# INFZ20, Introduction au monde d'Unix

AUDIBERT Laurent [1]

23 octobre 2003

1. Jeune équipe DELIC - Université de Provence - 29 Avenue Robert SCHUMAN - 13621 Aix-en-Provence Cedex 1 - laurent.audibert@up.univ-aix.fr

Ce cours est un cours d'introduction au monde d'Unix dispensé au Centre Informatique pour les Lettres et les Sciences Humaines (CILSH). Les objectifs de ce cours sont multiples.

Tout le monde a, un jour ou l'autre, entendu parler d'*Unix*, « ce système très puissant et fiable qui anime les machines sérieuses ». Les médias, et surtout la presse informatique, parlent de plus en plus de *Linux*, cet Unix gratuit que l'on peut mettre dans le PC de monsieur tout-le-monde et qui est actuellement le seul concurrent crédible, bien qu'encore très discret, de l'ogre Microsoft. Pourtant, peu de personnes ont déjà aperçu, et encore moins travaillé avec, une machine fonctionnant sous Unix ou Linux. Ce cours doit, tout d'abord, permettre aux étudiants d'avoir un premier contact réel avec le monde de Linux.

D'autre part, certains étudiants seront amenés à poursuivre leurs études par un « DESS informatique double compétence » où il seront certainement confrontés à des machines fonctionnant sous Unix. D'autre encore pourront être confrontés à Unix dans leur vie professionnelle. Ce cours devrait permettre à toutes ces personnes de ne pas se retrouver complètement déroutées et démunies face à cette situation possible à venir.

Enfin les utilitaires d'Unix, disponibles par la ligne de commande, permettent d'effectuer des opérations complexes et très utiles sur des fichiers, opérations qu'il est beaucoup plus fastidieux de réaliser dans le monde Windows. Ce cours devrait donc offrir une « boîte à outils » très utile pour des linguistes travaillant souvent avec de gros corpus écrits.

# Table des matières

# Chapitre 1

# Système d'exploitation

Les objectifs de ce premier chapitre sont :

1. rappeler sommairement ce qu'est un système d'exploitation ;
2. rappeler ce qu'est un système de fichier et, notamment, définir ce que sont les fichiers et les répertoires.

Pourquoi ne parlerons-nous pas d'Unix dans ce premier chapitre ou dans ce premier TP ?

Avant de travailler avec un ordinateur, il est essentiel de posséder certaines notions de base, notamment de bien comprendre ce que sont les répertoires et les fichiers. Vous êtes familiarisé avec un environnement graphique, Microsoft Windows, qui tente d'assister au maximum l'utilisateur en masquant par tous les moyens les considérations techniques, comme les notions de fichiers et de répertoires par exemple, et en éloignant l'utilisateur de la ligne de commande. Cette philosophie comporte ses avantages et ses inconvénients. Le monde d'Unix n'a pas la même philosophie. Nous allons donc commencer par redéfinir des notions de base qui sont aujourd'hui communes à tous les systèmes d'exploitation et allons nous rapprocher de la ligne de commande dans un environnement qui nous est plus familier, celui de Microsoft Windows. Toute les conditions seront alors remplies pour un passage en douceur vers le monde Unix.

## 1.1 Qu'est ce qu'un système d'exploitation ?

Le système d'exploitation (en anglais *Operating System*, ou OS) est le logiciel central d'un ordinateur. C'est lui qui réalise l'interaction entre le matériel, les programmes et les utilisateurs. Il sert à donner au programmeur et à l'utilisateur une vision de la machine indépendante de son infrastructure matérielle. Il gère notamment :

- tout le processus de (re)démarrage ;
- le ou les processeurs ;
- les mémoires ;
- l'exécution des programmes (allocation de ressources, communication entre les processus, contrôle d'accès) ;
- le système de fichiers ;
- les liens avec les périphériques (via des programme appelés "pilotes", ou "drivers") ;
- ...

Voici quelques uns des systèmes d'exploitation les plus connus :

- MS-DOS (Microsoft-Disk Operating Systems), c'est notamment le noyau d'environnements graphiques comme Windows 3.1, Windows 9$x$ ou encore Windows millennium ;
- Windows NT, Windows 2000, Windows XP ;
- Unix et Linux (qui désigne plutôt des familles d'OS) ;
- OS/2 développé par IBM ;
- MacOS pour les Macintosh ;
- Be-OS ;
- Openstep (anciennement Nextstep) ;
- ...

Les tâches d'un système d'exploitation ayant augmentées considérablement au fil des années, cet ensemble de programmes n'a cessé de croître. Il s'agit donc de logiciels volumineux nécessitant de gros efforts de programmation. Un système d'exploitation moderne est constitué de centaines de milliers, voire de millions de lignes de codes, requérant des milliers d'homme-années de travail.

Aujourd'hui, le système d'exploitation est devenu l'intermédiaire obligatoire entre l'utilisateur et la machine.

**Définition 1.1 -Système d'exploitation-** *Le système d'exploitation est l'ensemble des programmes qui se chargent de tous les problèmes relatifs à l'exploitation de l'ordinateur.*

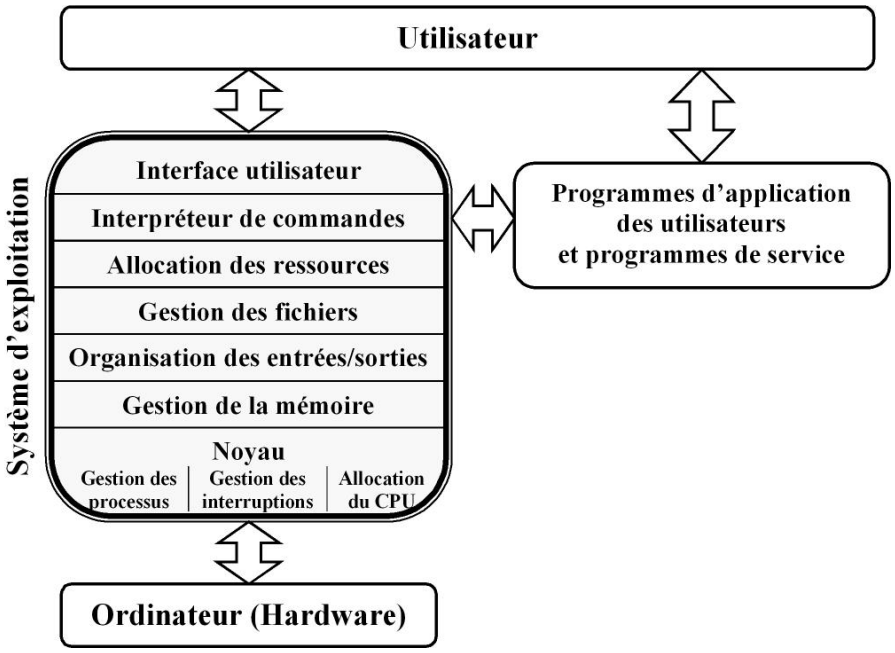


FIG. 1.1 – *Structure d'un système d'exploitation.*

En examinant les fonctions d'un système d'exploitation, on s'aperçoit qu'elles ne sont pas toutes d'un même niveau. On peut faire un modèle de système basé sur une superposition de couches fonctionnelles (figure 1.1) les couches les plus basses étant celles qui sont en interaction directe avec le matériel et, les plus hautes, celles qui servent d'interface avec l'utilisateur. Chaque couche utilise les fonctions définies par les couches inférieures.

Sur tous les systèmes d'exploitation, on distingue deux types d'interfaces utilisateurs :

- L'*interface ligne de commande* (CLI, de l'anglais *Command Line Interface*) est l'interface originelle et certainement celle qui reste la plus puissante. Sous Windows 2000 on l'appelle *invite de commandes* et sous Unix/Linux, on la nomme *shell*.
- L'*interface graphique* (GUI, de l'anglais *Graphical User Interface*) est la plus conviviale. Sous Windows 2000 cette interface est intégrée au système tandis qu'elle est indépendante sous Unix/Linux de sorte qu'il en existe plusieurs (Gnome, KDE, ...).

La partie la plus importante de tout système informatique est celle qui contient les données : les informations qui sont emmagasinées et manipulées par les programmes. Cette partie est confiée au système de fichier qui s'occupe de la gestion des fichiers.

## 1.2 Qu'est ce qu'un fichier ?

Dans la vie de tous les jours, le concept de fichier désigne une collection de feuilles de papier que l'on peut conserver dans une chemise par exemple. En informatique, un fichier est un ensemble de données stockées dans une mémoire non volatile (un disque, une bande, etc.) de façon à être lues et traitées par un ordinateur, et qui représente une entité pour l'utilisateur.

**Définition 1.2 -Fichier-** *Collection d'informations enregistrées de façon à être lues et traitées par un ordinateur.*

Un fichier d'ordinateur peut contenir un document, un programme, une feuille de tableur, une image, un message, un son, un vidéo, etc. Chaque fichier possède un nom servant à le référencer.

Aujourd'hui, on garde sur disque, ou disquette, les adresses des amis, les recettes de cuisine, le portefeuille de papiers-valeurs, etc. Le texte et les images de cette photocopie sont issus d'un fichier. Les banques gèrent des fichiers contenant l'état de nos finances. Nos informations personnelles concernant notre état civil, notre sécurité sociale, notre position fiscale, . . ., sont conservées dans d'innombrables fichiers entretenus par les administrations publiques.

La question de l'utilité d'un système de gestion des fichiers ne se pose plus. De toute évidence, on ne peut pas se passer des services du système de fichiers (en anglais : *file system*) qui, pour l'utilisateur, constitue la partie la plus visible du système d'exploitation. On exige d'un bon système d'exploitation qu'il soit doté d'un bon système de fichier aussi simple, sûr et fiable que possible.

## 1.3 Qu'est ce qu'un répertoire ?

Les répertoires servent à organiser les fichiers en un système hiérarchique. Pour cela, il faut rassembler des fichiers en des groupes et garder chaque groupe dans un répertoire

spécifique. Les répertoires peuvent être eux-mêmes considérés comme des fichiers ; un répertoire peut ainsi contenir d'autre répertoires. C'est ainsi que l'on crée une arborescence de répertoires.

Un *répertoire parent* est un répertoire qui contient d'autres répertoires. Un *sous-répertoire* est un répertoire qui se trouve dans un autre répertoire.

Dans le monde Windows, les *répertoires* sont appelés *dossiers*. On utilise aussi parfois le mot *catalogue*. Ces trois appellations désignent la même chose.

## 1.4 TP sur les fichiers et répertoires

Cette séance de travaux pratiques (TP) se déroulera sous Windows.

Les objectifs de cette première séance sont :

1. pérenniser les notions de fichiers et de répertoires en réalisant une série d'opération sur ces derniers ;
2. interagir, peut-être pour la première fois, avec une ligne de commande et son aide en ligne associée ;
3. Manipuler les caractères génériques de Windows.

### 1.4.1 Création d'une arborescence rationnelle

Imaginez la situation suivante. Vous possédez un ordinateur familial et désirez créer une arborescence de répertoires qui répondent aux besoins suivants :

- Vous désirez que tous les fichiers des utilisateurs se trouvent dans un répertoire unique[1], par exemple dans `c:\Utilisateurs`.
- Vous désirez également cloisonner vos données personnelles, celles de votre père, celles de votre mère et, le cas échéant, celles de vos frères et sœurs. Cette façon de procéder permettra à chacun de disposer de son propre répertoire sans empiéter sur celui des autres, donc sans mélanger ses fichiers avec ceux des autres, et permettra également à chacun de gérer sa sous-arborescence sans imposer son classement aux autres membres de la famille.
- Vous laisserez chacun classer ses fichiers dans son répertoire. Comme vous êtes très ordonné, dans votre répertoire, vous désirez classer les fichiers correspondant à vos cours par année universitaire (exemple: `DEUG1`, `DEUG2`, `Licence`, `Maîtrise`, etc.) puis, pour chaque année par module (exemple : `INFZ20`, `INFZ21`, `INFZ23`, `INFZ24`, etc.).

Proposez et réalisez une arborescence répondant à ce cahier des charges.

### 1.4.2 Sauvegarde, copie, déplacement, suppression d'un fichier

Avec le logiciel Word, créer un petit document (un texte) que vous sauverez dans le répertoire de votre choix. En utilisant le gestionnaire de fichier, exercez-vous à faire des copies, des déplacements, des suppressions de fichier.

1. Cette façon de procéder est très recommandée et permet, par exemple, de sauver et/ou de déplacer facilement toutes les données sensibles (importantes) lorsqu'il faut réinstaller le système d'exploitation ou encore lorsqu'il faut changer d'ordinateur.

Quand un fichier est supprimé, en réalité, il est déplacé dans la corbeille. S'il s'agit d'un fichier sensible que vous désirez supprimer définitivement, comment faites-vous ?

### 1.4.3 Mais où sont-ils ?

Lorsque vous créez un fichier sur le bureau, le fichier n'est pas conservé sur ce dernier, sinon il serait perdu un fois l'ordinateur éteint. Mais où, dans l'arborescence des répertoires, ce fichier est-il sauvé ? Il y a un moyen très simple de le localiser, saurez-vous le trouver ?

Même question pour les fichiers se trouvant dans le répertoire *Mes documents*.

### 1.4.4 Compréhension de l'arborescence

Observez l'arborescence de votre disque dur. Commentez sa structure (où se trouve le système d'exploitation, où se trouvent les applications, où se trouvent les données des utilisateurs, ...).

### 1.4.5 Première prise de contact avec une ligne de commande

Lancez une *Invite de commandes* [2]. Dans une telle fenêtre, la souris ne sert plus beaucoup, il vous faudra interagir avec le clavier.

Sous Windows 2000, la commande `help` sera votre précieuse alliée. Tapez simplement `help` puis validez pour obtenir la liste des commandes disponibles. Tapez `help [commande]` [3] pour obtenir des informations d'aide sur la commande `commande`.

La commande `dir` permet d'afficher la liste des fichiers du répertoire dans lequel vous vous trouvez. Quelque soit le sous-répertoire où vous vous trouvez, il y a au moins deux fichiers : « `.` » et « `..` ». « `.` » désigne le répertoire courant (le répertoire où vous vous trouvez). « `..` » désigne le répertoire parent. Pour changer de répertoire, utilisez la commande `cd`. Apprenez à vous déplacer dans l'arborescence.

Généralement, on installe toute les applications dans le répertoire `\Program Files`. Cependant ce répertoire est composé d'un nom long avec un espace. Autrefois la norme n'acceptait ni les noms de plus de 8 lettres, ni les espaces dans les noms. Aussi, par soucis de compatibilité avec les anciennes applications, chaque fichier possède également un nom court. Imaginez que vous vouliez installer une application, n'acceptant pas les noms longs, dans `\Program Files`. Il vous faut alors obtenir le nom court du répertoire `Program Files` pour pouvoir spécifier à l'application son répertoire d'installation. Dans une *Invite de commandes*, la commande `dir` vous permet justement d'afficher le nom court des fichiers. Utilisez `help dir` pour trouver comment. Quel est le nom court du répertoire `Program Files` ?

2. *Invite de commandes* sous Windows 2000 et *Commandes MS-DOS* sous Windows 98

3. Pour obtenir de l'aide en ligne sur une commande particulière, vous disposez de deux méthodes sous Windows 2000. Vous pouvez indiquer le nom de la commande sur la ligne `help` ou taper ce nom suivi du commutateur `/?` à l'invite de commandes. Pour vous renseigner sur la commande `dir`, par exemple, vous pouvez taper l'une ou l'autre des commandes suivantes : `help dir` ou `dir /?`. Sous Windows 98 la commande `help` n'existe pas. Pour obtenir l'aide en ligne sur une commande particulière il faut taper : `[commande] /?`.

### 1.4.6 Utilisation des caractères génériques sous Windows

Un caractère générique est un caractère du clavier, tel qu'un astérisque (`*`) ou un point d'interrogation (`?`), que vous pouvez utiliser pour représenter un ou plusieurs caractères véritables pour désigner des fichiers ou des dossiers. Les caractères génériques sont souvent utilisés à la place d'un ou plusieurs caractères quand vous ne savez pas quel est le véritable caractère ou que vous ne souhaitez pas taper le nom en entier.

Vous pouvez utiliser l'astérisque pour remplacer zéro ou plusieurs caractères. Si vous recherchez un fichier commençant par `gloss` mais dont vous ne vous souvenez plus la suite, tapez : `dir gloss*`. La commande `dir` dressera la liste de tous les fichiers, quel que soit leur type, commençant par `gloss`, y compris `Glossaire.txt`, `Glossaire.doc` et `Glossine.doc`. Pour limiter la recherche à un type de fichier spécifique, tapez : `dir gloss*.doc`. Dans ce cas, la commande `dir` dressera la liste tous les fichiers commençant par `gloss` mais dont l'extension est `.doc`, tels que `Glossaire.doc` et `Glossine.doc`.

Vous pouvez utiliser le point d'interrogation pour remplacer un seul caractère dans un nom. Par exemple, si vous avez tapé `dir gloss?.doc`, la commande `dir` peut afficher le fichier `Glosse.doc` ou `Gloss1.doc` mais pas `Glossaire.doc`.

Faites quelques essais pour vous familiariser avec ces deux caractères génériques.

# Chapitre 2

# Introduction au monde d'Unix

L'objectif de ce chapitre est de vous donner un premier aperçu du monde Unix. Cet aperçu dépasse celui d'une station de travail Linux comme celles que nous utiliserons en TP puisqu'il concerne le monde Unix tel que vous y seriez confronté au sein d'un réseau Unix de type professionnel. Nous parlerons des principes généraux de ce système. Nous apporterons également le vocabulaire minimal spécifique au monde Unix. Nous apprendrons enfin à nous connecter, à interagir avec quelques commandes de base, puis à nous déconnecter.

## 2.1 Historique

L'histoire d'Unix débute dans les années 60 et peut être résumée de la façon suivante.

**1966** les laboratoires Bell (un division d'AT&T) ont besoin, pour leur usage interne, d'un système d'exploitation pour le traitement de textes et le développement d'applications. Ken Thomson et son équipe sont chargés de ce travail. Ils développent au *Massachussets Institute of Technology* (MIT) un système baptisé *Multics*, acronyme de *MULTiplexed Information and Computing Service*.

**fin des années 60** L'objectif de *Multics* était d'offrir simultanément plusieurs services à un ensemble d'utilisateurs. Le résultat fut un gros système peu maniable. La direction de Bell décida de ne pas donner suite au projet *Multics* et rappela ses chercheurs.

**1969** Ken Thomson développa un petit système d'exploitation pour le PDP-7, un mini-ordinateur. Contrairement à *Multics*, le système de Thomson était plus petit, moins ambitieux et (du moins au début) utilisable par une seule personne à la fois. De plus, chaque partie du système était conçue pour effectuer une seule tâche, mais de la manière la plus propre possible. Désireux de lui donner un nom, Thomson compara sons système à *Multics* et, par opposition, appela sons système *Unics*, acronyme de *UNIplexed Information and Computing Service*. *Unics* fût ensuite rapidement changé en *Unix*. En d'autre termes, *Unix* est un calembour sur *Multics*.

**1970** Unix compte ses premiers utilisateurs (en dehors de ses créateurs).

**1972** Denis Ritchie réécrit entièrement *Unix* en *C*. Le *C* est un nouveau langage développé par Thompson et Ritchie pour augmenter la portabilité d'*Unix*. Ceci explique les liens profonds entre le langage *C* et *Unix*.

**1974** AT&T propose les 1$^{res}$ licences aux universités (en particulier, l'Université de Berkeley) ce qui apporta un enrichissement en extensions et en utilitaires variés à *Unix*. Cette date correspond au début de la popularité et de la diversité d'*Unix*.

**1978** AT&T présente à l'industrie les 1$^{res}$ versions commerciales.

**années 80** AT&T autorise le clonage d'Unix par d'autres constructeurs. Ainsi, apparaissent *Ultrix* sur DEC, *BSD* sur SUN, *AIX* sur IBM, etc. Ces versions constructeur dérivent toutes des 2 versions présentes à l'époque et qui sont :

- *System V* pour des configurations moyennes et petites ; USL (Unix Systems Labs, filiale d'AT&T) en est responsable actuellement ;
- *BSD* (Berkeley Software Distribution) pour des configurations importantes dans le domaine scientifique.

Une étape importante dans l'histoire d'*Unix* est également la création au début des années 80 de la Free Software Foundation (FSF) par Richard M. Stallman du MIT, dont l'objectif est de diffuser des logiciels libres (c'est à dire qui peuvent être librement adaptés, selon quelques règles de base toutefois, par leurs utilisateurs). L'idée est un peu de revenir à cet esprit de communauté qui a caractérisé les débuts d'Unix. Au sein de la FSF, il lance le projet GNU (GNU is Not Unix) qui consiste à développer un système d'exploitation dans l'esprit d'Unix, et distribué librement. Tous les logiciels développés dans le cadre de ce projet sont soumis à une licence particulière, la GPL (General Public License), qui autorise les utilisateurs du logiciel à avoir accès au code source du logiciel et à pouvoir le modifier et diffuser leurs modifications à condition que celles-ci soient elles-mêmes soumises à la licence GPL. Une grande partie des utilitaires d'Unix ont été réécrits sous cette licence et sont d'ailleurs généralement beaucoup plus puissants que leurs équivalents propriétaires.

**1991** Enfin, *Linux* en tant que tel apparaît en 1991 lorsque Linus Torvalds, un étudiant finlandais, décide d'optimiser un dérivé d'*Unix* pour PC, *Minix* (développé par Andrew Tanenbaum à l'Université d'Amsterdam). Rapidement, il réécrit une bonne partie du système et diffuse cette première version, *Linux 0.01* (qui tourne encore sous *Minix*), sur Internet. Un groupe de programmeurs s'intéresse alors à cet embryon de système et propose de nombreuses améliorations. Ainsi dès la fin de l'année 1991, *Linux* (version 0.99) est maintenant un système d'exploitation à part entière. *Linux* est distribué sous la licence GPL, ce qui fait que de nombreux programmeurs peuvent améliorer le système et redistribuer cette version améliorée que d'autres programmeurs pourront à nouveau améliorer, etc ...

**1991** Ainsi, la première version stable, *Linux 1.0*, sort en 1994. Suite aux diverses améliorations, nous en sommes maintenant à la version 2.2, et il évolue encore ...

## 2.2 Principes généraux

Une des raisons de la popularité d'Unix est que l'ordinateur sous Unix peut être relié à tout autre ordinateur fonctionnant avec le même système d'exploitation. Le résultat est un énorme réseau d'ordinateurs et une communauté Unix dispersée dans le monde entier.

### 2.2.1 Hôtes et terminaux

Unix est un système d'exploitation multi-utilisateurs, ce qui signifie que plusieurs utilisateurs peuvent être simultanément connectés à l'ordinateur. Cet ordinateur qui fait tout le travail s'appelle l'*hôte*. Une des tâches du système d'exploitation est de vérifier que les ressources de l'hôte sont correctement partagées entre les utilisateurs.

Pour travailler sous Unix, vous devez utiliser un *terminal*. Un terminal Unix comprend un moniteur, un clavier, et éventuellement une souris. Votre terminal est connecté physiquement à l'hôte. Quand vous entrez des caractères, ou que vous déplacez la souris, des signaux sont envoyés à l'hôte. Un programme, exécuté sur l'hôte, interprète ces signaux et réagit de manière appropriée.

Quand un programme en cours d'exécution sur la machine hôte doit afficher des résultats, il envoie un signal au terminal qui se charge alors d'imprimer sur l'écran les informations adéquates. Tous les systèmes Unix fonctionnent de cette manière.

### 2.2.2 Que se passe-t-il quand une touche est pressée ?

Chaque fois que vous pressez une touche du clavier, un signal est envoyé à l'hôte. L'hôte répond en renvoyant un signal vers le terminal lui indiquant d'afficher le caractère approprié sur l'écran. Ainsi, si vous entrez la commande `date` (pour afficher la date et l'heure), votre écran affiche les lettres `d a t e`. Le terminal n'affiche cependant pas de lettre tant que l'hôte ne lui a pas dit de le faire.

