`news:fr.comp.os.bsd`) a pour sujet les systèmes BSD. Il doit exister d'autres forums consacrés à d'autres *UNIX*. N'oubliez pas de lire un peu ce qui s'y dit avant d'y aller tout de suite de votre question !
- plusieurs associations ou groupes d'enthousiastes du logiciel libre proposent une assistance bénévole. Pour trouver ceux dont les membres habitent près de chez vous, la meilleure chose à faire est de consulter les sites Web spécialisés ou de lire les forums de discussions concernés.
- plusieurs *channels* IRC permettent une assistance en temps réel (mais en aveugle) par des *gourous*. Voir par exemple le *channel* `#linux` (`news:#linux`) sur la plupart des réseaux d'IRC, ou `#linuxhelp` (`news:#linuxhelp`) sur *IRCNET*.
- en dernier recours, demander à l'auteur du logiciel (s'il a mentionné son nom et son *adresse électronique* dans un fichier de la distribution) si vous êtes sûr d'avoir identifié un bogue, qui peut en fait n'être dû qu'à votre architecture (mais après tout, un logiciel libre est censé être transportable !).

### 12.6.3. Comment trouver des logiciels libres

Pour trouver des logiciels libres, quelques trucs peuvent être utiles à savoir :

- l'énorme site FTP `sunsite.unc.edu` (`sunsite.unc.edu`) ou l'un de ses clones;
- les sites Web suivants effectuent un catalogue de nombreux logiciels libres utilisables sur plates-formes *UNIX* (mais on y trouve aussi des logiciels propriétaires) :
  - `freshmeat` (`http://www.freshmeat.net/`) est sans doute le site le plus complet;
  - `linux-france` (`http://www.linux-france.org`) contient de nombreuses astuces pour les logiciels fonctionnant sous *GNU/Linux*. La plupart fonctionnent bien sûr sur les autres plates-formes *UNIX* libres;
  - `gnu.org` (`http://www.gnu.org/software/`) pour une liste exhaustive de tous les logiciels GNU. Bien sûr, tous sont libres et la plupart ont pour licence la GPL.
- vous pouvez de plus effectuer une recherche sur des moteurs comme `altavista` (`http://www.altavista.com/`) et faire une requête de type :

```
+<logiciel>
+download

ou

"download
<logiciel>"
```

## 12.7. Remerciements

- Relectures et commentaires désobligeants (et par ordre alphabétique) : Sébastien Blondeel, Mathieu Bois, Xavier Renault et Kamel Sehil.
- *Bêta-testing* : Laurent Bassaler





## Chapitre 13. Compilation et mise en place de nouveaux noyaux

Avec la notion de montage de systèmes de fichiers et la compilation des sources, le sujet que nous abordons ici est sans doute celui qui énerve le plus les débutants. La compilation d'un nouveau noyau n'est en général pas nécessaire, puisque les noyaux installés par **Mandrake Linux** contiennent le support pour un nombre conséquent de périphériques, ainsi qu'un bon nombre de correctifs logiciel (*patches*) mais...

Il peut arriver, pourquoi pas, que vous ayez envie de le faire, rien que pour voir « ce que ça fait » ! En fait, à part faire chauffer votre PC et votre cafetière un peu plus que d'habitude, ça ne change pas grand-chose ! Les raisons pour lesquelles vous pourriez désirer recompiler votre nouveau noyau sont de plusieurs ordres, allant de la simple désactivation d'une option à la compilation du tout dernier noyau expérimental. Ainsi donc, l'objectif de ce chapitre sera de faire en sorte que... votre cafetière fonctionne encore après la compilation :-)

De toute façon, plusieurs raisons valables justifient que vous vouliez vous lancer dans l'aventure : vous avez lu que le noyau que vous utilisez a un *bogue* au niveau de la sécurité, bogue corrigé dans une version plus récente ; un nouveau noyau prend en charge un périphérique dont vous avez grand besoin. Vous avez en effet le choix d'attendre des mises à jour ou bien de compiler vous-même un nouveau noyau. Optez donc pour la deuxième solution !

De toute façon, achetez-vous des filtres et du café !

### 13.1. Où trouver les sources du noyau

Les sources peuvent s'obtenir principalement de deux façons :

1. **Noyaux officiels Mandrake Linux.** Vous trouverez dans le répertoire SRPMS de tous les miroirs (<http://www.MandrakeLinux.com/en/cookerdevel.php3>) de Cooker les paquets suivants :

`kernel-2.4.??-?mdk-?-?mdk.src.rpm`

Les sources du noyau pour compiler celui utilisé dans la distribution. Il est amplement remanié pour offrir des fonctionnalités additionnelles.

`kernel-linus2.4-2.4.??-?mdk.src.rpm`

Le noyau brut tel que publié par le responsable du noyau *GNU/Linux*.

Si vous choisissez cette option (recommandée), il suffit de charger le paquetage source, l'installer (en tant que root) et sauter à *Configuration du noyau*, page 86.

2. **L'entrepôt officiel pour les noyaux Linux.** <ftp.kernel.org> (<ftp.kernel.org>) s'avère le site principal d'hébergement des sources du noyau, mais un nombre important de miroirs existe et tous ont pour nom <ftp.xx.kernel.org> (<ftp.xx.kernel.org>), où xx (xx) représente le code ISO du pays. Pour la France, ce code est fr (fr), et par conséquent, le miroir préférentiel sera <ftp.fr.kernel.org>. À partir de l'annonce officielle de la sortie du noyau, vous pouvez compter deux bonnes heures avant que tous les miroirs soient alimentés.

Sur tous ces serveurs FTP, les sources sont situées dans le répertoire `/pub/linux/kernel`. Allez ensuite dans le répertoire dont la série vous intéresse : ce sera certainement `v2.4`. Rien ne vous empêche d'essayer des noyaux 2.5, mais gardez à l'esprit que ce sont des noyaux expérimentaux. Le fichier contenant les sources du noyau est `linux-<version.du.noyau>.tar.bz2`, par exemple `linux-2.4.20.tar.bz2`.

Il existe également des *patches* à appliquer aux sources du noyau pour le mettre à jour de façon incrémentale : ainsi, si vous avez déjà les sources du noyau 2.4.20 et que vous voulez actualiser vers le noyau 2.4.22, vous pouvez vous dispenser de télécharger les sources en entier, et télécharger simplement les *patches* `patch-2.4.21.bz2` et `patch-2.4.22.bz2`. En règle générale, c'est une bonne idée puisque les sources occupent aujourd'hui plus de 26 Mo.

## 13.2. Décompactage des sources du noyau, correction éventuelle du noyau

Les sources du noyau sont à placer dans `/usr/src`. Il vous faut donc vous placer dans ce répertoire puis y décompresser les sources :

```
$ cd /usr/src
$ mv linux linux.old
$ tar xjf /path/to/linux-2.4.20.tar.bz2
```

La commande `mv linux linux.old` est nécessaire : en effet, vous disposez peut-être déjà des sources d'une autre version du noyau. Cette commande fait en sorte que vous ne les écrasiez pas. Dès l'archive décompressée, vous disposez d'un répertoire `linux-<version>` avec les sources du nouveau noyau. Vous pouvez créer un lien (`ln -s linux-<version> linux`) pour raisons de commodité.

Maintenant, les *patches*. Supposons que vous voulez effectivement « *patcher* » (corriger) du 2.4.20 vers le 2.4.22 et avez téléchargé les *patches* nécessaires pour ce faire : rendez-vous dans le répertoire `linux` nouvellement créé, puis appliquez les *patches* :

```
$ cd linux
$ bzcat -dc /path/to/patch-2.4.21.bz2 | patch -p1
$ bzcat -dc /path/to/patch-2.4.22.bz2 | patch -p1
$ cd ..
```

De façon générale, passer d'une version 2.4.x à 2.4.y requiert que vous appliquiez tous les *patches* 2.4.x+1, 2.4.x+2, ..., 2.4.y dans l'ordre. Pour retourner à la version précédente de 2.4.y à 2.4.x, répétez exactement la même procédure mais en appliquant les *patches* dans l'ordre inverse et avec l'option `-R` de `patch` (*Reverse*, soit inverse). Ainsi, pour repasser du noyau 2.4.22 au noyau 2.4.20, vous feriez :

```
$ bzcat /path/to/patch-2.4.22.bz2 | patch -p1 -R
$ bzcat /path/to/patch-2.4.21.bz2 | patch -p1 -R
```



Si vous souhaitez tester si un correctif s'appliquera correctement avant de l'appliquer vraiment, ajoutez l'option `--dry-try` à la commande `patch`.

Ensuite, dans un souci de clarté (et aussi pour s'y retrouver), vous pouvez renommer `linux` pour refléter la version du noyau et créer un lien symbolique :

```
$ mv linux linux-2.4.22
$ ln -s linux-2.4.22 linux
```

Il est maintenant temps de passer à la configuration. Pour ce faire, placez-vous dans le répertoire des sources :

```
$ cd linux
```

## 13.3. Configuration du noyau

En premier lieu, allez dans `/usr/src/linux`.

Petit tuyau avant de commencer. Si vous le désirez, vous pouvez choisir la version de votre noyau. La version du noyau est déterminée par les 4 premières lignes de votre `Makefile` :

```
$ head -4 Makefile
VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 22
EXTRAVERSION =
```

De plus dans le `Makefile`, est aussi présente la version du noyau :

```
KERNELRELEASE=$(VERSION) . $(PATCHLEVEL) . $(SUBLEVEL) $(EXTRAVERSION)
```

Vous n'avez qu'à modifier un de ces champs afin de changer votre version. Toutefois, il est préférable de ne changer que EXTRAVERSION. Supposons que vous le régliez à -toto, par exemple. Votre nouvelle version du noyau deviendra 2.4.22-toto. N'hésitez pas à changer ce champ à chaque fois que vous recompilez un noyau avec la même version. Vous pourrez ainsi tester différentes options tout en gardant les anciens essais.

Maintenant, pour configurer le noyau, vous avez le choix entre :

- `make xconfig` pour une interface graphique ;
- `make menuconfig` pour une interface basée sur ncurses ;
- `make config` pour l'interface la plus rudimentaire, ligne par ligne, section par section ;
- `make oldconfig` similaire à `config`, mais en se basant sur votre ancienne configuration. Voir *Sauvegarder et réutiliser vos fichiers de configuration du noyau*, page 87.

Malheureusement la configuration du noyau n'est pas encore internationalisée, tout est en anglais. Nous allons parcourir la configuration section par section, mais vous pouvez sauter des sections pour passer à celle qui vous intéresse si vous utilisez `menuconfig` ou `xconfig`. Le choix pour les options est **y** pour Yes (fonctionnalité compilée en dur dans le noyau), **m** pour Module (fonctionnalité compilée en module), ou **n** pour No (ne pas inclure dans le noyau).

`make xconfig` et `make menuconfig` présentent tous deux les options groupées par thèmes. Par exemple, `Processor family` est placé dans `Processor type and features`.

Pour `xconfig`, le bouton `Main Menu` sert à revenir au menu principal lorsque vous êtes à l'intérieur d'un groupe, `Next` sert à passer au groupe d'options suivant et `Prev`, au précédent. Pour `menuconfig`, servez-vous de la touche `Entrée` pour choisir une section ; déterminez les options avec **y**, **m** ou **n** pour en changer l'état ou bien appuyez sur la touche `Entrée` et sélectionnez les diverses options à choix multiples. `Exit` sortira d'une section et de la configuration si vous êtes dans le menu principal. Et évidemment, il y a `Help` (Aide).

Nous n'énumérerons pas toutes les options puisqu'il y en a quelques centaines. De plus, si vous vous êtes rendu au présent chapitre, vous savez probablement ce que vous faites. Donc, nous vous laisserons naviguer dans la configuration du noyau et vous laisserons activer ou désactiver à votre convenance les options. Toutefois, voici quelques conseils afin d'éviter que votre noyau devienne inutilisable :

1. À moins que vous n'utilisiez un ramdisk, **ne jamais** compiler les gestionnaires nécessaires pour monter votre système de fichiers racine (pilotes matériel et de systèmes de fichiers) en tant que modules ! De plus, si vous utilisez un ramdisk, répondez **Y** à la question concernant le support ext2FS en tant que système de fichiers utilisé pour le ramdisk. Vous aurez aussi besoin du support pour `initrd`.
2. Si vous possédez des cartes réseau, compilez leurs pilotes en tant que modules. De cette façon, vous pourrez choisir laquelle sera la première, puis la seconde et ainsi de suite, en affectant les alias appropriés dans `/etc/modules.conf`. Si vous compilez les pilotes dans le noyau, l'ordre dans lequel ils seront chargés dépendra de l'ordre dans lequel ils ont été connectés ce dernier ordre n'est peut-être pas celui que vous souhaitez.
3. Finalement : si vous ne connaissez pas les tenants et aboutissants d'une option, lisez l'aide ! Et si l'aide ne vous inspire toujours pas, laissez les options telles quelles. (Pour `config` et `oldconfig`, pressez la touche `?` pour lire l'aide).

Vous pouvez aussi consulter le fichier `/usr/src/linux/Documentation/Configure.help` qui reprend la totalité des textes d'aide par ordre d'apparition. Vous trouverez aussi au début une liste des traductions et leurs adresses sur le Web. Il y a, en particulier, une version française (<http://www.traduc.org/kernel/fr/>).

Et voilà ! La configuration est enfin terminée. Sauvegardez votre configuration et quittez.

### 13.4. Sauvegarder et réutiliser vos fichiers de configuration du noyau

Le fichier de configuration du noyau se situe dans le fichier `/usr/src/linux/.config`. Il en existe une sauvegarde dans `/boot/config-<version>`, il est bon de garder cette référence. Cependant, sauvegardez aussi les différentes configurations de vos noyaux, dans la mesure où il ne s'agit que de donner différents noms aux fichiers de configuration.

Une des possibilités est de nommer le fichier de configuration selon la version du noyau. Supposons que vous avez modifié la version du noyau comme dans *Configuration du noyau*, page 86, alors vous pouvez faire :

```
$ cp .config /root/config-2.4.22-toto
```

Si vous décidez de mettre à niveau votre noyau à la version 2.4.24 (par exemple), vous pourrez réutiliser ce fichier, puisque les différences entre ces deux versions du noyau seront fort minimes. Vous n'avez qu'à utiliser la copie de sauvegarde :

```
$ cp /root/config-2.4.22-toto .config
```

Mais le fait de copier le fichier de nouveau ne signifie pas que le noyau est prêt à être compilé pour autant. Vous devez invoquer la commande `make menuconfig` (ou quoi que ce soit que vous décidez d'utiliser) de nouveau, parce que certains fichiers nécessaires à la réussite de la compilation sont créés et modifiés par ces commandes.

