Cours SII

Année 2012–2013

Stage d'informatique pour l'ingénieur

Marc Baudoin

École nationale supérieure
de **techniques avancées**

ENSTA
ParisTech

Cours SII

Année 2012–2013

Stage d'informatique pour l'ingénieur

Marc Baudoin

École nationale supérieure de **techniques avancées**

32 boulevard Victor - 75739 PARIS CEDEX 15 - France

© Marc Baudoin, 2009, 2010, 2011, 2012

Ce document est mis à disposition selon les termes du contrat Creative Commons « Paternité - Pas d'utilisation commerciale - Partage des conditions initiales à l'identique » 2.0 France :

http://creativecommons.org/licenses/by-nc-sa/2.0/fr/

Vous êtes libre :
- de reproduire, distribuer et communiquer cette création au public ;
- de modifier cette création.

Selon les conditions suivantes :

Paternité. Vous devez citer le nom de l'auteur original de la manière indiquée par l'auteur de l'œuvre ou le titulaire des droits qui vous confère cette autorisation (mais pas d'une manière qui suggérerait qu'ils vous soutiennent ou approuvent votre utilisation de l'œuvre).

Pas d'utilisation commerciale. Vous n'avez pas le droit d'utiliser cette création à des fins commerciales.

Partage des conditions initiales à l'identique. Si vous modifiez, transformez ou adaptez cette création, vous n'avez le droit de distribuer la création qui en résulte que sous un contrat identique à celui-ci.

- À chaque réutilisation ou distribution de cette création, vous devez faire apparaître clairement au public les conditions contractuelles de sa mise à disposition. La meilleure manière de les indiquer est un lien vers cette page Web.
- Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droits sur cette œuvre.
- Rien dans ce contrat ne diminue ou ne restreint le droit moral de l'auteur ou des auteurs.

Ce qui précède n'affecte en rien vos droits en tant qu'utilisateur (exceptions au droit d'auteur : copies réservées à l'usage privé du copiste, courtes citations, parodie...).

À Patrick Rougeau,
qui m'a permis de devenir
ce que je suis.

Sommaire

Sommaire 9

Avant-propos 11

I Introduction 15

1 Le stage d'informatique pour l'ingénieur 17

2 Représentation et mesure de l'information 23

3 Premier contact 29

II Du texte et des éditeurs de texte 43

4 Le texte et son codage 45

5 L'éditeur de texte Emacs 57

6 Les éditeurs de texte `vi` et Vim 73

III L'interpréteur de commandes 85

7 L'interpréteur de commandes 87

8 Gestion, organisation et traitement des fichiers 107

9 Commandes, processus et redirections 173

10 Impression 229

11 Connexion à distance 245

12 Programmation 257

IV Création d'un site Web 279

13 Formats de documents et historique de la toile 281

14 HTML, XHTML 291

15 Feuilles de style CSS 313

Annexes 367

A Récapitulatif des commandes des éditeurs de texte 369

B Récapitulatif des commandes 375

C Récapitulatif des éléments HTML et des propriétés CSS 381

Bibliographie 385

Index 395

Index – Du texte et des éditeurs de texte 399

Index – L'interpréteur de commandes 401

Index – Création d'un site Web 407

Liste des tableaux 411

Table des figures 413

Table des listages 415

Table des matières 417

Avant-propos

> « Former des ingénieurs capables d'assurer la conception, la réalisation et la direction de systèmes complexes, sous des contraintes économiques fortes et dans un environnement international. »
>
> ENSTA PARISTECH

CE DOCUMENT est le support de cours du *stage d'informatique pour l'ingénieur* (SII) de l'École nationale supérieure de techniques avancées [1] (ENSTA ParisTech), destiné à enseigner aux futurs ingénieurs les rudiments de l'utilisation d'un système informatique. Il se veut le plus universel possible et une grande partie de son contenu s'applique à tout environnement informatique sous UNIX.

> Certaines particularités propres à celui de l'ENSTA ParisTech doivent néanmoins être mentionnées. Afin de les distinguer clairement, elles seront composées comme ce paragraphe, avec une barre verticale grise en bordure de la marge gauche et le logo de l'ENSTA ParisTech dans celle-ci.

En revanche, ce document ne se veut en aucun cas complet ou exhaustif. Il s'agit d'un document d'introduction que les plus curieux pourront compléter par les ouvrages mentionnés dans la bibliographie — en particulier le livre *La machine UNIX* [56], qui était le support du précédent professeur responsable de ce cours — ainsi que par les pages Web indiquées en fin de certaines sections et qui sont repérées par une barre verticale grise en bordure de la marge gauche. Beaucoup de ces pages sont des articles de la version française de l'encyclopédie Wikipédia. Il peut être intéressant de consulter également les articles correspondants en anglais — accessibles grâce à l'élément « Autres langues » du menu de navigation situé à la gauche de chaque page — car ils sont parfois plus détaillés.

1. http://www.ensta-paristech.fr/

Le ton de ce document est volontairement incisif. Il faut admettre que la situation de l'informatique aujourd'hui ne laisse augurer rien de bon pour le futur. En effet, depuis quelques années, toute diversité a quasiment disparu du marché informatique et un ordinateur a maintenant toutes les chances d'être un PC (dont le nombre de constructeurs est de plus en plus restreint) doté d'un microprocesseur Intel et fourni obligatoirement avec Windows (qu'on peut se faire rembourser avec les pires difficultés du monde si l'on souhaite utiliser un autre système d'exploitation). Aucun autre domaine industriel n'est victime d'une telle réduction des choix. Accepterait-on, au niveau mondial, de n'avoir qu'une poignée de constructeurs automobiles, un seul fabricant de pneus et un seul pétrolier ? Évidemment non. Cependant, cela ne choque absolument personne au niveau de l'informatique... Une certaine diversité existe encore dans l'informatique embarquée (téléphones mobiles, baladeurs MP3, GPS, boîtiers de raccordement ADSL, équipements de télécommunication, sondes spatiales, etc.) mais malheureusement sans que le grand public en ait réellement conscience.

Bien entendu, les positions exprimées par l'auteur [2] de ce document n'engagent que lui.

Comment lire ce document ?

Le stage d'informatique pour l'ingénieur étant une introduction pratique à l'informatique, il sera bien plus efficace de lire ce document devant un ordinateur, en reproduisant les exemples et en effectuant les commandes proposées. On a beau regarder le rugby à la télévision, on aura l'air malin sur un terrain si on n'a jamais pratiqué, c'est pareil en informatique, on apprend bien mieux en pratiquant. Il est donc vivement conseillé de lire la suite de ce document à proximité d'un ordinateur.

Par ailleurs, la version PDF de ce document offre l'avantage sur la version imprimée que les hyperliens (références croisées, liens vers les pages Web) y sont actifs et permettent donc de naviguer plus facilement en son sein et vers les pages Web indiquées.

Sur la toile

Le site de référence concernant ce cours est :

2. Et l'auteur en question a quelques avis bien tranchés en matière d'informatique...

> http://www.babafou.eu.org/ensta/sii/

Vous y trouverez :
- la dernière version de ce document au format PDF ;
- le programme des séances de petite classe ;
- les consignes pour le projet ;
- les présentations utilisées en cours magistral.

Terminologie

L'informatique fait un usage intensif de termes en anglais. On lit parfois et on entend souvent à son sujet une sorte de magma mélangeant sans vergogne français, anglais, néologismes et barbarismes [3].

J'essaierai autant que possible d'employer un français châtié, en indiquant les termes anglais équivalents lorsque cela s'avérera nécessaire.

Pour faire avancer le Schmilblick [4]

Je vous invite à me faire part de vos remarques concernant ce document par courriel à l'adresse <babafou+sii@babafou.eu.org>.

3. J'entends régulièrement des gens qui parlent de « déléter » un fichier. Quelle horreur...

4. Comme disait Pierre Dac : « Le Schmilblick des frères Fauderche est, il convient de le souligner, rigoureusement intégral, c'est-à-dire qu'il peut à la fois servir de Schmilblick d'intérieur, grâce à la taille réduite de ses gorgomoches, et de Schmilblick de campagne grâce à sa mostoblase et à ses deux glotosifres qui lui permettent ainsi d'urnapouiller les istioplocks même par les plus basses températures. Ça c'est clair, jusque là ! »

Première partie

Introduction

1

Le stage d'informatique pour l'ingénieur

TOUT INGÉNIEUR utilise aujourd'hui quotidiennement l'outil informatique. Que ce soit pour communiquer par courrier électronique, pour consulter des sites Web, pour rédiger des documents, pour utiliser des bases de données, pour faire de la CAO, pour effectuer des simulations, pour programmer, etc. L'ordinateur est un instrument incontournable.

Néanmoins, on en est venu à oublier le but de l'informatique. Rappelons que le mot « informatique » est un mot-valise né de la fusion des mots « information » et « automatique ». Si l'informatique donne effectivement accès, grâce à l'Internet, à une masse d'information considérable (où il est parfois difficile de faire la part des choses), le côté automatique est bien souvent oublié et un ordinateur n'est pour beaucoup qu'un appareil avec lequel on passe son temps à agiter une souris pour déplacer des icônes...

L'utilisation d'un ordinateur est de plus en plus répétitive alors qu'elle devrait être efficace. La faute en revient pour beaucoup aux interfaces graphiques qui, si elles ont l'avantage de la simplicité, ont beaucoup perdu en efficacité par rapport à ce qui existait avant l'introduction de la souris et des fenêtres.

Lorsqu'on parle d'interfaces graphiques, c'est souvent pour citer leur ergonomie et leur intuitivité. On oublie malheureusement que ce sont deux notions relatives à un environnement culturel donné. Utiliser une interface graphique est intuitif lorsqu'on en connaît déjà une autre, relativement semblable. Donnez des couverts à un Chinois, qui n'a jamais connu que des baguettes, il aura les pires difficultés à manger avec. Donnez-moi des baguettes, je finirai par avoir une crampe au poignet. Et chacun vous dira que sa façon de manger est plus ergonomique et intuitive que celle de l'autre. Tout est relatif.

http://fr.wikipedia.org/wiki/Informatique

1.1 Objectifs du cours

Dans la mesure où une grande partie des cours de l'ENSTA ParisTech utilise l'outil informatique, le premier objectif de ce cours est de permettre aux élèves d'utiliser au mieux les moyens informatiques de l'ENSTA ParisTech. Mais nous essaierons néanmoins de dégager des principes généraux applicables à tout système informatique dès que possible.

En particulier, seront abordés :

- l'utilisation d'un éditeur de texte ;
- l'interpréteur de commandes sous UNIX ainsi qu'un certain nombre de commandes usuelles ;
- la conception de pages Web, ce qui donnera lieu à la réalisation d'un projet noté.