A première vue cela peut paraître étrange. Quand vous pressez la touche `d`, le caractère `d` n'est pas affiché sur l'écran. Ce qui se passe en réalité est que le signal `d` est envoyé vers l'hôte, qui à son tour envoie un signal vers le terminal, signifiant à celui-ci d'afficher le bon caractère. Nous dirons que l'hôte fait un écho du caractère frappé sur votre écran. Dans la plupart des cas, cet échange de signaux se fait si rapidement que cela peut vous donner l'impression que le clavier est directement relié au moniteur.

Il est cependant également possible de se connecter à un ordinateur hôte se trouvant à des milliers de kilomètres. Dans de tels cas, il pourra arriver qu'un certain laps de temps s'écoule entre le moment où vous frapperez une touche du clavier et celui où le caractère correspondant s'affichera sur l'écran. Autrement dit, vous presserez des touches, mais vous ne verrez pas apparaître les caractères correspondants immédiatement à l'écran. C'est se qui se passe en général quand l'ordinateur auquel vous vous connectez est éloigné, ou quand la ligne de communication est lente.

### 2.2.3 L'administrateur système

Tous les systèmes Unix doivent être gérés et maintenus. La personne qui s'acquitte de ces charges se nomme l'*ingénieur système*, l'*administrateur système* ou encore le *root*. Dans les universités, ou dans les entreprises, l'ingénieur système est un employé au même titre que les autres. Maintenir un système Unix entier n'est pas une activité de tout repos. Dans le monde Unix, l'administration du système est souvent considérée, et c'est un euphémisme, comme une « activité non triviale ». En d'autres termes installer correctement et garder un système Unix en état de marche est un gros travail.

Quand vous vous faites enregistrer comme utilisateur d'un système Unix, l'administrateur système vous donne un nom qui vous identifie vis-à-vis du système. Ce nom est le

*userid* ou *identifiant d'utilisateur*. En même temps que ce nom, vous recevrez un *mot de passe*. Le mot de passe est un code secret que vous devrez entrer à chaque fois que vous vous connecterez au système.

### 2.2.4 La console

Presque tous les ordinateurs disposent d'un clavier et d'un écran. Pour Unix, ce clavier et cet écran ne sont rien d'autre qu'un terminal, mais doté d'un nom spécial, la *console*. La console fait partie de l'ordinateur, alors que les autres terminaux sont indépendants et doivent être connectés.

Un système Unix typique consiste en un ordinateur hôte disposé dans le bureau de l'administrateur du système. Cet ordinateur est connecté à une salle remplie de terminaux. Par commodité, l'administrateur système utilise la console, c'est-à-dire le clavier et l'écran intégrés de l'ordinateur, pour effectuer son travail, alors que tous les autres utilisateurs n'ont affaire qu'à des terminaux. Cependant, l'administrateur peut très bien utiliser un terminal pour faire son travail.

Vous pouvez alors légitimement vous demander si une console est réellement nécessaire à Unix? Et bien pas du tout. Il existe des ordinateurs sans clavier ni écran. L'administrateur du système utilise un terminal classique . L'ordinateur, qui n'est alors qu'une boîte, peut très bien être enfermé dans une armoire quelconque.

### 2.2.5 Station de travail

En principe, tous les systèmes Unix supportent plusieurs utilisateurs simultanés, mais il existe des ordinateurs Unix qui ne sont utilisés que par une seule personne à la fois, les *stations de travail*.

Une station de travail n'est connectée qu'à un seul terminal, la console. Ce qui signifie que si vous utilisez une station de travail, vous disposez de l'ordinateur entièrement pour votre usage privé. C'est le cas si vous possédez un ordinateur entièrement personnel fonctionnant sous Unix (sous Linux s'il s'agit d'un PC). Si vous disposez de votre propre système Unix et que vous êtes le seul à l'utiliser, vous en serez aussi l'ingénieur système, avec tous les inconvénients que cela comporte.

Il est également possible de mettre des stations de travail en réseau.

### 2.2.6 Relation Client-Serveur

Une des principales raisons à l'origine de l'existence des réseaux est le besoin de partager les ressources. Supposons qu'un ordinateur dispose d'un grand espace disque pour stocker des fichiers. Grâce au réseau, cet espace peut être partagé. Il se peut ainsi très bien que vos fichiers personnels ne soient pas sauvegardés sur votre ordinateur mais sur un autre. Quand vos programmes doivent accéder à ces fichiers, les données sont transmises à votre ordinateur par le réseau. De tels systèmes sont fréquents dans les grandes structures informatiques.

Dans la terminologie des réseaux, un programme qui offre une ressource s'appelle un *serveur*. Un programme qui utilise une ressource se nomme un *client*. Ainsi, le programme

qui permet l'accès à des fichiers à travers le réseau s'appelle un serveur de fichiers, celui qui contrôle l'accès à l'imprimante le serveur d'imprimante, et ainsi de suite.

Parfois, le nom *serveur* fait référence à l'ordinateur lui-même et non plus à un programme. C'est le cas quand un ordinateur est dédié à une tâche particulière. On désignera ainsi *serveur de fichier* l'ordinateur dédié à la gestion des fichiers sur lequel de gros disques durs sont connectés. On parlera de *serveur de courrier électronique* pour désigner un ordinateur particulier du réseau qui s'occupe de la gestion des courriers électroniques. On parlera de *serveur de calcul* pour désigner un ordinateur très puissant en terme de puissance de calcul (un ou plusieurs processeurs très rapides, mémoire très importante, etc.) dont la tâche est d'effectuer les traitements dont la complexité est trop importante pour être réalisés dans un intervalle de temps raisonnable sur un hôte classique.

## 2.3 Système X Window

Si vous avez déjà travaillé sur Macintosh, ou sur PC avec Windows, vous savez ce qu'est une *interface utilisateur graphique* (en anglais *Graphical User Interface* ou GUI). C'est un système avec lequel vous utilisez non seulement un clavier mais aussi une souris. Votre écran ne se borne pas à afficher des caractères, mais peut aussi afficher des boîtes (les fenêtres) et des images. Vous travaillez en manipulant les boîtes et les images comme des objets.

X Window est un système conçu pour supporter diverses Interfaces Utilisateur Graphiques. Pour des raisons de commodité, on fait souvent référence au système X Window par le seul biais de la lettre X.

Le système X Window provient originellement d'un système d'exploitation particulier conçu à l'université de Stanford. Ce système s'appelait *V*. Quand une interface à base de fenêtres fut développée pour ce système, elle prit le nom de *W*. Cette interface fut ensuite donnée à une personne du MIT qui développa un nouveau système graphique et lui donna le nom de *X*. Ce nom de X est depuis resté.

L'idée qui se cache derrière X est de proposer un ensemble de services standards permettant l'affichage de données graphiques. X est actuellement maintenu par une organisation indépendante nommée X Consortium.

L'interface utilisateur graphique n'est pas directement fournie par le système X lui-même, mais par un programme appelé *gestionnaire de fenêtres*. Ce gestionnaire contrôle l'apparence ainsi que le comportement des fenêtres et des images. X supporte plusieurs gestionnaires de fenêtres, sous Linux, les plus courants sont KDE et Gnome.

X est un système graphique conçu pour ne supporter que les écrans graphiques, mais la plupart du temps votre travail consistera à entrer des commandes Unix, l'une après l'autre, en réponse au *prompt* du *shell*. Pour cela, vous n'avez besoin que d'un simple terminal caractère. Ce besoin étant fondamental, X Window comprend un client X dont la seule fonction est d'émuler un terminal dans une fenêtre. Ce programme s'appelle `xterm`. Quand vous exécuterez `xterm`, cela fera apparaître une petite fenêtre, version réduite d'un terminal caractère.

## 2.4 Procédure de *login*

On suppose ici, que l'utilisateur a déjà été enregistré sur le système et que donc, son identité est présente dans des fichiers particuliers gérés par l'administrateur du système.

### 2.4.1 Se *loguer*

L'utilisateur va devoir s'identifier auprès du système. Cette identification se fait en deux étapes. L'utilisateur est dans premier temps confronté à l'affichage du message `Login :` après lequel il faut rentrer son identifiant d'utilisateur. Ensuite le message `Password :` est affiché, il faut alors rentrer son mot de passe. Celui-ci n'est pas affiché pendant la frappe pour éviter bien sûr que quelqu'un d'autre puisse l'apercevoir.

L'utilisateur est effectivement prêt à travailler quand il reçoit l'invite du système consistant en un marqueur en début de ligne. Ce marqueur est variable selon les machines (ex : `$` ou `nom_utilisateur@nom_machine>`)

### 2.4.2 Mot de passe

Le changement de mot de passe s'effectue avec la commande `passwd` (ou `yppasswd`). L'utilisation de l'une ou l'autre commande dépend du type de configuration du système d'enregistrement des utilisateurs d'un système Unix. Si ceux-ci sont les utilisateurs locaux d'une machine donnée, ils doivent utiliser la commande `passwd`.

Le changement de mot de passe s'effectue en entrant d'abord le mot de passe actuel puis en entrant le nouveau mot de passe que l'on doit retaper pour confirmation.

```
perrot@von-neumann>yppasswd
Changing NIS password for perrot on von-neumann.
Old password:
New password:
Retype new password:
NIS entry changed on von-neumann
```

Si le nouveau mot de passe est rejeté, la raison peut être que :

- la 2$^{e}$ entrée du nouveau mot de passe (pour confirmation) était différente de la première ;
- le nouveau mot de passe ne répond pas à des exigences de sécurité (voir ci-dessous).

Des personnes peuvent essayer d'accéder de façon illégale à un système Unix. Un des moyens dont ils disposent est d'entrer sur un compte utilisateur en déterminant son mot de passe. Pour cela, ils utilisent des chaînes de caractères appartenant à des dictionnaires ou correspondant à des informations personnelles sur l'utilisateur (prénom, nom, numéro de téléphone, etc.). Il est donc nécessaire que votre mot de passe respecte certaines règles de sécurité (ces règles sont valables pour tout mot de passe informatique) :

- il doit posséder au moins 7 caractères[1] et contenir au moins une lettre majuscule, un chiffre et un caractère de ponctuation, et ceux-ci à l'intérieur et non en début ou fin de mot de passe ;

1. Généralement, seuls les 8 premiers caractères sont pris en compte.

- il ne doit pas contenir des données relatives à votre identité comme votre nom d'utilisateur ou une information livrée par la commande `finger` ;
- il ne doit pas appartenir à des dictionnaires, tel quel ou sous sa forme canonique (c'est à dire, épuré de tous les caractères non-alphabétiques), à moins qu'il contienne des majuscules autres que le premier caractère ;
- il ne doit pas contenir des répétitions de caractère ;
- il doit être suffisamment simple pour s'en rappeler ;
- il ne faut pas le noter sur papier ou dans un fichier ni le donner à quelqu'un d'autre.

### 2.4.3 Se *déloguer*

Cette opération dépend du type de session qui a été ouverte. Si un environnement graphique (tel que celui créé par X Window, l'environnement multi-fenêtrage) est en place, il existe généralement un menu *logout* (ou *exit*) qui permet de quitter cet environnement et ainsi de terminer la session de travail. Parfois aussi, ce menu ne permet simplement que de quitter l'environnement graphique. Il faut ensuite procéder à l'étape de déconnexion ci- dessous.

En l'absence d'un environnement graphique, une simple commande telle que `logout` ou `exit` , entrée après l'invite du système, suffit pour terminer la session de travail.

## 2.5 Interagir avec le *shell* : les commandes

### 2.5.1 Interpréteur de commandes, ou *shell*

L'utilisateur peut taper des commandes lorsqu'il se trouve au niveau commande, c'est-à-dire lorsque l'invite du système apparaît. Toute commande entrée sera interprétée par l'*interpréteur de commandes* (ou *shell*).

Le terme shell veut dire coquille pour exprimer l'idée d'interface entre utilisateurs et système Unix et a été donné par opposition au noyau du système. Il existe des dizaines d'interpréteurs de commandes sous Unix mais les 2 principaux (qu'on retrouve sur la plupart des systèmes) sont le *Bourne-shell* (sh) et le *C-shell* (csh). Le choix de l'interpréteur activé à la connexion est fait à l'enregistrement de l'utilisateur dans le système. Dans la plupart des cas, c'est le C-shell.

### 2.5.2 Syntaxe des commandes

Celle-ci est généralement la suivante :

```
nom_commande [options] [arguments...]
```

Le caractère séparateur entre les différents éléments de la commande est le blanc (ESPACE).

Les options commencent habituellement par le caractère − (signe moins) suivi d'une ou plusieurs lettres-clés. Ces options vont modifier le comportement de la commande.

Les arguments spécifient les objets (fichiers ou variables) sur lesquels la commande va s'appliquer.

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

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

Il faut noter que tous les shells font la distinction entre les lettres minuscules et majuscules pour les commandes et les noms de fichiers contrairement au MS-DOS ou à Windows.

### 2.5.3 Complètement d'une ligne de commande

Le complètement est une fonctionnalité des plus pratiques et tous les *shell* modernes l'incluent désormais. Son but est de compléter la ligne de commande commencée par l'utilisateur. La demande de complètement se fait en pressant la touche <TAB>. Le *shell* ne peut compléter une ligne de commande que si celle-ci est non ambiguë. S'il y à ambiguïté, le complètement ne se fera que sur la partie non ambiguë si elle existe.

### 2.5.4 Manuel des commandes

Si on veut de l'aide sur les règles d'utilisation ou encore sur les fonctionnalités d'une commande, on peut utiliser l'aide en ligne grâce à la commande `man` (comme *MANual* pages) de la façon suivante :

```
man <nom_commande>
```

L'affichage des *pages du manuel* se fait à l'aide de la commande `more`, afficheur page par page dont on verra l'utilisation plus loin. Pour faire avancer l'affichage, il suffit de taper la barre d'espace, pour quitter l'affichage des *pages du manuel* il faut taper la lettre `q`.

Si on ne connaît pas la syntaxe de la commande, il est possible de faire une recherche par mot-clé dans le système des *pages du manuel*) à l'aide de la commande suivante :

```
man -k <mot_clé>
```

### 2.5.5 Caractères génériques

Certaines commandes acceptent plusieurs noms de fichiers en arguments aussi, il était intéressant d'avoir des notations permettant de raccourcir l'écriture d'une telle liste. Ainsi, il existe plusieurs caractères génériques qui, incorporés dans les noms de fichiers, ont la signification suivante :

- le caractère ? qui peut remplacer n'importe quel caractère ;
- le caractère $*$ (astérisque) qui peut remplacer n'importe quelle chaîne de caractères, y compris la chaîne vide.

Pour plus de renseignement, se référer à la section 4.5 sur les motifs d'englobement.

## 2.6 Quelques commandes

Nous avons déjà parlé des commandes `passwd`, `date`, `exit`, `logout` et `man`. Nous allons ici voir quelques nouvelles commandes qui nous seront utiles lors de la séance de TP.

### 2.6.1 La commande `who`

La commande `who` affiche la liste des utilisateurs connectés sur la machine où l'on travaille. Le nom d'utilisateur, le nom du terminal utilisé et la date de début de session de travail sont fournis.

```
perrot@vonneumann>who
Lartis ttyp1 Oct 23 22:25 (mistral.ERE.UMon)
perrot ttyp3 Oct 24 08:52 (SanA.grbb.polym)
```

Une utilisation pratique de la commande `who` est la commande `who am i` qui permet de connaître le nom de l'utilisateur connecté à partir d'un terminal et qui l'a quitté.

### 2.6.2 La commande `ls`

Pour énumérer l'ensemble des fichiers contenus dans un répertoire, utilisez la commande `ls` (liste). C'est l'une des commandes les plus utilisées, pour cette raison, elle dispose de beaucoup d'options contrôlant les résultats produits (cf. section 3.5 pour un complément sur le formatage des informations affichées). La syntaxe de cette commande est la suivante :

```
ls [options] [fichier...]
```

Voici quelques-unes des options les plus utiles :

**-a** affiche également les fichiers cachés (dont le nom commence par «.») ;

**-F** cette option permet de suffixer les noms de fichier avec un caractère d'identification ; les nom de répertoires sons suivis d'un /, les noms de fichiers ordinaires faisant référence à des programmes exécutables sont suivis d'une astérisque $*$, les autres types de fichiers ordinaires ne sont pas marqués ;

**-i** affiche le numéro d'i-nœud en face du fichier ;

**-l** affiche une information détaillée sur chacun des fichiers ;

**-R** fait une liste récursivement, par exemple, tous les fichiers et sous-répertoires des répertoires mentionnés sur la ligne de commande ;

**-s** affiche la taille en kilo-octets à côté de chaque fichier ;

### 2.6.3 La commande `cd`

Pour changer de répertoire de travail servez-vous de la commande `cd` (en anglais : change directory) dont la syntaxe est :

```
cd [rep]
```

où `rep` est le chemin dans lequel on désire aller.

Si vous entrez la commande sans chemin, `cd` vous positionnera par défaut dans votre répertoire de travail.

### 2.6.4 La commande `pwd`

Pour afficher le chemin complet du répertoire courant utilisez la commande `pwd`.

## 2.7 TP premier pas sous Linux

Cette première séance de TP sous Linux est plus ou moins libre.

1. Connectez vous au système Linux en vous loguant.
2. Par défaut, vous vous retrouverez sous X-Windows avec le gestionnaire de fenêtre KDE. Vous pouvez explorer ce nouvel environnement graphique.
3. Cependant, l'objectif de ce chapitre n'est pas de vous familiariser avec un nouvelle environnement graphique, l'objectif est de vous familiariser avec la ligne de commande du shell. Durant ce TP, vous devrez donc exécuter une *xterm* pour vous familiariser avec la ligne de commande et avec l'utilisation des commandes que nous avons abordées dans ce chapitre.

# Chapitre 3

# Système de fichier d'Unix

Les système de fichier est l'un des domaines où Unix diffère totalement de Windows et de la plupart des autres systèmes d'exploitation.

Le but de ce chapitre est de découvrir la plupart des particularités du système de fichier sous Unix/Linux. Ce chapitre nous emportera également dans l'exploration de l'arborescence des fichiers sous d'une installation classique de Linux.

## 3.1 Remarque sur les noms de fichier

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[1].

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. Cependant, ces extensions sont très pratiques.

Le caractère point (`.`) est un caractère comme les autres sous Unix. Les noms de fichiers commençant avec un point sous Unix sont des fichiers cachés.

## 3.2 Permission de fichiers

Chaque fichier est la propriété exclusive d'un utilisateur et du groupe auquel l'utilisateur appartient. 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 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.

Même si le concept de groupe est esthétique, sachez qu'ils sont très peu employés en pratique. Les ingénieurs système considèrent comme superflue la gestion de petits groupes d'utilisateurs. Vous pouvez donc ignorer la notion de groupe de travail en affectant à vos

1. Minuscule vs. majuscule.

fichiers les mêmes permissions à la catégorie groupe que celles que vous donnez au reste des utilisateurs.

On distingue trois types de droits :

1. Les **droits 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. Les **droits d'écriture** (`w` pour *Write* ou écriture). 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 desdits fichiers.
3. Les **droit d'exécution** (`x` pour *eXecute*ou executer). 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 !

Voir la section 4.4 pour la manipulation des permissions de fichier.

## 3.3 Différents types de fichier Unix

Sous Unix, la définition d'un fichier est élargie par rapport à la définition que nous avons vu section 1.2 : un fichier est une source de données qui peuvent être lues, ou une destination dans laquelle les données peuvent être écrites. Ainsi, le terme *fichier* fait référence non seulement à un dépôt de données comme un fichier disque, mais aussi à tout dispositif physique. En particulier, le clavier est un fichier (un fichier source), l'écran est un fichier (un fichier destination), ainsi que l'imprimante (également un fichier destination).

**Définition 3.1 -Fichier Unix-** *Sous Unix, le terme* fichier *fait référence à toute source (entrée) ou toute destination (sortie), et pas seulement à un dépôt de données.*

Ainsi, sous Unix, tout est fichier. Tout veut vraiment dire tout : un disque dur, une partition sur ce disque dur, un port parallèle, une connexion à un site Web, une carte Ethernet, un répertoire, etc. On peut donc différentier différents types de fichiers.

Il existe trois grands type de fichiers Unix : les fichiers ordinaires, les répertoires et les fichiers spéciaux. Le mot *fichier* est surtout employé pour faire référence aux fichiers ordinaires. Quand vous voyez ou entendez le mot *fichier*, vous devez déterminer sa signification en fonction du contexte.

### 3.3.1 Fichier ordinaire

Un *fichier ordinaire* est ce que la plupart des gens imaginent quand ils emploient le mot *fichier*. Les fichiers ordinaires contiennent des données et sont stockés sur disque ou, plus rarement, sur bande.

Dans la sortie de la commande `ls -l`, le caractère qui apparaît avant l'énoncé des droits d'accès représente le type du fichier. Sur la sortie d'un `ls -l` les fichiers ordinaires sont identifiés par le caractère `-`.

### 3.3.2 Répertoire

Un autre type de fichier est le *répertoire*. Un répertoire est stocké sur le disque et contient des informations qui sont utilisées pour organiser et accéder à d'autres fichiers. Un répertoire peut pointer vers d'autres répertoires. Cela vous permet d'organiser vos fichiers en un système hiérarchique dont vous pouvez spécifier la structure selon vos besoins.

Sur la sortie d'un `ls -l` les répertoires sont identifiés par la lettre `d`.

### 3.3.3 Fichiers spéciaux

Nous pouvons distinguer 5 types de fichiers spéciaux.

**Fichiers en mode caractère :** ces fichiers sont des fichiers spéciaux du système ou des périphériques (port série ou parallèle par exemple) dont le contenu (s'il existe) n'est pas conservé en mémoire. Sur la sortie d'un `ls -l` ces fichiers sont identifiés par la lettre `c`.

**Fichiers en mode bloc :** ces fichiers désignent des périphériques sauvegardé en mémoire (disque durs, partitions sur ces disques durs, lecteurs de disquettes, de CD-ROM, etc.). Sur la sortie d'un `ls -l` ces fichiers sont identifiés par la lettre `b`.

**Liens symboliques :** leur raison d'être est de lier symboliquement d'autres fichiers, c'est-à-dire qu'ils pointent (ou non) sur un fichier existant. Sur la sortie d'un `ls -l` ces fichiers sont identifiés par la lettre `l`.

**Tubes nommés :** ils sont très similaires aux tubes utilisés par le shell, mais avec la particularité que ceux-ci ont un nom. Sur la sortie d'un `ls -l` ces fichiers sont identifiés par la lettre `p`.

**Sockets :** ce type de fichier est celui de toutes les connexions réseau. Mais seules quelques unes d'entre elles portent des noms. Sur la sortie d'un `ls -l` ces fichiers sont identifiés par la lettre `s`.

## 3.4 I-nœuds et liens

Les *i-nœuds* (*inode* en anglais) sont, avec le paradigme **tout est fichier**, les fondements de tout système Unix. Le mot i-nœud est l'abréviation de *nœud d'index*. Les i-nœuds existent pour n'importe quel type de fichier susceptible d'être stocké sur un système de fichier d'où la fameuse phrase **l'i-nœud est le fichier**.

Quand Unix crée un fichier, il fait deux choses. Il détermine tout d'abord un espace sur le disque pour stocker les données dans le fichier, et crée une structure appelé un i-nœud destinée à contenir les informations fondamentales sur le fichier.

L'i-nœud contient toutes les informations dont Unix a besoin pour manipuler le fichier (en tant qu'utilisateur, il est inutile de connaître la structure réelle d'un i-nœud) :

- le nom du propriétaire qui possède le fichier ;
- le type du fichier (ordinaire, répertoire, ...) ;
- la taille du fichier ;
- où les données sont stockées ;
- les permissions du fichier ;

- la dernière date de modification du fichier ;
- la dernière date d'accès au fichier ;
- la dernière date de modification attributs de l'i-nœud ;
- le nombre de liens sur le fichier.

Unix conserve tous les i-nœuds dans une grande table. Dans cette table, chaque i-nœud est identifié par un nombre appelé *i-numéro* (*inumber* en anglais) ou numéro d'index. Nous dirons par exemple qu'un fichier particulier est décrit par l'i-nœud 24, ou qu'il à l'i-nombre 24.