Toutefois, indépendamment de la tâche qui consiste à passer à travers tous les menus de nouveau, vous risquez de manquer certaines nouvelles options intéressantes. Vous pouvez éviter cela en utilisant la commande `make oldconfig`, laquelle possède deux avantages notables :

1. C'est rapide.
2. Si une nouvelle option apparaît dans le noyau, qui n'était pas présente dans vos fichiers de configuration, le système vous demandera de choisir entre les deux.



Après avoir copié le fichier `.config` sur le répertoire de `root`, comme proposé ci-dessus, lancez la commande `make mrproper`. Ainsi, rien ne restera de l'ancienne configuration et vous aurez ainsi un noyau tout neuf.

Maintenant, place à la compilation.

## 13.5. Compilation et installation des modules

Tout d'abord une petite chose : si vous compilez un noyau dont la version est identique à une version déjà présente sur votre système, les modules de cette dernière doivent être effacés auparavant. Par exemple, si vous recompilez un noyau 2.4.22, il vous faudra effacer le répertoire `/lib/modules/2.4.22`.

La compilation du noyau et des modules, ainsi que l'installation des modules, se font en une seule ligne :

```
make dep
make clean bzImage modules
make modules_install install
```

Un peu de vocabulaire : `dep`, `bzImage`, etc., aussi bien que `oldconfig`, mais aussi ceux utilisés ci-dessus, sont appelés **targets** (soit cibles). Si vous spécifiez un certain nombre de cibles pour `make` tel que ci-dessus, elles seront exécutées selon l'ordre d'apparition. Mais si une cible échoue, `make` n'ira pas plus loin <sup>1</sup>.

Penchons-nous maintenant sur les différentes cibles et regardons ce qu'elles font :

- `dep` : ce sont des dépendances entre les différents fichiers sources. Il est nécessaire de faire ainsi chaque fois que vous changez votre configuration. Autrement, certains fichiers risquent de ne pas être créés et la compilation échouera ;
- `bzImage` : ceci construit le noyau. Notez que cette cible est seulement valable pour des processeurs *Intel*. Cette cible générera aussi le fichier `System.map` pour ce noyau. Nous verrons plus loin à quoi sert ce fichier ;
- `modules` : comme le dit son nom, cette cible générera des modules pour le noyau. Si vous ne choisissez pas ces modules, la cible ne fera rien ;
- `modules_install` : ceci installe les modules. Par défaut, les modules seront installés dans le répertoire `/lib/modules/<kernel-version>`. Cette cible calcule aussi les dépendances entre modules (ce qui n'est probablement pas le cas en 2.2.x) ;
- `install` : cette dernière cible va finalement copier le noyau et tous ses modules au bon endroit, et modifier la configuration des chargeurs de démarrage de telle sorte que le nouveau noyau soit disponible au déma-

1. Dans ce cas, si cela échoue, cela signifie qu'il y a un bogue dans le noyau... Dans ce cas, merci de nous le faire savoir !

rage. Ne l'utilisez pas si vous préférez effectuer une installation manuelle, telle que décrite dans *Installation du nouveau noyau*, page 89.

Tout est maintenant compilé et les modules sont installés. Mais ce n'est pas tout : vous devez également installer le noyau dans un endroit où votre programme de démarrage (*bootloader*), que ce soit *LILO* ou *grub*, pourra le trouver. C'est ce dont il est question dans la prochaine section.



L'ancienne cible *zImage* est désormais obsolète, et vous ne devriez plus l'utiliser.

## 13.6. Installation du nouveau noyau

Le noyau se situe dans `arch/i386/boot/bzImage` (ou *zImage* si vous avez choisi `make zImage` en place et lieu). Le répertoire standard dans lequel les noyaux sont installés est `/boot`. Vous devez aussi copier le fichier `System.map` afin d'assurer le bon fonctionnement de certains programmes (tels que `top` par exemple). Assurez-vous de bien nommer ces fichiers du nom de la version du noyau. Supposons que votre version du noyau est `2.4.22-toto`. L'ordre des commandes que vous devrez taper est :

```
$ cp arch/i386/boot/bzImage /boot/vmlinux-2.4.22-toto
$ cp System.map /boot/System.map-2.4.22-toto
```

Maintenant, il est nécessaire de dire au chargeur de démarrage où le nouveau noyau se trouve. Vous avez deux possibilités : *grub* ou *LILO*. Notez que **Mandrake Linux** est installé avec *LILO* par défaut.

### 13.6.1. Mise à jour de LILO

La manière la plus simple de mettre à jour *LILO* est d'utiliser *drakboot* (voir le chapitre *Changer vos paramètres de démarrage* du *Guide de démarrage*). Ou encore, vous pouvez modifier manuellement le fichier de configuration comme expliqué ci-dessous.

Pour *LILO*, il vous faudra mettre à jour `/etc/lilo.conf`. Évidemment, gardez la possibilité de démarrer votre noyau actuel ! Voici à quoi ressemble un fichier `lilo.conf` typique, après l'installation de votre distribution **Mandrake Linux** et avant modification :

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
vga=normal
default=linux
keytable=/boot/fr-latin1.klt
lba32
prompt
timeout=50
message=/boot/message
image=/boot/vmlinuz-2.4.20-17mdk
    label=linux
    root=/dev/hda1
    read-only
    append="devfs=mount"
    vga=788
other=/dev/hda2
    label=dos
    table=/dev/hda
```

Un fichier `lilo.conf` est composé d'une section principale, puis d'une section pour le lancement de chaque système d'exploitation. Dans l'exemple du fichier ci-dessus, la section principale est composée des directives suivantes :

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
vga=normal
default=linux
```

```
keytable=/boot/fr-latin1.klt
lba32
prompt
timeout=50
message=/boot/message
```

La directive `boot=` dit à *LILO* où il doit installer son secteur d'amorçage *boot* ; en l'occurrence, il s'agit du MBR (*Master Boot Record*, soit l'enregistrement d'amorçage maître) du premier disque dur IDE. Si vous voulez faire une disquette *LILO*, il vous suffira de remplacer `/dev/hda` par `/dev/fd0`. La directive `prompt` demande à *LILO* de présenter l'invite au démarrage et d'entamer la procédure après cinq secondes (`timeout=50`). Si vous retirez la directive `timeout=`, *LILO* attendra jusqu'à ce que vous ayez tapé quelque chose.

Puis vient une section *linux* :

```
image=/boot/vmlinuz-2.4.20-17mdk
label=linux
root=/dev/hda1
read-only
append="devfs=mount"
vga=788
```

Une section *linux* commence par la directive `image=`, puis par le chemin complet vers un noyau *GNU/Linux* valide. À l'instar de toute section, elle contient une directive `label=` pour l'identifier de façon unique. La directive `root=` dit à *LILO* quelle partition héberge le système de fichiers racine pour ce système *GNU/Linux*, elle pourra être différente chez vous. La directive `read-only` commande à *LILO* de monter ce système de fichiers racine en lecture seule au démarrage : s'il n'y a pas cette directive, vous aurez un message d'avertissement. la ligne `append` comporte des options à passer au noyau. Finalement, le paramètre `vga` configure le *frame buffer*.

Puis vient la section *windows* :

```
other=/dev/hda2
    label=dos
    table=/dev/hda
```

En fait, une section débutant par `other=` est utilisée par *LILO* pour démarrer tout système d'exploitation autre que *GNU/Linux* : l'argument de cette directive est l'emplacement du secteur d'amorçage de ce système, et en l'occurrence, il s'agit d'un système *Windows*. Pour trouver le secteur d'amorçage situé au début de la partition hébergeant cet autre système, *GNU/Linux* a également besoin de savoir où se trouve la table des partitions, qui lui permettra de localiser la partition en question, ce que fait la directive `table=`. La directive `label=`, comme pour une section *linux*, sert à identifier le système.

Maintenant, rajouter la section *GNU/Linux* vous permettra de démarrer sur le nouveau noyau. Dans cet exemple, elle sera mise en tête, mais rien ne vous empêche de la mettre à un autre endroit, sauf au milieu d'une autre section :

```
image=/boot/vmlinuz-2.4.22-toto
    label=toto
    root=/dev/hda1
    read-only
```

Adaptez évidemment cela à votre configuration ! C'est à dessein que nous avons pris une situation différente de celle de *grub* ci-dessus...

Si vous avez compilé votre noyau avec le *frame buffer*, consultez le paragraphe à ce sujet ci-dessus dans la section au sujet de *grub*. La différence est que désormais, l'option est sur une ligne à part :

```
vga=0x315
```

Voici donc à quoi ressemble notre fichier `lilo.conf` après modification, agrémenté de quelques commentaires en plus (toutes les lignes commençant par #), qui seront ignorés par *LILO* :

```
#
# Section principale
```

```

#
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
# Au démarrage, nous voulons fonctionner en mode VGA normal.
# Le frame buffer basculera dans cette résolution
# si nous utilisons :
vga=normal
# Notre message de démarrage...
message=/boot/message
# Qu'est-ce qui doit être démarré par défaut.
# Mettons notre propre noyau par défaut :
default=toto
# Afficher le prompt...
prompt
# ... attendre 5 secondes
timeout=50
#
# Notre nouveau noyau : image par défaut
#
image=/boot/vmlinuz-2.4.22-toto
    label=toto
    root=/dev/hda1
    read-only
# Si nous utilisons le frame buffer VESA :
    vga=0x315
#
# le noyau original
#
image=/boot/vmlinuz-2.4.20-17mdk
    label=linux
    root=/dev/hda1
    read-only
#
# La section Windows
#
other=/dev/hda2
    label=dos
    table=/dev/hda

```

Souvenez-vous d'adapter le fichier à votre configuration ! Dans cet exemple, le système de fichiers racine de *GNU/Linux* est `/dev/hda1`, mais il peut très bien se situer ailleurs chez vous (pareillement pour *windows*).

Maintenant que le fichier est modifié correctement, il faut dire à *LILO* de changer le secteur de démarrage :

```

$ lilo
Added toto *
Added linux
Added dos
$

```

Vous pouvez ainsi compiler autant de noyaux que vous le souhaitez, en rajoutant autant de sections *GNU/Linux* que nécessaire. Il ne vous reste plus qu'à redémarrer pour tester votre nouveau noyau.

### 13.6.2. Mise à jour de grub

Évidemment, il faut conserver la possibilité de charger votre ancien noyau ! La manière la plus simple de mettre à jour *grub* est d'utiliser *drakboot* (voir le chapitre Changer vos paramètres de démarrage du *Guide de démarrage*). Ou encore, vous pouvez modifier manuellement le fichier de configuration comme expliqué ci-dessous.

Il faut ici modifier le fichier `/boot/grub/menu.lst`. Voilà ce à quoi ressemble un `menu.lst` typique, avant toute modification :

```
timeout 5
color black/cyan yellow/cyan
i18n (hd0,4)/boot/grub/messages
keytable (hd0,4)/boot/fr-latin1.klt
default 0

title linux
kernel (hd0,4)/boot/vmlinuz root=/dev/hda5

title failsafe
kernel (hd0,4)/boot/vmlinuz root=/dev/hda5 failsafe

title Windows
root (hd0,0)
makeactive
chainloader +1

title floppy
root (fd0)
chainloader +1
```

Ce fichier est constitué de deux parties : l'en-tête avec les options communes de base (les cinq premières lignes), et les sections (ou entrées), chacune correspondant à un noyau différent ou à un autre système d'exploitation. `timeout 5` définit le temps dont vous disposez pour choisir un autre noyau ou système d'exploitation que celui défini par défaut. Cette valeur par défaut est spécifiée par `default 0` (ce qui signifie que la première section définie est donc celle dite « par défaut »). La ligne `color` définit les couleurs du menu ; la ligne `i18n` indique l'emplacement du message de bienvenue : `(hd0,4)` signifie qu'il se trouve sur la cinquième partition du premier disque dur. L'option `keytable (hd0,4)/boot/fr-latin1.klt` indique à *grub* le type de clavier utilisé si vous devez passer des options manuellement au démarrage.

Vient ensuite la section des entrées. Il en existe quatre ici : `linux`, `failsafe`, `windows` et `floppy`.

- La section `linux` commence par indiquer à *grub* le noyau à charger (`kernel (hd0,4)/boot/vmlinuz`), puis les options à passer à ce noyau. Dans notre cas, `root=/dev/hda5` indiquera au noyau que le système de fichier racine est situé sur `/dev/hda5`. En fait, `/dev/hda5` est l'équivalent de `hd(0,4)` pour *grub*, mais rien n'empêche le noyau de se situer sur une partition différente de celle du système de fichiers racine.
- L'entrée `failsafe` ressemble beaucoup à la précédente, sauf que nous passons un argument supplémentaire au noyau (`failsafe`), qui lui demande de passer en mode « single » ou mode « de secours ».
- L'entrée `Windows` demande à *grub* de simplement charger le secteur de démarrage de la première partition, qui contient probablement le secteur de démarrage de *windows*.
- La dernière section, `floppy`, démarre simplement votre système depuis la disquette du premier lecteur, quel que soit le système d'exploitation qui y réside. Ce peut être *Windows*, voir même un noyau *GNU/Linux* sur disquette.



Suivant le niveau de sécurité utilisé sur votre système, il se peut que certaines des entrées décrites n'existent pas dans votre fichier.

Allons maintenant droit au but ! Il nous faut rajouter une nouvelle section *GNU/Linux* afin de pouvoir démarrer sur notre nouveau noyau. Dans cet exemple, elle sera placée au début des entrées, mais rien ne vous empêche de la mettre ailleurs :

```
title toto
```



```
kernel (hd0,4)/boot/vmlinux-2.4.22-toto root=/dev/hda5
```

Souvenez-vous d'adapter le fichier à votre configuration ! Le système de fichier root *GNU/Linux* est ici sur `/dev/hda5`, mais il est sûrement ailleurs sur votre système. Votre ordinateur est prêt à redémarrer. Profitez bien de votre noyau tout neuf !

Et c'est tout. Contrairement à *LILO*, que nous verrons plus loin, il n'y a rien d'autre à faire. Redémarrez votre ordinateur et la nouvelle entrée que vous venez de définir apparaîtra. Sélectionnez-là dans le menu et votre nouveau noyau démarrera.