1.2 Qu'est-ce qu'un système d'exploitation ?

1.2.1 Généralités

Un ordinateur est un ensemble de composants matériels destiné à faire fonctionner des logiciels. Entre ces composants (microprocesseur, disque dur, etc.) et les logiciels d'application (ceux qui sont visibles pour l'utilisateur, tels les logiciels de bureautique), on intercale un logiciel particulier appelé *système d'exploitation* (*operating system* en anglais).

Celui-ci permet de masquer aux logiciels les particularités du matériel, il gère les ressources et arbitre les demandes des logiciels.

Il existe plusieurs systèmes d'exploitation, le plus répandu (mais certainement pas le mieux conçu) étant Windows. Les ordinateurs de l'ENSTA ParisTech, quant à eux, fonctionnent avec un système d'exploitation appelé UNIX.

> http://fr.wikipedia.org/wiki/Système_d'exploitation
> http://fr.wikipedia.org/wiki/Liste_des_systèmes_d'exploitation

1.2.2 UNIX

UNIX a été créé en 1969 dans le cadre d'un programme de recherche des Laboratoires Bell. Il emprunte de nombreux concepts à un autre système d'exploitation appelé Multics (*Multiplexed Information and Computing Service*). UNIX s'appelait

d'ailleurs à l'origine Unics (pour *Uniplexed Information and Computing Service*), par dérision pour la complexité de Multics, alors que les créateurs d'Unics visaient la simplicité.

Distribué très librement à l'époque, UNIX a engendré toute une famille de systèmes d'exploitation et de nombreux concepts introduits par UNIX ont été repris par d'autres systèmes d'exploitation.

Aujourd'hui, UNIX n'existe plus en tant que système d'exploitation, il s'agit d'une spécification à laquelle sont conformes un certain nombre de systèmes d'exploitation. Parmi ceux-ci, on peut citer (par ordre alphabétique) :

- AIX, l'UNIX d'IBM ;
- les dérivés de BSD (Berkeley Software Distribution, un UNIX grandement redéveloppé par l'université de Berkeley) : Darwin (qui est au cœur de Mac OS X), DragonFly BSD, FreeBSD, NetBSD et OpenBSD ;
- HP-UX, l'UNIX d'HP ;
- Linux, dont il existe de nombreuses distributions (Arch Linux, CentOS, Debian, Fedora, Gentoo, Linux Mint, Mandriva, openSUSE, Red Hat Enterprise Linux, Slackware, SUSE Linux Enterprise, Ubuntu, etc.) ;
- Oracle Solaris, l'UNIX d'Oracle (conçu à l'origine par Sun Microsystems, entreprise rachetée par Oracle en 2010).

http://www.unix.org/
http://fr.wikipedia.org/wiki/UNIX
http://fr.wikipedia.org/wiki/Philosophie_d'Unix
http://www.levenez.com/unix/
http://fr.wikipedia.org/wiki/Type_Unix
http://fr.wikipedia.org/wiki/Multics
http://fr.wikipedia.org/wiki/AIX
http://fr.wikipedia.org/wiki/Berkeley_Software_Distribution
http://www.freebsd.org/
http://www.netbsd.org/
http://www.openbsd.org/
http://www.hp.com/go/hpux/
http://www.kernel.org/
http://www.oracle.com/us/products/servers-storage/solaris/
http://cm.bell-labs.com/cm/cs/who/dmr/otherunix.html
http://en.wikipedia.org/wiki/The_UNIX-HATERS_Handbook

1.2.3 Pourquoi UNIX à l'ENSTA ParisTech ?

Dans la mesure où l'on estime que Windows équipe 95 % des ordinateurs de la planète, pourquoi donc utiliser UNIX à l'ENSTA ParisTech ? Il est en effet raisonnable de penser que les ingénieurs ENSTA ParisTech, dans leur grande majorité, travailleront sur un PC équipé de Windows dès leur premier emploi et n'auront plus de contact direct avec un système UNIX.

Il y a plusieurs raisons à cela.

On pourrait dire qu'on utilisait UNIX à l'ENSTA ParisTech bien avant la première version de Windows. Bien sûr, les ordinateurs n'avaient rien à voir avec ceux d'aujourd'hui et l'UNIX de l'époque était plus dépouillé. En particulier, il ne disposait d'aucune interface graphique. D'ailleurs, il est intéressant de s'arrêter un moment sur ce point. Windows a le défaut d'être un système conçu de manière relativement monolithique, où l'interface graphique est indissociable des autres services du système. Ceci peut sembler compréhensible pour un ordinateur de bureau, dont l'interface graphique est l'élément le plus utilisé mais qu'en est-il pour un serveur ? Un serveur, très souvent, est un ordinateur sans clavier, sans souris, sans écran, entreposé dans un local rarement visité par des êtres humains. En conséquence, un serveur est généralement géré via une connexion à distance, habituellement non graphique. On peut alors douter de l'utilité d'une interface graphique sur un serveur. Pourtant, beaucoup d'entreprises ont un parc de serveurs fonctionnant exclusivement sous Windows. Sous UNIX, en revanche, l'interface graphique est clairement dissociée du système d'exploitation et elle n'est pas nécessaire à son bon fonctionnement. Sur les serveurs sous UNIX, elle n'est d'ailleurs simplement pas installée.

Voilà donc un premier argument en faveur d'UNIX : sa conception est bien plus propre que celle de Windows. Et, dans le cadre de l'enseignement de l'informatique, il est préférable d'étudier des systèmes d'exploitation bien conçus.

Pour continuer dans les raisons historiques, UNIX est depuis toujours un système d'exploitation *multitâche*, c'est-à-dire qu'il est capable de faire fonctionner plusieurs logiciels en même temps. Il aura fallu attendre Windows 95 pour avoir une ébauche de fonctionnement multitâche.

UNIX est également un système d'exploitation *multi-utilisateur*, c'est-à-dire qu'il permet à plusieurs personnes d'utiliser le même ordinateur simultanément, soit par la connexion de plusieurs terminaux sur l'ordinateur (un terminal est un appareil regroupant un écran et un clavier, voire également une souris pour les terminaux graphiques), soit au moyen d'une connexion via un réseau. Il aura fallu attendre Windows NT 4.0 Terminal Server en 1998 pour qu'il soit possible d'ouvrir

une session à distance (uniquement par déport de l'interface graphique).

Un autre argument en faveur d'UNIX est qu'il en existe des versions gratuites (ce qui est financièrement intéressant lorsqu'on dispose d'un parc de 200 ordinateurs, comme à l'ENSTA ParisTech). FreeBSD, Linux, NetBSD et OpenBSD sont disponibles gratuitement. Mieux encore, leur code source est également disponible, ce qui permet de le décortiquer à des fins pédagogiques.

Il faut également insister sur la sécurité des systèmes UNIX, bien plus évoluée que celle de Windows. Il n'existe en effet pas de virus pour UNIX, pas qu'on ne puisse en concevoir mais leur portée serait extrêmement limitée en raison des contrôles d'accès imposés par UNIX.

Bref, même s'il faut, avec un réalisme désabusé, reconnaître que de nombreuses entreprises ont une informatique 100 % Windows et que la majorité des ingénieurs ENSTA ParisTech devra s'en contenter, les vertus d'UNIX pour l'enseignement justifient pleinement son choix. Par ailleurs, l'expérience montre que quelqu'un maîtrisant UNIX n'a aucun mal à utiliser Windows. Le contraire est faux...

> http://fr.wikipedia.org/wiki/Multitâche
> http://fr.wikipedia.org/wiki/Multi-utilisateur

1.3 L'informatique efficace

Il faut peut-être relativiser l'affirmation péremptoire clôturant le paragraphe précédent. Il existe pour UNIX des interfaces graphiques équivalentes, voire supérieures, à celle de Windows permettant à ses habitués de pouvoir utiliser un système UNIX de la même façon.

Néanmoins, est-ce la façon la plus efficace d'utiliser un système informatique ? Dans le cas d'une utilisation épisodique, cela ne fait pas de différence mais, pour une utilisation plus soutenue, il existe des outils bien plus efficaces.

En particulier, dans le cadre de ce cours, nous allons utiliser un *interpréteur de commandes*.

Un interpréteur de commandes est un logiciel dans lequel on exprime par écrit ce qu'on veut faire en tapant des commandes au clavier. La souris ne sert quasiment jamais et il n'y a aucun élément graphique (pas d'icônes). Les interpréteurs de commandes sont utilisés en informatique depuis une cinquantaine d'années, bien avant l'apparition des premières interfaces graphiques.

On peut estimer qu'il s'agit d'un outil barbare et rétrograde, qui nécessite un apprentissage certain et qui n'a pas les qualités ergonomiques d'une interface

graphique. C'est en partie vrai. En revanche, l'interpréteur de commandes est particulièrement efficace pour traiter un grand nombre de fichiers et enchaîner les traitements les uns aux autres, ce que les interfaces graphiques ne font pas ou alors difficilement et de manière limitée. En effet, l'automatisation d'un ensemble de commandes est très simple à faire (en gros, on les rassemble dans un fichier et on soumet ce fichier à l'interpréteur de commandes) alors que l'automatisation d'un ensemble de clics dans une interface graphique, s'il est parfois possible, pose néanmoins des problèmes (position des objets sur l'écran, différences d'une version d'un logiciel à un autre, voire d'un ordinateur à un autre, traitement des erreurs, etc.).

2

Représentation et mesure de l'information

DANS UN ORDINATEUR, l'information est représentée sous forme numérique. Le microprocesseur ne sachant réaliser que des opérations sur des nombres et des déplacements de données en mémoire, toute information à traiter doit être exprimée sous une forme numérique. En informatique, on ramène donc tout à des nombres.

Dans ce chapitre, nous allons étudier plusieurs systèmes de représentation des nombres, en nous limitant aux entiers naturels. Seront en particulier abordés les systèmes de représentation des nombres couramment utilisés en informatique. Nous étudierons également les unités permettant de mesurer la quantité d'information.

2.1 Représentation des nombres

La représentation des nombres fait appel à un ensemble de signes et à des règles permettant de les associer afin de transcrire chaque nombre en une suite de signes. Ces signes et ces règles forment un *système de numération*.

Les premiers systèmes de numération remontent à plusieurs milliers d'années.

Les Romains, par exemple, utilisaient le système de numération que tout le monde connaît. Sept lettres ayant chacune une valeur croissante et combinées entre elles avec un système d'écriture additif ou soustractif en fonction de la position des lettres, permettent d'exprimer les entiers naturels non nuls allant jusqu'à quelques milliers. Il est difficile avec un tel système de numération d'effectuer des additions et des soustractions, encore plus des multiplications (ne parlons pas des divisions).

> http://fr.wikipedia.org/wiki/Système_de_numération
> http://fr.wikipedia.org/wiki/Numération_romaine

2.1.1 Système décimal

Nous utilisons un système de numération beaucoup plus efficace que celui des Romains, basé sur la position des chiffres dans la représentation d'un nombre. Comme il y a dix chiffres, on appelle cette numération le *système décimal* (qu'on abrège en *décimal* dans le langage courant).