Quand nous travaillons avec des répertoires, nous considérons qu'ils contiennent réellement des fichiers. en réalité, le répertoire ne contient pas réellement des fichiers, mais seulement leurs noms et leurs numéro d'index.

Ainsi, sous Unix, **on n'identifie pas un fichier par son nom**, mais par son i-numéro. Un 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*.

## 3.5 Compléments sur la commande `ls`

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

```
$ ls -lFi
total 1
351206227 -rw-r--r--  1 Laurent  users   5107 Sep  5 13:33 un_fichier
343610197 drwxr-xr-x  3 Laurent  users      0 Sep  5 13:16 un_repertoire/
$
```

Les différents champs de sortie de cette commande sont les suivants (de gauche à droite) :

- le premier champ de la sortie correspond au i-numéro, ou au numéro d'i-nœud du fichier ;
- le deuxième champ est constitué de 10 caractères qui désignent successivement le type du fichier et les droits qui lui sont associé ; le premier caractère désigne le type de fichier (c'est un tiret `-` s'il s'agit d'un fichier ordinaire et un `d` si le fichier est un répertoire) ; 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 autre que le propriétaire, les trois derniers aux autres utilisateurs ; un tiret (`-`) signifie ici que le droit n'est pas octroyé ; voir la section 4.4 pour la manipulation des permissions de fichier ;
- le 3$^{e}$ champ correspond au nombre de liens du fichier ;
- viennent ensuite le nom de l'utilisateur propriétaire du fichier et le nom du groupe propriétaire ;

– ensuite 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 suivit d'un `/` s'il s'agit d'un répertoire.

## 3.6 Visite guidée 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 système de fichiers), et il en est actuellement à sa version 2.2.

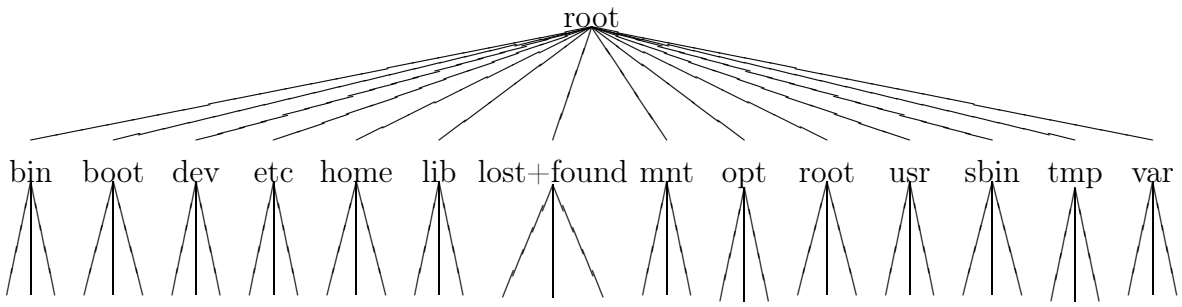


FIG. 3.1 – *Le système de fichier de Linux.*

Le répertoire racine (`root`) est à la base du système de fichier. Il contient surtout d'autres répertoires, mais peut contenir un fichier important : le programme du noyau d'Unix.

– Le répertoire `/bin` contient les programmes fondamentaux qui rendent Unix utilisable. C'est dans ce répertoire que la plupart des commandes Unix sont stockées. Le nom *bin* fait référence au fait que ces programmes sont principalement des fichiers binaires.
– Le répertoire `/boot` contient les fichiers nécessaires au gestionnaire de démarrage d'Unix. Cela peut ou non comprendre le noyau : s'il n'est pas ici, il doit être situé dans la racine.
– Le répertoire `/dev` contient les fichiers spéciaux qui représentent les dispositifs physiques (`dev` pour *DEVices*, périphérique). Vous n'aurez pas habituellement à employer ces fichiers, mais sachez qu'ils se trouvent là si vous en avez besoin.
– Le répertoire `/etc` est destiné à l'administrateur du système. Il contient les programmes employés pour gérer le système. Le fichier le plus connu de ce répertoire est `passwd`, le fichier des mots de passe.
– Le répertoire `/home` contient tous les répertoires personnels des utilisateurs du système.

- Le répertoire `/lib` contient les bibliothèques essentielles au système ainsi que les modules du noyau.
- Le répertoire `/lost+found` est utilisé par un programme particulier chargé de vérifier le système de fichier d'Unix. Ce programme permet de trouver les fichiers qui n'appartiennent à aucun répertoire (les accidents arrivent, même sur les meilleurs systèmes) et met les résultats dans de répertoire.
- Le répertoire `/mnt` contient les points de montage pour les fichiers temporaires tels que les CD-ROMs ou les disquettes.
- Le répertoire `/opt` contient les paquetages non nécessaires au fonctionnement du système.
- Le répertoire `/root` est le répertoire personnel de `root`, l'administrateur système.
- Le répertoire `/usr` est l'un des plus importants sous-répertoires du répertoire racine. Il contient lui-même un nombre important de sous-répertoires. Le nom *usr* indique que certains de ces sous-répertoires sont consacrés aux utilisateurs, afin de contenir leurs fichiers personnels.
- Le répertoire `/sbin` contient les binaires essentiels au démarrage du système, exploitables par `root`.
- Le répertoire `/tmp` est employé pour le stockage temporaire de fichiers. N'importe qui peut stocker des fichiers dans ce répertoire, ce qui fait que, de temps à autre, tous les fichiers dans ce répertoire sont détruits. Pratiquement, un programme se sert du répertoire `/tmp` pour stocker des fichiers nécessaires que pendant un court instant (fichiers intermédiaires de compilation par exemple).
- Le répertoire `/var` contient les données souvent modifiées par des programmes.

## 3.7 TP i-nœud, liens et arborescence

### 3.7.1 Créer un fichier : `touch`

Dans la section 3.5 nous avons analysé le résultat d'une commande `ls` avec, entre autre, l'option `-l`. Nous avons vu que, pour chaque fichier, sont affichés l'heure et la date de la dernière modification. La fonction de la commande `touch` est de remplacer cette date de modification par la date actuelle. Sur certains systèmes, `touch` vous permet de préciser la date et l'heure que vous voulez.

Si le fichier que vous précisez n'existe pas, la commande `touch` le créera. Ainsi, la commande `touch` est aussi utile pour créer un ensemble de fichiers temporaires destinés à expérimenter des commandes portant sur les fichiers Unix.

Par exemple, pour créer un fichier nommé `fichier`, faites :

```
$ touch fichier
```

### 3.7.2 Effacer un fichier : `rm`

Pour supprimer un fichier, il faut utiliser la commande `rm`. La syntaxe est : `rm <fichier>`. Profitez-en pour effacer les fichiers créés pour essayer la commande `touch`.

### 3.7.3 Créer un lien : `ln`

La commande `ln` permet d'ajouter un lien à un fichier ordinaire.

Syntaxe :

```
ln [options] <source> [dest]
```

où `source` est le nom d'un fichier ordinaire existant, et `dest` est le nom que vous voulez donner au lien. L'exécution de cette commande vous permet de récupérer deux noms de fichiers qui font référence au même fichier.

### 3.7.4 I-nœud et liens

Pour mieux comprendre ce que cache les notions d'i-nœud et de lien, passons à une illustration.

Créez à l'aide de la commande `touch` un fichier ordinaire `un_fichier`.

Affichez les informations relatives à ce fichier à l'aide de la commande `ls -il un_fichier`. Vous obtiendrez quelque chose du style :

```
$ pwd
/home/Laurent
$ touch un_fichier
$ ls -il un_fichier
17689 -rw-r--r--    1 Laurent  Aucun       0 Sep 23 14:23 un_fichier
$
```

Les champs qui nous intéressent ici sont le premier (le numéro d'i-nœud, ici 17689) et le troisième (le compteur de liens pour le fichier, ici 1).

On peut séparer la commande `touch un_fichier` en deux actions distincts :

- création d'un i-nœud, auquel le système a attribué le numéro 17689, dont le type est celui d'un fichier ordinaire ;
- création d'un lien vers cet i-nœud, nommé `un_fichier`, dans le répertoire courant, `/home/Laurent` ; par conséquent, le fichier appelé `/home/Laurent/un_fichier` est un lien vers l'i-nœud de numéro 17689 ; il est pour l'instant le seul et le compteur de liens indique donc 1.

Créez un lien nommé `un_lien` vers le fichier `un_fichier`. Affichez les informations relatives à ces fichiers à l'aide de la commande `ls -il un_fichier un_lien`.

Vous obtiendrez quelque chose du style :

```
$ ln un_fichier un_lien
$ ls -il un_fichier un_lien
17689 -rw-r--r--    2 Laurent  Aucun       0 Sep 23 14:45 un_fichier
17689 -rw-r--r--    2 Laurent  Aucun       0 Sep 23 14:45 un_lien
$
```

Nous avons créé un autre lien vers le même i-nœud. Comme on peut le constater, aucun fichier nommé `un_lien` n'a été créé, mais ce qui a été ajouté est un autre lien vers l'i-nœud de numéro 17689, dans le même répertoire, nommé `un_lien`. La sortie de `ls -il un_fichier un_lien` nous indique ainsi que le compteur de liens est maintenant à 2 et non plus à 1.

Effacez le fichier `un_fichier` à l'aide de la commande `rm un_fichier`.

Qu'observez vous ?

Ainsi, un fichier sous Unix n'a pas de nom. A la place, il a un ou plusieurs *liens*, dans un ou plusieurs répertoires et un numéro d'*i-nœud* unique.

### 3.7.5 L'arborescence

Explorez l'arborescence d'Unix en utilisant les commandes déjà vues `cd`, `ls` et `pwd`. Apprenez à vous familiariser avec ces commandes et leur diverses options. N'hésiter pas à utiliser la commande `man` pour approfondir l'utilisation de ces commandes. N'hésitez pas non plus à utiliser les caractères génériques conjointement à la commande `ls`. Observez l'arborescence telle que nous l'avons décrite et familiarisez-vous avec.

# Chapitre 4

# Opérations sur les fichier et répertoires

L'objectif de ce chapitre est de vous familiariser avec les commandes de manipulation des fichiers et des répertoires.

## 4.1 Chemin absolu et chemin relatif

Le chemin absolu d'un nom de fichier est formé de tous les noms de répertoires traversés depuis la racine pour l'atteindre, noms séparés par des obliques avant `/` (contrairement au MS-DOS ou à Windows qui utilise des obliques arrière \). On dira donc que le nom d'un fichier débutera par le `/`. Le chemin absolu correspondant à un nom de fichier est donc unique.

Un chemin relatif commence avec le répertoire de travail. L'idée est d'établir votre répertoire de travail suffisamment judicieusement pour utiliser autant que possible des chemins relatifs plus court.

Prenons comme exemple d'application l'arborescence de la figure 4.1. Plaçons nous dans le répertoire `/root/home/laurent`. Pour effacer le fichier `text1.tex` dans `courrier` nous pouvons, au choix, utiliser le chemin complet du fichier

```
rm /root/home/laurent/courrier/text1.tex
```

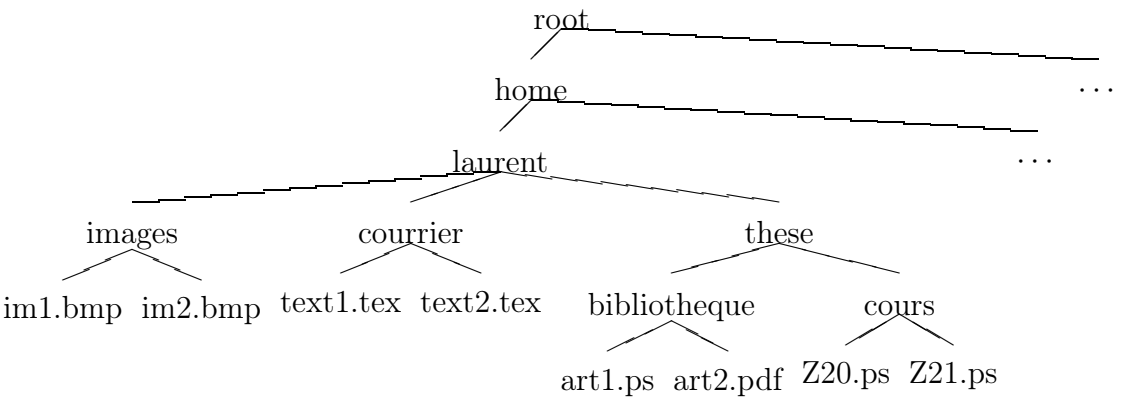


FIG. 4.1 – *Exemple classique d'arborescence d'un utilisateur.*

ou bien utiliser le nom relatif

```
rm courrier/text1.tex
```

beaucoup plus économique.

De plus, si le répertoire courant avait été `/root/home/laurent/courrier`, on aurait pu se contenter de saisir `rm text1.tex` pour effacer ce fichier.

## 4.2 Trois abréviations utiles : `..` `.` ~

Unix vous propose trois abréviations de chemin très utiles. La première est `..` qui est souvent appelé *point-point*. Dans un chemin d'accès, `..` fait référence au répertoire (*i.e.* est un lien vers le répertoire) parent de celui dans lequel vous êtes actuellement. Imaginons que l'on se trouve dans le répertoire `cours`, il est possible de désigner le fichier `art1.ps` du répertoire bibliothèque en utilisant le chemin absolu

```
/root/home/laurent/these/bibliotheque/art1.ps
```

ou le chemin relatif

```
../bibliotheque/art1.ps.
```

La seconde abréviation est `.` habituellement appelée *point*. `.` fait référence au répertoire (*i.e.* est un lien vers le répertoire) de travail lui-même. Par exemple, si on se trouve dans le répertoire `images` on peut faire référence au fichier `im1.bmp` en utilisant les trois chemins suivant :

```
/root/home/laurent/images/im1.bmp
./im1.bmp
im1.bmp
```

La troisième abréviation est ~ (tilde). Vous pouvez employer ce symbole comme synonyme de votre répertoire d'accueil. Par exemple, imaginez que vous vous trouviez dans le répertoire `cours` et que vous désiriez connaître le contenu de votre répertoire d'accueil `laurent`, vous pouvez indifféremment taper :

```
ls /root/home/laurent
ls ../..
ls ~
```

## 4.3 Manipulation des fichiers et répertoires

### Quelques conseils

Donnez à vos fichiers des noms qui signifient quelque chose pour vous. Quand vous ne vous êtes pas servi d'un fichier pendant un certain temps, le nom peut vous rappeler son contenu.

Quand vous nommez des fichiers ou des répertoires, n'utilisez que des lettres en minuscules.

### 4.3.1 Créer un fichier : `touch`

Nous avons déjà vu cette commande section 3.7.1.

### 4.3.2 Créer un répertoire : `mkdir`

La syntaxe est la suivante :

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

`mkdir` crée les répertoires demandés. L'option `-p` permet de créer des répertoires enfants (en créant également le répertoire parent si nécessaire).

Exemples :

- `mkdir yod` crée un répertoire du nom de `yod` dans le répertoire courant ;
- `mkdir -p yop/poy oye` crée un répertoire `poy` dans le répertoire `yop` après avoir créé ce dernier s'il n'existait pas ; crée également un répertoire `oye` dans le répertoire courant.

### 4.3.3 Copie de fichiers ou de répertoires : `cp`

La syntaxe est la suivante :

```
cp [options] <fichier|rép.> [fichier|rép...] <destination>
```

`cp` copie tous les fichiers contenus dans les différents `fichier|rép.` spécifié dans le répertoire `destination`.

L'option `-R` de copie récursive est obligatoire pour copier un répertoire, même vide.

Si `destination` est un nom de fichier, la syntaxe devient :

```
cp [options] <source> <destination>
```

Exemples :

- `cp toto titi` copie le fichier `toto` sous le nom `titi` dans le répertoire courant ;
- `cp ~/courrier/* ~` copie tous les fichiers du répertoire `~/courrier` dans le répertoire d'accueil.
- `cp -R ~/these ~/svg` copie tous les fichiers du répertoire `~/these` dans le répertoire `~/svg`.

### 4.3.4 Déplacer ou renommer un fichier ou un répertoire : `mv`

La syntaxe est la suivante :

```
mv [options] <fichier|rép.> [fichier|rép...] <destination>
```

Le fonctionnement de cette commande est le même que celle de la commande `cp` hormis qu'au lieu d'être copiés, les fichiers sont déplacés. Exemples :

- `mv toto titi` renomme le fichier `toto` sous le nom `titi` dans le répertoire courant ;
- `mv ~/courrier/* ~` déplace tous les fichiers du répertoire `~/courrier` dans le répertoire d'accueil.

### 4.3.5 Suppression de fichiers ou de répertoires : `rm`

La syntaxe est la suivante :

```
rm [options] <fichier|rép.> [fichier|rép...]
```

L'option `-R` ou `-r` de suppression récursive est obligatoire pour supprimer un répertoire, même vide.

Exemples :

– `rm toto titi` supprime le fichier `toto` et le fichier `titi` dans le répertoire courant ;

– `rm ~/courrier/*` supprime tous les fichiers du répertoire `~/courrier` ;

– `rm -r ~/these` supprime le répertoire `~/these`.

Quand vous utilisez la commande `rm` en ne précisant pas explicitement les fichiers à effacer (cas des répertoires, utilisation de caractère génériques, ...), pensez à vérifier d'abord les fichiers concernés avec la commande `ls`.

### 4.3.6 Suppression de répertoires : `rmdir`

La syntaxe est la suivante :

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

`rmdir` efface les répertoires demandés s'ils sont vides.

## 4.4 Permissions de fichiers et de répertoires : `chmod`

La syntaxe générale est la suivante :

```
chmod [options] <changement de mode> <fichier|rép.> [fichier|rép...]
```

Cette commande change les permissions en respectant les spécifications `<changement de mode>` sur tous les fichiers (ou répertoires) désigné par :

`<fichier|rép.> [fichier|rép...]`.

L'option `-R` permet un changement de permissions récursif.

Les spécifications de `<changement de mode>` peuvent prendre deux formes.

### 4.4.1 Changer les permissions en précisant le mode

Pour représenter les trois ensembles de permissions, Unix se sert d'un code sur trois chiffres. Ce code se nomme le *mode de fichier* ou *mode*. Chaque chiffre du mode correspond à l'un des trois ensembles de permissions. Le premier chiffre représente les permissions accordées au propriétaire. Le second fait référence aux permissions du groupe et le troisième au reste des utilisateurs.

Le codage est fondé sur l'association de valeurs numériques aux différentes permissions :

| | |
|---|---|
| permission en lecture | 4 |
| permission en écriture | 2 |
| permission en exécution | 1 |
| pas de permission | 0 |

Il suffit maintenant, pour chaque ensemble de permissions, d'ajouter les bonnes valeurs. Ainsi, pour indiquer une permission en lecture et en écriture, il faut ajouter 4 et 2.

Prenons l'exemple suivant où :

– le propriétaire a les permissions en lecture, écriture, exécution ;

– le groupe a les permissions en lecture et écriture ;

– le reste des utilisateurs a la permission en lecture.

Le calcul est le suivant :

| | | |
|---|---|---|
| Propriétaire : | lecture + écriture + exécution | $= 4 + 2 + 1 = 7$ |
| Groupe : | lecture + écriture | $= 4 + 2 + 0 = 6$ |
| Reste : | lecture | $= 4 + 0 + 0 = 4$ |

Le mode résultant est donc 764. La commande `chmod 764 *` affectera les permissions que nous venons de décrire à tous les fichiers du répertoire courant.

### 4.4.2 Changer les permissions en utilisant les catégories

La catégorie peut être une combinaison de :

**u :** *User*, soit utilisateur, ce sont les permission du propriétaire ;

**g :** *Group*, soit groupe, ce sont les permissions pour le groupe du propriétaire ;

**o :** *Other*, soit autres, ce sont les permissions 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 comme étant l'unique permission. Les droits d'accès sont définis par une ou plusieurs des lettres suivantes :

**r :** *Read*, pour le droit de lecture ;

**w :** *Write*, pour le droit d'écriture ;

**x :** *eXecute*, pour le droit d'exécution.

Par exemple, la commande `chmod u=rwx,g=rw,o=r *` produit le même résultat que la commande `chmod 764 *` vue précédemment.

## 4.5 Motifs d'englobement du shell

Nous avons déjà utilisé et décrit (cf. section 2.5.5) des caractères d'englobement pour désigner de manière concise des ensembles de nom de fichiers. Le but de cette section est d'étendre la description que nous avions précédemment faite. Les motifs d'englobement du shell sont en fait des *expressions régulières* (cf. section 8.9) simplifiées acceptées par le shell.

| | |
|---|---|
| `*` | correspond à n'importe quelle chaîne de caractères, y compris la chaîne vide. |
| `?` | 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 un mélange des deux, par exemple `[a-zBE5-7]` 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. |
| `{c1,c2}` | correspond à `c1` ou `c2`, où `c1` et `c2` sont également des motifs d'englobement. |

## 4.6 TP Manipulation des fichiers et répertoires

### 4.6.1 Chemin relatif et absolu

Vous êtes dans votre répertoire d'accueil. Dans quel répertoire vous-retrouvez vous après avoir lancé la commande (tentez de répondre avant de vérifier l'effet de la commande sur votre terminal) :

1. `cd /usr/spool`
2. `cd /usr/../var/spool`
3. `cd ../../mnt`
4. `cd ~`
5. `cd .`
6. `cd`

### 4.6.2 Création de répertoires et de fichiers

A l'aide des commandes `mkdir`, `touch` et `cd` créer l'arborescence décrite figure 4.2

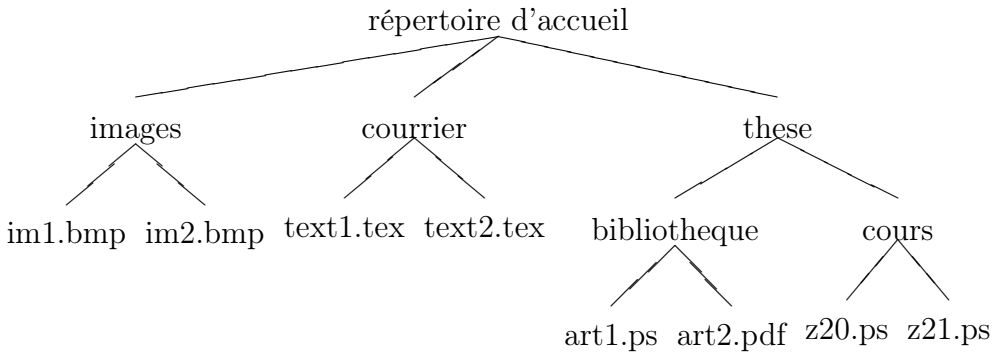


FIG. 4.2 – *Arborescence à créer.*

### 4.6.3 Utilisation des motifs d'englobement

Placez vous dans votre répertoire d'accueil. Le but de cet exercice est d'utiliser les motifs d'englobement.

1. Dressez la liste des fichiers du répertoire courant à l'aide de la commande `ls`.
2. Dressez la liste des fichiers du répertoire courant et des sous répertoires.
3. Dressez la liste des fichiers contenant le chiffre `1` du répertoire courant et des sous répertoires.
4. Dressez la liste des fichiers commençant par `art` ou `im` du répertoire courant et des sous répertoires.

### 4.6.4 Manipulation de répertoires et de fichiers

Effectuez les opérations qui suivent.

1. Placez vous dans le répertoire `cours` et faites une copie du fichier `z20.ps` dans le fichier `z20_copy.ps`
2. Copiez les fichiers `z20.ps` et `z21.ps`, en utilisant un motif d'englobement pour les désigner, dans votre répertoire d'accueil.
3. Depuis votre emplacement, déplacez les fichiers du répertoire `bibliotheque` dans votre répertoire courant.
4. Revenez dans votre répertoire d'accueil. Depuis ce répertoire, déplacez les fichiers `art1.ps` et `art2.pdf` du répertoire `cours` vers le répertoire `bibliotheque`.
5. Supprimez le répertoire `images`.
6. Renommez le fichier `z20.ps` dans votre répertoire d'accueil en `z20.pdf`.
7. Supprimez tous les fichiers commençant par la lettre `z` de votre répertoire d'accueil.
8. Déplacez le répertoire `courrier` dans le répertoire `these`.
9. Faites ce qui est nécessaire pour supprimer le répertoire `these` à l'aide de la commande `rmdir these`. Exécutez cette commande.