Si vous avez compilé votre noyau avec le *frame buffer*, vous aurez envie de l'exploiter : dans ce cas, rajoutez une directive à la section qui lui indique la résolution avec laquelle vous voulez démarrer. La liste des modes est disponible dans le fichier `/usr/src/linux/Documentation/fb/vesafb.txt` (pour le *frame buffer* VESA ! Sinon, reportez-vous au fichier correspondant). Pour du 800x600 en 32 bits<sup>2</sup>, le numéro du mode est 0x315, il faut donc rajouter la directive :

```
vga=0x315
```

Votre entrée ressemblera désormais à ceci :

```
title toto
kernel (hd0,4)/boot/vmlinux-2.4.22-toto root=/dev/hda5 vga=0x315
```

Pour plus de renseignements, consultez les pages d'info de *grub* (`info grub`).

---

2. 8 bits signifie ( $2^8$ ) couleurs, soit 256 ; 16 bits signifie ( $2^{16}$ ) couleurs, soit 64k, soit 65536 ; en 24 bits comme en 32 bits, la couleur est codée sur 24 bits, soit ( $2^{24}$ ) couleurs possibles, soit exactement 16M, ou un peu plus de 16 millions.



## Annexe A. La licence Publique Générale GNU (GPL)

Le texte qui suit est celui de la licence GPL qui s'applique à la plupart des programmes quise trouvent dans la distribution **Mandrake Linux**.

### A.1. Introduction

This is an unofficial translation of the GNU General Public License into French. It was not published by the Free Software Foundation, and does not legally state the distribution terms for software that uses the GNU GPL--only the original English text of the GNU GPL does that. However, we hope that this translation will help French speakers understand the GNU GPL better.

Voici (<http://www.linux-france.org/article/these/gpl.html>) une adaptation non officielle de la Licence Publique Générale du projet GNU. Elle n'a pas été publiée par la Free Software Foundation et son contenu n'a aucune portée légale car seule la version anglaise de ce document détaille le mode de distribution des logiciels sous GNU GPL. Nous espérons cependant qu'elle permettra aux francophones de mieux comprendre la GPL.

### Licence Publique Générale GNU Version 2, Juin 1991

Copyright © Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 États-Unis, 1989, 1991.

La copie et la distribution de copies exactes de ce document sont autorisées, mais aucune modification n'est permise.

### A.2. Préambule

Les licences d'utilisation de la plupart des programmes sont définies pour limiter ou supprimer toute liberté à l'utilisateur. À l'inverse, la Licence Publique Générale (General Public License) est destinée à vous garantir la liberté de partager et de modifier les logiciels libres, et de s'assurer que ces logiciels sont effectivement accessibles à tout utilisateur.

Cette Licence Publique Générale s'applique à la plupart des programmes de la Free Software Foundation, comme à tout autre programme dont l'auteur l'aura décidé (d'autres logiciels de la FSF sont couverts pour leur part par la Licence Publique Générale pour Bibliothèques GNU (LGPL)). Vous pouvez aussi appliquer les termes de cette Licence à vos propres programmes, si vous le désirez.

Liberté des logiciels ne signifie pas nécessairement gratuité. Notre Licence est conçue pour vous assurer la liberté de distribuer des copies des programmes, gratuitement ou non, de recevoir le code source ou de pouvoir l'obtenir, de modifier les programmes ou d'en utiliser des éléments dans de nouveaux programmes libres, en sachant que vous y êtes autorisé.

Afin de garantir ces droits, nous avons dû introduire des restrictions interdisant à quiconque de vous les refuser ou de vous demander d'y renoncer. Ces restrictions vous imposent en retour certaines obligations si vous distribuez ou modifiez des copies de programmes protégés par la Licence. En d'autre termes, il vous incombera en ce cas de :

- transmettre aux destinataires tous les droits que vous possédez,
- expédier aux destinataires le code source ou bien tenir celui-ci à leur disposition,
- leur remettre cette Licence afin qu'ils prennent connaissance de leurs droits.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Nous protégeons vos droits de deux façons : d'abord par le copyright du logiciel, ensuite par la remise de cette Licence qui vous autorise légalement à copier, distribuer et/ou modifier le logiciel.

En outre, pour protéger chaque auteur ainsi que la FSF, nous affirmons solennellement que le programme concerné ne fait l'objet d'aucune garantie. Si un tiers le modifie puis le redistribue, tous ceux qui en recevront une copie doivent savoir qu'il ne s'agit pas de l'original afin qu'une copie défectueuse n'entache pas la réputation de l'auteur du logiciel.

Enfin, tout programme libre est sans cesse menacé par des dépôts de brevets. Nous souhaitons à tout prix éviter que des distributeurs puissent déposer des brevets sur les Logiciels Libres pour leur propre compte. Pour éviter cela, nous stipulons bien que tout dépôt éventuel de brevet doit accorder expressément à tout un chacun le libre usage du produit.

Les dispositions précises et les conditions de copie, de distribution et de modification de nos logiciels sont les suivantes :

### A.3. Stipulations et conditions relatives à la copie, la distribution et la modification

1. La présente Licence s'applique à tout Programme (ou autre travail) où figure une note, placée par le détenteur des droits, stipulant que ledit Programme ou travail peut être distribué selon les termes de la présente Licence. Le terme Programme désigne aussi bien le Programme lui-même que tout travail qui en est dérivé selon la loi, c'est-à-dire tout ouvrage reproduisant le Programme ou une partie de celui-ci, à l'identique ou bien modifié, et/ou traduit dans une autre langue (la traduction est considérée comme une modification). Chaque personne concernée par la Licence Publique Générale sera désignée par le terme Vous.

Les activités autres que copie, distribution et modification ne sont pas couvertes par la présente Licence et sortent de son cadre. Rien ne restreint l'utilisation du Programme et les données issues de celui-ci ne sont couvertes que si leur contenu constitue un travail basé sur le logiciel (indépendamment du fait d'avoir été réalisé en lançant le Programme). Tout dépend de ce que le Programme est censé produire.

2. Vous pouvez copier et distribuer des copies conformes du code source du Programme, tel que Vous l'avez reçu, sur n'importe quel support, à condition de placer sur chaque copie un copyright approprié et une restriction de garantie, de ne pas modifier ou omettre toutes les stipulations se référant à la présente Licence et à la limitation de garantie, et de fournir avec toute copie du Programme un exemplaire de la Licence.

Vous pouvez demander une rétribution financière pour la réalisation de la copie et demeurez libre de proposer une garantie assurée par vos soins, moyennant finances.

3. Vous pouvez modifier votre copie ou vos copies du Programme ou partie de celui-ci, ou d'un travail basé sur ce Programme, et copier et distribuer ces modifications selon les termes de l'article 1, à condition de Vous conformer également aux conditions suivantes :

- a. Ajouter aux fichiers modifiés l'indication très claire des modifications effectuées, ainsi que la date de chaque changement.
- b. Distribuer sous les termes de la Licence Publique Générale l'ensemble de toute réalisation contenant tout ou partie du Programme, avec ou sans modifications.
- c. Si le Programme modifié lit des commandes de manière interactive lors de son exécution, faire en sorte qu'il affiche, lors d'une invocation ordinaire, le copyright approprié en indiquant clairement la limitation de garantie (ou la garantie que Vous Vous engagez à fournir Vous-même), qu'il stipule que tout utilisateur peut librement redistribuer le Programme selon les conditions de la Licence Publique Générale GNU, et qu'il montre à tout utilisateur comment lire une copie de celle-ci (exception : si le Programme original est interactif mais n'affiche pas un tel message en temps normal, tout travail dérivé de ce Programme ne sera pas non plus contraint de l'afficher).

Toutes ces conditions s'appliquent à l'ensemble des modifications. Si des éléments identifiables de ce travail ne sont pas dérivés du Programme et peuvent être raisonnablement considérés comme indépendants, la présente Licence ne s'applique pas à ces éléments lorsque Vous les distribuez seuls. Mais, si Vous distribuez ces mêmes éléments comme partie d'un ensemble cohérent dont le reste est basé sur un Programme

soumis à la Licence, ils lui sont également soumis, et la Licence s'étend ainsi à l'ensemble du produit, quel qu'en soit l'auteur.

Cet article n'a pas pour but de s'approprier ou de contester vos droits sur un travail entièrement réalisé par Vous, mais plutôt d'ouvrir droit à un contrôle de la libre distribution de tout travail dérivé ou collectif basé sur le Programme.

En outre, toute fusion d'un autre travail, non basé sur le Programme, avec le Programme (ou avec un travail dérivé de ce dernier), effectuée sur un support de stockage ou de distribution, ne fait pas tomber cet autre travail sous le contrôle de la Licence.

4. Vous pouvez copier et distribuer le Programme (ou tout travail dérivé selon les conditions énoncées dans l'article 1) sous forme de code objet ou exécutable, selon les termes des articles 0 et 1, à condition de respecter les clauses suivantes :

- a. Fournir le code source complet du Programme, sous une forme lisible par un ordinateur et selon les termes des articles 0 et 1, sur un support habituellement utilisé pour l'échange de données ; ou,
- b. Faire une offre écrite, valable pendant au moins trois ans, prévoyant de donner à tout tiers qui en fera la demande une copie, sous forme lisible par un ordinateur, du code source correspondant, pour un tarif n'excédant pas le coût de la copie, selon les termes des articles 0 et 1, sur un support couramment utilisé pour l'échange de données informatiques ; ou,
- c. Informer le destinataire de l'endroit où le code source peut être obtenu (cette solution n'est recevable que dans le cas d'une distribution non commerciale, et uniquement si Vous avez reçu le Programme sous forme de code objet ou exécutable avec l'offre prévue à l'alinéa b ci-dessus).

Le code source d'un travail désigne la forme de cet ouvrage sous laquelle les modifications sont les plus aisées. Sont ainsi désignées la totalité du code source de tous les modules composant un Programme exécutable, de même que tout fichier de définition associé, ainsi que les scripts utilisés pour effectuer la compilation et l'installation du Programme exécutable. Toutefois, l'environnement standard de développement du système d'exploitation mis en oeuvre (source ou binaire) -- compilateurs, bibliothèques, noyau, etc. -- constitue une exception, sauf si ces éléments sont diffusés en même temps que le Programme exécutable.

Si la distribution de l'exécutable ou du code objet consiste à offrir un accès permettant de copier le Programme depuis un endroit particulier, l'offre d'un accès équivalent pour se procurer le code source au même endroit est considéré comme une distribution de ce code source, même si l'utilisateur choisit de ne pas profiter de cette offre.

5. Vous ne pouvez pas copier, modifier, céder, déposer ou distribuer le Programme d'une autre manière que l'autorise la Licence Publique Générale. Toute tentative de ce type annule immédiatement vos droits d'utilisation du Programme sous cette Licence. Toutefois, les tiers ayant reçu de Vous des copies du Programme ou le droit d'utiliser ces copies continueront à bénéficier de leur droit d'utilisation tant qu'ils respecteront pleinement les conditions de la Licence.
6. Ne l'ayant pas signée, Vous n'êtes pas obligé d'accepter cette Licence. Cependant, rien d'autre ne Vous autorise à modifier ou distribuer le Programme ou quelque travaux dérivés : la loi l'interdit tant que Vous n'acceptez pas les termes de cette Licence. En conséquence, en modifiant ou en distribuant le Programme (ou tout travail basé sur lui), Vous acceptez implicitement tous les termes et conditions de cette Licence.
7. La diffusion d'un Programme (ou de tout travail dérivé) suppose l'envoi simultané d'une licence autorisant la copie, la distribution ou la modification du Programme, aux termes et conditions de la Licence. Vous n'avez pas le droit d'imposer de restrictions supplémentaires aux droits transmis au destinataire. Vous n'êtes pas responsable du respect de la Licence par un tiers.
8. Si, à la suite d'une décision de Justice, d'une plainte en contrefaçon ou pour toute autre raison (liée ou non à la contrefaçon), des conditions Vous sont imposées (que ce soit par ordonnance, accord amiable ou autre) qui se révèlent incompatibles avec les termes de la présente Licence, Vous n'êtes pas pour autant dégagé des obligations liées à celle-ci : si Vous ne pouvez concilier vos obligations légales ou autres avec les conditions de cette Licence, Vous ne devez pas distribuer le Programme.

Si une partie quelconque de cet article est invalidée ou inapplicable pour quelque raison que ce soit, le reste de l'article continue de s'appliquer et l'intégralité de l'article s'appliquera en toute autre circonstance.

Le présent article n'a pas pour but de Vous pousser à enfreindre des droits ou des dispositions légales ni en contester la validité ; son seul objectif est de protéger l'intégrité du système de distribution du

Logiciel Libre. De nombreuses personnes ont généreusement contribué à la large gamme de Programmes distribuée de cette façon en toute confiance ; il appartient à chaque auteur/donateur de décider de diffuser ses Programmes selon les critères de son choix.

Cet article est censé clarifier une conséquence supposée du reste de cette licence.

9. Si la distribution et/ou l'utilisation du Programme est limitée dans certains pays par des brevets ou des droits sur des interfaces, le détenteur original des droits qui place le Programme sous la Licence Publique Générale peut ajouter explicitement une clause de limitation géographique excluant ces pays. Dans ce cas, cette clause devient une partie intégrante de la Licence.
10. La Free Software Foundation se réserve le droit de publier périodiquement des mises à jour ou de nouvelles versions de la Licence. Rédigées dans le même esprit que la présente version, elles seront cependant susceptibles d'en modifier certains détails à mesure que de nouveaux problèmes se font jour.

Chaque version possède un numéro distinct. Si le Programme précise un numéro de version de cette Licence et « toute version ultérieure », Vous avez le choix de suivre les termes et conditions de cette version ou de toute autre version plus récente publiée par la Free Software Foundation. Si le Programme ne spécifie aucun numéro de version, Vous pouvez alors choisir l'une quelconque des versions publiées par la Free Software Foundation.

11. Si Vous désirez incorporer des éléments du Programme dans d'autres Programmes libres dont les conditions de distribution diffèrent, Vous devez écrire à l'auteur pour lui en demander la permission. Pour ce qui est des Programmes directement déposés par la Free Software Foundation, écrivez-nous : une exception est toujours envisageable. Notre décision sera basée sur notre volonté de préserver la liberté de notre Programme ou de ses dérivés et celle de promouvoir le partage et la réutilisation du logiciel en général.