Ainsi, le nombre 1 664 doit se comprendre comme :

$$1\,664 = 1 \times 10^3 + 6 \times 10^2 + 6 \times 10^1 + 4 \times 10^0$$

De manière générale, un nombre décimal écrit sous la forme $c_n \ldots c_2 c_1 c_0$ ($c_n \ldots c_2$, c_1 et c_0 étant les chiffres composant le nombre, de gauche à droite) doit se comprendre comme :

$$\sum_{i=0}^{n} c_i \cdot 10^i$$

Un tel système de numération permet d'effectuer facilement les opérations arithmétiques usuelles (addition, soustraction, multiplication et division).

http://fr.wikipedia.org/wiki/Système_décimal

2.1.2 Notation positionnelle et bases de numération

Dans le système décimal, la position de chaque chiffre dans un nombre détermine donc par quelle puissance de 10 il doit être multiplié. Cette façon de représenter les nombres est appelée *notation positionnelle*.

L'utilisation des puissances de 10 est un cas particulier, il est possible de généraliser ce système de numération en utilisant les puissances de n'importe quel entier supérieur ou égal à 2, qu'on appelle *base de numération*, et un ensemble de chiffres en nombre égal à la base. Ainsi, un nombre en base b écrit sous la forme $c_n \ldots c_2 c_1 c_0$ ($c_n \ldots c_2$, c_1 et c_0 étant les chiffres composant le nombre, de gauche à droite) doit se comprendre comme :

$$\sum_{i=0}^{n} c_i \cdot b^i$$

http://fr.wikipedia.org/wiki/Notation_positionnelle
http://fr.wikipedia.org/wiki/Base_(arithmétique)

2.1.3 Systèmes de numération utilisés en informatique

Trois systèmes de numération utilisant la notation positionnelle sont utilisés en informatique.

2.1.3.1 Système binaire

Le *système binaire* (qu'on abrège en *binaire* dans le langage courant) utilise la base 2, qui est la plus petite des bases de numération. Les deux chiffres utilisés en binaire sont le 0 et le 1, qu'on désigne conjointement sous le nom *bit* (contraction de l'anglais *binary digit*). Par conséquent, n bits permettent de représenter les nombres allant de 0 à $2^n - 1$.

Pour simplifier l'écriture des nombres codés sous forme binaire, on utilise d'autres bases de numération qui sont des puissances de 2. Ainsi, un nombre binaire codé sur n bits peut également être représenté par un unique chiffre en base 2^n. Les systèmes de numération utilisés en informatique selon ce principe sont le système octal (voir le paragraphe 2.1.3.2) et le système hexadécimal (voir le paragraphe 2.1.3.3 page suivante).

Le binaire est bien adapté à la représentation des nombres en électronique (et donc en informatique) parce qu'il n'utilise que deux chiffres, qui sont faciles à traduire en phénomènes physiques (présence ou absence de charge électrique ou de lumière, étant passant ou non passant, etc.).

En informatique, les bits sont généralement regroupés par huit pour former des octets (voir le paragraphe 2.2.1 page suivante).

> http://fr.wikipedia.org/wiki/Système_binaire
> http://fr.wikipedia.org/wiki/Bit

2.1.3.2 Système octal

Le *système octal* (qu'on abrège en *octal* dans le langage courant) utilise la base 8. Les huit chiffres utilisés en octal vont de 0 à 7 et ont la même valeur qu'en décimal.

Un chiffre octal permet de représenter un nombre binaire codé sur trois bits (puisque $2^3 = 8$).

L'octal est utilisé, entre autres, pour la représentation numérique des droits d'accès aux fichiers sous UNIX (voir le paragraphe 8.5.2 page 142).

http://fr.wikipedia.org/wiki/Système_octal

2.1.3.3 Système hexadécimal

Le *système hexadécimal* (qu'on abrège en *hexadécimal* dans le langage courant) utilise la base 16. Les seize chiffres utilisés en hexadécimal vont de 0 à 9 (et ont la même valeur qu'en décimal) et de A à F (qu'on peut également écrire en minuscules) pour les valeurs de 10 à 15. Ceci est résumé dans le tableau 2.1.

hexadécimal	0 ... 9	A	B	C	D	E	F
décimal	0 ... 9	10	11	12	13	14	15

TABLE 2.1 – *Chiffres utilisés en hexadécimal.*

Un chiffre hexadécimal permet de représenter un nombre binaire codé sur quatre bits (puisque $2^4 = 16$).

En informatique, le système hexadécimal est couramment utilisé car il permet de représenter les octets sur deux chiffres (voir le paragraphe 2.2.2).

http://fr.wikipedia.org/wiki/Système_hexadécimal

2.2 Unités de mesure de l'information

2.2.1 L'octet

Un bit (voir le paragraphe 2.1.3.1 page précédente) ne pouvant représenter que deux valeurs, il est rarement utilisé comme unité de mesure de l'information, même s'il s'agit de la quantité élémentaire d'information utilisée en informatique. On lui préfère un ensemble de huit bits, appelé *octet*.

Un octet contenant huit bits, il peut donc représenter les nombres allant de 0 à $2^8 - 1 = 255$.

http://fr.wikipedia.org/wiki/Octet

2.2.2 Représentation d'un octet en hexadécimal

L'hexadécimal est bien adapté à la représentation numérique des octets. En effet, chaque demi-octet contenant quatre bits, il peut être représenté par un chiffre en

base $2^4 = 16$, c'est-à-dire en hexadécimal. Un octet complet peut donc être représenté par deux chiffres hexadécimaux, allant de 00 à FF (255 en décimal). Réciproquement, tout nombre pouvant être représenté sous la forme de deux chiffres hexadécimaux (c'est-à-dire compris entre 00 et FF) est assuré de pouvoir être stocké dans un octet.

2.2.3 Unités multiples de l'octet

Les unités multiples de l'octet sont traditionnellement définies au moyen de puissances de 2, contrairement à l'usage établi par le système international d'unités (abrégé par la suite en SI), qui utilise des puissances de 10.

Ainsi, selon l'usage habituel, 1 kilooctet ne contient pas $10^3 = 1\,000$ octets mais $2^{10} = 1\,024$ octets. De même, 1 mégaoctet ne contient pas $10^6 = 1\,000\,000$ octets mais $2^{20} = 1\,048\,576$ octets et ainsi de suite.

Afin d'éviter toute confusion, il serait préférable d'utiliser les préfixes multiplicatifs indiqués dans la partie droite du tableau 2.2. Ainsi, ce qu'on appelle habituellement kilooctet est en fait un kibioctet. En pratique, ces unités sont rarement utilisées et l'on continue à utiliser à tort le kilooctet pour désigner un kibioctet, le mégaoctet pour désigner un mébioctet, etc.

Préfixe SI			**Préfixe binaire**		
Nom	**Symbole**	**Valeur**	**Nom**	**Symbole**	**Valeur**
kilooctet	ko	10^3	kibioctet	Kio	2^{10}
mégaoctet	Mo	10^6	mébioctet	Mio	2^{20}
gigaoctet	Go	10^9	gibioctet	Gio	2^{30}
téraoctet	To	10^{12}	tébioctet	Tio	2^{40}
pétaoctet	Po	10^{15}	pébioctet	Pio	2^{50}
exaoctet	Eo	10^{18}	exbioctet	Eio	2^{60}
zettaoctet	Zo	10^{21}	zébioctet	Zio	2^{70}
yottaoctet	Yo	10^{24}	yobioctet	Yio	2^{80}

TABLE 2.2 – *Unités multiples de l'octet.*

Les fabricants de disques durs l'ont d'ailleurs parfaitement compris. Ainsi, un disque dur vendu comme ayant une capacité de 500 Go, c'est-à-dire 466 Gio, sera reconnu par une grande partie des systèmes d'exploitation comme disposant d'une capacité de 466 Go (en utilisant la définition traditionnelle et incorrecte du gigaoctet). Bien que la mesure de la capacité du disque dur par le fabricant soit rigoureusement

exacte, elle ne correspond malheureusement pas aux habitudes et de nombreux acheteurs se sentent lésés en constatant que la capacité annoncée par le système d'exploitation est sensiblement inférieure à celle annoncée par le fabricant du disque dur.

> http://fr.wikipedia.org/wiki/Octet
> http://fr.wikipedia.org/wiki/Système_international_d'unités

3

Premier contact

AVANT d'entrer dans le vif du sujet, il convient de prendre rapidement contact avec les ordinateurs de l'ENSTA ParisTech et leur environnement. Le fonctionnement extérieur d'un ordinateur utilisant le système d'exploitation UNIX est en effet très semblable à celui d'autres systèmes d'exploitation mais certaines particularités doivent être signalées. Nous en profiterons pour faire connaissance avec l'interpréteur de commandes, qui sera étudié plus en détail au chapitre 7 page 87, afin de changer le mot de passe provisoire fourni à chaque nouvel utilisateur d'un système informatique, qui doit l'être rapidement afin d'éviter toute tentative d'usurpation d'identité.

3.1 La babasse

Dans le jargon des ingénieurs, le terme *babasse* désigne un ordinateur [1].

Les ordinateurs de l'ENSTA ParisTech sont en pratique des PC, identiques à ceux qu'on peut trouver dans le commerce, à cette différence près que leur système d'exploitation est UNIX (plus précisément, c'est l'une des distributions de Linux) et non pas Windows. Ils sont équipés de claviers AZERTY, comme il est d'usage en France.

3.1.1 Le clavier

L'habitude est pourtant d'utiliser des claviers QWERTY sous UNIX car de nombreux signes fréquemment utilisés y sont directement accessible alors que leur saisie nécessite l'utilisation de la touche Shift ou de la touche AltGr sur un clavier AZERTY. Par ailleurs, les symboles utilisés en programmation (parenthèses,

1. J'ai cru comprendre qu'à Lyon, le même terme désignait un flipper et que Frédéric Dard l'utilisait de façon particulièrement poétique...

crochets, accolades, par exemple) sont regroupés par paire et sont aisément accessibles sur un clavier QWERTY, ce qui n'est pas le cas sur un clavier AZERTY, où ces symboles sont éparpillés façon puzzle. On pourrait objecter que les claviers QWERTY ne disposent pas des touches permettant de saisir directement les lettres accentuées utilisées en français (du moins les claviers américains parce que les Québécois utilisent des claviers QWERTY adaptés à notre langue) mais il existe plusieurs méthodes pour saisir des lettres accentuées avec un clavier QWERTY.

Il est à noter qu'il n'existe à travers le monde que trois dispositions répandues pour les claviers utilisant l'alphabet latin. La disposition QWERTY, utilisée dans quasiment tous les pays (avec des variantes en fonction des caractères propres à chaque langue), la disposition AZERTY, utilisée en France et en Belgique, et la disposition QWERTZ, utilisée en Allemagne et dans plusieurs pays d'Europe centrale.