### 4.6.5 Permissions de fichier

1. Quel est le mode par défaut (lorsque Unix crée un fichier) pour un fichier ordinaire non exécutable?
2. Quel est le mode par défaut pour un répertoire?
3. Créez un fichier `mon_fichier_a_moi_tout_seul`. Vous voulez avoir tous les droits sur ce fichier, mais être le seul à avoir des droits sur ce fichier. Utilisez la commande `chmod` pour obtenir ce résultat.
4. Un fichier possède le mode 754 dans un répertoire dont le mode est 555. Peut-on le supprimer avec la commande `rm`? Peut-on effacer sont contenu?
5. Un fichier possède le mode 555 dans un répertoire dont le mode est 755. Peut-on le renomer avec `mv`?

# Chapitre 5

# Visualiser et éditer des fichiers

En tant qu'utilisateur Unix respectable, vous devez apprendre à utiliser un éditeur accessible par la ligne de commande. Les deux choix qui s'offrent sont `vi` et *Emacs*.

Emacs est actuellement un éditeur de texte parmi les plus puissants. Il permet de faire énormément de choses (lire son courrier, se promener sur le web, etc.) et est extensible à l'infini grâce à son langage de programmation inclus, s'appuyant sur *lisp*. Cependant, emacs n'est pas toujours disponible sur un système Unix.

`vi`, quand à lui, est disponible en standard sur tout système Unix et est largement suffisant pour travailler. Ses fonctionnalités n'ont pas changé depuis des années, et ne changeront plus jamais.

## 5.1 Editer avec `vi`

### 5.1.1 Historique

Le premier éditeur d'Unix, appelé `ed`, était un éditeur ligne à ligne. Les fichiers de texte étaient numérotées et vous deviez entrer les commandes en fonction de cette numérotation.

Plus tard, à l'université de Berkeley, Bill Joy développa un éditeur ligne à ligne plus puissant qu'il appela `ex` (*extended editor*, soit éditeur étendu). `ex` était beaucoup plus simple et puissant que `ed`.

Quand sont apparus les terminaux rapides, le besoin d'éditeurs plein écran se fit très vite sentir. Joy écrivit une interface plein écran pour `ex` qu'il appela `vi` (*Visual Editor* ou éditeur visuel). `vi` supportait toutes les commandes de `ex`, mais possédait ses propres conventions et disposait de commandes spéciales pour la gestion du plein écran. `vi` devient rapidement l'éditeur standard d'Unix puisque, comme nous l'avons dit, il fût intégré à tous les systèmes Unix.

Il est important de savoir que `vi` et `ex` sont en fait le même programme. Si vous lancez le programme avec la commande `ex`, vous aurez droit à l'interface ligne à ligne. Si vous lancez le programme avec la commande `vi`, vous obtiendrez un éditeur plein écran, `vi`, qui supporte toutes les commandes de `ex`.

### 5.1.2 Lancer `vi`

Pour lancer `vi`, tapez la commande :

vi [fichier]

où `fichier` est le nom du fichier que vous voulez écrire ou modifier. Si le fichier n'existe pas, `vi` le créera pour vous. Il est possible de lancer `vi` sans nom de fichier. `vi` vous permet de préciser un nom au moment de la sauvegarde des données.

Dans certaines situations, vous voudrez utiliser `vi` pour consulter un fichier important qui ne doit pas être modifié. Il y a deux manières de faire cela. La première est de lancer `vi` avec l'option `-R` (*Read-only*, soit lecture seule). Cette option indique à `vi` que vous ne voulez pas sauvegarder les données du fichier original. La seconde est de lancer l'éditeur au moyen de la commande `view` . Il n'y a aucune différence entre `vi -R` et `view`. Les deux commandes suivantes sont donc équivalentes :

```
vi -R <fichier_important>
view <fichier_important>
```

Utiliser `vi` de cette manière vous permet de ne pas détruire accidentellement des données importantes.

### 5.1.3 Mode d'insertion, mode de commande, commande `ex`

Quand vous travaillez avec `vi`, toutes les données sont gardées dans un tampon d'édition. Cela signifie que quand vous employez `vi` pour modifier un fichier existant, vous ne travaillez pas avec le fichier réel. `vi` copie le contenu du fichier dans le tampon d'édition. Ce n'est que lorsque vous dites à `vi` de sauvegarder vos données que le contenu de tampon est copié dans le fichier.

`vi` fonctionne selon deux modes, appelés modes saisie et mode commande. Quand `vi` est en mode saisie, tout ce que vous rentrez au clavier est inséré dans le tampon d'édition. Quand `vi` est en mode commande, les caractères que vous tapez sont interprétés comme des commandes.

Quand vous êtes en mode commande, vous pouvez passer en mode insertion en tapant `a` (insérer du texte derrière le curseur), `i` (insérer du texte devant le curseur), `A` (insérer du texte à la fin de la ligne courante) ou `I` (insérer du texte au début de la ligne courante).

Quand vous êtes en mode saisie, il faut presser la touche <Echap> pour passer en mode commande. Il n'existe aucun témoin vous indiquant à l'écran dans quel mode vous êtes (du moins avec la version originale de `vi`). Si vous êtes en mode commande et que vous pressez la touche <Echap>, `vi` vous répondra par un bip.

Toutes les commandes `ex` commencent par `:` et sont saisie en mode commande.

### 5.1.4 Sortir de `vi`

Vous pouvez sortir de `vi` de trois manières différentes.

La première, si le fichier n'a pas été modifié, vous pouvez sortir en tapant la commande `:q`. Cette commande ne permettra cependant pas de quitter `vi` si le fichier à été modifié et que les dernières modifications n'ont pas été sauvées.

La seconde façon de sortir est de taper la commande `:wq` ou `ZZ`. La commande `ZZ` quite `vi` lors de la saisie du deuxième `Z`, donc pas besoin de valider. Ces deux commandes sauvent le travail puis quitte l'éditeur.

La dernière façon de quitter `vi` est de taper la commande `:q!` qui permet de sortir de l'éditeur sans enregistrer les modifications effectuées sur le fichier édité.

### 5.1.5 Inventaire des commandes les plus importantes de `vi`

| | |
|---|---|
| `a` | Passe en mode insertion. Insertion après la position du curseur. |
| `A` | Passe en mode insertion. Insertion en fin de ligne. |
| `i` | Passe en mode insertion. Insertion à la position du curseur. |
| `I` | Passe en mode insertion. Insertion en début de ligne. |
| `o` | Rajoute une ligne vide après la ligne courante et passe en mode insertion. |
| `O` | Rajoute une ligne vide avant la ligne courante et passe en mode insertion. |
| `x` | Efface le caractère situé sous le curseur. |
| `X` | Efface le caractère précédant le curseur. |
| `dd` | Efface la ligne courante. |
| `D` | Efface la fin de la ligne à partir du curseur. |
| `w` | déplace le curseur sur le premier caractère du mot suivant. |
| `dw` | Efface le mot courant à partir de la position du curseur. |
| `cw` | Passe en mode insertion. Insertion à la position du curseur, le texte tapé remplace le mot courant à partir de la position du curseur. |
| `C` | Passe en mode insertion. Insertion à la position du curseur, le texte tapé remplace la fin de la ligne à partir du curseur. |
| `u` | Annule l'action précédente. Suivant les versions de vi, on ne peut effacer que l'action précédant immédiatement celle-ci, ou toutes les actions dans l'ordre inverse de celui dans lequel elles ont été effectuées. |
| `.` | Répète l'action précédente. |
| `$` | Déplace le curseur à la fin de la ligne. |
| `^` | Déplace le curseur au début de la ligne. |
| *flèches de direction* | Déplace le curseur d'une position dans le sens indiqué. |
| `[n]G` | Positionne le curseur sur la ligne n. Dernière ligne si n non indiqué. |
| `<nombre><commande>` | Répète la commande (sauf `G`) autant de fois qu'indiqué. Exemple : `3x` efface trois caractères. |
| `y` | Copier du texte. Par exemple, `y12w` copie les 12 mots après le curseur. |

| | |
|---|---|
| `d` | Couper du texte. Par exemple, `d12w` coupe les 12 mots après le curseur. |
| `p` | Insère le texte après le curseur. |
| `P` | Insère le texte devant le curseur. |
| `:wq` ou `ZZ` | sauvegarde le fichier puis quitte `vi`. |
| `:w [nom]` | Sauvegarde le fichier, sous le nom indiqué ici, ou sous le nom utilisé lors de l'appel à vi si aucun nouveau nom n'est spécifié. Un message d'erreur est indiqué si le nouveau fichier existe déjà. |
| `:w! [nom]` | Idem que `:w [nom]`, mais ne produit pas d'erreur si le fichier existe déjà. |
| `:q` | Quitte `vi` sauf si les modifications n'ont pas été sauvegardées. |
| `:q!` | Quitte `vi` sans condition. |
| `/expression/` | Recherche l'expression indiqué et place le curseur en début de l'occurrence trouvée. Si aucune expression spécifiée, répète la dernière recherche. Recherche à partir du curseur vers la fin du texte. |
| `?expression?` | Idem que `/expression/`, mais vers le début du texte. |
| `n` | Répète la dernière recherche. |
| `:s/expression1/ expression2/option` | Remplace la première occurrence d'`expression1` dans la ligne courante par `expression2`. Si `expression1` non spécifié, utilise l'expression utilisée lors de la dernière recherche. L'option `g` permet de remplacer l'ensemble des occurrences dans la ligne courante. |
| `:help` | Afficher l'aide de `vi`. |

## 5.2 Editer avec Emacs

### 5.2.1 Présentation générale

Pour lancer Emacs, tapez la commande :

emacs [fichier]

où `fichier` est le nom du fichier que vous voulez écrire ou modifier. Si le fichier n'existe pas, Emacs le créera pour vous. Il est possible de lancer Emacs sans nom de fichier. Emacs vous permet de préciser un nom au moment de la sauvegarde des données.

Emacs repose complètement sur l'emploi de deux touches particulières : <Ctrl> et <Meta>. Cet éditeur de textes a défini sa propre représentation des caractères de contrôle. L'abréviation `C-` signifie <Ctrl>, l'abréviation `M-` signifie <Meta>. Ainsi, pour désigner la combinaison de touche <Ctrl-C>, parfois notée `^C`, Emacs utilise la représentation `C-c`. Cette notation signifie « pressez la touche <Ctrl> puis, sans relâcher la touche <Ctrl>, pressez la touche <C> ».

La touche <Meta> correspond en fait à la touche d'échappement <Echap>. Vous ne la manipulerez pas comme la touche <Ctrl>. Vous pressez d'abord <Echap>, puis la touche suivante.

Pour obtenir l'aide en ligne, après voir lancé Emacs, il faut entrer `C-h t`.
Pour sortir d'Emacs, faites `C-x C-c`.

### 5.2.2 Inventaire des commandes les plus importantes d'Emacs

| | |
|---|---|
| `C-h t` | Obtenir l'aide en ligne. |
| `C-x C-s` | Enregistrer les modification. |
| `C-x C-w` | Enregistrer sous un nouveau nom. |
| `C-x C-c` | Quitter d'Emacs. |
| `C-x u` | Annuler l'opération précédente. |
| `C-a` | Aller en début de ligne. |
| `C-e` | Aller en fin de ligne. |
| `C-Espace` | Marque le début du tampon. |
| `C-w` | Couper le texte entre la marque de début du tampon et la position actuelle du curseur. |
| `C-y` | Coller le texte copié. |
| `C-s` | Rechercher du texte. |

## 5.3 Visualiser : les premiers filtres

Pour toutes les commandes qui suivent, pensez à consulter la pages du manuel (avec `man`) pour obtenir plus d'information.

### 5.3.1 Afficher le début d'un fichier : `head`

Servez vous de la commande `head` pour visualiser le début d'un fichier. La syntaxe est :

```
head [options] [fichier...]
```

L'option `-n<nb>` ou directement `-<nb>` permet d'afficher les `nb` premières lignes (*liNes* en anglais) du fichier.

L'option `-b<nb>` permet d'afficher les `nb` premier octets (*Bytes* en anglais) du fichier.

Par défaut, `head` consulte les 10 premières lignes, ce qui peut être utile si vous voulez rapidement vous faire une idée du contenu d'un fichier.

### 5.3.2 Afficher la fin d'un fichier : `tail`

Servez vous de la commande `tail` pour visualiser la fin d'un fichier. La syntaxe est :

```
tail [options] [fichier...]
```

Par défaut, `tail` consulte les 10 dernières lignes.

L'option `-n<nb>` ou directement `-<nb>` permet d'afficher les `nb` dernières lignes (*liNes* en anglais) du fichier.

L'option `-c<nb>` permet d'afficher les `nb` derniers octets du fichier.

Si on remplace le `-` par un plus `+` le début de l'affichage n'est plus compté à partir de la fin du fichier mais à partir du début. Par exemple `tail -10 fic.txt` affiche les 10

dernières lignes du fichier `fic.txt` tandis que la commande `tail +10 fic.txt` affiche de la ligne 10 jusqu'à la dernière ligne du fichier.

### 5.3.3 Les programmes de pagination

Dans les trois sections suivantes, nous aborderons les programmes que vous pouvez utiliser pour visualiser un fichier à l'écran. Ces programmes sont `more`, `less` et `pg`.

La caractéristique principale de ces programmes est qu'ils affichent des fichiers entiers sous la forme de plusieurs écrans successifs, chaque écran affiché effaçant le précédent, d'où leur nom de programme de pagination.

Pendant la lecture, vous aurez à votre disposition un certain nombre de commandes, parmi lesquelles celle qui permet de chercher une chaîne ou un motif particulier dans le texte. Les trois programmes disposent de leurs propres commandes internes.

La commande `man` affiche la page de manuel d'une commande en faisant un tube avec (ie. en se servant de) un programme de pagination qui peut être soit `more` soit `pg`. Sur la plupart des systèmes vous n'aurez pas le choix, ce qui fait qu'il est plus judicieux de connaître les principes de base associés à chaque programme de pagination.

### 5.3.4 Visualiser un fichier avec `more`

`more` est ainsi nommé parce qu'il affiche le *prompt* « `more` » (« plus ») une fois qu'un écran a été rempli, invitant ainsi l'utilisateur à continuer la lecture.

Le programme `more` peut être utilisé de la manière suivante :

```
more [-dpcs] [+/motif] [+ligne] [fichier...]
```

`motif:` désigne le motif que l'on recherche dans les fichiers ;

`ligne:` désigne le numéro de la ligne à laquelle vous voulez commencer ;

`fichier...:` le ou les fichiers que vous voulez visualiser ;

`-d:` pour que `more` affiche le message `[Press space to continue, 'q' to quit.]` à la fin de chaque écran et `[Press 'h' for instructions.]` plutôt qu'un *bip* quand une touche illégale est enfoncée ;

`-p:` pour ne pas avoir de scrolling d'affichage avec un effacement de l'écran précédant ;

`-c:` pour ne pas avoir de scrolling d'affichage avec un écrasement de l'écran précédant ;

`-s:` pour remplacer des lignes blanches successives par une seule ligne blanche ;

Le programme `more` affiche le contenu des fichiers passés en paramètre. Les données sont affichées sous la forme d'écrans successifs. Quand un écran est rempli, un *prompt* (une invite) est affichée au bas de l'écran, vous invitant à continuer. Le *prompt* a la forme :

```
--More-- (40%)
```

Le nombre entre parenthèses indique la quantité de données déjà affichées.

Quand `more` est utilisé dans un pipe-line (cf. section **??**), il n'affiche pas de pourcentage (c'est le cas avec la commande `man` par exemple), car le programme n'a aucune idée de la quantité de données contenus dans le pipe-line.

Au niveau du *prompt* vous avez la possibilité d'entrer des commandes. Dans la plupart des cas, vous n'avez pas à presser <Entrée> après la commande. La commande la plus

utilisée est celle qui affiche l'écran suivant : <Espace>. Une autre commande importante est <h> (*Help*, aide) qui affiche un résumé des commandes disponibles. Le tableau 5.1 récapitule les commandes `more` les plus utiles.

| | |
|---:|---|
| `h` | Afficher l'aide. |
| <Espace> | Passer à l'écran suivant. |
| `q` | Quitter le programme. |
| <Entrée> | Avancer d'une ligne. |
| $n$ <Entrée> | Avancer de $n$ ligne. |
| `d` | Avancer d'un demi-écran. |
| $n$`f` | Avancer de $n$ écran. |
| `b` | Retourner d'un écran en arrière. |
| $n$`b` | Retourner de $n$ écran en arrière. |
| `v` | Lancer `vi` sur le fichier visualisé. |
| /*motif* <Entrée> | Rechercher le motif *motif* vers l'aval du texte. |
| `n` | Répéter la commande précédente. |
| `!`*commande* <Entrée> | Exécuter la commande de *shell commande*. |
| `=` | Afficher le numéro de ligne courant. |
| `.` | Répéter la commande précédente. |

TAB. 5.1 – *Résumé des commandes* `more` *les plus utiles.*

### 5.3.5 Visualiser un fichier avec pg

`pg` est une abréviation de *pager*, terme anglais désignant un programme de pagination. Le programme `pg` peut être utilisé de la manière suivante :

```
pg [-cn] [+/motif] [+ligne] [fichier...]
```

`motif:` désigne le motif que l'on recherche dans les fichiers ;

`ligne:` désigne le numéro de la ligne à laquelle vous voulez commencer ;

`fichier...:` le ou les fichiers que vous voulez visualiser ;

`-c:` pour ne pas avoir de scrolling d'affichage avec un écrasement de l'écran précédant ;

`-n:` indique à `pg` d'exécuter les commandes composées d'une lettre sans devoir presser la touche <Entrée> (comme `more`).

Le programme `pg` affiche le contenu des fichiers passés en paramètre. Les données sont affichées sous la forme d'écrans successifs. Quand un écran est rempli, un *prompt* (une invite) est affichée au bas de l'écran, vous invitant à continuer. Le *prompt* a la forme :

:

Au niveau du *prompt* vous avez la possibilité d'entrer des commandes. La commande la plus utilisée est celle qui affiche l'écran suivant : <Entrée>. Une autre commande importante est <h> (*Help*, aide) qui affiche un résumé des commandes disponibles. Dans tous les cas (sauf si vous avez précisé l'option `-n`), vous devrez presser <Entrée> après la commande. Le tableau 5.2 récapitule les commandes `pg` les plus utiles.

| | |
|---|---|
| `h` | Afficher l'aide. |
| <Entrée> | Passer à l'écran suivant. |
| `q` | Quitter le programme. |
| *n*`l` | Aller à la ligne *n*. |
| `l` | Aller à la ligne suivante. |
| `+`*n*`l` | Avancer de *n* ligne. |
| `-`*n*`l` | Retourner de *n* ligne en arrière. |
| `d` | Avancer d'un demi-écran. |
| `-d` | Retourner d'un demi-écran en arrière. |
| `1` | Aller à la première ligne. |
| `$` | Aller à la dernière ligne. |
| `/`*motif* | Rechercher le motif *motif* vers l'aval du texte. |
| `?`*motif* | Rechercher le motif *motif* vers l'amont du texte. |
| `!`*commande* | Exécuter la commande de *shell commande*. |

TAB. 5.2 – *Résumé des commandes* `pg` *les plus utiles.*

### 5.3.6 Visualiser un fichier avec `less`

`less` est le pendant des commandes `pg` et `more`. Son nom vient de l'expression anglaise *more or less* (plus ou moins). L'ironie de la chose vient de ce que `less` est bien plus puissant que `more`.

Le programme `less` peut être utilisé de la manière suivante :

```
less [-cmsM] [+xtabulation] [+commande] [fichier...]
```

`tabulation:` permet de préciser la taille désirée des tabulations en nombre de caractères ;

`commande:` permet de spécifier une commande à exécuter automatiquement ;

`fichier...:` le ou les fichiers que vous voulez visualiser ;

`-c:` pour ne pas avoir de scrolling d'affichage avec un écrasement de l'écran précédant ;

`-m:` permet d'afficher un *prompt* semblable à celui de la commande `more` en affichant le pourcentage de fichier déjà parcouru ;

`-s:` pour remplacer des lignes blanches successives par une seule ligne blanche ;

`-M:` cette option affiche encore plus d'informations que `-m`, elle affiche le nom du fichier, le numéro de ligne et le pourcentage.

`less` ne fait pas partie du système standard, il se peut qu'il ne soit pas disponible sur votre système.

La syntaxe complète de la commande `less` est bien plus complexe en raison de la multitude de fonctionnalités que possède cette commande. Tapez `man less` pour vous en convaincre.

De manière générale, `less` fonctionne comme les deux autres programmes, `pg` et `more`, mais il dispose de fonctionnalités plus sophistiquées. Un des principaux avantages de `less` est qu'il facilite les déplacements en avant et en arrière dans le fichier. Les commandes de `less` sont celles de `more` et de l'éditeur de textes `vi`.

Le programme `less` affiche le contenu des fichiers passés en paramètre. Les données sont affichées sous la forme d'écrans successifs. Quand un écran est rempli, un *prompt* (une invite) est affichée au bas de l'écran, vous invitant à continuer. Le *prompt* a la forme :

:

Au niveau du *prompt* vous avez la possibilité d'entrer des commandes. Dans la plupart des cas, vous n'avez pas à presser <Entrée> après la commande. La commande la plus utilisée est celle qui affiche l'écran suivant : <Espace>. Une autre commande importante est <h> (*Help*, aide) qui affiche un résumé des commandes disponibles. Le tableau 5.3 récapitule les commandes `more` les plus utiles.

| | |
|---:|---|
| `h` | Afficher l'aide. |
| <Espace> | Passer à l'écran suivant. |
| `q` | Quitter le programme. |
| <Entrée> | Avancer d'une ligne. |
| $n$ <Entrée> | Avancer de $n$ ligne. |
| `b` | Retourner d'un écran en arrière. |
| $n$`y` | retourner de $n$ lignes en arrière. |
| `d` | Avancer d'un demi-écran. |
| `u` | Retourner d'un demi-écran en arrière. |
| `g` | Aller à la première ligne. |
| $n$`g` | Aller à la ligne $n$. |
| `G` | Aller à la dernière ligne. |
| $n$`p` | Aller à la ligne indiquant le pourcentage $n$. |
| `v` | Lancer `vi` sur le fichier visualisé. |
| /*motif* <Entrée> | Rechercher le motif *motif* vers l'aval du texte. |
| ?*motif* <Entrée> | Rechercher le motif *motif* vers l'amont du texte. |
| `n` | Répéter la commande précédente. |
| !*commande* <Entrée> | Exécuter la commande de *shell commande*. |
| `=` | Afficher le numéro de ligne courant et le nom du fichier. |
| `.` | Répéter la commande précédente. |
| -*option* <Entrée> | Changer d'option. |
| _*option* <Entrée> | Afficher la valeur courante d'une option. |

TAB. 5.3 – *Résumé des commandes* `less` *les plus utiles.*

## 5.4 Compléments sur la commande `man`

### 5.4.1 Connaître rapidement la fonction d'une commande : `whatis`

La commande `man` d'Unix affiche la page de manuel complète, alros qu'il vous suffit souvent d'une brève description.

Chaque page de manuel comporte une première section *NAME* qui donne en une ligne le nom et la fonction de la commande. Si vous ne désirez visualiser que cette brève description, vous devez préciser l'option `-f` après la commande `man`.

Par commodité, la commande `whatis` a exactement le même rôle que `man -f`. Ainsi, si vous souhaitez afficher l'heure mais que vous ne savez plus si la commande appropriée est `date` ou `time`, entrez : `whatis time date`.

### 5.4.2 Chercher une commande `apropos`

Quand vous désirez en savoir plus sur une commande, vous devez utiliser `man`. Mais que pouvez-vous faire quand vous savez ce que vous voulez mais que vous ignorez la commande à utiliser ?