## LIMITATION DE GARANTIE

12. Parce que l'utilisation de ce Programme est libre et gratuite, aucune garantie n'est fournie, comme le permet la loi. Sauf mention écrite, les détenteurs du copyright et/ou les tiers fournissent le Programme en l'état, sans aucune sorte de garantie explicite ou implicite, y compris les garanties de commercialisation ou d'adaptation dans un but particulier. Vous assumez tous les risques quant à la qualité et aux effets du Programme. Si le Programme est défectueux, Vous assumez le coût de tous les services, corrections ou réparations nécessaires.
13. Sauf lorsqu'explicitement prévu par la Loi ou accepté par écrit, ni le détenteur des droits, ni quiconque autorisé à modifier et/ou redistribuer le Programme comme il est permis ci-dessus ne pourra être tenu pour responsable de tout dommage direct, indirect, secondaire ou accessoire (pertes financières dues au manque à gagner, à l'interruption d'activités ou à la perte de données, etc., découlant de l'utilisation du Programme ou de l'impossibilité d'utiliser celui-ci).

FIN DES TERMES ET CONDITIONS

## Annexe B. Licence de documentation libre GNU

### B.1. A propos de cette traduction française de la GFDL

Ceci est une traduction française non officielle de la Licence de documentation libre GNU. Elle n'a pas été publiée par la Free Software Foundation, et ne fixe pas légalement les conditions de redistribution des documents qui l'utilisent – seul le texte original en anglais le fait. Nous espérons toutefois que cette traduction aidera les francophones à mieux comprendre la FDL GNU.

### B.2. Licence de documentation libre GNU

Traduction française non officielle de la GFDL Version 1.1 (Mars 2000)

Copyright (C) 2000 Free Software Foundation, inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Traduction adaptée de la Version 1.1.2 FR (Christian Casteyde et César Alexanian, mars 2001)

Chacun est libre de copier et de distribuer des copies conformes de cette Licence, mais nul n'est autorisé à la modifier.

## 0. PRÉAMBULE

L'objet de cette Licence est de rendre tout manuel, livre ou autre document écrit « libre » au sens de la liberté d'utilisation, à savoir : assurer à chacun la liberté effective de le copier ou de le redistribuer, avec ou sans modifications, commercialement ou non. En outre, cette Licence garantit à l'auteur et à l'éditeur la reconnaissance de leur travail, sans qu'ils soient pour autant considérés comme responsables des modifications réalisées par des tiers.

Cette Licence est une sorte de « copyleft », ce qui signifie que les travaux dérivés du document d'origine sont eux-mêmes « libres » selon les mêmes termes. Elle complète la Licence Publique Générale GNU, qui est également une Licence copyleft, conçue pour les logiciels libres.

Nous avons conçu cette Licence pour la documentation des logiciels libres, car les logiciels libres ont besoin d'une documentation elle-même libre : un logiciel libre doit être accompagné d'un manuel garantissant les mêmes libertés que celles accordées par le logiciel lui-même. Mais cette Licence n'est pas limitée aux seuls manuels des logiciels ; elle peut être utilisée pour tous les documents écrits, sans distinction particulière relative au sujet traité ou au mode de publication. Nous recommandons l'usage de cette Licence principalement pour les travaux destinés à des fins d'enseignement ou devant servir de document de référence.

## 1. APPLICABILITÉ ET DÉFINITIONS

Cette Licence couvre tout manuel ou tout autre travail écrit contenant une notice de copyright autorisant la redistribution selon les termes de cette Licence. Le mot « Document » se réfère ci-après à un tel manuel ou travail. Toute personne en est par définition concessionnaire, et est référencée ci-après par le terme « Vous ».

Une « Version modifiée » du Document désigne tout travail en contenant la totalité ou seulement une portion de celui-ci, copiée mot pour mot, modifiée et/ou traduite dans une autre langue.

Une « Section secondaire » désigne une annexe au Document, ou toute information indiquant les rapports entre l'auteur ou l'éditeur et le sujet (ou tout autre sujet connexe) du document, sans toutefois être en rapport direct avec le sujet lui-même (par exemple, si le Document est un manuel de mathématiques, une Section secondaire ne traitera d'aucune notion mathématique). Cette section peut contenir des informations relatives à l'historique du Document, des sources documentaires, des dispositions légales, commerciales, philosophiques, ou des positions éthiques ou politiques susceptibles de concerner le sujet traité.

Les « Sections inaltérables » sont des sections secondaires considérées comme ne pouvant être modifiées et citées comme telles dans la notice légale qui place le Document sous cette Licence.

Les « Textes de couverture » sont les textes courts situés sur les pages de couverture avant et arrière du Document, et cités comme tels dans la mention légale de ce Document.

Le terme « Copie transparente » désigne une version numérique du Document représentée dans un format dont les spécifications sont publiquement disponibles et dont le contenu peut être visualisé et édité directement et immédiatement par un éditeur de texte quelconque, ou (pour les images composées de pixels) par un programme de traitement d'images quelconque, ou (pour les dessins) par un éditeur de dessins courant.

Ce format doit pouvoir être accepté directement ou être convertible facilement dans des formats utilisables directement par des logiciels de formatage de texte. Une copie publiée dans un quelconque format numérique ouvert mais dont la structure a été conçue dans le but exprès de prévenir les modifications ultérieures du Document ou dans le but d'en décourager les lecteurs n'est pas considérée comme une Copie Transparente. Une copie qui n'est pas « Transparente » est considérée, par opposition, comme « Opaque ».

Le format de fichier texte codé en ASCII générique et n'utilisant pas de balises, les formats de fichiers Texinfo ou LaTeX, les formats de fichiers SGML ou XML utilisant une DTD publiquement accessible, ainsi que les formats de fichiers HTML simple et standard, écrits de telle sorte qu'ils sont modifiables sans outil spécifique, sont des exemples de formats acceptables pour la réalisation de Copies Transparentes. Les formats suivants sont opaques : PostScript, PDF, formats de fichiers propriétaires qui ne peuvent être visualisés ou édités que par des traitements de textes propriétaires, SGML et XML utilisant des DTD et/ou des outils de formatage qui ne sont pas disponibles publiquement, et du code HTML généré par une machine à l'aide d'un traitement de texte quelconque et dans le seul but de la génération d'un format de sortie.

La « Page de titre » désigne, pour les ouvrages imprimés, la page de titre elle-même, ainsi que les pages supplémentaires nécessaires pour fournir clairement les informations dont cette Licence impose la présence sur la page de titre. Pour les travaux n'ayant pas de Page de titre comme décrit ci-dessus, la « Page de titre » désigne le texte qui s'apparente le plus au titre du document et situé avant le texte principal.

## 2. COPIES CONFORMES

Vous pouvez copier et distribuer le Document sur tout type de support, commercialement ou non, à condition que cette Licence, la notice de copyright et la notice de la Licence indiquant que cette Licence s'applique à ce Document soient reproduits dans toutes les copies, et que vous n'y ajoutiez aucune condition restrictive supplémentaire. Vous ne pouvez pas utiliser un quelconque moyen technique visant à empêcher ou à contrôler la lecture ou la reproduction ultérieure des copies que vous avez créées ou distribuées. Toutefois, vous pouvez solliciter une rétribution en échange des copies. Si vous distribuez une grande quantité de copies, référez-vous aux dispositions de la section 3.

Vous pouvez également prêter des copies, sous les mêmes conditions que celles précitées, et vous pouvez afficher publiquement des copies de ce Document.

## 3. COPIES EN NOMBRE

Si vous publiez des copies imprimées de ce Document à plus de 100 exemplaires, et que la Licence du Document indique la présence de Textes de couverture, vous devez fournir une couverture pour chaque copie, qui présente les Textes de couverture des première et dernière pages de couverture du Document. Les première et dernière pages de couverture doivent également vous identifier clairement et sans ambiguïté comme étant l'éditeur de ces copies. La première page de couverture doit comporter le titre du Document en mots d'importance et de visibilité égale. Vous pouvez ajouter des informations complémentaires sur les pages de couverture. Les copies du Document dont seules la couverture a été modifiée peuvent être considérées comme des copies conformes, à condition que le titre du Document soit préservé et que les conditions indiquées précédemment soient respectées.

Si les textes devant se trouver sur la couverture sont trop importants pour y tenir de manière claire, vous pouvez ne placer que les premiers sur la première page et placer les suivantes sur les pages consécutives.

Si vous publiez plus de 100 Copies opaques du Document, vous devez soit fournir une Copie transparente pour chaque Copie opaque, soit préciser ou fournir avec chaque Copie opaque une adresse réseau publiquement accessible d'une Copie transparente et complète du Document, sans aucun ajout ou modification, et à laquelle tout le monde peut accéder en téléchargement anonyme et sans frais, selon des protocoles réseau communs et standards. Si vous choisissez cette dernière option, vous devez prendre les dispositions nécessaires, dans la limite du raisonnable, afin de garantir l'accès non restrictif à la Copie transparente durant une année pleine après la diffusion publique de la dernière Copie opaque (directement ou via vos revendeurs).

Nous recommandons, mais ce n'est pas obligatoire, que vous contactiez l'auteur du Document suffisamment tôt avant toute publication d'un grand nombre de copies, afin de lui permettre de vous donner une version à jour du Document.



## 4. MODIFICATIONS

Vous pouvez copier et distribuer une Version modifiée du Document en respectant les conditions des sections 2 et 3 précédentes, à condition de placer cette Version modifiée sous la présente Licence, dans laquelle le terme « Document » doit être remplacé par les termes « Version modifiée », donnant ainsi l'autorisation de redistribuer et de modifier cette Version modifiée à quiconque en possède une copie. De plus, vous devez effectuer les actions suivantes dans la Version modifiée :

- A. Utiliser sur la Page de titre (et sur les pages de couverture éventuellement présentes) un titre distinct de celui du Document d'origine et de toutes ses versions antérieures (qui, si elles existent, doivent être mentionnées dans la section « Historique » du Document). Vous pouvez utiliser le même titre si l'éditeur d'origine vous en a donné expressément la permission.
- B. Mentionner sur la Page de titre en tant qu'auteurs une ou plusieurs des personnes ou entités responsables des modifications de la Version modifiée, avec au moins les cinq principaux auteurs du Document (ou tous les auteurs si il y en a moins de cinq).
- C. Préciser sur la Page de titre le nom de l'éditeur de la Version modifiée, en tant qu'éditeur du Document.
- D. Préserver intégralement toutes les notices de copyright du Document.
- E. Ajouter une notice de copyright adjacente aux autres notices pour vos propres modifications.
- F. Inclure immédiatement après les notices de copyright une notice donnant à quiconque l'autorisation d'utiliser la Version modifiée selon les termes de cette Licence, sous la forme présentée dans l'annexe indiqué ci-dessous.
- G. Préserver dans cette notice la liste complète des Sections inaltérables et les Textes de couverture donnés avec la notice de la Licence du Document.
- H. Inclure une copie non modifiée de cette Licence.
- I. Préserver la section nommée « Historique » et son titre, et y ajouter une nouvelle entrée décrivant le titre, l'année, les nouveaux auteurs et l'éditeur de la Version modifiée, tels que décrits sur la Page de titre, ainsi qu'un descriptif des modifications apportées depuis la précédente version.
- J. Conserver l'adresse réseau éventuellement indiquée dans le Document permettant à quiconque d'accéder à une Copie transparente du Document, ainsi que les adresses réseau indiquées dans le Document pour les versions précédentes sur lesquelles le Document se base. Ces liens peuvent être placés dans la section « Historique ». Vous pouvez ne pas conserver les liens pour un travail datant de plus de quatre ans avant la version courante, ou si l'éditeur d'origine vous en accorde la permission.
- K. Si une section « Dédicaces » ou une section « Remerciements » sont présentes, les informations et les appréciations concernant les contributeurs et les personnes auxquelles s'adressent ces remerciements doivent être conservées, ainsi que le titre de ces sections.
- L. Conserver sans modification les Sections inaltérables du Document, ni dans leurs textes, ni dans leurs titres. Les numéros de sections ne sont pas considérés comme faisant partie du texte des sections.
- M. Effacer toute section intitulée « Approbations ». Une telle section ne peut pas être incluse dans une Version modifiée.
- N. Ne pas renommer une section existante sous le titre « Approbations » ou sous un autre titre entrant en conflit avec le titre d'une Section inaltérable.

Si la Version modifiée contient de nouvelles sections préliminaires ou de nouvelles annexes considérées comme des Sections secondaires, et que celles-ci ne contiennent aucun élément copié depuis le Document, vous pouvez à votre convenance en désigner une ou plusieurs comme étant des Sections inaltérables. Pour ce faire, ajoutez leurs titres dans la liste des Sections inaltérables au sein de la notice de Licence de la Version modifiée. Ces titres doivent être distincts des titres des autres sections.

Vous pouvez ajouter une section nommée « Approbations », à condition que ces approbations ne concernent que les modifications ayant donné naissance à la Version modifiée (par exemple, comptes-rendus de revue de document, ou acceptation du texte par une organisation le reconnaissant comme étant la définition d'un standard).

Vous pouvez ajouter un passage comprenant jusqu'à cinq mots en première page de couverture, et jusqu'à vingt-cinq mots en dernière page de couverture, à la liste des Textes de couverture de la Version modifiée. Il n'est autorisé d'ajouter qu'un seul passage en première et en dernière page de couverture par personne ou groupe de personnes ou organisation ayant contribué à la modification du Document. Si le Document

comporte déjà un passage sur la même couverture, ajouté en votre nom ou au nom de l'organisation au nom de laquelle vous agissez, vous ne pouvez pas ajouter de passage supplémentaire ; mais vous pouvez remplacer un ancien passage si vous avez expressément obtenu l'autorisation de l'éditeur de celui-ci.

Cette Licence ne vous donne pas le droit d'utiliser le nom des auteurs et des éditeurs de ce Document à des fins publicitaires ou pour prétendre à l'approbation d'une Version modifiée.

## 5. FUSION DE DOCUMENTS

Vous pouvez fusionner le Document avec d'autres documents soumis à cette Licence, suivant les spécifications de la section 4 pour les Versions modifiées, à condition d'inclure dans le document résultant toutes les Sections inaltérables des documents originaux sans modification, et de toutes les lister dans la liste des Sections inaltérables de la notice de Licence du document résultant de la fusion.

Le document résultant de la fusion n'a besoin que d'une seule copie de cette Licence, et les Sections inaltérables existant en multiples exemplaires peuvent être remplacées par une copie unique. S'il existe plusieurs Sections inaltérables portant le même nom mais de contenu différent, rendez unique le titre de chaque section en ajoutant, à la fin de celui-ci, entre parenthèses, le nom de l'auteur ou de l'éditeur d'origine, ou, à défaut, un numéro unique. Les mêmes modifications doivent être réalisées dans la liste des Sections inaltérables de la notice de Licence du document final.

Dans le document résultant de la fusion, vous devez rassembler en une seule toutes les sections « Historique » des documents d'origine. De même, vous devez rassembler les sections « Remerciements » et « Dédicaces ». Vous devez supprimer toutes les sections « Approbations ».