Les différences entre ces trois dispositions sont par ailleurs peu nombreuses, quelques lettres étant seulement placées à des endroits différents sur le clavier. Les dispositions AZERTY et QWERTZ sont en effet dérivées de la disposition QWERTY, mise au point dans les années 1870 afin de ralentir la frappe pour éviter des blocages mécaniques sur les premières machines à écrire, ce qui est un comble maintenant que les mêmes claviers sont utilisés sur des ordinateurs qui ne présentent évidemment pas ce genre de problème.

Il existe bien des dispositions, dites dispositions Dvorak, du nom de leur inventeur, permettant une frappe rapide mais elles sont peu utilisées. Les dispositions Dvorak sont basées sur l'analyse fréquentielle des caractères, qui varie en fonction de la langue et il existe donc une disposition pour chaque langue. La disposition bépo, inspirée des dispositions Dvorak, a été conçue au cours des années 2000 spécifiquement pour la saisie du français.

Enfin, remarquez que l'utilisation de la touche Caps Lock et d'une lettre accentuée génère sous UNIX la majuscule accentuée correspondante (ce qui est nécessaire à l'écriture d'un français correct, contrairement à une croyance répandue, voir dans la bibliographie le *Lexique des règles typographiques en usage à l'Imprimerie nationale*, page 12, et *Orthotypographie*, rubrique *Accentuation*[2]). Sous Windows, cela est facile pour l'accent circonflexe et le tréma, qui sont obtenus grâce à une touche morte, mais la saisie des autres majuscules accentuées nécessite une gymnastique particulièrement désagréable.

> http://fr.wikipedia.org/wiki/Disposition_des_touches_des_claviers_informatiques

2. http://www.orthotypographie.fr/volume-I/academie-accentuation.html#Accentuation

http://fr.wikipedia.org/wiki/AZERTY
http://fr.wikipedia.org/wiki/QWERTY
http://fr.wikipedia.org/wiki/QWERTZ
http://fr.wikipedia.org/wiki/Disposition_Dvorak
http://bepo.fr/
http://fr.wikipedia.org/wiki/Disposition_bépo
http://www.artlebedev.com/everything/optimus/

3.1.2 Le gestionnaire d'affichage

Le *gestionnaire d'affichage* (*display manager* en anglais) est le logiciel qui contrôle l'ouverture d'une session (voir le paragraphe 3.3 page suivante) sur un ordinateur, c'est-à-dire le fait de pouvoir l'utiliser au moyen de son écran, de son clavier et de sa souris.

Le gestionnaire d'affichage apparaît sur l'écran de tout ordinateur disponible.

Le gestionnaire d'affichage utilisé à l'ENSTA ParisTech est représenté dans la figure 3.1.

FIGURE 3.1 – *Gestionnaire d'affichage utilisé à l'ENSTA ParisTech.*

http://en.wikipedia.org/wiki/X_display_manager_(program_type)

3.2 Le compte

Avant de pouvoir utiliser un ordinateur sous UNIX, il faut y disposer d'un *compte* (*account* en anglais). Ceci comprend :

- un *identifiant* (*login* en anglais) ;
- un *mot de passe* (*password* en anglais) ;
- un espace de stockage pour vos fichiers, qu'on appelle *répertoire personnel* (*home directory* en anglais).

Ce compte est créé par le service informatique. Le mot de passe qui vous est fourni à la création de votre compte est provisoire et il convient d'en changer rapidement (voir le paragraphe 3.6 page 37) pour un mot de passe connu de vous seul.

http://fr.wikipedia.org/wiki/Compte#Informatique

3.3 Ouverture de session

Une *session* représente le fait d'attribuer des ressources informatiques à un utilisateur. Dans notre cas, il s'agit de pouvoir utiliser un ordinateur au moyen de son écran, de son clavier et de sa souris.

L'ouverture de session se fait en saisissant son identifiant et son mot de passe dans le gestionnaire d'affichage. Selon les environnements, le mot de passe n'apparaît pas à l'écran ou alors seulement sous forme de points ou d'astérisques, pour des raisons évidentes de confidentialité. Attention, UNIX fait la différence entre majuscules et minuscules dans l'identifiant et le mot de passe.

Si l'identifiant et le mot de passe sont corrects, l'utilisateur se retrouve dans un *environnement de bureau* (voir le paragraphe 3.4.4 page 35), qui est l'un des composants de l'interface graphique.

http://fr.wikipedia.org/wiki/Session_(informatique)

3.4 L'interface graphique

L'interface graphique s'est répandue à partir du milieu des années 1980. En micro-informatique, l'introduction du Macintosh, en 1984, a marqué le début de l'ère de

l'interface graphique. La première version de Windows, en 1985, est passée plutôt inaperçue et il a fallu attendre Windows 3.0, en 1990, pour que l'interface graphique se généralise sur PC. Sous UNIX, l'interface graphique repose habituellement sur le X Window System, dont la première version date de 1984.

> http://en.wikipedia.org/wiki/History_of_the_graphical_user_interface

3.4.1 X Window System

X Window System (également connu sous les noms X11 et X, nous utiliserons ce dernier par la suite) est un ensemble logiciel permettant la gestion des services de base de l'interface graphique sous UNIX.

X sait uniquement afficher des fenêtres, gérer les interactions avec l'utilisateur (déplacement de la souris et utilisation de ses boutons, saisie au clavier) et dessiner des objets graphiques élémentaires (pixels, lignes, rectangles, arcs de cercle ou d'ellipse, texte).

Le *serveur X* est le logiciel qui contrôle l'affichage sur l'écran et qui gère le clavier et la souris.

Les *clients X* sont les logiciels qui utilisent le serveur X pour permettre leur affichage.

Serveur X et clients X peuvent fonctionner sur des ordinateurs différents et communiquer via le réseau informatique. En pratique, lorsqu'on travaille sur un ordinateur (celui sur lequel fonctionne le serveur X), on peut donc exécuter des logiciels (les clients X) sur d'autres ordinateurs et les faire s'afficher sur son écran. Si l'on travaille sur un poste de bureau banal, on peut donc exécuter des logiciels réclamant une grande puissance de calcul sur des serveurs adéquats et les faire s'afficher sur l'écran de son poste.

> http://www.x.org/
> http://en.wikipedia.org/wiki/X_Window_System

3.4.2 Les boîtes à outils

Comme indiqué précédemment, X ne sait dessiner que des objets graphiques élémentaires. Il ne sait pas ce qu'est un bouton, un menu, ou tout autre *composant d'interface graphique* (*widget* en anglais).

Pour cela, on lui superpose une *boîte à outils* (*toolkit* en anglais), qui va permettre de gérer des composants d'interface graphique. Les boîtes à outils les plus répandues aujourd'hui sont GTK+ et Qt.

Une boîte à outils utilise les services de bas niveau fournis par X pour proposer des services de plus haut niveau au logiciel qui l'utilise. Par exemple, l'affichage d'un bouton par une boîte à outils utilise des tracés de rectangles de différentes couleurs pour obtenir un effet de relief.

Une telle architecture, composée de plusieurs niveaux et où chaque niveau utilise les services fournis par le niveau inférieur pour proposer des services plus complexes au niveau supérieur, est dite *architecture en couches*. Ce type d'architecture est très utilisé en informatique.

> http://fr.wikipedia.org/wiki/Composant_d'interface_graphique
> http://fr.wikipedia.org/wiki/Xaw
> http://www.opengroup.org/motif/
> http://fr.wikipedia.org/wiki/Motif_(bibliothèque_graphique)
> http://en.wikipedia.org/wiki/XView
> http://www.gtk.org/
> http://fr.wikipedia.org/wiki/GTK+
> http://qt.nokia.com/
> http://fr.wikipedia.org/wiki/Qt

3.4.3 Le gestionnaire de fenêtres

Le *gestionnaire de fenêtres* (*window manager* en anglais) est le logiciel qui entoure les fenêtres d'une bordure permettant de les déplacer et d'en modifier les dimensions.

Il permet également de s'affranchir des limites imposées par la taille de l'écran en proposant des bureaux virtuels. Un *bureau virtuel* (*virtual desktop* en anglais) est un espace de la taille de l'écran où l'on peut agencer ses fenêtres comme on le fait habituellement. Néanmoins, dans la mesure où l'on dispose de plusieurs bureaux virtuels, il n'est pas nécessaire de superposer ses fenêtres lorsqu'il y en a beaucoup, on peut tout à fait les répartir sur plusieurs bureaux virtuels afin d'avoir sur chacun des fenêtres juxtaposées et non superposées, ce qui évite d'avoir à chercher la fenêtre dont on a besoin et qui, bien évidemment, est cachée sous plusieurs autres fenêtres.

Certains gestionnaires de fenêtres disposent également de barres d'outils placées le long d'un ou de plusieurs bords de l'écran.

http://fr.wikipedia.org/wiki/Gestionnaire_de_fenêtres
http://fr.wikipedia.org/wiki/Bureau_virtuel_(interface_graphique)
http://xwinman.org/

3.4.4 L'environnement de bureau

Un *environnement de bureau* (*desktop environment* en anglais) est un ensemble logiciel comprenant un gestionnaire de fenêtres, un gestionnaire de fichiers et diverses applications utilisant la même boîte à outils afin d'avoir une apparence cohérente.

Sous UNIX, les environnements de bureau les plus répandus sont GNOME, KDE et Xfce.

L'environnement de bureau utilisé par défaut à l'ENSTA ParisTech, représenté dans la figure 3.2, est GNOME. Il est possible de choisir d'autres environnements depuis le gestionnaire d'affichage.

FIGURE 3.2 – *Environnement de bureau utilisé à l'ENSTA ParisTech.*

> http://fr.wikipedia.org/wiki/Environnement_de_bureau
> http://www.gnome.org/
> http://fr.wikipedia.org/wiki/GNOME
> http://www.kde.org/
> http://fr.wikipedia.org/wiki/KDE
> http://www.xfce.org/
> http://fr.wikipedia.org/wiki/Xfce

3.5 L'interpréteur de commandes

Même s'il sera étudié au chapitre 7 page 87, il est nécessaire de présenter dès maintenant les rudiments du fonctionnement de l'interpréteur de commandes.

> Dans l'environnement de bureau GNOME utilisé à l'ENSTA ParisTech, le lancement de l'interpréteur de commandes s'effectue en cliquant sur le fond de l'écran avec le bouton droit de la souris puis en choisissant l'élément « Ouvrir un terminal ».

L'interpréteur de commandes ouvre une fenêtre quasiment vide, à l'exception de sa première ligne qui contient une *invite* (*prompt* en anglais) suivie du curseur symbolisant l'emplacement du prochain caractère qui sera tapé au clavier. L'invite est un repère visuel indiquant que l'interpréteur de commandes attend que l'utilisateur lui ordonne quoi faire. La signification du contenu de l'invite sera détaillée au paragraphe 7.2.2 page 92. Pour simplifier les choses, nous utiliserons simplement par la suite `$` comme invite.