La solution à ce problème est d'employer la commande `man` avec l'option `-k` afin de répertorier les commandes dont la description contient certains mots-clés. Supposons par exemple que vous souhaitez trouver toutes les pages du manuel qui ont quelque chose à voir avec le manuel lui même. Pour cela, entrez : `man -k manual`. Par commodité, vous avez accès à la commande `apropos`, dont la fonction est identique à `man -k` : `apropos manual`.

## 5.5 TP Visualiser et éditer des fichiers

### 5.5.1 Edition d'un fichier

Familiarisez vous à l'édition de fichier avec `emacs`.

### 5.5.2 Début et fin d'un fichier

Apprenez à utiliser les commandes `head` et `tail`.

### 5.5.3 Programme de pagination

Apprenez à consulter (avancer et retourner) et à rechercher des informations dans un fichier avec la commande `more`.

# Chapitre 6

# Installation de Linux et de Cygwin/XFree86

## 6.1 Cygwin/XFree86, mettez un Unix dans votre Windows

### 6.1.1 Introduction

Si vous êtes obligés de travailler sous Windows mais avez envie d'une ligne de commande de type Linux, alors cet ensemble de programmes est fait pour vous !

Cygwin est un ensemble de logiciels de développement GNU. Cygwin sert à utiliser (simuler) une API de type UNIX dans une API Win32. En d'autres mots, Cygwin est un émulateur de système Unix pour Windows. Il permet de reproduire les fonctions Unix sur votre système Windows et en particulier de compiler et d'exécuter sur le PC/Windows des programmes développés pour Unix.

### 6.1.2 Installation de *Cygwin*

Nous supposons ici que Cygwin à déjà été intégralement téléchargé. Cygwin peut-être téléchargé depuis le site `http://www.cygwin.com/`. Cygwin peut même être installé directement depuis ce site.

1. Exécutez `setup.exe` puis cliquez sur *Suivant >*.
2. Cochez la case *Install from Local Directory* puis cliquez sur *Suivant >*.
3. Désignez le répertoire d'installation. Attention, cet utilitaire ne supporte pas les noms avec des espaces. Si vous voulez installer *Cygwin* dans le répertoire

   ```
   C:\Program Files\cygwin
   ```

   il faut spécifier

   ```
   C:\PROGRA~1\cygwin
   ```

   Cliquez ensuite sur *Suivant >*.
4. Désignez le répertoire depuis lequel vous installez *Cygwin* puis cliquez sur *Suivant >*.
5. Vous devez maintenant préciser les paquetages que vous voulez installez. Pour tout installer Cliquez sur *Default* en face de *All*. *Install* s'affiche alors en face *All*. Cliquez

ensuite sur *Suivant >*. L'installation commence et peut prendre plusieurs dizaines de minutes.

6. Quand l'installation est terminée cliquez sur *Terminer* puis, une fois les dernières opérations de configuration effectuées (encore quelques minutes) cliquez sur *Ok*.
7. Vous devez ajouter le chemin `C:\PROGRA~1\cygwin\bin` (remplacez `C:\PROGRA~1` par votre répertoire d'installation) dans votre variable d'environnement `PATH`. Sous windows 2000 la procédure est la suivante :
   - Clic droit sur *Poste de travail* puis *Propriétés*.
   - Cliquez sur l'onglet *Avancé*.
   - Cliquez sur le bouton *Variables d'environnement...*
   - Dans la zone *Variables système* sélectionnez *Path* puis cliquez sur le bouton *Modifier...*
   - A la fin ajoutez `;C:\PROGRA~1\cygwin\bin`
   - Cliquez sur *Ok* trois fois.

   C'est fini.

### 6.1.3 Installation de *XFree86* pour CygWin

Cette installation est facultative et destinée à ceux qui ont besoin de l'environnement graphique X-Window.

1. Pour installer, *XFree86* vous devez préalablement installer CygWin.
2. Allez dans le répertoire `\Cygwin\XFree86\4.1.0` contenant les fichiers d'installation de *XFree86* et copiez-les (tous si vous désirez, référez vous au fichier files.txt) dans un sous-répertoire du répertoire d'installation de cygwin, par exemple :

   `C:\PROGRA~1\cygwin\xinstall`.
3. Lancez le shell de CygWin (un lien a certainement été placé sur votre bureau ou dans le menu démarrer).
4. Allez dans le répertoire (`cd /xinstall`).
5. Décompressez le fichier `extract.exe.bz2` avec la commande :

   ```
   bunzip2 extract.exe.bz2
   ```
6. Copiez le fichier ainsi décompressé dans `/bin` :

   ```
   cp extract.exe /bin
   ```
7. Exécutez le script d'installation :

   ```
   sh Xinstall.sh
   ```

   Répondez par l'affirmative (y) aux questions posées (il faut être patient).
8. Copiez le fichier `startup-scripts.tgz` dans `/usr/X11R6/bin` :

   ```
   cp startup-scripts.tgz /usr/X11R6/bin
   ```
9. Allez dans ce répertoire :

   ```
   cd /usr/X11R6/bin
   ```
10. Décompressez le fichier :

    ```
    tar -xzf startup-scripts.tgz
    ```

11. Pour démarrer X Window exécutez le script :

    `\cygwin\usr\X11R6\bin\startxwin.bat`

    Nous vous conseillons d'en faire un raccourci sur le bureau.
12. L'installation proprement dite est terminée.

## 6.2 Installation de Mandrake Linux 9.1 avec DrakX

### 6.2.1 Introduction

#### 6.2.1.1 Qu'est-ce que DrakX ?

*DrckX* est le logiciel d'installation pour Mandrake Linux. La tâche de DrackX est de simplifier au maximum l'installation de Mandrake Linux.

#### 6.2.1.2 Distribution et documentation

Avant de procéder à l'installation de Mandrake Linux, il faut vous procurer les CD-ROM d'installation. Pour cela rendez-vous sur le site de Mandrake :

`http://www.mandrakelinux.com/`.

Profitez-en pour vous procurer de la documentation. Pour cela, dans le menu sur la gauche du site de Mandrake, dans le section *Support*, cliquez sur *Documentation*. Vous y trouverez principalement les documentations suivante :

- **Un guide d'installation de Mandrake Linux :** sur le site, il est appelé « Guide de démarrage rapide » ou « *Quick_ Startup* » en anglais. Ce guide décrit pas-à-pas les étapes de l'installation de Mandrake Linux.
- **Un guide sur l'utilisation de la distribution Mandrake Linux :** sur le site, il est appelé « Guide de démarrage » ou « *Starter* » en anglais. Ce guide est un manuel indispensable pour utiliser Mandrake Linux une fois celui-ci installé sur votre ordinateur.
- **Un guide sur les principales commandes d'Unix/Linux :** sur le site, il est appelé « Manuel de ligne de commande » ou « *Command-Line* » en anglais. Ce guide est un manuel des principales commandes d'Unix/Linux et est relativement indépendant d'une distribution particulière de Linux.

N'hésitez pas à la consulter toute cette documentation pour obtenir plus d'information.

#### 6.2.1.3 Note sur le *plug'n'play*

L'apparition du *plug'n'play* et sa démocratisation font que tous les *BIOS* modernes ont la possibilité d'initialiser de tels périphériques, mais encore faut-il le leur demander ! Pour cela, recherchez l'option adéquat dans votre *BIOS*. Elle est souvent intitulée `PnP OS installed`. Mettez cette option à `No`. Le *BIOS* initialisera ainsi les périphériques *plug'n'play*, cela peut aider Linux à reconnaître certains périphériques de votre machine.

Si votre *BIOS* est capable de démarrer à partir du CD-ROM, vous devez faire en corte que le *BIOS* cherche d'abord à démarrer depuis le CD-ROM. Sinon, il vous faudra créer une disquette de démarrage (cf. documentation de Mandrake Linux).

### 6.2.2 Installation *classique* de Mandrake Linux

#### 6.2.2.1 Avertissement !

La procédure de redimensionnement de la partition Windows est dangereuse. En cas de problème (que ce soit un problème au niveau du redimensionnement ou une erreur de votre part au niveau du choix de la partition d'installation) vous pouvez très bien vous retrouver avec un système qui ne fonctionne plus et même perdre vos données personnelles stockées sur le disque dur. L'assurance ultime contre les problèmes est de toujours **sauvegarder vos données** sur un support externe (disque ZIP, CD-ROM, disque externe, ...).

Pour que le redimensionnement se passe dans les meilleures des conditions il est recommandé d'effectuer les opérations qui suivent.

- Vous devez lancer scandisk sur votre disque Windows : le programme de redimensionnement peut détecter certaines erreurs, mais scandisk est plus adapté ;
- Pour une sécurité optimale des données, vous devriez aussi lancer defrag sur le disque. Cela réduit encore le risque de pertes de données : ce n'est pas obligatoire, mais fortement recommandé. Cela rendra aussi le redimensionnement plus rapide ;

**Attention !** Les utilisateurs de Windows 2000 , NT ou XP doivent être encore plus attentifs de NE PAS redimensionner une partition NTFS avec Linux. Vos données seront endommagées. Dans ce cas, utilisez une application de redimensionnement sur Windows comme Partition Magic.

#### 6.2.2.2 Installation

Laissez vous guider par le logiciel d'installation DrakX.

**Soyez vigilant lors de l'étape** « L'assistant de partitionnement a trouvé les solutions suivantes ». En effet, votre choix peut vous amener à perdre le système précédemment installé ainsi que toutes vos données personnelles. Par exemple, en choisissant « Effacer tout le disque » vous effacez toutes les données et les applications installées sur votre système et les remplacez par votre nouveau système Mandrake Linux. Soyez prudent car ce choix est irréversible et **l'ensemble du contenu de votre disque peut être perdu** !

Lors de l'étape « Créer un compte utilisateur » pensez à ajouter au moins un utilisateur : vous. En effet, vous n'utiliserez le compte `root` que pour des opérations spécifiques et ponctuelles.

### 6.2.3 Installation Linux avec conservation du menu de démarrage de Windows 2000 ou XP

#### 6.2.3.1 Avertissements !

**N'oubliez pas que vous devez posséder une partition libre** sur l'un de vos disque dur.

Pour réaliser cette installation vous aurez à effectuer des opérations sensibles sur votre système :

- partitionnement de disque dur ;
- choix de partition d'installation de linux;
- édition de fichier sensibles ; ...

En cas d'erreur de manipulation vous pouvez très bien vous retrouver avec un système qui ne fonctionne plus et même perdre vos données personnelles stockées sur disque dur. L'assurance ultime contre les problèmes est de toujours **sauvegarder vos données** sur un support externe (disque ZIP, CD-ROM, disque externe, ...).

Pour l'installation de Mandrake 9.1 lisez la section 6.2.3.3 avant d'effectuer les opérations décrites dans la section 6.2.3.2.

#### 6.2.3.2 Principe de l'installation de Linux avec conservation du menu de démarrage de Windows 2000 ou XP

1. Installez linux dans la partition libre de votre disque dur. Vous devez **spécifier votre partition root Linux comme support de boot** car le Master Boot Record (MBR) de votre disque dur est déjà occupé par celui de Windows 2000 et c'est lui que nous désirons conserver. Vous devez également **créer une disquette de démarrage !** En effet, comme Lilo n'est pas installé sur la partition principale, il est impossible de démarrer Linux sans disquette de démarrage tant que le menu de démarrage de Windows n'est pas configuré.
2. Loguez vous sous Linux (vous devez booter depuis la disquette de démarrage pour démarrer Linux).
3. Copiez le secteur de boot de votre partition de boot Linux dans un fichier. Pour cela il vous faut les pouvoirs d'administrateur (`root`). Vous avez deux solutions :
   - soit vous vous loguez en tant que root ;
   - soit vous prenez ses pouvoirs avec la commande `su` (solution préférable).

   Avec `/dev/hda2` comme partition Linux, la commande est :

   ```
   $ dd if=/dev/hda2 of=/bootsect.lnx bs=512 count=1
   ```

   Quelque chose ne va pas si votre `bootsect.lnx` fait plus de 512 octets.
4. Maintenant, copiez le fichier `bootsect.lnx` sur une disquette formatée pour DOS, ou sur un disque visible de Windows.

   Vous pouvez le copier avec sur une disquette avec la commande :

   ```
   $ mcopy /bootsect.lnx /mnt/floppy
   ```

   Vous pouvez le copier directement au bon endroit (`C:\`) avec un commande du style :

   ```
   $ cp /bootsect.lnx /mnt/win_c/
   ```
5. Loguez vous sous Windows et, si ce n'est déjà fait, copiez depuis la disquette le fichier vers : `C:\bootsect.lnx`.
6. Nous allons maintenant modifier le fichier `C:\boot.ini` qui est à Windows 2000 ce que `lilo.conf` est à linux. Changez lui ses attributs system et de lecture seules, avant de le modifier avec, dans une fenêtre d'invite de commande, la commande :

   ```
   C:\>attrib -S -R c:\boot.ini
   ```
7. Maintenant, changez le fichier boot.ini avec un éditeur, notepad par exemple, rajoutez à la fin la ligne suivante :

   ```
   C:\bootsect.lnx="Linux"
   ```
8. Remettez les bons attributs après avoir sauvé `boot.ini` comme ceci :

   ```
   C:\>attrib +S +R c:\boot.ini
   ```

Après un redémarrage de votre Windows vous verrez une nouvelle ligne `Linux` dans votre menu de démarrage.

Une nouvelle copie de `bootsect.lnx` doit être transférée vers le fichier `C:\bootsect.lnx` chaque fois que le secteur de boot de votre partition linux est modifiée. Ceci arrive quand vous installez un nouveau noyau avec lilo.

#### 6.2.3.3 Remarque importante concernant l'installation de Mandrake 9.1

Laissez vous guider par le logiciel d'installation DrakX.

**Soyez vigilant lors de l'étape** « L'assistant de partitionnement a trouvé les solutions suivantes ». Choisissez « Partitionnement personnalisé ». Sélectionnez la partition sur laquelle vous voulez installer Linux. **Toutes les informations qui se trouve sur cette partition seront perdue**. Cliquez sur l'action *Supprimer* puis sur *Partitionnement automatique* puis sur le bouton *Terminer*. Votre partition va être subdivisée et préparée à l'installation de Mandrake Linux.

Avec le logiciel d'installation DrakX de Mandrake 9.1, il est impossible de se conformer facilement au point 1 de la section 6.2.3.2, c'est-à-dire de spécifier votre partition root Linux comme support de boot (probablement un bug au niveau de l'installation). Le master boot record de votre disque dur, déjà occupé par celui de Windows 2000, est ainsi écrasé (nous le restaurerons par la suite).

Lors de l'étape « Créer un compte utilisateur » pensez à ajouter au moins un utilisateur : vous. En effet, vous n'utiliserez le compte `root` que pour des opérations spécifiques et ponctuelles.

Une fois l'installation achevée, loguez sous Linux. Dans une `xterm`, prenez les pouvoirs d'administrateur (avec la commande `su` par exemple), puis restaurez le master boot record originale :

```
lilo -u /dev/hda
```

Ouvrez le « Panneau de Contrôle Mandrake », entrez le mot de passe de l'administrateur, sélectionnez l'item « Démarrage » puis l'icône « DrakBoot : configuration de l'amorçage ». Dans « Menu d'amorçage » cliquez sur le bouton *Configurer*. Cliquez sur le bouton *Avancé*. Pour le « Périphérique d'amorçage » **spécifiez votre partition root Linux comme support de boot**. **Créez une disquette de démarrage !** En effet, comme Lilo n'est plus installé sur la partition principale, il est impossible de démarrer Linux sans cette disquette tant que le menu de démarrage de Windows n'est pas configuré. Pour cela, cocher la case « Disquette d'amorçage ». Validez en cliquant sur le bouton *Ok*

Pour finir l'installation, reprenez les étapes de la section précédente à partir de l'item 2.

# Chapitre 7

# Redirections, tubes et filtres : principes

## 7.1 Philosophie Unix

Les deux personnes qui ont conçu le premier Unix ont tout de suite compris qu'il était important d'éviter la complexité de Multics et des systèmes équivalents. Ainsi, ils prirent résolument une attitude spartiate. Ils décidèrent que chaque programme devait être un outil unique avec, dans certains cas, quelques options de base. Un programme ne devait faire qu'une chose, mais devait la faire bien. Une fois confronté à une tâche plus complexe, l'idée principale est qu'il fallait combiner les outils existants et non écrire un nouveau programme.

S'il y a un seul concept que vous devez comprendre pour utiliser Unix de manière efficace, c'est bien celui d'entrée standard et de sortie standard. L'idée fondamentale est simple : chaque programme doit être capable d'accepter toute source de données en entrée et d'acheminer les données de sortie vers n'importe quel autre programme. Imaginons par exemple que vous disposez d'un programme qui trie des lignes de données. Vous devez avoir le choix entre taper vous-même les données au clavier, lire ces données depuis un fichier, ou même prendre en entrée les données produites par un autre programme. De la même manière, le programme de tri doit être capable d'afficher ses résultats sur votre écran, de les écrire dans un fichier, ou de les envoyer vers un autre programme pour d'autres traitements.

Un tel système présente deux avantages appréciables. Le premier est qu'en tant qu'utilisateur, vous disposez d'une grande liberté de manœuvre. Vous pouvez définir l'entrée et la sortie d'un programme en fonction de vos besoins.

Vous ne devez de plus apprendre à utiliser qu'un seul outil pour chaque type de tâche. Le même programme d'affichage de données triées est capable de sortir ses résultats sur l'écran, vers un fichier, ou vers un autre programme.

Le deuxième avantage est que la conception et l'écriture des programmes devient bien plus facile. Quand vous écrivez un programme, vous n'avez pas à vous soucier de toutes les variations possibles de conditionnement des données d'entrée et de sortie. Vous pouvez vous concentrer sur les détails de votre programme et laisser Unix gérer les flux d'entrée et de sortie pour vous.

L'idée cruciale est que l'entrée et la sortie ne sont jamais précisées par le programmeur. Celui-ci conçoit le programme de manière générale en pouvant lire des données en entrée et écrire des données en sortie. Ultérieurement, lorsque vous lancerez le programme résultant,

le *shell* se chargera de relier la source appropriée à l'entrée et la destination appropriée à la sortie.

## 7.2 Redirections et tubes

### 7.2.1 Redirection de la sortie standard

Quand vous vous *loguez*, le *shell* affecte automatiquement l'entrée standard à votre clavier, et la sortie standard à votre écran. Cela signifie que, par défaut, la plupart des programmes liront les données depuis votre clavier et écriront les résultats sur votre écran.

Cependant, à chaque fois que vous entrez une commande et juste pour celle-ci, vous pouvez indiquer au *shell* de modifier l'entrée ou la sortie standard.

Voici comment cela fonctionne : si vous souhaitez que la sortie se fasse sur l'écran, vous n'avez rien à faire. Le *shell* fait cela automatiquement.

Si vous voulez que la sortie standard se fasse dans un fichier, saisissez un `>` (chevron supérieur) suivi du nom de fichier, à l afin de la commande. Par exemple, la commande `ls > fic` envoie le résultat de la commande `ls` dans le fichier `fic`. Le caractère `>` est assez représentatif de ce qui se passe, car on peut le considérer comme une flèche montrant le chemin suivi par les données produites en sortie.

Si le fichier n'existe pas, Unix le créera. S'il existe déjà, son contenu sera remplacé. Si vous employez deux caractères chevron supérieur `>>` à la suite, Unix va ajouter les données produites au fichier déjà existant.

Quand nous envoyons envoyons la sortie standard dans un fichier, nous disons que nous le *redirigeons*. Les deux commandes précédentes font donc une redirection de la sortie standard vers un fichier.

C'est à vous de choisir, quand vous redirigez la sortie, entre le remplacement avec `>` ou l'ajout avec `>>`. Si le fichier n'existe pas, ces deux commandes sont équivalentes.

### 7.2.2 Redirection de l'entrée standard

Par défaut, l'entrée standard est liée à votre clavier. Quand vous entrez une commande qui doit lire des données, Unix s'attend à ce que vous les rentriez au clavier, une ligne à la fois. Quand vous avez fini d'entrer des données, il vous faut presser `^D` (le code de fin de fichier). L'envoi de ce code indique qu'il n'y à plus de données disponibles.

Dans bien des cas, vous aurez envie de rediriger l'entrée standard depuis un fichier. Autrement dit, vous demanderez au *shell* de lire ses données depuis un fichier et non depuis le clavier. Pour cela, rentrez `<` (le chevron inférieur), suivi du nom de fichier à la fin de la commande.

Par exemple, pour trier les données contenues dans un fichier nommé `temp`, utilisez la commande suivante : `sort < temp`. Comme vous pouvez le voir, le caractère `<` indique assez clairement le sens du transfert depuis le fichier indiqué vers le programme.

Comme vous pouvez l'imaginer, il est possible d'utiliser simultanément la redirection de l'entrée standard et celle de la sortie standard. Par exemple, la commande : `sort < rawdata > names` lit les données depuis le fichier nommé `rawdata`, le trie, et écrit le résultat dans un fichier nommé `names`.

### 7.2.3 Tubes (*pipes*)

Si vous voulez que la sortie d'un programme soit communiquée comme entrée d'un autre programme, rentrez le caractère | (barre verticale) suivie du nom de programme. Par exemple, pour envoyer la sortie du programme de tri `sort` vers le programme d'impression `lpr`, faites `sort | lpr`. Quand nous envoyons la sortie standard d'un programme vers un autre programme, nous disons que nous établissons un *tube de communication* entre les programmes. Ainsi, la commande `sort` envoie ses résultats à la commande `lpr` via un tube.

Une fois que vous savez comment établir un tube, vous pouvez bâtir une commande dans laquelle les sorties des programmes sont acheminées d'un programme à l'autre en séquence formant ainsi un pipe-line de tubes. L'image d'un pipe-line est adaptée si l'on songe que celui-ci sert à transporter de la matière (pétrole ou gaz) d'un endroit à un autre. Une image encore meilleure est celle d'une chaîne d'assemblage où chaque programme modifie les données de manière différente. Les données non encore traitées entrent dans la chaîne d'assemblage par l'entrée d'un tube tandis que les données produites apparaissent en sortie du dernier programme de la chaîne.

Bien utiliser Unix c'est savoir quand et comment résoudre un problème en combinant des programmes avec des tubes.

### 7.2.4 Tés (*tee*)

Il se peut que vous ayez envie de rediriger la sortie standard d'un programme vers deux sorties différentes simultanément. La solution à ce problème est de vous servir d'un *tee*. Le `tee` est un mécanisme qui lit l'information sur son entrée standard et la recopie simultanément sur la sortie standard et dans un ou plusieurs fichiers.

Pour créer un *tee*, employez la commande `tee`. La syntaxe de la commande `tee` est : `tee [-a][fichiers...]`. Si le ou les fichiers existent déjà, `tee` détruira leur contenu et le remplacera par celui de l'entrée standard. Si vous souhaitez ajouter des données à la fin d'un fichier au lieu de le remplacer, servez-vous de l'option `-a`.

### 7.2.5 Exercices

1. Vous désirez lister le contenu de répertoire `/bin`. Pour cela vous utilisez la commande `ls /bin`. Cependant, votre objectif n'est pas d'afficher le résultat sur la sortie standard, l'écran, mais d'enregistrer ces informations dans le fichier `bin.txt`. Comment faites-vous ?
2. Comment faites-vous pour afficher les 5 premières lignes du fichier `bin.txt` ? Comment faites-vous pour afficher les 5 dernières ?
3. Vous désirez extraire les 5 premières ligne du fichier `bin.txt` à l'aide de la commande `tail` pour les enregistrer dans le fichier `5pl_bin.txt`. Comment faites-vous ?
4. Vous désirez afficher les 15 premiers fichiers de répertoire `/lib`. Comment faites-vous en utilisant les commandes `ls`, `tail` et un tube ?
5. En plus d'afficher les 15 premiers fichiers de répertoire `/lib` vous voulez les enregistrer dans le fichier `15pl_lib.txt`. Pour cela il vous faudra utiliser un *tee*. Quelle est la ligne de commande à saisir ?