## 6. REGROUPEMENTS DE DOCUMENTS

Vous pouvez créer un regroupement de documents comprenant le Document et d'autres documents soumis à cette Licence, et remplacer les copies individuelles de cette Licence des différents documents par une unique copie incluse dans le regroupement de documents, à condition de respecter pour chacun de ces documents l'ensemble des règles de cette Licence concernant les copies conformes.

Vous pouvez extraire un document d'un tel regroupement, et le distribuer individuellement sous couvert de cette Licence, à condition d'y inclure une copie de cette Licence et d'en l'ensemble des règles concernant les copies conformes.

## 7. AGRÉGATION AVEC DES TRAVAUX INDÉPENDANTS

La compilation du Document ou ses dérivés avec d'autres documents ou travaux séparés et indépendants sur un support de stockage ou sur un média de distribution quelconque ne représente pas une Version modifiée du Document tant qu'aucun copyright n'est déposé pour cette compilation. Une telle compilation est appelée « agrégat », et cette Licence ne s'applique pas aux autres travaux indépendants compilés avec le Document, s'ils ne sont pas eux-mêmes des travaux dérivés du Document.

Si les exigences de la section 3 concernant les Textes de couverture sont applicables à ces copies du Document, et si le Document représente un volume inférieur à un quart du volume total de l'agrégat, les Textes de couverture du Document peuvent être placés sur des pages de couverture qui n'encadrent que le Document au sein de l'agrégat. Dans le cas contraire, ils doivent apparaître sur les pages de couverture de l'agrégat complet.

## 8. TRADUCTION

La traduction est considérée comme une forme de modification, vous pouvez donc distribuer les traductions du Document selon les termes de la section 4. Vous devez obtenir l'autorisation spéciale des auteurs des Sections inaltérables pour les remplacer par des traductions, mais vous pouvez inclure les traductions des Sections inaltérables en plus des textes originaux. Vous pouvez inclure une traduction de cette Licence à condition d'inclure également la version originale en anglais. En cas de contradiction entre la traduction et la version originale en anglais, c'est cette dernière qui prévaut.

## 9. CADUCITÉ

Vous ne pouvez pas copier, modifier, sous-licencier ou distribuer le Document autrement que selon les termes de cette Licence. Toute autre acte de copie, modification, sous-licence ou distribution du Document est sans objet et vous prive automatiquement des droits que cette Licence vous accorde. En revanche, les personnes qui ont reçu de votre part des copies ou les droits sur le document sous couvert de cette Licence ne voient pas leurs droits caducs tant qu'elles en respectent les principes.

## 10. RÉVISIONS FUTURES DE CETTE LICENCE

La Free Software Foundation peut publier de temps en temps de nouvelles versions révisées de cette Licence. Ces nouvelles versions seront semblables à la présente version dans l'esprit, mais pourront différer sur des points de détail en fonctions de nouvelles questions ou problèmes. Voyez <http://www.gnu.org/copyleft/> (<http://www.gnu.org/copyleft/>) pour plus de détails.

Chaque version de cette Licence est dotée d'un numéro de version distinct. Si un Document spécifie un numéro de version particulier de cette Licence, et porte la mention « ou toute autre version ultérieure », vous pouvez choisir de suivre les termes de la version spécifiée ou ceux de n'importe quelle version ultérieure publiée par la Free Software Foundation. Si aucun numéro de version n'est spécifié, vous pouvez choisir n'importe quelle version officielle publiée par la Free Software Foundation.

### B.3. Comment utiliser cette Licence pour vos documents

Pour utiliser cette Licence avec un document que vous avez écrit, incorporez une copie du texte de cette Licence en anglais et placez le texte ci-dessous juste après la page de titre :

Copyright (c) ANNÉE VOTRE NOM Permission vous est donnée de copier, distribuer et/ou modifier ce document selon les termes de la Licence GNU Free Documentation License, Version 1.1 ou ultérieure publiée par la Free Software Foundation ; avec les sections inaltérables suivantes :

LISTE DES TITRES DES SECTIONS INALTÉRABLES

Avec le texte de première page de couverture suivant :

TEXTE DE PREMIÈRE PAGE DE COUVERTURE

Avec le texte de dernière page de couverture suivant :

TEXTE DE DERNIÈRE PAGE DE COUVERTURE

Une copie de cette Licence est incluse dans la section appelée GNU Free Documentation License de ce document.

Si votre Document ne comporte pas de section inaltérable, de textes de première et dernière pages de couverture, veuillez insérer les mentions suivantes dans les sections adéquates :

- pas de section inaltérable
- pas de texte de première page de couverture
- pas de texte de dernière page de couverture

Vous pouvez également fournir une traduction de la Licence GNU FDL dans votre document, mais celle-ci ne doit pas remplacer la version anglaise. La section intitulée GNU Free Documentation License doit contenir la version anglaise de la Licence GNU FDL, c'est la seule qui fait foi.

Si votre Document contient des exemples non triviaux de code programme, nous recommandons de distribuer ces exemples en parallèle sous Licence GNU General Public License, qui permet leur usage dans les logiciels libres.



## Annexe C. Glossaire

### *adresse IP*

est une adresse numérique composée de quatre séquences de un à trois chiffres qui identifie un ordinateur sur un réseau et notamment sur l'Internet. Les adresses IP sont structurées de manière hiérarchique, avec des domaines supérieurs et nationaux, domaines, sous-domaines, et adresses de chaque machine individuelle. Une adresse IP ressemble à 192.168.0.1.. L'adresse d'une machine personnelle peut être de deux types: statique ou dynamique. Les adresses IP statiques sont des adresses qui ne changent pas, alors que les adresses dynamiques seront réactualisées à chaque nouvelle connexion au réseau. Les utilisateurs de modem ou de câble ont généralement des adresses IP dynamiques, alors que certaines connexions DSL et d'autres à large bande fournissent des adresses IP statiques.

### *alias*

mécanisme utilisé dans un *shell* pour lui faire substituer une chaîne par une autre avant d'exécuter une commande. Vous pouvez voir tous les alias définis dans la session courante en tapant la commande *alias* à l'invite.

### *anneau tueur*

( *kill ring* ) sous *Emacs*, c'est l'ensemble des zones de texte copiées ou coupées depuis le démarrage de l'éditeur, que l'on peut rappeler pour les insérer de nouveau, et qui est organisé sous forme d'anneau. On peut aussi l'appeler « cercle des morts ».

### *APM*

*Advanced Power Management* (Gestion Avancée de l'Énergie): fonctionnalité utilisée par quelques *BIOS* pour faire entrer la machine dans un état de latence après une période d'inactivité donnée. Sur les ordinateurs portables, l'APM est aussi chargé de reporter le statut de la batterie et, si l'ordinateur le permet, la « durée de vie » restante estimée.

### *arp*

*Address Resolution Protocol*: Protocole de Résolution d'Adresses. Le protocole *Internet* utilisé pour faire automatiquement correspondre une adresse *Internet* et une adresse physique (matérielle) sur un réseau local. Cela est limité aux réseaux supportant la diffusion ( *broadcasting* ) matérielle.

### *arrière-plan*

dans le contexte du *shell*, un processus tourne en arrière-plan si vous pouvez envoyer des commandes pendant que ledit processus continue de fonctionner.

Voir aussi : *job*, avant-plan.

### *ASCII*

*American Standard Code for Information Interchange*: Code standard américain pour l'échange d'information. Le code standard utilisé pour stocker des caractères, y compris les caractères de contrôle, sur un ordinateur. Beaucoup de codes 8-bit (tels que ISO 8859-1, l'ensemble des caractères par défaut de *GNU/Linux*) contiennent ASCII sur leur moitié inférieure.

Voir aussi : .

### *ATAPI*

( « AT Attachment Packet Interface » ) Une extension des spécifications ATA ( « Advanced Technology Attachment », communément connue sous le nom d'IDE, *Integrated Drive Electronics* ) qui propose des commandes supplémentaires pour contrôler les lecteurs de CD-ROM et de bandes magnétiques. Les contrôleurs IDE proposant cette extension sont aussi appelés contrôleurs EIDE ( *Enhanced IDE* ).

### *atomique*

une série d'opérations est dite atomique si elle est exécutée en une seule fois, sans interruption.

### *avant-plan*

dans le contexte du *shell*, le processus au premier plan est celui qui est en train d'être exécuté. Vous devez attendre qu'un tel processus ait fini pour pouvoir entrer à nouveau des commandes.

Voir aussi : *job*, arrière-plan.

### *batch*

est un mode de gestion pour lequel les travaux ( *jobs* ) sont soumis au processeur séquentiellement jusqu'au dernier, le processeur est alors libéré pour une autre liste de processus.

**beta test**

est le nom donné à la procédure visant à tester la version beta (préliminaire) d'un programme. Ce dernier passe généralement par des phases dites « alpha » puis « beta » de test avant de sortir officiellement ( *release* ).

**bibliothèque**

est un ensemble de procédures et de fonctions au format binaire utilisé par les programmeurs dans leurs programmes (si la licence le leur permet). Le programme responsable du chargement des bibliothèques partagées au démarrage est appelé l'éditeur de liens dynamique ( *dynamic linker* ).

**bip**

est le petit bruit aigu émis par l'ordinateur pour attirer votre attention sur une situation ambiguë ou une erreur. Il est utilisé en particulier lorsque le complètement automatique d'une commande propose plusieurs choix.

**bit**

signifie *Binary digIT* ( *Chiffre Binaire* ). Un simple chiffre pouvant prendre les valeurs 0 ou 1, car on calcule en base deux.

**block (fichier en mode)**

fichiers dont le contenu est mis en tampon ( *buffer* ). toutes les opérations d'entrée/sortie sur de tels fichiers passent par le tampon, ce qui permet des écritures asynchrones sur le matériel sous-jacent, et des lectures directes sur le tampon, donc plus rapides.

*Voir aussi* : tampon, cache mémoire, caractère (fichiers en mode).

**bogue**

( *bug* ) comportement illogique ou incohérent d'un programme dans un cas particulier, ou comportement qui n'est pas en accord avec sa documentation. Souvent, dans un programme, de nouvelles fonctionnalités introduisent de nouveaux bogues. Le mot « bogue » est une francisation du mot anglais *bug*. Historiquement, ce terme remonte au temps des cartes perforées: une punaise (l'insecte!) qui se serait glissée dans un trou d'une carte perforée et aurait engendré un dysfonctionnement du programme ; Ada Lovelace, voyant cela, déclara « C'est un bug » ; l'expression est restée depuis.

**boot**

la procédure qui s'enclenche au démarrage d'un ordinateur lorsque les périphériques sont reconnus un par un, et que le système d'exploitation est chargé en mémoire.

**BSD**

*Berkeley Software Distribution* Distribution Logicielle de Berkeley: variante d' *Unix* développée au département informatique de l'université de Berkeley. Cette version a toujours été considérée plus avancée technologiquement que les autres, et a apporté beaucoup d'innovations au monde de l'informatique en général et à *Unix* en particulier.

**bureau**

Si vous utilisez l'environnement graphique *X* , le bureau est l'endroit de l'écran avec lequel vous travaillez et sur lequel sont placées les icônes et les fenêtres. Il peut aussi être appelé le fond d'écran, et est généralement rempli par une simple couleur, un gradient de couleur ou même une image.

*Voir aussi* : bureau virtuel.

**bureau virtuel**

Dans le système de fenêtrage *X*, le gestionnaire de fenêtres peut vous proposer plusieurs bureaux. Cette fonctionnalité pratique vous permet d'organiser vos fenêtres en limitant le nombre de fenêtres se superposant l'une à l'autre. Cela fonctionne de la même manière que si vous possédiez plusieurs écrans. Vous pouvez passer d'un bureau virtuel à un autre par le clavier ou la souris, selon le gestionnaire de fenêtres que vous utilisez.

*Voir aussi* : gestionnaire de fenêtres, bureau.

**cache mémoire**

élément crucial du noyau d'un système d'exploitation, il a pour rôle de maintenir les tampons à jour, de les effacer lorsqu'ils sont inutiles, de réduire la taille de l'antémémoire si nécessaire. Etc.

*Voir aussi* : tampon.

**canaux IRC**

il s'agit des « points de rencontre » à l'intérieur des serveurs IRC où vous pouvez converser avec d'autres utilisateurs. Les canaux sont créés dans les serveurs IRC et les utilisateurs se connectent à ces canaux afin de pouvoir communiquer entre eux. Les messages écrits sur un canal ne sont visibles que par les personnes connectées à ce canal. Deux ou plusieurs utilisateurs peuvent aussi créer des canaux « privés » afin de ne pas être dérangés par les autres utilisateurs. Les noms de canaux commencent par un #.

**caractère (fichiers en mode)**

fichiers dont le contenu n'est pas mis en tampon ( *buffer* ). Lorsqu'associé à un périphérique physique, toutes les entrées/sorties sont immédiatement effectuées. Certains périphériques caractères spéciaux sont créés par le système ( */dev/zero*, */dev/null* et d'autres). Ils correspondent aux flux de données. Voir aussi : *block* (fichier en mode).

**casse**

dans le contexte de chaînes de caractères, la casse est la différenciation entre lettres minuscules et majuscules (ou capitales).

**CHAP**

*Challenge-Handshake Authentication Protocol* (Protocole d'Authentification par Poignée de main-défi): protocole utilisé par les FAI pour authentifier leurs clients. Dans ce schéma, une valeur est envoyée au client (la machine qui se connecte), le client calcule un *hash* à partir de cette valeur qu'il envoie au serveur, et le serveur compare le *hash* avec celui qu'il a calculé.

Voir aussi : *PAP*.

**Chargeur de démarrage (bootloader)**

est le programme responsable pour le chargement du système d'exploitation. De nombreux chargeurs de démarrage vous donnent la possibilité de charger plus d'un système en vous proposant un menu. Les chargeurs tels que *grub* sont communs grâce à cette fonctionnalité et sont très utiles sur les systèmes multi-systèmes.

**chemin**

est une affectation d'un fichier ou un répertoire au système de fichiers. Les différents niveaux d'un chemin sont séparés par le « *slash* » ou caractère « */* ». Il y a deux types de chemins sous *GNU/Linux* . Le chemin **relatif** est la position d'un fichier ou un répertoire par rapport au répertoire courant. Le chemin **absolu** est la position d'un fichier ou un répertoire par rapport au répertoire racine.

**cible (target)**

désigne l'objet d'une compilation, généralement le fichier binaire devant être généré par le compilateur.