Lorsqu'on tape la commande `date` (ce que l'utilisateur tape au clavier est figuré en gras dans les exemples), suivie de la touche Enter, l'interpréteur de commandes affiche à la ligne suivante la date et l'heure puis une nouvelle invite, indiquant qu'il attend à nouveau que l'utilisateur lui demande quelque chose :

```
$ date Enter
Thu Jun 25 15:16:19 CEST 2009
$ ■
```

Ceci nous suffira pour le moment. Rendez-vous au chapitre 7 page 87 pour de nouvelles aventures.

3.6 Changement du mot de passe

Avant toute chose, il faut choisir un nouveau mot de passe. L'objectif du mot de passe est de protéger l'accès à votre compte, il est donc essentiel que ce mot de passe ne puisse pas être facilement deviné par autrui. Il existe également des logiciels qui tentent de trouver les mots de passe en essayant diverses combinaisons à partir de mots du dictionnaire, ce qui rend le choix du mot de passe encore plus difficile.

Certains systèmes informatiques imposent une longueur minimale, la présence d'au moins quelques lettres (attention, on fait la différence entre majuscules et minuscules), chiffres ou signes de ponctuation.

En particulier, à l'ENSTA ParisTech, le mot de passe doit respecter les contraintes suivantes :

- comprendre entre 8 et 20 caractères ;
- comporter au moins 4 caractères différents ;
- contenir au moins une lettre minuscule ;
- contenir au moins un chiffre.

En revanche, il faut éviter les lettres accentuées, car elles peuvent être codées différemment d'un système à un autre (voir le paragraphe 4.3 page 47).

Comment donc choisir un bon mot de passe ?

Un bon mot de passe est un mot de passe suffisamment compliqué pour qu'il ne puisse pas être deviné mais avec un truc qui permette de s'en souvenir. En effet, si l'on oublie son mot de passe, il n'y a aucun moyen technique simple pour le retrouver, il faut demander au service informatique de le réinitialiser. Pour éviter cela, il ne faut surtout pas noter son mot de passe quelque part (le grand classique est le pense-bête collé sur l'écran ou sous le clavier), il faut s'en souvenir, c'est tout.

Quels trucs peut-on donc utiliser pour se souvenir de son mot de passe ? Chacun a les siens mais voici quelques exemples :

- une formule mathématique, physique ou chimique, par exemple `eipi+1=0` ($e^{i\pi} + 1 = 0$, identité d'Euler) ou `CnH2n+2` (C_nH_{2n+2}, formule des alcanes non cycliques) ;
- les initiales de chaque mot d'une citation, d'un proverbe ou d'un vers, par exemple `Qjaafa1nuas!` (*Que j'aime à faire apprendre un nombre utile aux sages !*) ;
- une phrase en écriture phonétique ou approximative, par exemple `Jai2loq` (que la décence m'interdit d'expliquer).

Par ailleurs, plus un mot de passe est ancien, plus il a de chances d'être découvert. Il convient donc d'en changer régulièrement. Certains centres informatiques imposent à leurs utilisateurs de changer leur mot de passe à intervalle régulier.

> Ce n'est pas le cas à l'ENSTA ParisTech mais ce n'est pas une raison pour passer toute votre scolarité avec le même mot de passe.

Enfin, évitez de changer votre mot de passe le soir, en fin de semaine ou avant de partir en vacances. Le fait de ne pas l'utiliser tout de suite est propice à son oubli.

Il est maintenant possible de changer votre mot de passe. Pour cela, utilisez la commande `passwd` (pour *password*) :

```
$ passwd [Enter]
Changing password for user sii.
Enter login(LDAP) password: mot de passe actuel [Enter]
New UNIX password: nouveau mot de passe [Enter]
Retype new UNIX password: nouveau mot de passe [Enter]
LDAP password information changed for sii
passwd: all authentication tokens updated successfully.
$ ■
```

Il vous est demandé de saisir (chaque saisie est à valider au moyen de la touche [Enter]) :

1. votre mot de passe actuel ;
2. votre nouveau mot de passe (une première fois) ;
3. votre nouveau mot de passe (une seconde fois).

Aucun des trois mots de passe n'apparaît à l'écran, ni de manière directement lisible, ni en affichant quelque chose (un gros point ou un astérisque) à la place de chaque caractère. Ceci est parfaitement normal et a pour but de ne fournir aucune information concernant les mots de passe, pas même leur longueur, à un observateur indiscret.

Notez également que le nouveau mot de passe est à saisir deux fois à l'identique pour se prémunir d'une possible erreur d'inattention lors de sa saisie ou de l'appui inopiné sur une mauvaise touche.

Si la commande `passwd` n'affiche pas de message d'erreur et revient à l'invite, votre mot de passe a été correctement changé. Dans le cas contraire, le message d'erreur indique la nature du problème, par exemple :

```
$ passwd [Enter]
Changing password for user sii.
Enter login(LDAP) password: mot de passe erroné [Enter]
LDAP Password incorrect: try again
Enter login(LDAP) password: ■
```

ou bien :

```
$ passwd [Enter]
Changing password for user sii.
Enter login(LDAP) password: mot de passe actuel [Enter]
New UNIX password: nouveau mot de passe [Enter]
BAD PASSWORD: it is too short
New UNIX password: ■
```

ou bien encore :

```
$ passwd [Enter]
Changing password for user sii.
Enter login(LDAP) password: mot de passe actuel [Enter]
New UNIX password: nouveau mot de passe [Enter]
Retype new UNIX password: mot de passe différent [Enter]
Sorry, passwords do not match.
New UNIX password: ■
```

> http://fr.wikipedia.org/wiki/Mot_de_passe
> http://en.wikipedia.org/wiki/Passwd_(command)

3.7 Verrouillage et fin de session

Si vous quittez temporairement ou définitivement l'ordinateur que vous utilisez, il est respectivement nécessaire de verrouiller ou de fermer votre session. En effet, en votre absence, n'importe qui pourrait l'utiliser et, en particulier, manipuler vos fichiers, ce qui n'est pas souhaitable.

Le verrouillage de la session permet de la protéger des curieux en masquant l'écran. Il est ensuite nécessaire de saisir votre mot de passe pour déverrouiller la session.

Fermer la session permet de revenir au gestionnaire d'affichage. Quelqu'un d'autre peut alors se connecter pour utiliser l'ordinateur.

Le verrouillage de la session n'a de sens que si vous vous absentez pendant peu de temps (ce qui est toujours subjectif). En cas d'absence plus longue, fermer la session est préférable car ceci permet à quelqu'un d'autre de pouvoir utiliser l'ordinateur.

> Dans l'environnement de bureau GNOME utilisé à l'ENSTA ParisTech, ces opérations s'effectuent en allant dans le menu « Système », en haut de l'écran, puis en choisissant :
> – soit l'élément « Verrouiller l'écran » ;
> – soit l'élément « Fermer la session ».
>
> Encore une fois, il est impératif de verrouiller ou de fermer la session en cas d'absence sinon gare aux JBM...

N'éteignez l'ordinateur en aucun cas. Non seulement, cela nécessite d'utiliser une procédure non accessible aux utilisateurs mais l'ordinateur peut être utilisé par quelqu'un d'autre via le réseau (voir ci-après).

3.8 L'environnement informatique de l'ENSTA ParisTech

Tous les ordinateurs de l'ENSTA ParisTech sont reliés entre eux par un réseau local, lui-même relié à l'Internet.

Les ordinateurs des salles de cours sont banalisés, c'est-à-dire que vous pouvez les utiliser indifféremment au moyen du même identifiant et du même mot de passe. Vous retrouverez alors vos fichiers et votre environnement de travail. Ceci est possible en raison de l'utilisation d'un système d'authentification centralisé (qui fait que tous les ordinateurs peuvent reconnaître identifiant et mot de passe) et de serveurs de fichiers (qui font que vos fichiers ne sont pas stockés sur les ordinateurs des salles de cours mais sur ces serveurs, eux-mêmes accédés par ces ordinateurs).

En revanche, les ordinateurs des salles de cours ne sont pas tous semblables, certains sont plus anciens (donc moins puissants) que d'autres. Néanmoins, il est possible de se connecter via le réseau d'un ordinateur à un autre (pour exploiter les capacités d'un ordinateur plus puissant mais situé dans une salle de cours éloignée,

par exemple), de lancer des logiciels sur l'ordinateur distant et de les faire s'afficher sur l'écran du sien. C'est en particulier pour cela qu'il ne faut pas éteindre un ordinateur, quelqu'un d'autre peut l'utiliser à distance.

Sur un réseau informatique, chaque ordinateur est repéré au moyen d'un nom, choisi arbitrairement. Celui-ci est affiché dans le gestionnaire d'affichage. À l'ENSTA ParisTech, pour faire simple, les noms des ordinateurs sont construits à partir du nom de la salle, suivi d'un numéro à partir de 1, croissant selon la disposition des ordinateurs dans la salle.

Par ailleurs, de nombreuses informations concernant l'environnement informatique de l'ENSTA ParisTech sont accessibles, après identification et authentification avec les mêmes identifiant et mot de passe que pour utiliser son compte, sur le site du service informatique édition :

> https://sie.ensta.fr/

Deuxième partie

Du texte et des éditeurs de texte

4

Le texte et son codage

SOUS UNIX, on aime bien manipuler des fichiers contenant du texte et on ramène beaucoup de choses à cela. Par exemple, là où sous Windows, beaucoup de logiciels se configurent exclusivement au moyen de formulaires contenant des boutons à cliquer et des cases à remplir, stockant le résultat on ne sait où ni de quelle façon, sous UNIX, on préfère utiliser un fichier contenant ces informations sous forme de texte (quitte à avoir des formulaires graphiques permettant de générer ce fichier pour les utilisateurs novices).

L'avantage des fichiers texte est qu'ils sont facilement compréhensibles et qu'il est possible au besoin de générer ou de modifier de tels fichiers par des procédures automatiques, ce qui est impossible à faire si la seule façon de gérer ces informations est d'utiliser une interface graphique.

4.1 Le texte, c'est quoi ?

Avant de poursuivre, il faut définir un certain nombre de choses et comprendre ce qu'on entend par « texte ».

Tout d'abord, un *fichier texte* est, tout simplement, un fichier qui contient du texte.

Mais alors, un fichier Word, qui contiendrait par exemple un document tout simple, sans images, est-il un fichier texte ? Vous pouvez faire l'expérience, ouvrez Word, tapez « coucou » suivi d'un retour à la ligne, sauvez le document et regardez la taille du fichier. Faute [1] de disposer de Word, OpenOffice fait également l'affaire, à condition de sauvegarder le fichier au format Word 97/2000/XP, ce qui produit un fichier d'un peu plus de 9 ko. Nous reviendrons dessus mais, généralement, un caractère est codé sur un octet et, « coucou » faisant six caractères, notre fichier devrait donc avoir une taille avoisinant les sept octets (en comptant le retour à la ligne), en tout cas de cet ordre de grandeur. Si la taille du fichier est de 9 ko, c'est

1. Mais est-ce vraiment une faute ?

qu'il contient bien plus que son contenu visible. En pratique, c'est le cas, un fichier Word contient également des informations sur l'auteur d'un document, ses versions successives, les informations de présentation (gras, italique, savoir si un paragraphe est justifié, etc.).