## 7.3 Manipulation des filtres

### 7.3.1 Introduction

Dans la section précédente (section 7.2), nous avons décrit comment la philosophie de développement des outils Unix a conduit à la présence de nombreux programmes, chaque outil réalisant une fonction bien déterminée. Nous avons montré comment se servir de la redirection depuis la source (ou vers la destination) au moyen des entrée/sortie standards, et présenté la manière de construire des pipe-lines, suite de tubes formant ensemble une ligne d'assemblage dans laquelle les données sont acheminées d'outil en outil.

En appliquant les concepts de la section 7.2 et en utilisant les filtres, qui font l'objet du chapitre suivant, vous serez en mesure de définir vos propres outils pour résoudre les problèmes que vous rencontrerez dans votre travail quotidien.

### 7.3.2 Filtres et tubes

La commande suivante vous montre un exemple de pipe-line où les données sont traitées en séquence par 4 programmes :`cat`, `grep`, `sort` et `lpr` :

```
cat newnames oldnames | grep Harley | sort | lpr
```

Vous devriez maintenant être en mesure d'apprécier l'utilité d'un programme conçu pour être une partie d'un pipeline. Un tel programme peut lire des données, effectuer un certain traitement, puis écrire le résultat. Ce type de programme est un *filtre*. Une manière plus formelle de définir un filtre est de le présenter comme un programme qui lit ses, données depuis l'entrée standard et qui écrit ses résultats sur la sortie standard. De manière plus informelle, un filtre ne doit faire qu'une chose, mais doit la faire bien.

Pour un programmeur, la réalisation d'un filtre est une activité assez simple. Vous pouvez vous servir d'un langage de programmation (comme le C++) ou du langage intégré au *shell*. Il vous suffit de vous assurer que le programme lit bien ses données depuis l'entrée standard et écrit ses résultats sur la sortie standard. En d'autres termes, si votre programme lit ses données depuis l'entrée standard et écrit ses résultats sur la sortie standard, il peut faire partie d'un pipe-line.

Remarquez bien que le premier et le dernier programme du pipe-line n'agissent pas vraiment comme des filtres. Dans l'exemple précédent, le programme `lpr` imprime les résultats fournis par la commande `sort`. Nous voyons bien que les résultats de `lpr` ne sont pas affichés sur la sortie standard, mais imprimés (plus précisément, la sortie est redirigée vers un fichier du système qui attendra d'être imprimé). De la même manière, la première commande du pipe-line, `cat` (qui concatène des fichiers), ne lit pas de données depuis l'entrée standard. Dans notre exemple, `cat` lit ses données depuis deux fichiers, `newnames` et `oldnames`.

La figure 7.1 en donne la liste des filtres les plus importants.

### 7.3.3 Filtres et redirections

Un filtre lit ses données depuis l'entrée standard, effectue un certain traitement, puis écrit ses résultats sur sa sortie standard. Le filtre le plus simple est celui qui ne fait aucun traitement, `cat`. `cat` ne fait que recopier l'entrée standard vers la sortie standard.

| Filtre | Fonction |
|---|---|
| `awk` | commande complexe au possibilités nombreuses |
| `cat` | combiner des fichiers ; copier entrée std vers sortie std |
| `cmp` | comparaison de fichiers |
| `colrm` | enlever des colonnes précises dans chaque ligne de données |
| `comm` | comparaison de fichiers |
| `crypt` | encoder ou décoder des données avec une clé |
| `cut` | extraire des portions précises de chaque ligne de données |
| `diff` | différences entre deux fichiers |
| `grep` | extraire les lignes qui contiennent un certain motif |
| `head` | afficher les premières lignes d'un fichier |
| `join` | opération de jointure entre fichiers |
| `less` | afficher les données un écran à la fois |
| `look` | rechercher les lignes commençant par un motif |
| `more` | afficher les données un écran à la fois |
| `paste` | combiner des colonnes de données |
| `pg` | afficher les données un écran à la fois |
| `rev` | inverser l'ordre des caractères d'une ligne de données |
| `sed` | éditeur non interactif |
| `sort` | trier ou combiner des données |
| `spell` | vérifier l'orthographe d'un mot |
| `tail` | afficher la fin d'un fichier |
| `tr` | remplacer ou détruire les caractères spécifiés |
| `uniq` | rechercher les lignes dupliquées |
| `wc` | compter le nombre de lignes, de mots ou de caractères |

FIG. 7.1 – *Liste des filtres les plus importants*

Examinez l'exemple suivant, en entrant la commande `cat`. Le système attend que vous rentriez des données sur l'entrée standard de `cat`. `cat` attend donc que vous tapiez des caractères sur votre clavier (rappelez-vous que l'entrée standard est le clavier). Quand vous pressez <Entrée> à la fin de chaque ligne, celle-ci est envoyée à `cat` qui la copie vers la sortie standard (l'écran). Le résultat est que chaque ligne que vous tapez est affichée deux fois :

```
ligne 1
ligne 1
ligne 2
ligne 2
```

Quand vous avez fini de rentrer du texte, pressez `^D` (le code `eof` de fin de fichier) pour indiquer à Unix qu'il n'y a plus de données disponibles. La commande `cat` se termine et vous revenez au *prompt* du *shell*.

L'intérêt d'un filtre qui ne fait rien est limité, mais vous pouvez tirer avantage du concept d'entrée/sortie standard pour créer rapidement des petits fichiers.

Examinons la commande suivante : `cat > data`. Dans cette commande, l'entrée standard reste le clavier, mais la sortie standard a été redirigée vers le fichier `data`. Chaque ligne que vous rentrez est copiée dans ce fichier. Si le fichier n'existe pas, il est créé. S'il

existe, son contenu est remplacé. Vous pouvez rentrer autant de lignes que vous le souhaitez, et terminer en pressant `^D`. De cette manière, vous pouvez rapidement créer un fichier contenant une petite quantité de données (malheureusement, si vous faites une erreur et que vous pressez <Entrée>, vous devrez retaper la commande et le texte entier).

Si vous voulez ajouter des données à la fin d'un fichier existant, vous pouvez vous servir de `>>` pour rediriger la sortie standard : `cat >> data` (Comme nous l'avons expliqué à la section 7.2.1, la redirection au moyen de `>>` ajoute les résultats de la sortie standard d'un programme à la fin d'un fichier).

`cat` peut aussi servir à afficher un petit fichier. Il vous suffit de rediriger l'entrée standard depuis le fichier que vous voulez visualiser. Ainsi : `cat < data` affiche le contenu du fichier data sur votre écran.

Enfin, `cat` peut aussi servir à copier un fichier, en redirigeant l'entrée standard et la sortie standard. La commande suivante présente un exemple où le fichier `data` est copié dans le fichier `newdata` : `cat < data > newdata`.

Bien sûr, il existe des programmes de pagination (`more` ou `pg`) qui affichent le contenu d'un fichier, des éditeurs de textes (comme `vi` ou `emacs`) qui permettent de créer des fichiers, et le programme `cp` qui effectue des copies de fichiers. Il est néanmoins important de bien comprendre ces exemples, qui vous aident à saisir la puissance de l'entrée/sortie standard. Pour vous en convaincre, examinez bien tout ce que nous avons pu faire avec un filtre qui ne fait rien. A partir d'un concept simple, les données sont communiquées de l'entrée standard vers la sortie standard, un utilisateur peut effectuer simplement une grande variété d'actions.

### 7.3.4 Accroître la puissance des filtres

En permettant à l'utilisateur de spécifier les noms des fichiers d'entrée, il est possible d'améliorer la puissance des filtres de manière significative. Comme vous le savez, la définition stricte d'un filtre suppose que celui-ci lit ses données depuis l'entrée standard. Si vous souhaitez lire des données depuis un fichier, vous devez rediriger l'entrée standard depuis un fichier.

Supposons maintenant qu'il soit possible de lire les données depuis un fichier passé comme paramètre d'une commande, comme dans l'exemple suivant : `cat data` au lieu de `cat < data`. A priori, la différence semble mineure. La ligne de commande devient plus simple, mais cela au détriment de la simplicité du programme `cat`. Celui-ci doit dorénavant être capable de lire ses données depuis l'entrée standard et depuis un fichier. De plus, en enrichissait la commande `cat`, nous perdons la beauté et la simplicité d'un filtre pur.

Sachez cependant que bien des filtres sont modifiés de manière à accepter un fichier en paramètre. La raison n'est pas que cela simplifie la lecture depuis un fichier, mais plutôt que cela autorise la lecture de plusieurs fichiers.

En autorisant la commande `cat` à lire depuis un fichier, nous lui avons automatiquement permis de pouvoir lire depuis plusieurs fichiers. Cela signifie que si l'utilisateur spécifie plusieurs fichiers en paramètre de cette commande, chaque fichier sera à tour de rôle ouvert, lu et écrit vers la sortie standard.

En d'autres termes, `cat` peut servir à concaténer (ajouter le contenu à la suite) les

données de plusieurs fichiers. Examinons les exemples suivants :

```
cat nom adresse telephone
cat nom adresse telephone > info
cat nom adresse telephone | sort
```

Le premier exemple combine les données de trois fichiers (`nom`, `adresse` et `telephone`) et écrit le résultat à l'écran. Le second exemple combine les mêmes données mais les écrit dans le fichier `info`. Le troisième exemple combine les trois fichiers et les communique au programme `sort` au moyen d'un tube.

Nous avons déjà exprimé le fait que les autres filtres, en plus de `cat`, sont aussi en mesure de lire leurs données depuis plusieurs fichiers, mais cette possibilité n'est techniquement pas nécessaire. Si nous souhaitons effectuer un certain traitement sur plusieurs fichiers, il est tout à fait possible de rassembler les données avec `cat` et d'employer un tube en entrée du filtre voulu, comme avec la commande : `cat nom adresse telephone | sort`. Cependant, pour des raisons de confort et au détriment de la simplicité de conception des programmes, les filtres acceptent plusieurs fichiers en paramètre.

# Chapitre 8

# Filtres

Dans la présentation de chaque filtre, rappelez-vous bien que nous ne présentons que le fonctionnement global du filtre. Si vous voulez en savoir plus, ou examiner certains détails ou options particuliers, servez-vous du manuel Unix. Il se peut aussi que certains de ces filtres ne soient pas disponibles sur votre système. Si vous avez des doutes, consultez le manuel ou utilisez la commande `whatis`.

## 8.1 Concaténer des fichiers : `cat`

Le programme `cat` copie des données, sans aucune modification, vers la sortie standard. Les données d'entrée peuvent provenir soit de l'entrée standard, soit d'un ou de plusieurs fichiers.

La syntaxe est :

```
cat [-bns] [fichier...]
```

où `fichier` est le nom d'un fichier.

`cat` sert à concaténer plusieurs fichiers, comme :

```
cat nom adresse telephone
cat nom adresse telephone > info
cat nom adresse telephone | sort
```

La commande `cat` permet d'afficher un ou plusieurs fichiers :

```
cat nom
cat nom adresse telephone
```

Pour créer un fichier : `cat > nouveaufichier`, pour ajouter des données à un fichier existant : `cat >> ancienfichier`, et pour copier un fichier : `cat < data > newdata`

Vous devez absolument éviter de commettre l'erreur courante qui consiste à rediriger la sortie standard vers un des fichiers d'entrée. Si vous voulez ajouter le contenu des fichiers `adresse` et `telephone` au fichier `nom`, vous ne devez pas faire :

```
cat nom adresse telephone > nom
```

pour la bonne raison qu'Unix ouvre le fichier `nom` en écriture avant de lancer la commande `cat`. Le fichier est donc vide avant que `cat` ne combine les données d'entrée, et `cat` ne verra qu'un fichier vide. Si vous entrez une telle commande, vous aurez droit à un message comme : `cat : input name is output` (`cat` : le nom d'entrée est une sortie) mais il sera déjà trop tard. Le contenu original de `nom` sera perdu.

Pour ajouter en toute sécurité les contenus de `adresse` et `telephone` au fichier `nom`, entrez plutôt : `cat adresse telephone >> nom`

L'option `-n` de `cat` ajoute un numéro de ligne avant chaque ligne. L'option `-b` est utilisée en conjonction avec l'option `-n` et indique à `cat` de ne pas numéroter les lignes blanches. L'option `-s` précise à `cat` de remplacer les lignes blanches consécutives par une ligne blanche unique.

Même si vous pouvez visualiser le contenu d'un fichier avec `cat`, vous devez prendre l'habitude d'utiliser un programme de pagination comme `more`, qui affiche un écran de données à la fois. Les utilisateurs aiment bien la commande `cat` car le nom est simple et facile à taper, mais vous n'aurez jamais le temps de lire les données avant qu'elles n'aient disparu de l'écran après défilement.

La plupart des utilisateurs pensent que `cat` tire son origine du terme *concaténer*, mais cela n'est pas le cas. Le terme `cat` vient du latin *catena*, qui signifie chaîne.

**Exercices**

1. Vous désirez afficher la liste des fichier du répertoire `/bin`. Quel est la commande ?
2. Lorsque vous tapez la commande précédente, vous n'avez pas le temps de voir tous les fichiers. Comment faites-vous pour afficher la liste page par page ?
3. Vous désirez en plus que chaque fichier soit numéroté pour pouvoir savoir à chaque page à combien de fichiers vous en êtes et à la fin combien de fichier en tout ont étés listés. Quelle commande devez vous taper ?

## 8.2 Comparaison de fichiers

### 8.2.1 Commande : `cmp`

Il est parfois nécessaire de connaître les différences entre 2 versions d'un fichier. des trois commandes de comparaison de fichiers (`cmp`, `comm` et `diff`), `cmp` est la plus basique. Elle indique si deux fichiers sont identiques. En tapant :

```
cmp <fichier1> <fichier2>
```

Si les deux sont identiques, la commande ne génère aucune sortie, s'ils sont différents la commande indique la position de la première différence (ligne et caractère), avec une sortie du genre :

```
fichier1 fichier2 differ : char 34, line 2
```

### 8.2.2 Commande : `comm`

La `comm` est un peu plus évolué que `cmp`, elle recherche les lignes identiques à 2 fichiers.
Syntaxe :

```
comm [-123] <fichier1> <fichier2>
```

Sa sortie s'effectue sur 3 colonnes. La première contient les lignes uniques du premier fichier, la seconde les lignes uniques du deuxième fichier et la troisième les lignes communes. Elle dispose de paramètres qui précisent si l'on veut éliminer une de ces colonnes.

`-1`, `-2` et `-3` indiquent à `comm` de ne pas afficher la première, la seconde ou la troisième colonne.

### 8.2.3 Différences entre deux fichiers : `diff`

Cette commande permet de rechercher les différences entre deux fichiers. La syntaxe est la suivante :

```
diff [-biw] <fichier1> <fichier2>
```

`diff` fait en sorte de vous donner des indications pour que le `fichier1` soit identique au `fichier2`.

Les options principales sont les suivantes :

**-b :** ignore les espaces.

**-i :** contrairement à la plupart des commandes, l'option `-i` ne signifie pas interactif mais ignore-case, c'est à dire que `diff` ne fera pas de différence entre minuscules et majuscules.

**-w :** ignore les espaces et les tabulations.

#### Exercice

Créez un fichier, nommé `texte1.txt`, contenant un petit texte. Faites une copie de ce fichier vers un fichier nommé `texte2.txt`. Apportez quelques modifications à `texte2.txt` à l'aide de l'éditeur de votre choix. Expérimentez ensuite les trois commandes `cmp`, `comm` et `diff`.

## 8.3 Manipuler des colonnes de données

### 8.3.1 Détruire des colonnes de données : `colrm`

La commande `colrm` lit ses données depuis l'entrée standard, détruit les colonnes de texte spécifiées puis écrit le résultat sur la sortie standard.

La syntaxe de cette commande est :

```
colrm [coldebut [colfin]]
```

où `coldebut` et `colfin` indiquent l'intervalle des colonnes devant être détruites. La numérotation commence à la colonne 1.

Supposons que vous êtes un professeur. L'examen est terminé et vous disposez d'un fichier contenant la liste des notes, comme celui-ci, appelé étudiants :

```
123-45-6789  Barton, Barbara  10  P
234-56-7890  Canby, Charles   17  TB
345-67-8901  Danfield, Anne   15  B
```

Chaque ligne de ce fichier contient le numéro de l'étudiant, son nom, la note et la mention.

Vous voulez créer à partir de ce fichier une nouvelle liste où ne figurent pas les noms des étudiants. Pour constituer cette liste, entrez : `colrm 14 30 < etudiants`. La sortie est alors :

```
123-45-6789  10  P
234-56-7890  17  TB
345-67-8901  15  B
```

Pour imprimer cette liste, vous n'avez qu'à établir un tube avec le programme `lpr` : `colrm 14 30 < etudiants | lpr`.

Si vous ne spécifiez que la colonne de départ, `colrm` détruit toutes les données comprises entre cette colonne et la fin de ligne. Ainsi : `colrm 14 < etudiants` affiche le résultat suivant :

```
123-45-6789
234-56-7890
345-67-8901
```

Si vous ne spécifiez aucune colonne en paramètre, `colrm` n'aura aucun effet.

**Exercices**

La commande `ls -l` affiche un grand nombre d'informations. Vous voulez lister les fichiers du répertoire `/lib` et stocker le résultat dans le fichier `lib.txt`. Cependant, seul la taille et le nom des fichiers vous intéresse et vous ne désirez pas que le fichier `lib.txt` contienne autre chose. Quelle commande, utilisant le filtre `colrm`, tapez-vous ?

### 8.3.2 Extraire des colonnes précises dans chaque ligne : `cut`

La commande `cut`, très commode, extrait des colonnes de données d'un flux de caractères d'entrée. Vous pouvez extraire des colonnes particulières d'une ligne ou des parties d'une ligne appelées des champs. Si vous êtes un expert en base de données, vous pouvez considérer `cut` comme la réalisation sous Unix de la projection d'une relation.

La syntaxe de `cut` est la suivante :

```
cut [-cliste] [fichier...]
```

où `liste` est une liste de colonnes à extraire et `fichier` le nom du fichier d'entrée.

La liste sert à indiquer à `cut` les champs ou colonnes à extraire. Si vous ne voulez récupérer que la colonne 10, rentrez simplement `10`. Pour extraire les colonnes 1, 8 et 10, entrez `1,8,10`. Vous pouvez aussi spécifier un intervalle de colonnes au moyen d'un trait d'union. Pour extraire les colonnes 10 à 15, entrez `10-15`. Pour extraire les colonnes 1,8 et 10 à 15, entrez `1,8,10-15`.

Pour donner un exemple d'utilisation de la commande `cut`, imaginons que vous disposez d'un fichier nommé info contenant des informations sur un groupe de personnes. Chaque ligne contient des données pertinentes sur un individu. En particulier, les colonnes 14 à 30 contiennent le nom et les colonnes 42 à 49 contiennent le numéro de téléphone. Le texte suivant est extrait du fichier `info` :

```
123-45-6789  Barton, Barbara  01/01/72  04 29 38 47 36
234-56-7890  Canby, Charles   02/02/73  05 13 24 35 46
345-67-8901  Danfield, Anne   03/03/74  06 59 72 27 95
```

Pour n'afficher que les noms, entrez : `cut -c14-30 info` et vous obtiendrez :

```
Barton, Barbara
Canby, Charles
Danfield, Anne
```

Pour afficher les noms et numéros de téléphone, entrez : `cut -c14-30,41-55 info` et vous aurez :

```
Barton, Barbara  04 29 38 47 36
Canby, Charles   05 13 24 35 46
Danfield, Anne   06 59 72 27 95
```

Pour sauvegarder ces informations, vous pouvez rediriger la sortie vers un fichier :

```
cut -c14-30,41-55 info > listetelephone
```

Si vous souhaitez modifier l'ordre des colonnes d'une table, utilisez `cut` en conjonction avec `paste` (décrite plus loin dans ce chapitre).

**Exercices**

Même exercice que celui de la question précédente, mais un utilisant le filtre `cut`.

### 8.3.3 Combiner des colonnes de données : `paste`

La commande `paste` combine des colonnes de données, et est très utile, car elle vous aide à constituer rapidement une grande table à partir de plusieurs plus petites. Vous pouvez aussi combiner des lignes de données consécutives pour construire plusieurs colonnes.

La syntaxe de la commande est :

```
paste [-d car] [fichier...]
```

où `car` est le caractère à utiliser comme séparateur et `fichier` le nom du fichier d'entrée.

Par défaut, `paste` insère une tabulation entre chaque entrée de colonne. Unix fait l'hypothèse que les tabulations sont aux positions 1, 9, 17, 25, etc. Si vous voulez insérer un autre caractère entre les colonnes, il faut utiliser l'option `-d` (délimiteur) suivie du caractère entre apostrophes.

Prenons, par exemple, les trois fichiers suivant :

| num | noms | notes |
|---|---|---|
| 123-45-6789 | Barton, Barbara | 10 |
| 234-56-7890 | Canby, Charles | 17 |
| 345-67-8901 | Danfield, Anne | 15 |

La commande `paste num noms notes` produira :

```
123-45-6789     Barton, Barbara 10
234-56-7890     Canby, Charles  17
345-67-8901     Danfield, Anne  15
```

tandis que la commande `paste -d ';' num noms notes` produira :

```
123-45-6789;Barton, Barbara;10
234-56-7890;Canby, Charles;17
345-67-8901;Danfield, Anne;15
```

## 8.4 Manipuler des lignes de donnée

### 8.4.1 Extraire les lignes commençant par un motif : `look`

La commande `look` parcourt les données d'entrée triées par ordre alphabétique et affiche toutes les lignes qui commencent par un modèle donné.

La syntaxe de look est la suivante :

```
look [-df] motif [fichier...]
```

où `motif` est le motif à rechercher et `fichiez` le nom d'un fichier.

`look` ne peut pas lire ses données depuis l'entrée standard (`look` n'est donc pas un filtre standard et ne peut être utilisé dans un pipe-line). La raison est que `look` s'appuie sur une méthode de recherche particulière, la recherche binaire, qui nécessite un accès simultané à toutes les données (or, avec l'entrée standard, vous ne pouvez accéder qu'à une ligne de données à la fois). Pour pallier à ce problème, il suffit de préparer vos données, de les sauvegarder dans un fichier, puis d'utiliser `look`.

Deux options de `look` permettent de contrôler la comparaison. L'option `-d` (dictionnaire) demande à `look` de ne considérer que les caractères (majuscules et minuscules), les chiffres, les tabulations et les espaces.

L'option `-f` indique à `look` d'ignorer la distinction entre majuscules et minuscules.

La seconde manière d'utiliser `look` est de ne spécifier que le motif, sans fichier en paramètre. Dans ce cas, `look` examine le fichier `/usr/dict/words` contenant un dictionnaire des mots anglais, et qui est utilisé par la commande `spell` (présentée plus loin). Comme ce fichier est trié, look peut l'utiliser de manière efficace. Il vous suffit d'employer la commande : `look -df motif /usr/dict/words` pour rechercher le motif `motif` dans ce dictionnaire. Comme `-df` est l'option par défaut et `/usr/dict/words` le fichier par défaut, il vous suffit en réalité de rentrer :`look motif`. Si vous ne savez pas quelle est l'orthographe exacte de *simultaneous* (Le dictionnaire ne comprend que les mots anglais), vous pouvez rentrer : `look simu`. Le résultat sera la liste suivante :

```
Simula
Simulate
Simulated
Simulates
Simulating
Simulation
Simulations
Simulator
Simulators
simulcast
simultaneity
simultaneous
simultaneously
```

### 8.4.2 Extraire les lignes qui contiennent un motif : `grep`

Une fois lue cette section, n'oubliez pas de lire la section 8.9 sur les expressions régulières, qui vous permettent de spécifier des motifs de recherche pour `grep`.

La commande `grep` cherche toutes les lignes dans un ensemble de données qui contiennent un motif spécifié puis écrit celles-ci sur la sortie standard.

La commande `grep` appartient en fait à une famille de commandes, dont les autres membres sont `fgrep` et `egrep`. A l'époque où `grep` a été développé, la mémoire et la vitesse de calcul du processeur étaient limités. Chaque programme de cette famille a été développé pour résoudre l'une on l'autre de ces limitations.