**CIFS**

*Common Internet FileSystem* ( *Système de Fichiers Internet Commun* ): prédécesseur du système de fichiers de SMB , utilisé sur les systèmes *DOS* .

**client**

programme ou ordinateur qui se connecte de façon épisodique et temporaire à un autre programme ou ordinateur pour lui donner des ordres ou lui demander des renseignements. C'est l'une des composantes d'un système **client/serveur**.

**code objet**

est le code généré par le processus de compilation devant être lié avec les autres codes objets et bibliothèques pour former un fichier exécutable. Le code objet est lisible par la machine.

Voir aussi : *compilation*, *liaison*.

**compilation**

est l'une des étapes de la traduction du code source (en langage compréhensible avec un peu d'entraînement) écrit en un langage de programmation ( *C* , par exemple) en un fichier binaire lisible par la machine.

**complètement**

néologisme (substantif masculin) désignant la capacité du *shell* à étendre une sous-chaîne en un nom de fichier, nom d'utilisateur ou autre, de façon automatique, si la sous-chaîne n'est pas ambiguë.

### **compression**

est un moyen de diminuer la taille des fichiers ou le nombre de caractères transmis lors d'une connexion. Certains programmes de compression de fichiers sont *compress*, *zip*, *gzip*, et *bzip2*.

### **compte**

sur un système *Unix*, un nom de connexion, un répertoire personnel, un mot de passe et un *shell* qui autorisent une personne à se connecter sur ce système.

### **console**

est le nom donné à ce que l'on appelait autrefois « terminal ». Ils constituaient les postes utilisateurs des gros ordinateurs centraux ( *mainframe* ). Sur les , le terminal physique est le clavier plus l'écran.

Voir aussi : consoles virtuelles.

### **consoles virtuelles**

Sur les systèmes *GNU/Linux*, les consoles virtuelles sont utilisées pour vous permettre de lancer plusieurs sessions sur un seul écran. Vous disposez par défaut de six consoles virtuelles qui peuvent être activées en pressant les combinaisons de touches: **ALT-F1** à **ALT-F6**. Il-y-a une septième console virtuelle par défaut, **ALT-F7**, qui vous permet de lancer le serveur X (interface graphique) Depuis X, vous pourrez activer les consoles virtuelles en pressant **CTRL-ALT-F1** à **CTRL-ALT-F6**.

Voir aussi : console.

### **cookies**

fichiers temporaires écrits sur le disque dur local par un site *web* distant. Cela permet au serveur d'être prévenu des préférences de l'utilisateur quand celui-ci se connecte à nouveau.

### **courrier électronique**

aussi appelé « mail », « email », « mèl » ou encore « courriel », il désigne un message que l'on fait parvenir à un autre utilisateur d'un même réseau informatique par voie électronique. Similaire au courrier traditionnel (dit courrier « escargot »), le courrier électronique a besoin des adresses de l'expéditeur et du destinataire pour être envoyé correctement. L'expéditeur doit avoir une adresse du type « moi@chez.moi » et le destinataire une adresse « lui@chez.lui ». Le courrier électronique est un moyen de communication très rapide (généralement quelques minutes quelque soit la destination). Afin de pouvoir écrire un courrier électronique, vous avez besoin d'un client de courrier du type de *pine* ou *mutt* qui sont en mode texte, ou des clients en mode graphique comme *kmail*.

### **DHCP**

*Dynamic Host Configuration Protocol* ( *Protocole Dynamique de Configuration d'Hôtes* ): protocole conçu pour que les machines sur un réseau local puissent se voir allouer une adresse IP dynamiquement.

### **disquette de démarrage**

disquette amorçable ( « *bootable* » en anglais) contenant le code nécessaire pour démarrer le système d'exploitation présent sur le disque dur (parfois, elle se suffit à elle-même).

### **distribution**

est le terme utilisé pour distinguer les différents produits proposés par les fournisseurs *GNU/Linux*. Une distribution est constituée du noyau *Linux*, et d'utilitaires, ainsi que de programmes d'installation, programmes de tiers, et parfois même des programmes propriétaires.

### **DMA**

*Direct Memory Access* Accès Direct à la Mémoire ): fonctionnalité utilisée sur l'architecture *PC*; qui permet à un périphérique de lire ou d'écrire dans la mémoire principale sans l'aide du processeur. Les périphériques PCI utilisent le *bus mastering* ( *contrôle du bus par un périphérique en lieu et place du processeur* ) et n'ont pas besoin de DMA.

### **DNS**

*Domain Name System*: système de nom de domaine. Le mécanisme de correspondance nom/adresse utilisé sur l' *Internet*. C'est ce mécanisme qui permet de mettre en correspondance un nom de domaine et une adresse IP, qui vous laisse rentrer un nom de domaine sans connaître l'adresse IP du site. DNS permet aussi d'effectuer une recherche inversée, de sorte que vous pouvez obtenir l'adresse IP d'une machine à partir de son nom.

### **DPMS**

*Display Power Management System* ( *Système de Gestion de l'Alimentation de l'Affichage* ). Protocole utilisé par tous les écrans modernes pour gérer les fonctionnalités de gestion d'énergie. Les moniteurs disposant de ces fonctionnalités sont souvent appelés des moniteurs « verts ».



**drapeau**

désigne un indicateur (généralement un seul bit) utilisé pour signaler une condition particulière à un programme. Par exemple, un système de fichier a, entre autres, un drapeau indiquant s'il doit être inclus dans une sauvegarde ; lorsque le drapeau est levé le système de fichier est sauvegardé, il ne l'est pas si le drapeau est désactivé.

**dépendances**

sont les étapes de la compilation nécessaires pour passer à l'étape suivante dans la compilation finale d'un programme.

**ELF**

*Executable and Linking Format* ( *Format d'Exécutables et de Liaisons* ) C'est le format binaire utilisé par la plupart des distributions *GNU/Linux* de nos jours.

**englobement**

dans le *shell* , la capacité de regrouper un certain ensemble de noms de fichiers avec un motif d'englobement.

*Voir aussi* : motif d'englobement.

**entrée standard**

le descripteur de fichier numéro 0, ouvert par tous les processus, utilisé par convention comme le descripteur de fichier par lequel le processus reçoit ses données.

*Voir aussi* : erreur standard, canal d', sortie standard.

**environnement**

est le contexte d'exécution d'un processus. Cela inclut toutes l'information dont le système d'exploitation a besoin pour gérer le processus, et ce dont le processeur a besoin pour exécuter correctement ce processus.

*Voir aussi* : processus.

**erreur standard, canal d'**

descripteur de fichier numéro 2, ouvert par tous les processus, utilisé par convention pour les messages d'erreur et, par défaut, l'écran du terminal.

*Voir aussi* : entrée standard, sortie standard.

**expression rationnelle**

un outil théorique très puissant utilisé pour la recherche et la correspondance de chaînes de texte. Il permet de spécifier des motifs auxquels les chaînes recherchées doivent se conformer. Beaucoup d'utilitaires *Unix* l'utilisent: *sed* , *awk* , *grep* et *perl* entre autres.

**ext2**

abréviation de « *second extended filesystem* »: système de fichier étendu 2. ext2 est le système de fichiers natif de *GNU/Linux* et possède toutes les caractéristiques d'un système de fichiers *Unix* : support des fichiers spéciaux (périphériques caractères, liens symboliques...), permissions sur les fichiers et propriétaires, etc.

**FAI**

*Fournisseur d'Accès Internet*. Une société qui revend un accès *Internet* à ses clients, que l'accès soit par lignes téléphoniques ou par lignes dédiées.

**FAQ**

*Frequently Asked Questions* (Questions Fréquemment Posées, mais aussi Foire Aux Questions): document contenant une série de questions/réponses sur un domaine spécifique. Historiquement, les FAQ s sont apparues dans les groupes de discussion, mais cette sorte de document est maintenant disponible sur des sites *web* divers. Même des produits commerciaux ont leur FAQ . Généralement, ce sont de très bonnes sources d'informations.

**FAT**

*File Allocation Table*. (Table d'Allocation des Fichiers). Système de fichiers utilisé par *DOS* / *Windows* .

**FDDI**

*Fiber Distributed Digital Interface* ( *Interface Numérique Distribuée par Fibre* ): couche réseau matérielle à haut débit, qui utilise des fibres optiques pour la communication. Seulement utilisée sur les gros réseaux et surtout à cause de son prix.

### FHS

*Filesystem Hierarchy Standard* ( *Standard pour la Hiérarchie des Systèmes de fichier* ): document contenant des lignes de conduite pour une arborescence des fichiers cohérente sur les systèmes *Unix* . **Mandrake Linux** se conforme à ce standard.

### FIFO

*First In, First Out* ( *Premier Entré, Premier Sorti* ): structure de données ou un tampon matériel depuis lequel les éléments sont enlevés dans l'ordre de leur insertion. Les tubes *Unix* sont l'exemple le plus courant de FIFO s.

### focus

le fait qu'une fenêtre reçoive les événements claviers (tels que les pressions ou les relâches des touches) et les clics de la souris, à moins que ces derniers ne soient interceptés par le gestionnaire de fenêtres.

### framebuffer

projection de la RAM d'une carte graphique dans la mémoire principale. Cela autorise les applications à accéder à la mémoire vidéo en évitant les complications liées à la communication directe avec la carte. Toutes les stations graphiques de haut niveau utilisent des framebuffers, par exemple.

### FTP

*File Transfer Protocol* (Protocole de Transfert de Fichiers). C'est le protocole *Internet* standard pour transférer des fichiers d'une machine à une autre.

### gestionnaire de fenêtres

programme responsable de l'allure générale d'un environnement graphique et qui s'occupe des barres et cadres des fenêtres, des boutons, des menus issus de l'image de fond, et de certains raccourcis clavier. Sans lui, il serait difficile ou impossible d'avoir des bureaux virtuels, de changer la taille des fenêtres à la volée, de déplacer ces dernières, ...

### GIF

*Graphics Interchange Format*. (Format Graphique d'Échange): format de fichier image, très utilisé sur le *web* . Les images GIF sont compressées, et elles peuvent même être animées. Pour des questions de droits d'auteur, il est conseillé de remplacer ce format d'image par le plus récent format PNG .

### GNU

*GNU is Not Unix* ( *GNU N'est pas Unix* ). Le projet GNU est un projet initié par Richard Stallman au début des années 80, dont le but est de concevoir un système d'exploitation libre et complet. Aujourd'hui, la plupart des outils sont disponibles, excepté... le noyau. Le noyau du projet GNU , *Hurd* , n'est pas encore prêt à sortir du laboratoire. *Linux* emprunte, entre autres, deux choses au projet GNU : son compilateur *C* , *gcc* , et sa licence, la GPL .  
Voir aussi : GPL.

### GPL

*General Public License* Licence Publique Générale: licence de nombreux programmes libres, notamment du noyau *Linux* . Elle va à l'encontre de toutes les licences propriétaires puisqu'elle ne donne aucune restriction en ce qui concerne la copie, la modification et la redistribution du logiciel, aussi longtemps que le code source est disponible. La seule restriction, si on peut l'appeler ainsi, est que les personnes à qui vous le redistribuez doivent aussi bénéficier des mêmes droits.

### HTML

*HyperText Markup Language* ( *Langage de Balisage HyperTexte* ): langage utilisé pour créer les documents *web* .

### HTTP

*HyperText Transfer Protocol* ( *Protocole de Transfert HyperTexte* ): protocole utilisé pour se connecter à des sites *web* et retirer des documents HTML ou des fichiers.

### hôte

se réfère à un ordinateur et est généralement utilisé pour des ordinateurs reliés à un réseau.

### i-nud

point d'entrée menant au contenu d'un fichier sur un système de fichiers *Unix* . Un i-nud est identifié de façon spécifique par un numéro, et contient des méta-informations sur le fichier auquel il se réfère, tels que ses temps d'accès, son type, sa taille, **mais pas son nom!**

**icône**

est un petit dessin (généralement en 16 x 16, 32 x 32, 48 x 48, et parfois 64 x 64 pixels) qui représente, sous un environnement graphique, un document, un fichier ou un programme.

**IDE**

*Integrated Drive Electronics Électronique Intégrée au Disque*: le plus utilisé des bus sur les PC d'aujourd'hui pour les disques durs. Un bus IDE peut contenir jusqu'à deux périphériques ; sa vitesse est limitée par le périphérique au bus qui a la file de commandes la plus lente (et pas la vitesse de transfert la plus lente!). Voir aussi : ATAPI.

**Interface graphique: GUI**

*Graphical User Interface*. Interface d'un ordinateur constituée de menus, boutons, icônes, et autres éléments graphiques. La plupart des utilisateurs préfèrent une interface graphique plutôt qu'une CLI (*Command Line Interface*: Interface en ligne de commande) pour sa facilité d'utilisation, même si cette dernière est plus polyvalente.

**Internet**

désigne un immense réseau qui connecte des ordinateurs tout autour du monde.

**invite**

*prompt* dans un *shell* , c'est la chaîne de caractères avant le curseur. Lorsqu'elle est visible, il est possible de taper vos commandes.

**IRC**

*Internet Relay Chat ( Conversation Relayée par l'Internet )*: une des rares normes sur l' *Internet* pour la conversation en direct. Elle autorise la création de canaux, des conversations privées et aussi l'échange de fichiers. Elle est aussi conçue pour être capable de faire se connecter les serveurs entre eux, et c'est pourquoi plusieurs réseaux IRC existent aujourd'hui: *Undernet*, *DALnet*, *EFnet* pour en citer quelques-uns.

**ISA**

*Industry Standard Architecture ( Architecture Standard pour l'Industrie )*: premier bus utilisé sur les , il est lentement abandonné au profit du bus PCI . Cependant, quelques fabricants de matériel l'utilisent toujours. Il est encore très courant que les cartes SCSI fournies avec des scanners, graveurs, etc. soient ISA .

**ISDN**

*Integrated Services Digital Network* ou RNIS: Réseau numérique à intégration de services. Un ensemble de standards de communication permettant à un unique câble ou fibre optique de transporter de la voix, des services de réseau numérique et de la vidéo. Il a été conçu afin de remplacer le système téléphonique actuel, connu sous l'acronyme RTC ( *Réseau Téléphonique Commuté* ).

**ISO**

*International Standards Organisation ( Organisation Internationale de Standards )*: groupement de compagnies, de consultants, d'universités et autres sources qui élabore des standards dans divers domaines, y compris l'informatique. Les documents décrivant les standards sont numérotés. Le standard numéro 9660, par exemple, décrit le système de fichiers utilisé par les CD-ROM .