Un fichier Word n'est donc pas un fichier texte, bien qu'il en contienne un peu. On peut alors affiner notre définition en disant qu'un fichier texte est un fichier qui ne contient *que* du texte.

C'est un peu mieux. Il nous reste encore à définir ce qu'est un fichier et ce qu'est du texte.

http://fr.wikipedia.org/wiki/Fichier_texte

4.2 Qu'est-ce qu'un fichier ?

Pour quasiment tous les systèmes d'exploitation, dont UNIX, un fichier est simplement une suite d'octets (voir le paragraphe 2.2.1 page 26) donc de nombres entre 0 et 255. La signification qui est faite de ces nombres dépend du *format* du fichier. Considérons le fichier indiqué en figure 4.1 (les espaces et retours à la ligne ne sont là que pour en faciliter la lecture).

```
10 10
76 70 74 84 77 32 48 97 64 76
76 76 70 74 84 77 32 48 97 64
64 76 76 70 74 84 77 32 48 97
97 64 76 76 70 74 84 77 32 48
48 97 64 76 76 70 74 84 77 32
32 48 97 64 76 76 70 74 84 77
77 32 48 97 64 76 76 70 74 84
84 77 32 48 97 64 76 76 70 74
74 84 77 32 48 97 64 76 76 70
70 74 84 77 32 48 97 64 76 76
```

FIGURE 4.1 – *Exemple de fichier.*

Sans informations supplémentaires sur sa nature, ce fichier pourrait contenir n'importe quoi : la température de mon four mesurée toutes les minutes, la vitesse du vent pendant une tempête...

Maintenant, disons que ce fichier contient une image en niveaux de gris. Les deux premiers octets indiquent sa largeur et sa hauteur (ce qui limite de fait la taille des images à 256 pixels par 256 mais ce n'est pas bien grave) et les octets suivants indiquent le niveau de gris du pixel en question : 0 pour noir, 255 pour blanc, en passant par tous les gris intermédiaires.

Cet ensemble d'informations, permettant de savoir comment interpréter les octets d'un fichier, c'est ce qu'on appelle son *format*.

On peut poursuivre en imaginant un format pour des images en couleurs, indiqué en figure 4.2.

```
10 10
--- --- --- --- --- --- --- --- --- ---
--- --- --- --- --- --- --- --- --- ---
--- --- --- --- --- --- --- --- --- ---
--- --- --- --- --- --- --- --- --- ---
--- --- --- --- --- --- --- --- --- ---
--- --- --- --- --- --- --- --- --- ---
--- --- --- --- --- --- --- --- --- ---
--- --- --- --- --- --- --- --- --- ---
--- --- --- --- --- --- --- --- --- ---
--- --- --- --- --- --- --- --- --- ---
```

FIGURE 4.2 – *Exemple de fichier contenant une image en couleurs.*

Il s'agit toujours d'une image de 10 pixels par 10, chaque bloc de trois octets (ici, chaque tiret figure un octet) représente un pixel et, dans chaque bloc, le premier octet le niveau de rouge (entre 0 et 255), le deuxième le niveau de vert et le troisième le niveau de bleu (en synthèse additive des couleurs).

> http://fr.wikipedia.org/wiki/Fichier_informatique
> http://fr.wikipedia.org/wiki/Format_de_données

4.3 Codage du texte

De la même façon, on peut vouloir représenter du texte sous forme d'une suite de nombres et stocker tout ceci dans un fichier. Il s'agira donc d'un fichier au format texte (ou fichier texte, on y arrive).

À chaque caractère, on associe un nombre, c'est ce qu'on appelle le *codage* du texte, et l'affaire est réglée. Sauf que les choses ne sont pas si simples, on va le voir.

> http://fr.wikipedia.org/wiki/Codage_des_caractères

4.3.1 Le codage ASCII

Le codage ASCII (*American Standard Code for Information Interchange*), historiquement l'un des premiers codages du texte, puisqu'il date de 1967, est encore largement utilisé aujourd'hui.

Comme indiqué dans le tableau 4.1 page suivante, les nombres de 0 à 31 servent à coder des caractères dits *de contrôle* (retour à la ligne, saut de page, etc.). 32 code l'espace, suivie par des signes de ponctuation, les chiffres, encore de la ponctuation, les majuscules dans l'ordre alphabétique, quelques symboles, les minuscules dans l'ordre alphabétique et quelques derniers symboles.

Tout ceci occupe en pratique les nombres de 0 à 127, soit 128 valeurs, ce qui tient donc sur 7 bits. ($128 = 2^7$). En ASCII, le huitième bit est toujours inutilisé.

Le codage ASCII présente un problème majeur. Il ne permet de coder que les lettres non accentuées de l'alphabet latin et est donc insuffisant pour quasiment toutes les langues autres que l'anglais.

> http://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange

4.3.2 Les codages ISO 8859

Or l'ASCII n'utilise que la moitié des nombres qu'on peut représenter avec un octet. L'envie a donc été forte d'étendre l'ASCII pour coder d'autres caractères dans cette plage libre allant de 128 à 255.

C'est l'objet d'un ensemble de normes internationales, les normes ISO 8859 (développées de 1987 à 2004), qui sont décrites dans le tableau 4.2 page 50.

Sans rentrer dans les détails, la partie basse (de 0 à 127) de chaque norme ISO 8859 est en fait exactement l'ASCII et la partie haute (de 128 à 255) code pour les lettres accentuées de chaque ensemble de langues (ISO 8859-1 pour le français, l'italien, etc., ISO 8859-2 pour le hongrois, le polonais, etc.) ou les alphabets des langues non latines (ISO 8859-5 à ISO 8859-8).

C'est un progrès indéniable par rapport à l'ASCII mais il reste deux problèmes :

- on ne peut pas facilement mélanger plusieurs codages dans un même fichier texte ou alors il faut indiquer quelque part quel codage est utilisé et les éventuels changements en cours de fichier mais, dans ce cas, ce n'est plus un fichier texte puisqu'il comporte des indications en plus du texte ;

0	nul	1	soh	2	stx	3	etx	4	eot	5	enq	6	ack	7	bel
8	bs	9	ht	10	nl	11	vt	12	np	13	cr	14	so	15	si
16	dle	17	dc1	18	dc2	19	dc3	20	dc4	21	nak	22	syn	23	etb
24	can	25	em	26	sub	27	esc	28	fs	29	gs	30	rs	31	us
32	sp	33	!	34	"	35	#	36	$	37	%	38	&	39	'
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	del

TABLE 4.1 – *Le codage ASCII.*

ISO 8859-1	langues de l'Europe de l'ouest
ISO 8859-2	langues de l'Europe du centre et de l'est
ISO 8859-3	langues de l'Europe du sud
ISO 8859-4	langues de l'Europe du nord
ISO 8859-5	langues utilisant l'alphabet cyrillique
ISO 8859-6	langue arabe
ISO 8859-7	langue grecque moderne (monotonique)
ISO 8859-8	langue hébraïque
...	
ISO 8859-15	quasiment identique à ISO 8859-1

TABLE 4.2 – *Les codages ISO 8859.*

- les 128 codes supplémentaires ne suffisent pas pour coder les caractères des langues asiatiques (chinois, japonais et coréen, par exemple).

Concernant ce dernier point, il faut savoir qu'il existe par ailleurs de nombreux codages — sur 16 bits compte tenu du nombre de symboles à coder — pour les langues asiatiques.

http://fr.wikipedia.org/wiki/ISO_8859

4.3.3 Unicode et le codage UTF-8

Avec le développement de l'informatique et des télécommunications, le besoin se fit rapidement sentir de disposer d'un système de codage universel des caractères, permettant de coder de manière homogène l'ensemble des caractères, glyphes et symboles des langues de la planète.

Ceci aboutit en 1991 à la norme internationale ISO 10646, plus connue sous son petit nom : *Unicode*. Cette norme définit un système de codage universel des caractères, *Universal Character Set* (UCS).

Il semblait évident que continuer à utiliser des nombres sur 8 bits ne suffirait pas donc les premières versions d'Unicode utilisaient des valeurs sur 16 bits et les suivantes sont rapidement passées à 32 bits. 2^{32} est de l'ordre de 4×10^9 donc cela devrait laisser suffisamment de marge afin de coder une bonne fois pour toutes l'ensemble des systèmes d'écriture de la planète [2].

2. Et même plus. Certaines plages d'Unicode sont réservées pour des utilisations « exotiques ». Ainsi le tengwar de Tolkien et le klingon de Star Trek disposent de plages réservées permettant de les coder...

Comme pour les codages ISO 8859, afin de faciliter la reprise de l'existant, les codes de 0 à 127 d'Unicode correspondent aux caractères ASCII équivalents, à la différence qu'ils sont codés sur quatre octets et non plus sur un seul.

Unicode est une excellente idée sur le papier mais, en pratique, le codage des caractères sur 32 bits est tout sauf optimal. Prenons un exemple.

La chaîne de caractères « coucou » est codée en ASCII par les nombres : 99 111 117 99 111 117, sur six octets, comme on peut le voir dans le tableau 4.3 (où chaque case correspond à un octet).

c	o	u	c	o	u
99	111	117	99	111	117

TABLE 4.3 – *Codage de la chaîne de caractères « coucou » en ASCII.*

En Unicode, la chaîne de caractères « coucou » est codée par les mêmes nombres mais, chacun occupant quatre octets, le total nécessite $6 \times 4 = 24$ octets, comme on peut le voir dans le tableau 4.4.

c				o				u			
0	0	0	99	0	0	0	111	0	0	0	117
c				o				u			
0	0	0	99	0	0	0	111	0	0	0	117

TABLE 4.4 – *Codage de la chaîne de caractères « coucou » en Unicode.*

Ceci fait qu'un fichier texte codé initialement en ASCII verrait sa taille quadruplée si on le codait en Unicode et ce pour rajouter trois octets nuls entre chaque octet significatif. Ce n'est pas acceptable.

En réalité, aucun fichier n'est codé directement en Unicode sur 32 bits par caractère parce que, malgré ses vertus, Unicode en lui-même est loin d'offrir un stockage optimal. C'est pourquoi des méthodes de codages plus économes mais plus complexes sont utilisées.

La plus répandue est *UCS Transformation Format - 8 bits* (qu'on abrège en UTF-8). Inventée en 1992 par Ken Thompson, l'un des créateurs d'UNIX, UTF-8 peut paraître déroutant par rapport aux codages précédents. En effet, les caractères n'y sont pas codés sur une longueur fixe mais sur un, deux, trois ou quatre octets. Les caractères ASCII conservent les mêmes codes et sont stockés dans un octet (qui a donc son huitième bit à zéro). Tout fichier ASCII est donc un fichier UTF-8. Les

autres caractères sont stockés dans deux, trois ou quatre octets, dont le premier a son huitième bit à un et dont la valeur indique le nombre d'octets qui le suivent.