`grep` a été conçu pour être un programme d'usage général. Il peut chercher parmi des données un motif composé de caractères connus ou d'expressions plus complexes comme «H[a-z]*» qui correspond à une chaîne de caractères commençant par un `H` majuscule suivi de plusieurs (ou d'aucune) lettres contenues dans l'intervalle [`a` *minuscule*, `z` *minuscule*].

Le nom `grep` est un acronyme de *Global Regular Expression Print* (affichage d'expression régulière globale). *global* nous indique que `grep` cherche un modèle parmi toutes les données en entrée. `print` signifie traditionnellement sous Unix *affichage*. Le nom `grep` signifie donc que cette commande cherche toutes les occurrences d'une expression régulière sur sa source d'entrée, puis affiche les résultats. Avec l'éditeur de texte `ed`, la commande a exécuter pour effectuer une telle recherche était `g/`, suivi de la *regular expression* (expression régulière), suivi de `/p`. Autrement dit, la commande `ed` était : `g/re/p` où `re` est une expression régulière. Le nom reprend directement la syntaxe de la commande `ed`.

`fgrep` a été conçu pour être rapide, et pour cette raison, n'accepte que les motifs constitués de caractères connus ou exacts. Le nom `fgrep` signifie *fixed characteres grep* (`grep` à caractères fixes).

`egrep` a été conçu pour être le programme le plus performant. Il peut chercher des motifs plus complexes que `grep`, et est généralement le plus rapide de la famille. L'inconvénient de cette commande est qu'elle consomme plus de mémoire que `grep` ou `fgrep`. Le nom `egrep` signifie *extended grep* (`grep` étendu).

Aujourd'hui, les ordinateurs sont plus rapides et disposent de plus de mémoire, `fgrep` est tombé en désuétude et les utilisateurs expérimentés considèrent que `egrep` est le bon choix. La plupart des utilisateurs continuent cependant à utiliser `grep` pour deux raisons : la première est que `grep` est traditionnellement connu, la deuxième qu'il est plus court que `egrep`.

La syntaxe de `grep` est :

```
grep [-cilnvw] <motif> [fichier...]
```

où `motif` est le motif à rechercher et `fichier` le nom du fichier d'entrée.

`grep` lit toutes les données d'entrée et sélectionne toutes les lignes qui contiennent le motif spécifié. Aucun message n'est affiché si `grep` ne trouve pas de ligne contenant le motif. Au même titre que les autres commandes Unix, `grep` est concis. S'il n'a rien à dire, il ne dit rien.

Quand vous spécifiez un motif contenant des signes de ponctuation ou des caractères spéciaux, prenez bien soin de les mettre entre apostrophes pour que le *shell* interprète la commande correctement. Pour chercher le motif «: ␣» (deux points suivi d'un espace) dans le fichier `info`, vous devez entrer : `grep ': ' info`.

La puissance de la commande `grep` vient du fait qu'il est possible de spécifier, outre des chaînes de caractères fixes, des expressions de motif plus généraux. Vous devez utiliser pour cela des *expressions régulières*. Comme les expressions régulières sont une caractéristique très importante du système Unix, nous les présenterons à part (section 8.9).

Dans un pipe-line, la commande `grep` sert à réduire les données brutes en un petit ensemble de données utiles.

La commande `grep` dispose de plusieurs options intéressantes. L'option `-c` (compte) permet d'afficher le nombre de lignes extraites, plutôt que les lignes elles-mêmes.

L'option `-i` indique à `grep` de ne pas tenir compte de la case (vous verrez plus loin que pour les commandes `look` et `sort`, l'option équivalente est `-f`).

L'option `-n` affiche un numéro de ligne relatif au début de chaque ligne affichée. Il est inutile que vos données d'entrée contiennent des numéros de ligne, car `grep` les compte au fur et à mesure de la recherche. Cette option est utile quand vous effectuez des recherches dans un grand fichier. Si vous devez modifier les lignes recherchées avec un éditeur, les numéros de ligne peuvent vous être très utiles pour trouver rapidement les données.

L'option `-l` (liste) est pratique quand vous cherchez un même motif dans plusieurs fichiers. Cette option permet d'afficher le nom des fichiers contenant le motif pendant le traitement.

L'option `-w` indique à `grep` de ne faire des recherches que sur des mots complets.

Considérons l'exemple du fichier `memo` suivant :

```
Unix est un systeme
que chacun peut utiliser
comme il l'entend.
```

Supposons que vous recherchez toutes les lignes qui contiennent le mot «un». Si vous entrez : `grep un memo` le résultat sera :

```
Unix est un systeme
que chacun peut utiliser
```

`grep` n'a donc pas fait la distinction entre «un» et «chacun». Si maintenaut vous entrez : `grep -w un memo` vous obtiendrez :

```
Unix est un systeme
```

Enfin, l'option `-v` vous permet de sélectionner les lignes qui ne contiennent pas le motif.

**Exercices**

N'oubliez pas de lire la section 8.9 sur les expressions régulières !

1. Comment feriez-vous pour dresser la liste des fichiers qui contiennent un `a` dans le répertoire `/bin` ?
2. Ceux qui commencent par un `a` ?
3. Ceux qui contiennent un `a` et un `e` dans cet ordre ?
4. Comment feriez-vous pour trouver les noms des fichiers stockés dans le répertoire `/bin` composés de deux lettres (avec une commande `ls` et une `grep`) ?
5. Même question, mais ce qui vous intéresse, c'est le nombre de ces fichiers (toujours avec une commande `ls` et une `grep`).

### 8.4.3 Trier et combiner des lignes de données : `sort`

La commande `sort` a deux fonctions. Elle permet d'une part de trier des données, et d'autre part de lire et de combiner des données précédemment triées en un seul grand

fichier. La commande `sort` est très utile pour trier des fichiers de données ou dans un pipe-line.

`sort` vous permet de comparer des lignes entières ou des parties de lignes appelés champs.

La syntaxe de cette commande pour trier des données est :

```
sort[-dfnru] [-o fichiersortie] [fichierentrée...]
```

où `fichiersortie` est le fichier qui contiendra le résultat du tri et `fichierentrée` le nom du fichier qui contient les données d'entrée.

La syntaxe de la commande sort pour combiner des données est :

```
sort -m [-o fichiersortie] <fichiertrié...>
```

où `fichiersortie` est le fichier qui contiendra le résultat trié et `fichiertrié` le nom du fichier qui contient des données triées.

La manière la plus simple d'utiliser la commande sort est de trier un seul fichier et d'afficher le résultat sur votre écran. Si le fichier `noms` contient les données suivante :

```
Danfield, Anne
Barton, Barbara
Canby, Charles
```

vous pouvez le trier par ordre alphabétique en rentrant la commande `sort noms`. Pour sauvegarder le résultat du tri dans un fichier `liste`, vous pouvez rediriger la sortie standard : `sort noms > liste`.

Dans certaines situations, vous voudrez petit-être sauvegarder le résultat du tri dans le fichier contenant les données originales non triées. Malheureusement, nous avons vu que la commande suivante : `sort noms > noms` ne fonctionne pas car le fichier `noms` est à la fois source et destination. Pour résoudre ce problème, `sort` dispose d'une option spéciale, `-o`, qui permet de sauvegarder le résultat du tri dans un fichier. Aussi, pour trier le fichier et sauvegarder les résultats dans le même fichier, vous pouvez rentrer : `sort -o noms noms`. Cette commande remplace le fichier original `noms` par les données triées.

Si vous devez combiner les données triées de plusieurs fichiers, il vous suffit de spécifier leurs noms. Pour trier les données des fichiers `noms`, `vieuxnoms` et `nouveauxnoms` et sauvegarder le résultat dans le fichier `liste`, entrez :

```
sort noms vieuxnoms nouveauxnoms > liste
```

Pour trier ces mêmes fichiers mais sauvegarder les résultats dans le fichier `noms`, utilisez l'option `-o` :

```
sort -o noms noms vieuxnoms nouveauxnoms
```

Par défaut, les données sont triées par ordre croissant suivant le code ASCII. Le code ASCII contient une description des 128 caractères différents qui composent l'alphabet d'Unix. Cet ensemble contient les lettres majuscules et minuscules, les chiffres, les signes de ponctuation et d'autres symboles. Sont aussi inclus l'espace, la tabulation et les caractères de contrôle. Il est important de noter que l'alphabet d'Unix classe les caractères par ordre croissant comme l'alphabet classique et que c'est cet ordre qui est utilisé pour les comparaisons lors du tri. L'ordre des caractères est le suivant :

- les caractères de contrôle
- le caractère espace
- les symboles `! " # $ % £ ' ( ) * + , - . /`

- les chiffres `0 1 2 3 4 5 6 7 8 9`
- d'autres symboles `; < = > ? @`
- les majuscules `A B C ... Z`
- d'autres symboles `[ \ ] ^ _ ‘`
- les minuscules `a b c ... z`
- d'autres symboles `{ | } ~`

De cette liste, mémorisez surtout que l'espace est avant les chiffres, que les chiffres sont avant les majuscules et que les majuscules sont avant les minuscules.

La commande sort dispose de certaines options qui affectent l'ordre de tri.

L'option `-d` (dictionnaire) indique à sort de ne considérer que les lettres, les chiffres et les espaces. Tous les autres caractères sont ignorés.

L'option `-f` permet de ne pas tenir compte de la case.

L'option `-n` reconnaît les nombres en début de ligne et les trie par ordre numérique. Ces nombres peuvent inclure des espaces, des signes `-` ou des points de décimale (aux USA, la virgule d'un nombre réel est remplacée par un point).

L'option `-r` demande à sort de trier les données dans l'ordre inverse (décroissant).

Enfin, l'option `-u` remplace toutes les occurrences d'une ligne identique par une occurrence simple.

`sort` permet aussi de combiner des fichiers contenant des données triées. Pour cela, utilisez l'option `-m`.

**Exercices**

Vous désirez obtenir un listing, de tous les fichiers présent dans les répertoire `/lib` et `/bin`, trié par ordre alphabétique. Comment faites-vous?

### 8.4.4 Rechercher les lignes répétées : `uniq`

La commande `uniq` examine les données d'entrée ligne par ligne et détermine les lignes consécutives dupliquées. `uniq` fonctionne selon 4 modes :

- ne retenir que les lignes dupliquées,
- ne retenir que les lignes uniques,
- éliminer les lignes dupliquées,
- compter l'indice de répétition des lignes.

En effectuant ses comparaisons, `uniq` considère soit la ligne entière, soit des parties de cette ligne. Nous ne décrirons ici que la comparaison sur les lignes entières.

La syntaxe de la commande `uniq` est :

```
uniq [-cud] [fichierentrée] [fichiersortie]
```

où `fichierentrée` est un fichier d'entrée et `fichiersortie` le nom d'un fichier de sortie.

L'option `-d` permet de ne retenir que les lignes dupliquées qui sont consécutives.

L'option `-u` (unique) permet de ne retenir que les lignes non dupliquées.

Sans options, `uniq` fonctionne comme si `-d` et `-u` étaient utilisées, ce qui permet d'éliminer les lignes dupliquées.

L'option `-c` permet de compter l'indice de répétition des lignes.

Le véritable intérêt de la commande `uniq` est qu'elle peut être insérée dans un pipe-line contenant des données triées. Quand les données sont triées, toutes les lignes dupliquées sont consécutives.

Supposons que vous disposez de deux fichiers `c1` et `c2` correspondant aux noms des étudiants qui suivent deux de vos cours. Pour afficher la liste de ceux qui suivent les deux cours, entrez : `sort c1 c2 | uniq -d`. Pour connaître ceux qui ne suivent qu'un cours, faites : `sort c1 c2 | uniq -u`. Pour avoir la liste des étudiants, sans doublons, entrez : `sort c1 c2 | uniq` (mais vous auriez aussi pu utiliser `sort -u c1 c2`). Enfin, pour afficher une liste d'étudiants et du nombre de cours qu'ils suivent, entrez : `sort c1 c2 | uniq -c`.

**Exercice**

Imaginons que vous ayez un fichier `noms.txt` dont le contenu est le suivant :

```
Anne
Barbara
Charles
Anne
Charles
```

Quelle commande taperiez-vous pour éliminer les noms en doublon ?

### 8.4.5 Inverser l'ordre des caractères d'une ligne : `rev`

La commande `rev` permet d'intervertir l'ordre des caractères dans chaque ligne disponible sur l'entrée standard. Les données peuvent provenir du clavier ou d'un fichier.

La syntaxe de cette commande est :

```
rev [fichier...]
```

où `fichier` est le nom d'un fichier.

La sortie de `rev` est disponible sur la sortie standard, et le fichier original n'est jamais modifié.

Supposons que vous disposez du fichier `data` suivant :

```
12345
abcde
AxAXA
```

Si vous entrez `rev data` vous obtiendrez :

```
54321
edcba
AXAxA
```

### 8.4.6 Remplacer ou détruire des caractères d'une ligne : `tr`

La commande `tr` lit les données sur l'entrée standard et remplace les caractères spécifiés par d'autres caractères. Elle permet aussi de détruire certains caractères d'une ligne. Avec `tr`, vous pouvez facilement remplacer les caractères majuscules par des minuscules, ou détruire toutes les parenthèses.

La syntaxe de `tr` est :

```
tr [-cds] [ensemble1 [ensemble2]]
```

où `ensemble1` et `ensemble2` sont des ensembles de caractères.

`tr` fonctionne en lisant les données sur l'entrée standard et en vérifiant si les caractères lus appartiennent à `ensemble1`. Quand `tr` rencontre un caractère appartenant à `ensemble1`, il le remplace par le caractère correspondant dans `ensemble2`.

Imaginons que vous avez stocké des informations dans le fichier `olddata`. Vous voulez remplacer tous les caractères `a` par `A` et sauvegarder le résultat dans un fichier `newdata`. Pour cela, entrez : `tr a A < olddata > newdata`.

En définissant des ensembles de données plus conséquents pour `ensemble1` et `ensemble2`, vous pouvez remplacer simultanément plusieurs caractères.

La commande suivante permet ainsi de remplacer `a` par `A`, `b` par `B` et `c` par `C`: `tr abc ABC < olddata > newdata` Si `ensemble2` contient moins de caractères que `ensemble1`, le dernier caractère d'`ensemble2` est dupliqué autant de fois que nécessaire.

Si vous voulez préciser des caractères qui ont une signification spéciale pour le *shell*, n'oubliez pas de les insérer entre apostrophes. Les caractères entre apostrophes sont traités littéralement. Si vous souhaitez remplacer dans un texte tous les deux-points, point-virgule et point d'interrogation par un simple point, entrez :`tr ':;?' '.' < olddata > newdata`.

Vous avez aussi la possibilité de définir un intervalle de caractères au moyen d'un trait d'union. `a-z` spécifie ainsi l'ensemble des lettres comprises entre `a` et `z`.

La commande suivante permet donc une conversion de majuscules en minuscules : `tr A-Z a-z < olddata > newdata`

Si vous voulez utiliser un caractère qui n'est pas facilement accessible, vous pouvez rentrer directement son code ASCII. A chaque code ASCII correspond un nombre de trois chiffres. Ce nombre indique la position dans le code ASCII, exprimée en octal (base 8). Pour utiliser une telle valeur, rentrez juste le caractère  suivi des trois chiffres du code.

Vous utiliserez principalement les codes suivants :

| Nom | Valeur en octal |
|---|---|
| *backspace* | 010 |
| tabulation | 011 |
| fin de ligne | 012 |

Pour remplacer toutes les tabulations d'un fichier `olddata` par des espaces et ainsi créer un fichier `newdata`, vous rentrerez : `tr '\011' ' ' < olddata > newdata`.

La commande `tr` dispose de trois options modifiant le traitement des données. L'option `-d` permet de détruire les caractères spécifiés. Avec l'option `-d`, seul un ensemble de caractères est nécessaire.

L'option `-s` permet de remplacer toutes les occurrences de caractères répétés par une seule occurrence. Ainsi, la commande : `tr -s ' ' ' ' < olddata > newdata` permet de remplacer tous les espaces consécutifs du texte `olddata` par un seul espace.

Enfin, l'option `-c` demande à `tr` de remplacer tous les caractères qui ne sont pas dans `ensemble1`. `-c` signifie "complément". La théorie mathématique des ensembles définit ainsi tous les éléments qui ne font pas partie d'un ensemble.

**Exercice**

Supposons que vous disposez d'un fichier `texte.txt` dont le contenu est le suivant :

```
Au lever, je fais le voeu de renoncer cette année, au plus possible
de choses, et, au déjeuner, je commence par me bourrer d'oie aux
marrons et de galettes de plomb. Et l'histoire d'une ferme, est-ce
que vous ne croyez pas que, malgré Virgile, il reste à la faire ?
```

comment ferriez-vous pour segmenter ce texte en supprimant toute ponctuation et en ne conservant qu'un mot par ligne ?

## 8.5 Editeur non interactif : `sed`

### 8.5.1 Présentation et syntaxe générale

`sed` est une évolution de l'éditeur `ed` lui même précurseur de `vi`. `sed` est un éditeur non interactif. Cette commande permet d'appliquer un certain nombre de commandes sur un fichier (ou sur l'entrée standard) puis d'en afficher le résultat (sans modification du fichier de départ) sur la sortie standard.

La syntaxe de `sed` est la suivante :

```
sed [-n] [-e commandes] [-f fichier_de_commandes] [fichiers]
```

**-n :** écrit seulement les lignes spécifiées (par l'option /p) sur la sortie standard, sinon `sed` écrit tout même les lignes non traitées.

**-e :** permet de spécifier les commandes à appliquer sur le fichier. Cette option est utile lorsque vous appliquez plusieurs commandes. Afin d'éviter que le shell interprète certains caractères, il faut mieux encadrer la commande avec des ' ou des " .

**-f :** les commandes sont lu à partir d'un fichier.

Pour chaque ligne, on applique les commandes (si cela est possible) puis on affiche sur la sortie standard la ligne modifiée ou non.

La syntaxe générale des commandes est de la forme `<lignes><action>`. Le champ `<lignes>` permet de préciser les lignes sur lesquelles porteront l'action de la commande ce qui permet de restreindre son champ d'action.

La commande peut également être de la forme `<lignes>!<action>`. Dans ce cas, la commande ne porte pas sur les lignes désignées par `<lignes>` mais sur toutes les lignes à l'exclusion des lignes désignées par `<lignes>`.

La désignation des lignes `<lignes>` se fait de la façon suivante :

***rien* :** toutes les lignes.

***n* :** la ligne *n*.

***n1, n2*** les lignes entre les lignes *n1* et *n2*.

***regex* :** les lignes qui contiennent une sous-chaîne décrites par l'expression régulière *regex*.

***regex1, regex2* :** les lignes entre la première ligne qui contiennent une sous-chaîne décrites par l'expression régulière *regex1* et la première ligne qui contiennent une sous-chaîne décrites par l'expression régulière *regex1*.

Les actions `<action>` possibles sont multiples nous en décrirons deux dans les sections qui suivent. Pour une information plus détaillée, reportez vous à la commande `man`, faites une recherche sur internet ou reportez vous à un ouvrage spécialisé.

### 8.5.2 La commande de substitution s

**Syntaxe**

`<lignes>s/regex/chaine/flags`. Cette commande remplace la portion décrite par `regex` par la chaîne de remplacement `chaine` en se restreignant aux lignes ciblées par `<lignes>`. `flags` peut prendre plusieurs valeurs :

- `g` pour global, c'est à dire toutes les portions décrite par `regex` ;
- par défaut (sans `flags`) seule la première portions décrite par `regex` est remplacée ;
- `nb` pour remplacer la `nb`$^{e}$ portions décrite par `regex` ;
- `p` pour que seules les lignes ayant subies une modification soit écrites sur la sortie standard (utile avec l'option -n) ;
- `w fichier` écrit la ligne dans le fichier spécifié en plus de la sortie standard.

**Exemples**

Tous ces exemples écrivent leur résultat sur la sortie standard et opèrent leur traitement depuis le fichier `fichier`.

- `sed "s/toto/TOTO/" fichier` : va changer la première occurrence, et uniquement celle-ci, de la chaîne `toto` par `TOTO`.
- `sed "s/toto/TOTO/3" fichier` : va changer la troisième occurrence, et uniquement celle-ci, de la chaîne `toto` par `TOTO`.
- `sed "12,59s/toto/TOTO/g" fichier` : va changer toutes les occurrences de la chaîne `toto` des lignes 12 à 59 par `TOTO` (les chaînes `toto` qui ne se trouvent pas sur les lignes 12 à 59 ne sont pas concernées).
- `sed "s/toto/TOTO/p" fichier` : dans le cas où un remplacement dans la ligne est effectué, la ligne concernée est affichée sur la sortie standard.
- `sed "s/toto/TOTO/w resultat" fichier` : dans le cas où un remplacement dans la ligne est effectué, la ligne concernée est inscrite dans un fichier `resultat`.

### 8.5.3 La commande de suppression d (*delete*)

**Syntaxe**

`<lignes>d`. Cette commande efface les lignes ciblées par `<lignes>`.

**Exemples**

- `sed "20,30d" fichier` : cette commande recopie sur la sortie standard le contenu du fichier `fichier` en excluant les lignes 20 à 30 du fichier `fichier`.
- `sed "/toto/d" fichier` : on peut très bien utiliser les expressions régulières, ici cette commande supprime (sur la sortie standard) les lignes contenant la chaîne `toto`.
- `sed "/toto/!d" fichier` : permet, au contraire, de ne conserver que les lignes contenant la chaîne `toto` (toutes les autres sont supprimées).

## 8.6 Opération de jointure : `join`

La commande unix `join` permet de faire une jointure (au sens base de donnée du terme) de deux fichiers. Il s'agit en faite de fusionner les lignes de deux fichiers triés en se basant sur une colonnes de données particulière dans chacun des deux fichiers.

La syntaxe de la commande est :

```
join [options] <fichier1> <fichier2>
```

`join` fusionne les lignes en correspondance de deux fichier, `fichier1` et `fichier2`, qui contiennent des colonnes de données, parfois appelées champs, triées en utilisant la même règle. Le résultat est affiché sur la sortie standard. Par défaut, les champs sur lesquels se base l'opération de jointure sont les premiers (ie. les premières colonnes) de chaque fichier. Pour chaque ligne en correspondance (dont le premier champ est identique dans les deux fichiers), la commande `join` écrit le champ commun suivi de tous les champs du premier fichier `fichier1` suivi de tous les champs du deuxième fichier `fichier2`.

Voici une liste des options les plus utiles :

`-a no_fic :` affiche également les lignes non appariées du fichier `no_fic` (`no_fic`=1 ou `no_fic`=2);

`-e EMPTY :` remplace les champs manquant par `EMPTY` ;

`-i` ne tient pas compte de la case lors de la comparaison des champs ;

`-j CHAMP :` équivalent à `-1 CHAMP -2 CHAMP` ;

`-t CHAR :` utilise le caractère `CHAR` comme délimiteur de champs ;

`-1 CHAMP :` opère la jointure en utilisant la colonne `CHAMP` du premier fichier ;

`-2 CHAMP :` opère la jointure en utilisant la colonne `CHAMP` du deuxième fichier.

`-o FORMAT :` les champs du fichier de sortie apparaîtront dans un ordre définit par `FORMAT`. Le format de `FORMAT` étant une suite de `noFichier.noChamps` (par exemple, pour afficher sur chaque ligne, l'information de la première puis de la deuxième colonne du premier fichier suivie de l'information de la deuxième colonne du deuxième fichier il faut spécifier : `-o 1.1 1.2 2.2`).

`-v no_fic :` comme `-a no_fic` mais n'affiche pas les lignes appariées.

Par défaut (si l'option `-t CHAR` n'a pas été utilisée), le délimiteur de champs est le caractère espace. Les champs (colonnes) des fichiers sont numérotée à partir de 1.