**ISO 8859**

Le standard ISO 8859 inclut plusieurs extensions 8-bit à l'ensemble de caractères ASCII. Il y a notamment ISO 8859-1, l' « Alphabet Latin No. 1 », largement utilisé, qui peut en fait être considéré comme le remplaçant de facto du standard ASCII.

ISO 8859-1 reconnaît les langues suivantes: afrikaans, allemand, anglais, basque, catalan, danois, hollandais, écossais, espagnol, Féroais, finlandais, français, gallois, islandais, irlandais, italien, norvégien, portugais, et suédois.

Notez bien que les caractères ISO 8859-1 sont aussi les 256 premiers caractères de ISO 10646 (Unicode). Néanmoins, il lui manque le symbole EURO et ne reconnaît pas complètement le finlandais ni le français. L'ISO 8859-15 est une modification de ISO 8859-1 qui couvre ces besoins.

Voir aussi : ASCII.

**job**

dans le contexte du *shell* , un *job* est un processus fonctionnant en arrière-plan. Vous pouvez avoir plusieurs *jobs* dans un même *shell* , et contrôler ces *jobs*.

Voir aussi : avant-plan, arrière-plan.

**joker (wildcard)**

Les caractères « \* » et « ? » sont utilisés comme caractères dit jokers car ils peuvent représenter n'importe quoi. Le « \* » représente un nombre quelconque de caractères, alors que « ? » représente exactement un caractère. Les jokers sont utilisés fréquemment dans les expressions ordinaires.

**JPEG**

*Joint Photographic Experts Group* ( *Regroupement d'Experts de la Photographie* ) : autre format de fichier image très commun. JPEG est surtout habilité à compresser des scènes réelles, et ne fonctionne pas très bien avec les images non réalistes.

**lancer**

action d'invoquer, ou de démarrer un programme.

**langage assembleur**

est le langage de programmation le plus proche de l'ordinateur, d'où son nom de langage de programmation de « bas niveau ». L'assembleur a l'avantage de la vitesse puisque les programmes sont écrits directement sous la forme d'instructions pour le processeur, de sorte qu'aucune ou peu de traduction n'est nécessaire pour en faire un programme exécutable. Son inconvénient majeur est qu'il est fondamentalement dépendant du processeur (ou de l'architecture). Écrire des programmes complexes est donc très long. Ainsi l'assembleur est le langage de programmation le plus rapide, mais il n'est pas transportable entre architectures.

**LDP**

*Linux Documentation Project* ( *Project de Documentation pour Linux* ) : organisation à but non lucratif qui maintient de la documentation sur *GNU/Linux* . Ses documents les plus connus sont les *HOWTO* , mais il produit aussi des FAQ , et même quelques livres.

**lecture seule (read-only mode)**

signifie, pour un fichier, qu'il ne peut être modifié. On pourra en lire le contenu, mais pas le modifier.  
*Voir aussi* : lecture-écriture (read-write mode).

**lecture-écriture (read-write mode)**

Pour un fichier, cela signifie que l'on peut le modifier. Ce type d'autorisation permet à la fois de lire et de modifier un fichier.  
*Voir aussi* : lecture seule (read-only mode).

**liaison**

dernière étape du processus de compilation, consistant à lier ensemble les différents fichiers objet de façon à produire un fichier exécutable, et à résoudre les symboles manquant avec les bibliothèques dynamiques (à moins qu'une liaison statique ait été demandée, auquel cas le code de ces symboles sera inclus dans l'exécutable).

**libre (logiciel) open source**

nom donné au code source libre d'un programme qui est rendu disponible à la communauté de développement, et au public en général. La théorie sous-jacente est qu'en autorisant que le code source soit utilisé et modifié par un groupe plus large de programmeurs, cela produira un produit plus utile pour plus de personnes. On peut citer parmi les programmes libres les plus célèbres *Apache* , *sendmail* et *GNU/Linux* .

**lien**

désigne un i-nud dans un répertoire, donnant par là un nom (de fichier) à cet i-nud. Des exemples d'i-nuds n'ayant pas de liens (et donc aucun nom) sont: les tubes anonymes (utilisés par le *shell* ), les sockets (connexions réseau), périphériques réseau, etc.

**ligne de commande**

ce que fournit un *shell* et permet à l'utilisateur de taper des commandes directement. C'est également le sujet d'une bataille éternelle entre ses adeptes et ses détracteurs :-)

**Linux**

est un système d'exploitation du type *Unix* adapté à une grande variété d'architectures; il est utilisable et modifiable à volonté. Linux (le noyau) a été écrit par Linus Torvalds.

**login**

nom de connexion de l'utilisateur sur un système *Unix* , et l'action même de se connecter.

**loopback**

interface réseau virtuelle d'une machine avec elle-même, qui permet aux programmes en fonctionnement de ne pas devoir prendre en compte le cas particulier où deux entités réseau correspondent à la même machine.

**majeur**

numéro caractéristique de la classe de périphériques considérée.

**mandataire**

( *proxy* ) machine qui se situe entre un réseau et l' *Internet* , dont le rôle est d'accélérer les transferts de données pour les protocoles les plus utilisés ( HTTP et FTP par exemple). Il maintient un tampon des demandes précédentes, ce qui évite le coût impliqué par une nouvelle demande de fichier alors qu'une autre machine a fait cette requête récemment. Les serveurs mandataires sont très utiles sur les réseaux à petite vitesse (comprenez: connexions modems). Quelquefois, le mandataire est la seule machine capable d'atteindre l'extérieur.

**masquage IP**

est la technique utilisée lorsque vous utilisez un pare-feu pour cacher la véritable adresse IP de votre ordinateur de l'extérieur. Généralement, les connexions faites en dehors du réseau hériteront de l'adresse IP du pare-feu lui-même. Cela est utile dans les cas où vous avez une connexion *Internet* rapide avec une seule adresse IP officielle, mais souhaitez partager cette connexion avec d'autres ordinateurs d'un réseau local ayant des adresses IP privées.

**MBR**

*Master Boot Record* ( *Secteur de Démarrage Maître* ). Nom donné au premier secteur d'un disque dur amorçable. Le MBR contient le code utilisé pour charger le système d'exploitation en mémoire ou un chargeur de démarrage (tel que *LILO* ), et la table des partitions de ce disque dur.

**menu déroulant**

est un menu qui peut s' « enrouler » et se « dérouler » à volonté à l'aide d'un bouton situé à l'une de ses extrémités. Il sert généralement à choisir une des valeurs proposées dans ce menu.

**MIME**

*Multipurpose Internet Mail Extensions* ( *Extensions de Courrier pour l'Internet à Usages Multiples* ): chaîne de la forme *type/sous-type* décrivant le contenu d'un fichier attaché dans un courrier électronique. Cela autorise les lecteurs de courrier reconnaissant le MIME à effectuer des actions dépendantes du type du fichier.

**mineur**

numéro précisant le périphérique dont il est question.

**mode commande**

sous *Vi* ou l'un de ses clones, c'est l'état du programme dans lequel la pression sur une touche (ceci concerne surtout les lettres) n'aura pas pour effet d'insérer le caractère correspondant dans le fichier en cours d'édition, mais d'effectuer une action propre à la touche enfoncée (à moins que le clone que vous utilisez ne permette de personnaliser la correspondance entre touches et actions, et que vous ayez choisi cette fonctionnalité). On en sort en enfonçant l'une des touches ramenant au mode insert, comme **i**, **I**, **a**, **A**, **s**, **S**, **o**, **O**, **c**, **C**, ...

**mode insertion**

sous *Vi* ou l'un de ses clones, c'est l'état du programme dans lequel la pression sur une touche aura pour effet d'insérer le caractère correspondant dans le fichier en cours d'édition (sauf cas pathologiques comme complètement d'une abréviation, calibrage à droite en fin de ligne, ...). On en sort par une pression sur la touche **Échap** (ou **Ctrl-[** ).

**mot de passe**

mot secret ou combinaison de lettres, de chiffres et de symboles, utilisé pour protéger quelque-chose. Les mots de passe sont utilisés de concert avec les noms d'utilisateur ( *login* ) pour les systèmes multi-utilisateurs, sites *web* , FTP , etc. Les mots de passe devraient être des phrases difficiles à deviner, ou des combinaisons alpha-numériques, et ne doivent en aucun cas être basées sur des mots du dictionnaire. Les mots de passe empêchent que d'autres personnes puissent se connecter sur un ordinateur ou un site en utilisant votre compte.

**motif d'englobement**

une chaîne de caractères composée de caractères normaux et de caractères spéciaux. Les caractères spéciaux sont interprétés et étendus par le *shell* .

**MPEG**

*Moving Pictures Experts Group* ( *Groupe d'Experts en Images Animées* ): comité ISO qui génère des normes de compression audio et vidéo. MPEG est aussi le nom de leurs algorithmes. Malheureusement, la licence de ce format est très restrictive, par conséquent il n'existe aucun visualisateur MPEG sous licence libre...

**MSS**

La **MSS** ( *Maximum Segment Size*, « dimensioni massime di un segmento » ) è la massima quantità di dati che può essere trasmessa in un momento dato. Se volete evitare la frammentazione locale, la MSS deve essere pari alla intestazione MTU-IP.

**MTU**

The Maximum Transmission Unit ( **MTU** ) is a parameter that determines the largest datagram than can be transmitted by an IP interface without it needing to be broken down into smaller units. The MTU should be larger than the largest datagram you wish to transmit unfragmented. Note, this only prevents fragmentation locally, some other link in the path may have a smaller MTU and the datagram will be fragmented there. Typical values are 1500 bytes for an ethernet interface, or 576 bytes for a SLIP interface.

**NCP**

*NetWare Core Protocol* ( *Protocole de Base de NetWare* ): protocole défini par **Novell** pour accéder aux services de fichiers et d'impression de Novell Netware.

**NFS**

*Network FileSystem* ( *Système de Fichiers Réseau* ): système de fichiers réseau créé par **Sun Microsystems** pour partager des fichiers le long d'un réseau en toute transparence.

**NIC**

*Network Interface Controller* ( *Contrôleur d'Interface Réseau* ): adaptateur installé dans un ordinateur qui fournit une connexion physique à un réseau, comme une carte *Ethernet* .

**NIS**

*Network Information System* ( *Système d'Informations par Réseau* ). NIS était aussi connu sous le nom de « Yellow Pages » ( *Pages Jaunes* ), mais **British Telecom** possède un copyright sur ce nom. NIS est un protocole conçu par **Sun Microsystems** pour partager des informations communes le long d'un **domaine** NIS , qui peut regrouper un réseau local complet, quelques machines de ce réseau ou plusieurs réseaux locaux. Il peut exporter des bases de données de mots de passe, de services, d'informations de groupe, etc.

**niveau d'exécution (runlevel)**

désigne une configuration d'un système logiciel, qui ne permet que certains processus. Les processus autorisés sont définis pour chaque niveau dans le fichier */etc/inittab*. Il y a huit niveaux prédéfinis: 0, 1, 2, 3, 4, 5, 6, S et passer de l'un à l'autre ne peut se faire que par l'administrateur en exécutant les commandes *init* et *telinit*.

**nom d'utilisateur (username)**

appelé aussi *login*, est un nom (ou plus généralement un mot) qui identifie un utilisateur dans un système. Chaque nom d'utilisateur est associé à un unique UID ( *user ID* : IDentificateur d'Utilisateur)  
*Voir aussi* : *login*.

**nommage**

néologisme couramment employé dans le milieu de l'informatique pour nommer une méthode de désignation de certains objets. On parle souvent de « convention de nommage » pour des fichiers, des fonctions dans un programme, etc.

**noyau**

largement connu sous son nom anglais *kernel*, il est le cur du système d'exploitation. Le noyau est chargé de l'allocation des ressources et de la gestion des processus. Il prend en charge toutes les opérations de bas-niveau qui permettent aux programmes de communiquer directement avec le matériel de l'ordinateur.

**nul (caractère)**

le caractère ou octet de numéro 0, il est utilisé pour marquer la fin d'une chaîne de caractères.

**octet**

huit bits consécutifs. Il est interprété comme un nombre, en base deux, compris entre 0 et 255.  
 Voir aussi : bit.

**page de manuel**

un petit document contenant la définition d'une commande et son utilisation, à consulter avec la commande `man`. La première chose à (savoir) lire lorsqu'on entend parler d'une commande inconnue :-)

**PAP**

*Password Authentication Protocol* ( *Protocole d'Authentification par Mot de Passe* ): protocole utilisé par les FAI pour authentifier leurs clients. Dans ce schéma, le client (c'est vous) envoie une paire identifiant/mot de passe au serveur, non cryptée.  
 Voir aussi : CHAP.

**pare-feu**

( *firewall* ) machine qui est l'unique point d'entrée et de sortie avec le réseau extérieur dans la topologie d'un réseau local, et qui filtre ou contrôle l'activité sur certains ports, ou les réserve à des interfaces IP précises.

**patch, patcher**

*correctif*, fichier contenant une liste de modifications à apporter à un code source dans le but d'y ajouter des fonctionnalités, d'en ôter des bogues, ou d'y apporter toute autre modification souhaitée. L'action d'appliquer ce fichier à l'archive du code source.

**PCI**

*Peripheral Components Interconnect* ( *Interconnexion de Composants Périphériques* ): bus créé par **Intel** et qui est aujourd'hui le bus standard pour les architectures, mais d'autres architectures l'utilisent également. C'est le successeur de l'ISA, et il offre de nombreux services: identification du périphérique, informations de configuration, partage des IRQ, bus mastering, etc.

**PCMCIA**

*Personal Computer Memory Card International Association* (Association Internationale des Cartes Mémoires pour Ordinateurs Personnels): de plus en plus souvent appelé « PC Card » pour des raisons de simplicité; c'est la norme pour les cartes externes attachées aux ordinateurs portables: modems, disques durs, cartes mémoire, cartes *Ethernet*, etc. L'acronyme est quelquefois étendu avec humour en *People Cannot Memorize Computer Industry Acronyms* (Les gens ne peuvent pas mémoriser les acronymes de l'industrie informatique)...

**plein-écran**

Ce terme est utilisé pour désigner les applications qui prennent toute la place disponible de votre affichage.

**plugin**

programme d'appoint utilisé pour afficher ou enclencher un contenu multimédia proposé sur un document *web*. Il est généralement facile à télécharger lorsque le navigateur est encore incapable d'afficher ce type d'information.

**PNG**

*Portable Network Graphics* ( *Graphiques Réseau Portables* ): format de fichier image créé principalement pour l'utilisation sur le *Web*, il a été conçu comme un remplacement de GIF (sans les problèmes de brevets et avec des fonctionnalités supplémentaires).