Sans rentrer dans les détails, donnons juste un exemple permettant d'illustrer le codage en UTF-8. Le tableau 4.5 illustre comment la chaîne de caractères « pâté » est codée en ISO 8859-1, un octet par caractère.

p	â	t	é
112	226	116	233

TABLE 4.5 – *Codage de la chaîne de caractères « pâté » en ISO 8859-1.*

Le tableau 4.6 illustre comment la même chaîne de caractères est codée en UTF-8. Les deux caractères accentués sont codés chacun sur deux octets, dont le premier a une valeur strictement supérieure à 127, alors que les deux autres caractères sont codés sur un seul octet, dont la valeur est inférieure ou égale à 127, puisqu'il s'agit de caractères ASCII.

p	â		t	é	
112	195	162	116	195	169

TABLE 4.6 – *Codage de la chaîne de caractères « pâté » en UTF-8.*

On le voit bien dans cet exemple, un texte rédigé en français et codé en UTF-8 donne donc un fichier plus volumineux que s'il était codé en ISO 8859-1, puisque chaque caractère accentué est codé sur deux octets en UTF-8 contre un seul en ISO 8859-1.

C'est d'ailleurs le cas général pour toutes les langues utilisant des caractères en dehors de l'ASCII, leur codage en UTF-8 produit des fichiers un peu plus volumineux qu'avec d'autres codages plus limités (ISO 8859 ou codages sur 16 bits pour les langues asiatiques). En contrepartie, avec UTF-8, il est possible d'utiliser tous les caractères possibles et imaginables dans le même fichier, ce qui est plus qu'appréciable.

Tous les logiciels modernes gèrent parfaitement l'UTF-8 et c'est sans conteste le codage à utiliser dorénavant.

> http://www.unicode.org/
> http://fr.wikipedia.org/wiki/UTF-8

4.4 Codage des fichiers

Lorsqu'on dispose d'un fichier texte, comment déterminer son codage ?

L'idéal est qu'il soit indiqué directement dedans. C'est le cas par exemple des fichiers XML, dont la première ligne indique quel codage est utilisé pour le fichier :

```
<?xml version="1.0" encoding="UTF-8"?>

[...]
```

Le codage peut également être indiqué à l'extérieur du document. C'est le cas pour les messages électroniques, dont le codage est indiqué dans l'en-tête :

```
[...]
MIME-Version: 1.0
Content-Type: text/plain; charset=utf-8
Content-Disposition: inline
Content-Transfer-Encoding: 8bit
[...]

[ corps du message ]
```

4.5 Conversion d'un codage vers un autre

4.5.1 La commande `iconv`

Il peut être utile de convertir du texte d'un codage vers un autre. Imaginez par exemple que avez un fichier codé en ISO 8859-1 que vous voulez convertir en UTF-8. Il existe pour cela sous UNIX un petit utilitaire appelé `iconv` et qui s'utilise comme ceci :

```
$ iconv -f iso8859-1 -t utf-8 -o tata toto
```

Nous n'aborderons l'utilisation de l'interpréteur de commandes qu'au chapitre 7 page 87 mais la séquence ci-dessus est assez simple à comprendre. Elle appelle la commande `iconv` en lui demandant de convertir le fichier `toto` (indiqué en fin de

ligne), dont le codage d'origine (option `-f` pour *from*) est ISO 8859-1 vers le codage de destination (option `-t` pour *to*) UTF-8 et de stocker le résultat (option `-o` pour *output*) dans le fichier `tata`.

> http://en.wikipedia.org/wiki/Iconv

4.5.2 Dans un logiciel

On peut également être amené à effectuer des conversions directement au sein d'un logiciel. Il est en effet possible d'avoir à manipuler des fragments de texte provenant d'origines diverses et codés de façons différentes. Ces fragments sont habituellement convertis le plus tôt possible vers un codage plus universel, UTF-8 étant en l'occurrence un bon candidat.

Le listage 4.1 montre un exemple de conversion de codage de caractères dans le langage de programmation Perl.

```
use Encode ;

[...]

# conversion en UTF-8

$nom = Encode::encode ( 'utf8' , $nom ) ;
$prenom = Encode::encode ( 'utf8' , $prenom ) ;
```

LISTAGE 4.1 – *Conversion de codage de caractères en Perl.*

4.6 Les éditeurs de texte

Un *éditeur de texte* est un logiciel permettant de saisir et de modifier des fichiers texte. À ne pas confondre avec un logiciel de traitement de texte, qui, comme indiqué au paragraphe 4.1 page 45, ne travaille pas sur des fichiers texte.

En raison de l'omniprésence des fichiers texte sous UNIX, il y existe de très nombreux éditeurs de texte. Nous étudierons par la suite les deux plus répandus, et dont beaucoup d'autres ont repris les caractéristiques : Emacs et `vi` (prononcer les lettres à l'anglaise : « vie ail »).

Il existe aussi des éditeurs de texte sur d'autres systèmes d'exploitation qu'UNIX. C'est le cas pour Windows, qui est fourni avec l'éditeur de texte Notepad, ainsi que pour Mac OS X, qui est fourni avec l'éditeur de texte TextEdit.

> http://fr.wikipedia.org/wiki/Éditeur_de_texte
> http://fr.wikipedia.org/wiki/Guerre_d'éditeurs
> http://en.wikipedia.org/wiki/Comparison_of_text_editors
> http://fr.wikipedia.org/wiki/Bloc-notes_(Windows)
> http://fr.wikipedia.org/wiki/TextEdit

5

L'éditeur de texte Emacs

EMACS est probablement l'éditeur de texte le plus répandu sous UNIX. Simple d'emploi et néanmoins très puissant, il convient aux débutants comme aux utilisateurs expérimentés. Emacs dépasse même la simple fonction d'éditeur de texte puisqu'un langage d'extension (que nous n'aborderons pas ici) permet de lui faire faire à peu près n'importe quoi.

> http://www.gnu.org/software/emacs/
> http://fr.wikipedia.org/wiki/Emacs

5.1 Lancement d'Emacs

Emacs se lance depuis un interpréteur de commandes au moyen de la commande `emacs` suivie du nom du fichier à éditer (les fichiers texte ont traditionnellement une extension `.txt`) :

```
$ emacs toto.txt [Enter]
```

Si le fichier `toto.txt` existe, son contenu est affiché et peut être modifié. S'il n'existe pas, Emacs affiche un contenu vierge et le fichier sera créé lors de sa première sauvegarde.

En ce qui nous concerne, le fichier `toto.txt` n'existe pas donc Emacs ouvre une nouvelle fenêtre semblable à celle de la figure 5.1 page suivante.

Vous remarquez, en haut de la fenêtre, des menus et une rangée d'icônes. Nous n'utiliserons quasiment pas ces outils. Pourquoi ? Emacs est un éditeur de texte. Le périphérique d'entrée privilégié pour saisir du texte est, bien évidemment, le clavier. Lorsqu'on utilise un éditeur de texte, les deux mains sont donc principalement situées sur le clavier. Déplacer une main pour agiter la souris puis la ramener vers le clavier représente donc une perte de temps.

FIGURE 5.1 – *Emacs.*

C'est pourquoi nous allons apprendre à utiliser Emacs quasi exclusivement au moyen du clavier. Certaines rares manipulations sont plus efficacement faites avec la souris et ce seront les seules occasions où celle-ci sera utilisée.

Vous remarquez également deux lignes au bas de la fenêtre. La dernière, tout en bas, est appelée *minibuffer* (c'est le terme consacré et il n'a pas d'équivalent établi en français). Le minibuffer sert à Emacs pour communiquer avec l'utilisateur, soit pour lui afficher des informations soit pour lui poser des questions.

La ligne du dessus est appelée *ligne de mode* (*mode line* en anglais). Elle affiche diverses informations qui sont mises à jour automatiquement si nécessaire.

Enfin, la partie principale de la fenêtre est vide. Si le fichier `toto.txt` avait existé, son contenu y aurait été affiché.

Revenons à la ligne de mode. Elle contient :
- des indicateurs, ici `-U:---` :
  - le `U` indique que le fichier est (ou sera s'il n'existe pas encore) codé en UTF-8 (ce serait un `1` pour de l'ISO 8859-1),
  - les deux tirets suivant le deux-points se transforment en astérisques ** lorsque le contenu du fichier a été modifié mais pas encore sauvegardé sur disque, il s'agit d'un repère visuel permettant de savoir si le fichier a été sauvegardé ou pas ;
- le nom du fichier, ici `toto.txt` ;
- la position dans le fichier, qui peut être :
  - `All` lorsque l'ensemble du fichier est visible dans la fenêtre,
  - `Top` lorsque le fichier est trop long pour être entièrement visible dans la fenêtre et qu'on en voit le haut,
  - `Bot` (pour *bottom*) lorsque le fichier est trop long pour être entièrement visible dans la fenêtre et qu'on en voit le bas,
  - un pourcentage indiquant où l'on se trouve dans le fichier si l'on est quelque part à l'intérieur ;
- le numéro de la ligne sur laquelle se trouve le curseur ;
- le mode d'édition du fichier entre parenthèses, c'est de là que la ligne de mode tire son nom.

Emacs est en effet capable de reconnaître sur quel type de fichier texte on travaille (texte simple, programme, etc.) et d'adapter son comportement en fonction. Le mode entre parenthèses indique ce type. Ici, Emacs a reconnu à son extension que le fichier `toto.txt` est un fichier texte simple et s'est donc mis en mode `Text`.

5.2 Saisie du texte

Dans Emacs, la saisie du texte se fait de manière habituelle. La touche ⟵ efface le caractère précédant le curseur, Enter permet de passer à la ligne et les touches fléchées de se déplacer.

Afin d'expérimenter les fonctionnalités d'Emacs, nous aurons besoin d'un peu de texte. Tapez ce qui vous passe par la tête, au moins trois paragraphes (un paragraphe est séparé du suivant par une ligne blanche) contenant chacun trois lignes de texte, avant d'aller plus loin.

Si l'une des lignes est trop longue pour tenir dans la fenêtre, elle se continue sur la ligne suivante, comme on peut voir dans la figure 5.2 page suivante. Il ne s'agit pas de deux lignes mais bien d'une seule et même ligne. Les petites flèches

recourbées présentes dans les marges (et qui sont agrandies dans la figure) sont des repères visuels permettant de le savoir.

FIGURE 5.2 – *Emacs : continuation de ligne.*

Les lignes dépassant du bord de la fenêtre donnent un aspect inesthétique au fichier, il est donc préférable de les éviter.

5.3 Les commandes

Les manipulations plus compliquées que la simple saisie du texte sont faites grâce à des combinaisons de touches appelées *commandes*.