### Exemple

Dans cet exemple, on suppose disposer de deux fichiers `test1` et `test2` :

| `test1` contient : | `test2` contient : |
|---|---|
| `andrea 92 A` | `andrea 89 B+` |
| `bobby 87 B+` | `bobby 94 A` |
| `zach 90 A-` | `zach 84 B` |

Une opération de jointure sur ces deux fichiers donnera :

```
$ join test1 test2
andrea 92 A 89 B+
bobby 87 B+ 94 A
zach 90 A- 84 B
```

## 8.7 La commande awk

Le nom awk provient du nom de ses créateurs: Alfred Aho, Brian Kernighan et Peter Weinberger.

L'utilitaire awk d'unix peut servir pour une multitude de tâches, mais il est particulièrement efficace lorsqu'il traite et manipule des fichiers de données formatés, tels que des bases de données de fichiers plats ou des fichiers de tableurs.

Bien que nous utilisions le terme commande awk, awk n'est pas le genre de chose qui est habituellement appelé une commande. En fait awk est un langage de programmation, avec une syntaxe proche du C à beaucoup d'égards. C'est un langage interprété et l'interpréteur awk traduit des instructions.

En raison de sa complexité, nous n'étudierons pas cette commande dans ce chapitre.

## 8.8 Quelques commandes particulières

### 8.8.1 Compter les lignes, les mots et les caractères : wc

La commande wc permet de compter les lignes, les mots et les caractères appliqués en entrée. Les données peuvent provenir de l'entrée standard ou d'un fichier.

La syntaxe de la commande est :

```
wc [-lwc] [fichier...]
```

où fichier est le nom d'un fichier.

Le fonctionnement de wc est simple. Il ne fournit que trois nombres : le nombre de lignes, le nombre de mots et le nombre de caractères. Si vous précisez un nom de fichier, wc écrit le nom de celui-ci après les trois nombres, si vous donnez plusieurs fichiers en paramètre, wc vous donne un tableau récapitulatif.

Pour wc, un mot est une séquence de caractères délimités par des espaces, des tabulations ou des caractères «fin de ligne» (Le caractère «fin de ligne» marque la fin de la ligne).

Vous ne risquez pas d'oublier l'ordre de ces trois nombres car il y a toujours plus de caractères que de mots, et plus de mots que de lignes. Le nombre de caractères prend en compte les «fin de ligne» en fin de ligne.

Les options -l (lignes), -w (mots), et -c (caractères) permettent de n'afficher respectivement que les lignes, les mots et les caractères.

wc est utile dans deux occasions. La première survient lorsque vous souhaitez avoir une estimation rapide de la taille d'un fichier que vous voulez, par exemple, transférer par le réseau. Supposons que le fichier est important et que vous voulez vérifier qu'il est bien arrivé intact. Pour cela, lancez wc sur votre fichier original et communiquez au destinataire le résultat de cette commande. Celui-ci lance alors la même commande. Si les nombres correspondent, le transfert a de bonnes chances d'être un succès. Dans le cas contraire, le transfert est un échec.

La deuxième utilisation de wc est encore plus importante. Vous pouvez établir un tube entre une commande et wc pour vérifier le nombre de lignes générées. La plupart des commandes génèrent une information par ligne. En comptant celles-ci, vous pouvez connaître la quantité d'informations produites.

Voyons un exemple. La commande `ls` liste les noms de tous les fichiers d'un répertoire. Si vous entrez `ls /etc` vous verrez les noms de tous les fichiers du répertoire `/etc`. Cette commande `ls` dispose de nombreuses options, mais aucune ne permet de compter le nombre de fichiers. Pour résoudre ce problème, il suffit d'établir un tube entre la commande `ls` et `wc`. Pour compter le nombre de fichiers du répertoire `/etc`, entrez : `ls /etc | wc -l`.

Cet exemple démontre un principe important. Quand vous examinerez le fonctionnement de la commande `ls`, vous verrez que cette-ci affiche les noms de fichiers sur plusieurs colonnes. Il est intéressant de voir malgré cela que la commande `ls` «sait» qu'il ne faut générer qu'une seule colonne lorsqu'un tube est établi entre la sortie standard et la commande suivante, de sorte qu'il y a bien un nom par ligne. En d'autres termes, la commande `ls` est suffisamment intelligente pour être coopérative quand elle est insérée dans un pipeline, en écrivant ses données d'une manière facilement exploitable.

Une bonne partie des commandes Unix (et notamment les filtres) génèrent leurs résultats sous la forme d'une information par ligne. L'exploitation de ces résultats par d'autres programmes en est facilitée.

### 8.8.2 Vérifier l'orthographe des données : `spell`

`spell` n'est pas présent sur tous les systèmes.

La commande `spell` lit les données et génère une liste de tous les mots anglais du texte qui semblent mal orthographiés.

La syntaxe de la commande `spell` est :

```
spell [-b] [fichier...]
```

où `fichier` est le nom d'un fichier d'entrée.

`spell` est un filtre simple qui prend un fichier en entrée et le parcourt à la recherche des mots mal orthographiés.

Si vous disposez d'un fichier nommé `document`, vous pouvez le «vérifier» en entrant : `spell document`. Chaque mot de la liste générée n'apparaît qu'une fois, même s'il en existe plusieurs occurrences dans le texte original.

Par défaut, `spell` fonctionne selon l'orthographe américaine. Si vous souhaitez utiliser l'orthographe britannique/canadienne, ajoutez l'option `-b`. L'orthographe du mot *color* deviendra *colour*.

Le fichier contenant le dictionnaire de `spell` est `/usr/dict/words`. Vous pouvez le parcourir (en partie, car il est grand) au moyen de la commande : `more /usr/dict/words`.

### 8.8.3 Encoder et décoder des données : `crypt`

La commande `crypt` permet de coder des données. Pour cela, vous devez rentrer un mot de passe appelé la *clé*. `crypt` se sert de la clé pour créer une version encodée des données appliquées en entrée. Les données encodées ne sont absolument pas lisibles mais, si vous connaissez la clé, vous pouvez réappliquer `crypt` pour décoder les données encodées et reconstituer le message original.

Même si Unix dispose d'un moyen de protection des fichiers (les permissions décrites section 4.4), il se peut qu'une personne trouve un moyen détourné de lire vos fichiers personnels. Pour cette raison, vous avez intérêt à crypter vos données sensible ou confidentielles.

Pour ceux que cela intéresse, sachez que la commande `crypt` utilise le même principe que la machine de cryptage allemande Enigma, mise au point et utilisée pendant la dernière guerre mondiale. La seule différence réside dans le fait que, contrairement à la machine allemande qui disposait d'un tambour de codage à un élément, l'algorithme de cryptage employé par `crypt` émule en fait un tambour à 256 éléments.

Cette commande n'est pas disponible sur tous les systèmes. En particulier, et pour des raisons de sécurité nationale, la commande est supposée ne pas être distribuée hors des USA (cela peut sembler idiot, mais les américains dorment mieux la nuit en sachant cela).

La syntaxe de la commande `crypt` est :

```
crypt [cle]
```

où `cle` est le mot de passe utilisé pour coder les données.

L'utilisation de `crypt` est simple. Si vous ne rentrez pas de clé, `crypt` vous en demandera une, puis produira le résultat codé sur la sortie standard.

Si vous voulez créer un message codé dans le fichier message, entrez la commande :

```
crypt > message
```

Le programme vous demandera de rentrer une clé : `Enter key :` (Entrez la clé :). Entrez alors la clé de votre choix et pressez <Entrée>. Pour des raisons évidentes de sécurité, la clé ne sera pas affichée à l'écran, comme le mot de passe au login. Vous pouvez alors rentrer votre message sur autant de lignes que nécessaire. Pressez `^D` (le code `eof`) pour indiquer enfin à `crypt` qu'il n 'y a plus de données disponibles. `crypt` encode alors votre message et l'écrit dans le fichier spécifié.

Pour visualiser le message encodé, entrer : `crypt < message`. Une fois encore, `crypt` vous demande la clé. Une fois celle-ci communiquée au programme, `crypt` décode le message et l'affiche sur la sortie standard, l'écran.

Pour les maniaques de la sécurité, afin d'être très prudent, n'entrez jamais la clé comme paramètre de la commande. La personne qui veut vous espionner peut tout à fait utiliser les commandes `w`, `ps` ou regarder par dessus votre épaule pour voir la ligne de commande entière et consulter votre clé.

Enfin, gardez toujours votre clé secrète et ne la perdez pas, car vous seriez alors dans l'impossibilité de recouvrer les données originales.

## 8.9 Les expressions régulières

### 8.9.1 Introduction

Le terme *expression régulière* est issu de la théorie informatique et fait référence à un ensemble de règles permettant de définir un ensemble de chaîne de caractères.

Une expression régulière constitue donc une manière compacte de définir un ensemble de chaîne de caractères. Nous dirons qu'une portion de texte est décrite par une expression régulière si cette portion est un élément de l'ensemble de chaîne de caractères défini par l'expression régulière.

Vous pouvez, par exemple, avec une expression régulière, chercher au moyen de `grep` toutes les occurrences de chaînes de caractères commençant par un `H`, suivi d'un certain nombre de lettres et finalement de la lettre `y`.

Les expressions régulières sont disponibles dans de nombreux outils, comme `vi`, ou les programmes de pagination (comme `more`). Soyez cependant averti que les types d'expressions régulières acceptés varient légèrement suivant les programmes (la commande `egrep` est ainsi plus complète que la commande `grep`).

Les expressions régulières font partie intégrante d'Unix et vous devez savoir les utiliser. Dans cette section, nous présenterons les expressions régulières utilisables avec `egrep`, qui sont les plus générales. Si vous connaissez ces règles, vous ne rencontrerez que des difficultés mineures dans l'utilisation d'autres programmes.

### 8.9.2 Formalisme

Comme nous allons le voir, dans une expression régulière, certains symboles ont une signification spéciale. Dans ce qui suit, *<expreg>*, *<expreg_1>*, *<expreg_2>* désignent des expressions régulières.

***<caractère>*** : un caractère est une expression régulière qui désigne lui-même, excepté pour les caractères `.`, `?`, `+`, `*`, `{`, `|`, `(`, `)`, `^`, `$`, `\`, `[`, `]` qui sont des méta-caractères et ont une signification spéciale ; pour désigner ces méta-caractères, il faut les faire précéder d'un antislash (`\.`, `\?`, `\+`, `\*`, `\{`, `\|`, `\(`, `\)`, `\^`, `\$`, `\\`, `\[`, `\]`).

**`[`*<liste_de_caractères>*`]`** : est une expression régulière qui décrit l'un des caractères de la liste de caractères, par exemple `[abcdf]` décrit le caractère `a`, le `b`, le `c`, le `d` ou le `f` ; le caractère `-` permet de décrire des ensembles de caractères consécutifs, par exemple `[a-df]` ≡ `[abcdf]` ; la plupart des méta-caractères perdent leur signification spéciale dans une liste, pour insérer un `]` dans une liste, il faut le mettre en tête de liste, pour inclure un `^`, il faut le mettre n'importe où sauf en tête de liste, enfin un `-` se place à la fin de la liste.

**`[^`*<liste_de_caractères>*`]`** : est une expression régulière qui décrit les caractères qui ne sont pas dans la liste de caractères.

**`[:alnum:]`** : à l'intérieur d'une liste, décrit un caractère alpha-numérique (`[[:alnum:]]` ≡ `[0-9A-Za-z]`) ; sur le même principe, on a également `[:alpha:]`, `[:cntrl:]`, `[:digit:]`, `[:graph:]`, `[:lower:]`, `[:print:]`, `[:punct:]`, `[:space:]`, `[:upper:]` et `[:xdigit:]`.

**`.`** : est une expression régulière et un méta-caractère qui désigne n'importe quel caractère.

**`\w`** : est une expression régulière qui est synonyme de `[[:alnum:]]`.

**`\W`** : est une expression régulière qui est synonyme de `[^[:alnum:]]`.

**`^`** : est une expression régulière et un méta-caractère qui désigne le début d'une chaîne de caractères.

`$` : est une expression régulière et un méta-caractère qui désigne la fin d'une chaîne de caractères.

`\<` : est une expression régulière qui désigne le début d'un mot.

`\>` : est une expression régulière qui désigne la fin d'un mot.

*<expreg>*`?` : est une expression régulière qui décrit zéro ou une fois *<expreg>*.

*<expreg>*`*` : est une expression régulière qui décrit *<expreg>* un nombre quelconque de fois, zéro compris.

*<expreg>*`+` : est une expression régulière qui décrit *<expreg>* au moins une fois.

*<expreg>*`{`$n$`}` : est une expression régulière qui décrit *<expreg>* $n$ fois.

*<expreg>*`{`$n$`,}` : est une expression régulière qui décrit *<expreg>* au moins $n$ fois.

*<expreg>*`{`$n$`,`$m$`}` : décrit *<expreg>* au moins $n$ fois et au plus $m$ fois.

***<expreg_1><expreg_2>*** : est une expression régulière qui décrit une chaîne constituée de la concaténation de deux sous-chaînes respectivement décrites par *<expreg_1>* et *<expreg_2>*.

***<expreg_1>*`|`*<expreg_2>*** : est une expression régulière qui décrit toute chaîne décrite par *<expreg_1>* ou par *<expreg_2>*.

`(`*<expreg>*`)` : est une expression régulière qui décrit ce que décrit *<expreg>*.

`\n` : où `n` est un chiffre, est une expression régulière qui décrit la sous-chaîne décrite par la $n^e$ sous-expression parenthèsée de l'expression régulière.

**Remarque :** la concaténation de deux expressions régulières (*<exp reg_1><expreg_2>*) est une opération prioritaire sur l'union (*<expreg_1>*`|`*<expreg_2>*).

### 8.9.3 Exemples

Nous allons maintenant donner quelques exemples décrivant l'utilisation des expression régulières, en considérant que tous ces exemples portent sur la commande `grep` et le fichier de données `data`.

Un caractère, qui n'est pas un méta-caractère, se décrit lui-même. Ce qui signifie que si vous cherchez les lignes du fichier qui contiennent «voiture», vous pouvez directement entrer : `grep voiture data`

Si vous ne cherchez que les motifs situés en début de ligne, utilisez le symbole `^`. Pour chercher toutes les lignes qui commencent par «voiture», entrez : `grep '^voiture' data`. Notez que nous avons mis le motif entre apostrophes. Vous devrez toujours mettre une expression régulière entre apostrophes si vous souhaitez que celle-ci soit correctement interprétée par le shell. Les apostrophes préviennent le shell de laisser les caractères tels qu'ils sont et de les communiquer directement au programme (ici, `grep`).

Le signe `$` (dollar) indique que vous souhaitez trouver les motifs en fin de ligne. Ainsi : `grep 'voiture$' data` recherche toutes les occurrences de "voiture" en fin de ligne.

Vous pouvez aussi spécifier un motif qui doit obligatoirement débuter sur un mot ou finir sur un mot. Le début de mot se note `\<`. Pour trouver toutes les occurrences du motif «voi» qui débutent sur un mot, servez-vous de la commande : `grep '\<voi' data`. Pour trouver toutes les occurrences du motif «ure» qui finissent sur un mot, faites `grep 'ure\>' data`

Pour chercher des mots complets, utilisez conjointement `\<` et `\>`. Utiliser `grep` avec les options `\<` et `\>` revient à employer l'option `-w`.

Le symbole `.` (point) remplace n'importe quel caractère excepté la «fin de ligne». Pour trouver toutes les occurrences du motif composé des lettres `vo`, de trois lettres quelconques, et de la lettre `e`, utilisez : `grep 'vo...e' data`. Cette commande permet de trouver des chaînes comme : `voyagent`, `voyage`, `voyager`, `voyageur`, `vous e`.

Vous pouvez aussi définir un ensemble de lettres en les insérant entre crochets `[ ]`. Pour chercher toutes les lignes qui contiennent les lettres `P` ou `p` suivie de `rince`, entrez :`grep '[Pp]rince' data`.

Si vous voulez spécifier un intervalle de caractères, servez vous d'un trait d'union pour délimiter le début et la fin de l'intervalle. Quand vous définissez un intervalle, vous devez, respecter l'ordre du code ASCII. Vous pouvez aussi définir plusieurs intervalles simultanément (par exemple `[A-Za-z]` désigne toutes les lettres de l'alphabet quelque soit la case).Notez bien qu'un intervalle ne correspond qu'à un caractère dans le texte.

Le symbole `*` est utilisé pour définir zéro ou plusieurs occurrences du caractère précédent.

Si vous souhaitez qu'un symbole soit interprété littéralement il faut le préfixer par un `\`. Pour trouver toutes les lignes qui contiennent le symbole `$`, entrez : `grep '\$' data`

# Chapitre 9

# Pratique des filtres, redirections et tubes

## 9.1 Fréquence des fichiers par date de modification

Vous vous intéressez au fichier du répertoire `/bin`. Plus précisément vous vous intéressez à leur date de dernière modification. En fait vous aimeriez connaître pour chaque date combien de fichiers ont été modifiés à cette date. Nous allons effectuer cette requête en une seule ligne de commande : un pipe-line.

1. Nous commençons par lister l'ensemble des fichiers du répertoire `/bin` avec la commande `ls -l /bin`.
2. Vous ne désirez obtenir que les fichiers ordinaires (ie. les lignes qui commencent par un `-`) : complétez la commande dans cette optique.
3. Pour chaque fichiers, vous ne désirez conserver que la date (ie. le mois, le jour et l'année), complétez la ligne de commande.
4. Vous désirez compter les lignes identiques (commande `uniq`). Attention, pour cela, les données doivent être triées. Complétez la ligne de commande.
5. Vous aimeriez enfin que les dates soient classées de la moins fréquente à la plus fréquente. Quelle est la ligne de commande finale ?

## 9.2 Fréquence des mots d'un corpus

On désire afficher les mots du fichier `corpus_1.txt` accompagnés de leur fréquence dans l'ordre de fréquence croissante.

- La première étape consiste à segmenter par des retours chariots les mots du fichier `corpus_1.txt`. Pour cela, la méthode la plus simple, bien que très grossière, consiste à remplacer tous les caractères non alphabétiques par des retours chariots. Vous pouvez effectuer cette action avec le filtre `tr`. Ecrivez la ligne de commande adéquat.
- Il ne reste plus maintenant qu'a trier ces mots par ordre alphabétique, compter les lignes consécutives identiques et trier à nouveau le résultat. Ecrivez la commande complète effectuant tout le traitement en un seul pipe-line.

## 9.3 Génération d'un lexique

1. L'objectif est de dresser un lexique à partir des corpus `corpus_1.txt` et `corpus_2.txt`. Ce lexique doit contenir tous les mots qui apparaissent dans les corpus `corpus_1.txt` et `corpus_2.txt`. Il ne doit y avoir qu'un mot par ligne. Toutes les majuscules doivent être converties en minuscules. Il ne doit pas y avoir de doublon. Le résultat doit être conservé dans le fichier `lexique.txt`. Ecrivez le pipe-line qui permet d'effectuer cette opération.
2. Vous aimeriez avoir un nouveau lexique `lexique-rev.txt` basé sur `lexique.txt` où les mots sont classés par ordre alphabétique en partant de la fin du mot (d'abord les mots qui finissent par `a`, ...). Comment faites-vous ?
3. Comptez le nombre de mots dans le lexique *lexique.txt* avec la commande `wc -w`. Comment ferriez vous pour estimer le nombre moyen de lettres par mot dans ce lexique ?
4. Pour chaque mot, vous aimeriez que le lexique vous indique également sa fréquence. Cependant, vous ne voulez pas que la fréquence figure devant le mot mais derrière le mot. Réalisez cet objectif dans le fichier `lexique-frec.txt`.

## 9.4 Recherches dans un lexique

A partir du lexique `lexique.txt` vous désirez :

1. afficher les mots qui commencent par `pri` en utilisant la commande `look` puis la commande `grep` ;
2. afficher les mots qui contiennent la sous-chaîne `pri` ;
3. afficher les mots qui finissent par les lettres `aux` ;
4. dénombrer le nombre de mots qui finissent par les lettres `aux` ;
5. afficher les mots qui contiennent la lettre `w` ou la lettre `x`
6. afficher les mots qui contiennent la lettre `v` et la lettre `x` dans cet ordre mais pas forcément consécutives ;
7. afficher les mots qui contiennent la lettre `v` et la lettre `x` dans un ordre quelconque et pas forcément consécutives ;
8. dénombrer le nombre de mots de trois lettres ;
9. afficher les anagrammes de trois lettres ;
10. afficher les anagrammes de quatre lettres ;
11. dénombrer le nombre de mots qui commence par deux lettres quelconques et finissent par ces deux mêmes lettres mais dans l'ordre inverse (ex : `revolver`) ;
12. obtenir les quatre combinaisons possibles des deux dernières lettres des mots du lexique les plus fréquentes, accompagnées de leur fréquences, par ordre de fréquence croissante ; le résultat devrait ressembler à :

```
218 it
242 er
356 es
469 nt
```

## 9.5 Comparaisons de lexiques

Etant donnés les deux lexique `lex1` et `lex2` effectuez les opérations qui suivent.

1. Testez les commandes `cmp`, `comm` et `diff` sur ces deux fichiers et vérifiez le résultat obtenu.
2. Affichez les mots qui sont à la fois dans `lex1.txt` et dans `lex2.txt` à l'aide des commandes `sort` et `uniq`.
3. Même question mais en utilisant la commande `join`.
4. Affichez les mots qui sont dans `lex1` ou dans `lex2` mais pas dans les deux à l'aide des commandes `sort` et `uniq`.
5. Même question mais en utilisant la commande `join`.
6. Affichez les mots qui sont dans `lex1.txt` mais pas dans `lex2.txt`, puis ceux qui sont dans `lex2.txt` mais pas dans `lex1.txt`, en utilisant la commande `join`.

## 9.6 Tableau des fréquences des mots de deux corpus

L'objectif de cet exercice est d'obtenir un tableau `tab` des fréquences des mots des corpus `corpus_1.txt` et `corpus_2.txt`. Sur chaque ligne on trouvera un mot suivit de sa fréquence dans `corpus_1.txt` puis de sa fréquence dans `corpus_2.txt`.

1. Réalisez un lexique (un mot par ligne, pas de répétition d'un même mot, pas de majuscule) des mots du corpus `corpus_1.txt` avec la fréquence de chaque mots. Ce fichier doit être trié par ordre alphabétique des mots du lexique. Faire la même chose avec `corpus_2.txt`.
2. On désire réaliser un tableau avec le format suivant :

| Mot | Fréquence `Corpus_1` | Fréquence `Corpus_2` |
|---|---|---|
| absence | 1 | 0 |
| absolument | 1 | 7 |
| acceptable | 0 | 3 |
| ⋮ | ⋮ | ⋮ |

   Pour réaliser cet objectif, quel est le défaut de la commande ci-dessous ?

```
join -a 1 -a 2 -e 0 -j 2 -o 1.2 1.1 2.1 freq1 freq2 > tab
```

3. Pour remédier a ce problème, dans un premier temps, on ne considérera pas les lignes non appariées de `freq2`. La commande est donc la même sans l'option `-a 2`. Dans un second temps on complète le fichier `tab` (*i.e.* `>> tab` au lieu de `> tab`) en ne considérant que les lignes non appariées de `freq2` (avec l'option `-v 2`). Il ne reste plus qu'a retrier le fichier. Réalisez ces trois oppérations.

# Chapitre 10

# Alias et Scripts

## 10.1 TP scripts

# Bibliographie

Aho, A., & Ullman, J. (1992). *Concepts fondamentaux de l'informatique.* Paris: Dunod.

Hahn, H. (1993). *Unix guide de l'étudiant.* Paris: Dunod.

MandrakeSoft. (2002a). *Mandrake linux 8.2 : Guide d'installation et de l'utilisateur.* http://www.mandrakelinux.com/.

MandrakeSoft. (2002b). *Mandrake linux 8.2 : Manuel de référence.* http://www.-mandrakelinux.com/.

Mouret, P. (2002). *Z20 - linux / perl.* http://www.up.univ-mrs.fr/~mouret/z20/-index.html. (Université de Provence)

Perrot, G. (1994). *Cours d'introduction a UNIX.* (Ecole polytechnique de Montréal, institut de génie biomédical)

Zanella, P., & Ligier, Y. (1998). *Architecture et technologie des ordinateurs* (3 ed.). Paris: Dunod. (ISBN 2 10 003801 X)

# Index