**PNP**

*Plug'N'Play* ( *Brancher Et Utiliser* ). Conçu en premier lieu pour l'ISA pour ajouter des informations de configuration pour les périphériques, c'est devenu un terme plus générique qui regroupe tous les périphériques capables de rapporter leurs paramètres de configuration. Tous les périphériques PCI sont Plug'n'Play.

**processus**

dans un contexte *Unix*, un processus est l'instance d'un programme en cours d'exécution, avec son environnement.

**propriétaire**

dans le contexte des utilisateurs et de leurs fichiers, le propriétaire d'un fichier est celui qui a créé ce fichier.

Dans le contexte des groupes, le groupe propriétaire d'un fichier est le groupe auquel appartient le créateur de ce fichier.

**RAID**

*Redundant Array of Independent Disks* ( *Ensemble Redondant de Disques Indépendants* ): projet initié par le département informatique de l'université de Berkeley, et dans lequel le stockage des données est réparti sur un ensemble de disques.

**RAM**

*Random Access Memory* ( *Mémoire à Accès Aléatoire* ). Terme utilisé pour identifier la mémoire principale d'un ordinateur.

**RFC**

*Request For Comments* ( *Appel À Commentaires* ). Les RFC sont les documents officiels des standards de l'*Internet* . Ils décrivent tous les protocoles, leur utilisations, les prérequis imposés, etc. Pour comprendre le fonctionnement d'un protocole, allez chercher le RFC correspondant.

**route**

Is the path that your datagrams take through the network to reach their destination. Is the path between one machine and another in a network.

**RPM**

*Redhat Package Manager* ( *Gestionnaire de Paquetages de Red Hat* ). Un format d'emballage développé par **Red Hat** pour créer des paquetages logiciels, et utilisé par beaucoup de distributions *GNU/Linux* , y compris **Mandrake Linux** .

**répertoire**

Partie de la structure du système de fichiers. Un répertoire est un contenant pour les fichiers et éventuellement d'autres répertoires. Ces derniers sont alors appelés sous-répertoires (ou branches) du premier répertoire. On y fait souvent référence sous le terme d'arborescence. Si vous souhaitez voir le contenu d'un répertoire, vous pouvez soit le lister, soit y pénétrer. Les fichiers d'un répertoire sont appelés « feuilles » et les sous-répertoires « branches ». Les répertoires suivent les mêmes restrictions que les fichiers, bien que la signification des autorisations y soit parfois différente. Les répertoires spéciaux « . » et « .. » font respectivement référence au répertoire même et à son parent.

**répertoire personnel**

très souvent abrégé par « *home* », même en français, c'est le nom donné au répertoire d'un utilisateur donné.

Voir aussi : compte.

**réseau local**

Aussi appelé LAN *Local Area Network*. Nom générique donné à un réseau de machines physiquement connectées au même câble.

**sauvegarde**

est un moyen visant à protéger vos données importantes en les conservant sur un support et un endroit fiables. Les sauvegardes devraient être faites régulièrement, tout particulièrement pour les informations critiques et les fichiers de configuration (les premiers répertoires à sauvegarder sont */etc*, */home*, et */usr/local* ). Généralement, on utilise *tar* avec *gzip* ou *bzip2* pour sauvegarder des répertoires et des fichiers. Il existe d'autres outils ou programmes tels que *dump* et *restore* , ainsi qu'une quantité d'autres solutions libres ou commerciales pour la sauvegarde de nos documents.

**SCSI**

*Small Computers System Interface* ( *Interface Système pour Petits Ordinateurs* ): bus avec une grande bande passante mis au point pour autoriser plusieurs types de périphériques. Contrairement à l' *IDE* , un bus SCSI n'est pas limité par la vitesse à laquelle les périphériques acceptent les commandes. Seules les machines de haut niveau intègrent un bus SCSI directement sur la carte mère; une carte additionnelle est donc nécessaire pour les *PC* .



**serveur**

programme ou ordinateur qui propose une fonctionnalité ou service et attend les connexions des **clients** pour répondre à leurs ordres ou leur fournir les renseignements qu'ils demandent. C'est l'une des composantes d'un système **client/serveur**.

**shadowr, mots de passe**

un système de gestion des mots de passe dans laquelle le fichier contenant les mots de passe chiffrés n'est plus lisible par tout le monde, alors qu'il l'est quand on utilise le système normal de mots de passe.

**SMB**

*Server Message Block* ( *Serveur de Messages par Blocs* ). Protocole utilisé par les machines *windows* ( *9x* or *NT* ) pour le partage de fichiers le long d'un réseau.

**socket**

type de fichier correspondant à tout ce qui est connexion réseau.

**sortie standard**

le descripteur de fichier numéro 1, ouvert par tous les processus, utilisé par convention comme le descripteur de fichier dans lequel le processus écrit les données qu'il produit.

*Voir aussi* : erreur standard, canal d', entrée standard.

**SVGA**

*Super Video Graphics Array* ( *Super Affichage Graphique Vidéo* ) : norme d'affichage vidéo définie par VESA pour l'architecture *PC* . La résolution est de 800 x 600 x 16 couleurs.

**switch (options)**

Les switches sont utilisés pour modifier le comportement des programmes, et sont aussi appelés : options de ligne de commande ou arguments. Pour déterminer si un programme propose des switches en option, lisez sa page de man pages ou essayez de lui passer l'option `--help` (ie. `program --help` ).

**symboliques, liens**

fichiers particuliers, ne contenant qu'une chaîne de caractères. Tout accès à ces fichiers est équivalent à un accès au fichier dont le nom est donné par cette chaîne de caractères, qui peut ou non exister, et qui peut être précisé par un chemin relatif ou absolu.

**système client/serveur**

système ou protocole composé d'un **serveur** et d'un ou plusieurs **clients**.

**système d'exploitation**

est l'interface entre les applications et le matériel sous-jacent. Les tâches de tout système d'exploitation sont en premier lieu de gérer toutes les ressources spécifiques à une machine. Sur un système *GNU/Linux* , cela est fait pas le noyau et les modules chargeables. D'autres systèmes d'exploitation connus sont *AmigaOS* , *MacOS* , *FreeBSD* , *OS/2* , *Unix* , *Windows NT* , et *Windows 9x* .

**système de fichiers**

schéma utilisé pour stocker des fichiers sur un support physique (disque dur, disquette) d'une manière consistante. Des exemples de systèmes de fichiers sont la FAT , ext2fs de *Linux* , iso9660 (utilisé par les CD-ROM ), etc.

**tampon**

( *buffer* ) Zone de mémoire de taille fixe, pouvant être associée à un fichier en mode bloc, une table du système, un processus etc. La cohérence de tous les tampons est assurée par le cache mémoire.

**thémable**

pour une application graphique, cela indique qu'elle peut changer son apparence en temps réel. Beaucoup de gestionnaires de fenêtres sont également thématiques.

**traverser**

pour un répertoire sur un système *Unix* , cela signifie que l'utilisateur est autorisé à passer à travers ce répertoire et, si possible, de se rendre dans ses sous-répertoires. Cela requiert que l'utilisateur ait le droit d'exécution sur ce répertoire.

**tube**

Type de fichiers spécial d' *Unix* . Un programme écrit des données dans le tube, et un autre programme lit les données à l'autre bout. Les tubes *Unix* sont FIFO , donc les données sont lues à l'autre bout dans l'ordre où elles ont été envoyées. Très utilisés dans le *shell* . Voir aussi **tube nommé**.

**tube nommé**

Tube *Unix* qui est lié, contrairement aux tubes utilisés dans le *shell*.  
Voir aussi : tube, lien.

**URL**

*Uniform Resource Locator* ( *Localisateur Uniforme de Ressources* ) : ligne avec un format spécial utilisée pour identifier une ressource sur l' *Internet* d'une façon univoque. La ressource peut être un fichier, un serveur et autres. La syntaxe d'un URL est  
protocole://nom.du.serveur[:port]/chemin/vers/ressource.  
Quand un nom de machine est donné seulement et que le protocole est http://, cela équivaut à retirer l'éventuel fichier intitulé index.html du serveur par défaut.

**valeurs discrètes**

Sont des valeurs non continues ou qui ne se suivent pas, comme s'il existait une sorte d' « espace » entre deux valeurs consécutives.

**variables**

chaînes utilisées dans les fichiers *Makefile* pour être remplacées par leur valeur chaque fois qu'elles apparaissent. Elles sont généralement définies au début du fichier *Makefile* et sont utilisées pour simplifier le *Makefile* et la gestion de l'arborescence des fichiers source.

De manière plus générale, en programmation, les variables sont des mots qui font référence à d'autres entités (nombres, chaînes, tableaux de valeurs, etc.) qui sont susceptibles de varier au cours de l'exécution du programme.

**variables d'environnement**

une partie de l'environnement d'un processus. Les variables d'environnement sont directement visibles depuis le *shell* .

Voir aussi : processus.

**verbeux**

Pour les commandes, le mode verbeux fait que la commande va afficher sur la sortie standard (ou erreur) toutes les actions engagées et les résultats de ces actions. Les commandes offrent parfois un « niveau de volubilité », ce qui signifie que la quantité d'information fournie peut être contrôlée.

**VESA**

*Video Electronics Standards Association* ( *Association pour les Standards des matériels Vidéo Électroniques* ) : association de normes de l'industrie orientée vers l'architecture . Elle est l'auteur de la norme SVGA , par exemple.

**visionneuse**

( *pager* ) programme présentant un fichier texte page écran par page écran, et proposant des facilités de déplacement et de recherche dans ce fichier. Nous conseillons *less* .

**volée (à la)**

On dit qu'une action est réalisée « à la volée » lorsqu'elle est faite en même temps qu'une autre sans que l'on s'en rende compte ou sans qu'on l'ait explicitement demandé.

**WAN : réseau étendu**

*Wide Area Network*: réseau à large portée. Ce réseau, bien que similaire au réseau local ( LAN ), connecte des ordinateurs sur un réseau qui n'est pas relié physiquement aux mêmes brins, et sont séparés par une large distance.

**échappement**

dans le contexte du *shell* , désigne l'action d'encadrer une chaîne entre guillemets pour empêcher son interprétation. Par exemple, pour utiliser des espaces sur une ligne de commande, puis rediriger le résultat à une autre commande, il faudra mettre la première commande entre guillemets ( « échapper » la commande) sinon le *shell* l'interprétera mal, ce qui empêchera le bon fonctionnement.

***écho***

voir un écho signifie voir à l'écran les caractères qui sont frappés au clavier. Par exemple, lorsque l'on tape un mot de passe, on n'a généralement pas d'écho mais de simple étoiles « \* » pour chaque caractère tapé.

***éditeur***

Ce terme désigne les programmes spécialisés dans la modification des fichiers textes. Les éditeurs les plus connus sous *GNU/Linux* sont GNU Emacs ( *Emacs* ) et l'éditeur *Unix* : *Vi* .



# Index

- .bashrc, 14
- /dev/hda, 11
- éditeur
  - Emacs, 23
  - vi, 26
- applications
  - ImageMagick, 19
  - terminal, 19
- attribut
  - fichier, 15
- caractère
  - englobement, 17
- caractères
  - spéciaux, 19
- command
  - sed, 18
- commande
  - at, 38
  - bzip2, 40, 72
  - cat, 7
  - cd, 5
  - chgrp, 15
  - chmod, 16
  - chown, 15
  - configure, 73
  - cp, 15
  - crontab, 37
  - find, 35
  - grep, 32
  - gzip, 40
  - init, 67
  - less, 7, 18
  - ls, 7
  - make, 75
  - mkdir, 13
  - mount, 52
  - mv, 14
  - patch, 86
  - pwd, 5
  - rm, 13
  - tar, 38, 71
  - touch, 13
  - umount, 53
  - wc, 18
- commandes
  - rmdir, 14
- complètement, 19
- compte, 1
- console, 1
- contributeurs, ii
- conventions
  - typographiques, ii
- disques, 9
- Docbook, ??
- documentation, ii
- englobement
  - caractère, 17
- environment

- variable, 6
- environnement
  - processus, 63
- FHS, 47
- fichier
  - attribut, 15, 62
  - copier, 15
  - créer, 13
  - déplacement, 14
  - effacement, 13
  - lien, 57, 58
  - mode bloc, 57
  - mode caractère, 57
  - mode bloc, 60
  - mode caractère, 60
  - propriétaire, 15
  - renommer, 14
  - socket, 57
  - tube, 57
- framebuffer, 93
- Free Software Foundation, ??
- GFDL, 99
- GID, 2
- GNU Free Documentation License, ??
- GNU/Linux, 1
- GPL, 95
- groupe, 1
  - changement, 15
- grub, 92
- home
  - partition, 10
- horodatage
  - atime, 13
  - ctime, 13
  - mtime, 13
- IDE
  - périphériques, 11
- inoeud, 57
- internationalisation, ii
- invite, 2
- lien
  - dur, 61
  - symbolique, 61
- ligne de commande
  - introduction, 13
- lilo, 89
- Makefile, 70, 76
- Mandrake
  - listes de diffusion, i
- Mandrake Secure, i
- MandrakeClub, i
- MandrakeExpert, i
- MandrakeSoft, ??
- MandrakeSoft S.A., ??
- MandrakeStore, ii
- modules, 65
- mot de passe, 1
- ordre de tri, 17
- paquetage, ii
- partition, 9

- logique, 11
- primaire, 11
- étendue, 11
- permission, 16
- PID, 4
- Pierre Pingus, iv
- pipe, 18
- process, 63
- processus, 4, 20
- programmation, ii
- prompt, 5
- propriétaire
  - changement, 15
- racine
  - répertoire, 47, 64
- RAM, mémoire, 10
- redirection, 18
- Reine Pingusa, iv
- root
  - partition, 10
  - utilisateur, 2
- runlevel, 67
- répertoire
  - copier, 15
  - créer, 13
  - déplacement, 14
  - effacement, 13
  - renommer, 14
- SCSI
  - disques, 11
- secteur, 9
- shell, 5, 13
  - motifs d'englobement, 17
- Soundblaster, 11
- standard
  - entrée, 17
  - erreur, 17
  - sortie, 17
- supermount, 54
- swap, 9
  - partition, 10
  - taille, 10
- système de fichiers
  - Devfs (Device File System, 12
- Torvalds, Linus, ??
- tube
  - anonyme, 59
  - nommé, 59
- UID, 2
- UNIX, 1
- usr
  - partition, 10
- utilisateur, 1
- utilisateurs
  - génériques, iv
- utilitaires
  - manipulation de fichiers, 13
- valeurs
  - discrètes, 17
- virus, 4