Celles-ci font appel à deux touches du clavier, la touche Ctrl et la touche Meta. Autant la touche Ctrl vous est familière, autant il est inutile de chercher la touche Meta sur le clavier, elle n'y figure pas. Elle existait sur le clavier de l'auteur d'Emacs, Richard Stallman, et elle figure sous ce nom dans toutes les documentations sur Emacs donc le terme est resté.

Sur des claviers modernes, la touche Meta est remplacée par deux touches, qu'on peut utiliser indifféremment en fonction de ses préférences :

- la touche Alt utilisée en conjonction avec une autre touche, c'est-à-dire qu'on appuie d'abord sur Alt et on la laisse enfoncée pendant qu'on appuie sur l'autre touche puis on relâche les deux touches ;
- la touche Esc suivie d'une autre touche, c'est-à-dire qu'on tape d'abord sur Esc (en la relâchant) puis sur l'autre touche.

La notation des commandes est particulière aux documentations sur Emacs et vous les trouverez partout représentées de la même façon.

Une commande introduite par la touche Ctrl se note `C-` suivi du caractère à saisir, une commande introduite par la touche Meta se note `M-` suivi du caractère à saisir.

Par exemple, `C-a` représente l'appui sur la touche Ctrl, qu'on maintient enfoncée en appuyant sur A.

On parle bien ici de caractère et pas seulement de touche. Ainsi, il existe une commande `M-%` (il s'agit de la commande de recherche et de remplacement, qui sera abordée plus loin). `M-%` représente l'appui sur la touche Alt, qu'on maintient enfoncée en appuyant sur Shift ⇧ et % (ou Esc puis Shift ⇧ et % comme indiqué plus haut). L'emploi de la touche Shift ⇧ a son importance car, sans elle, on aurait un 5 sur un clavier QWERTY ou un ù sur un clavier AZERTY et pas un %.

Les commandes les plus utilisées sont introduites directement par `C-` ou `M-`, ce qui revient comme on vient de le voir à l'appui sur deux touches, voire trois. Malheureusement, le nombre de caractères disponibles sur le clavier est limité et il est en pratique nécessaire d'avoir plus de commandes qu'il n'y a de caractères sur le clavier. Certaines commandes moins utilisées sont donc formées de deux combinaisons de touches, dont la première est appelée *préfixe*. Par exemple, la commande `C-x C-c` permet de quitter Emacs. Dans celle-ci, `C-x` est le préfixe. Un préfixe n'est jamais utilisé seul, il a toujours besoin d'être suivi par quelque chose

pour former une commande complète. Emacs attend que la commande complète ait été saisie avant de faire quoi que ce soit. Si l'on tarde à finir la commande après avoir saisi le préfixe, Emacs affiche celui-ci dans le minibuffer, comme indiqué dans la figure 5.3. On peut donc reprendre où l'on était si l'on a été distrait ou interrompu.

FIGURE 5.3 – *Emacs : affichage dans le minibuffer d'une commande partiellement saisie.*

Il existe généralement un moyen mnémotechnique pour retenir les commandes d'Emacs, basé sur la signification (en anglais) du caractère suivant `C-` ou `M-`. Par exemple, `C-d` est la commande permettant d'effacer (`d` pour *delete*) un caractère. Certaines commandes existent en deux versions, l'une introduite par `C-`, l'autre par `M-`. Dans ce cas, la version introduite par `M-` a toujours une portée plus importante que celle introduite par `C-`. Pour reprendre le même exemple, là où `C-d` efface un

caractère, `M-d` efface la fin du mot.

Dans la suite de ce tour d'horizon, nous n'allons pas aborder toutes les commandes d'Emacs mais seulement la vingtaine de commandes qui sert le plus. Il est important d'expérimenter ces commandes sur les quelques lignes de texte saisies plus haut, afin de bien comprendre leur fonctionnement.

5.4 Commandes de déplacement

Il est rare de rédiger un document de manière linéaire. Il est donc utile de pouvoir se déplacer dans le texte de manière efficace. Les touches fléchées peuvent être utilisées pour les petits déplacements mais elles ne permettent pas de grands déplacements rapides. Utiliser la souris n'est pas non plus une bonne solution en raison de la perte de temps liée aux déplacements de la main.

Heureusement, Emacs fournit plusieurs commandes permettant des déplacements efficaces.

La commande `C-a` permet de se déplacer en début de ligne (`a` est la première lettre de l'alphabet), la commande `C-e` en fin de ligne (`e` pour *end*).

La commande `C-` suivie d'une touche fléchée permet de se déplacer de mot en mot (un mot étant séparé d'un autre par une espace) pour les flèches horizontales et de paragraphe en paragraphe (un paragraphe étant séparé d'un autre par une ligne blanche) pour les flèches verticales.

Dans un long document, il est fréquent de devoir se positionner sur une ligne précise, identifiée par son numéro, par exemple parce qu'une erreur a été détectée à cet endroit. La commande `M-g M-g` (ou `M-g g`) affiche `Goto line:` dans le minibuffer, demandant qu'on saisisse le numéro de la ligne où aller, suivi de Enter .

> http://www.gnu.org/software/emacs/manual/html_node/emacs/Moving-Point.html

5.5 Commandes de suppression

La commande `C-d` (`d` pour *delete*) permet de supprimer le caractère situé sous le curseur, la commande `M-d` la fin du mot à partir de la position du curseur jusqu'à la prochaine espace ou jusqu'au prochain signe de ponctuation.

À partir de la position du curseur, la commande `C-k` (`k` comme *kill*) supprime la fin de la ligne.

> http://www.gnu.org/software/emacs/manual/html_node/emacs/Deletion.html
> http://www.gnu.org/software/emacs/manual/html_node/emacs/Words.html
> http://www.gnu.org/software/emacs/manual/html_node/emacs/Killing-by-Lines.html

5.6 Exécution multiple d'une commande

Il est possible d'exécuter *n* fois la même commande en la préfixant par `M-`*n*. Ainsi, on peut effacer 5 lignes au moyen de `M-5 C-k`.

On peut répéter la dernière commande à l'identique, sans avoir besoin de la retaper, au moyen de `C-x z` (attention, c'est `C-x z`, pas `C-x C-z`). Immédiatement après avoir utilisé `C-x z`, il suffit de taper une nouvelle fois `z` pour répéter la même commande et ainsi de suite.

> http://www.gnu.org/software/emacs/manual/html_node/emacs/Arguments.html
> http://www.gnu.org/software/emacs/manual/html_node/emacs/Repeating.html

5.7 Copier-coller

La procédure décrite ci-dessous, permettant de copier puis de coller du texte, n'est pas spécifique à Emacs, même s'il semble opportun de la décrire ici. En effet, cette procédure fonctionne avec n'importe quel logiciel manipulant du texte sous UNIX.

C'est l'une des rares occasions où nous allons utiliser la souris. Il existe bien des commandes d'Emacs permettant de sélectionner puis de coller du texte en n'utilisant que le clavier mais il faut reconnaître qu'utiliser la souris est, dans ce cas précis, plus efficace.

Pour copier du texte, il suffit de le sélectionner à la souris. On positionne la souris au début du texte à copier, on appuie sur le bouton de gauche, on le maintient ainsi et on déplace la souris à la fin du texte à copier puis on relâche le bouton. Rien que de plus classique ici.

Petite astuce, lorsqu'on souhaite copier un mot, un double clic quelque part sur ce mot le sélectionne sans avoir à faire tout cela. Pour copier une ligne, c'est l'affaire d'un triple clic.

Pour coller le texte préalablement copié, il suffit de déplacer la souris à l'endroit où l'on veut le voir apparaître et d'appuyer sur le bouton du milieu (qui est bien souvent la molette).

Si le même texte est à coller à plusieurs endroits, il suffit de se repositionner et de cliquer à nouveau sur le bouton du milieu, le texte copié restant en mémoire tant qu'une autre portion de texte n'est pas copiée.

Et c'est tout ! Cette méthode est bien plus simple que ce qui se fait sur Macintosh ou sous Windows, qui implique l'utilisation de la souris et du clavier. Sous UNIX, tout se fait à la souris.

5.8 Couper-coller

En revanche, il n'existe pas de procédure équivalente utilisant uniquement la souris pour réaliser un couper-coller. La sélection du texte se fait de la même manière que pour le copier-coller. Puis on utilise la commande `C-w` pour couper le texte sélectionné et on le colle comme pour le copier-coller.

> http://www.gnu.org/software/emacs/manual/html_node/emacs/Other-Kill-Commands.html

5.9 Commande d'annulation des modifications

L'erreur est humaine. Et il arrive régulièrement de commettre une erreur par mégarde. Il est alors possible d'annuler la dernière modification apportée au document en utilisant la commande `C-/` ou la commande `C-_` (les deux commandes ont le même effet, la première est plus facile à saisir sur un clavier QWERTY, la seconde est plus facile à saisir sur un clavier AZERTY). En répétant cette commande, on annule l'avant-dernière modification et ainsi de suite.

> http://www.gnu.org/software/emacs/manual/html_node/emacs/Undo.html

5.10 Commandes de gestion de fichiers

Nous avons maintenant suffisamment travaillé sur notre document pour vouloir le sauvegarder sur disque.

Il est important, et ceci vaut pour tous les logiciels, d'effectuer des sauvegardes régulières. En effet, entre deux sauvegardes, les modifications apportées au document

ne sont stockées que dans la mémoire vive de l'ordinateur et l'on n'est jamais à l'abri d'une coupure de courant ou de quelqu'un qui trébuche sur le câble d'alimentation en le débranchant au passage, ce qui fait irrémédiablement perdre ces modifications. Par ailleurs, il vaut mieux ne pas compter sur les sauvegardes automatiques que certains logiciels effectuent, elle se font rarement quand il le faut.

La commande `C-x C-s` (`s` pour *save*) permet de sauvegarder le fichier sur disque.

À l'occasion de la sauvegarde du fichier, on peut remarquer que les deux astérisques des indicateurs de la ligne de mode ont été remplacés par deux tirets, comme on peut le voir dans la figure 5.4.

FIGURE 5.4 – *Emacs : état de la ligne de mode après sauvegarde du fichier.*

Si jamais on a lancé Emacs sans argument, ce qui est mal, et qu'on souhaite sauvegarder le fichier, il faut lui donner un nom en le sauvegardant. Pour cela, on

utilise la commande `C-x C-w` (`w` pour *write*). Elle demande de saisir le nom à donner au fichier, suivi de Enter, dans le minibuffer.

Si jamais on a lancé Emacs sans argument et qu'on souhaite charger un fichier, on utilise la commande `C-x C-f` (`f` poue *find*). Elle demande de saisir le nom du fichier à charger, suivi de Enter, dans le minibuffer.

> http://www.gnu.org/software/emacs/manual/html_node/emacs/Save-Commands.html
> http://www.gnu.org/software/emacs/manual/html_node/emacs/Visiting.html

5.11 Commandes de recherche et de remplacement

Emacs a une façon particulière et extrêmement efficace d'effectuer une recherche par mot clé.

La commande `C-s` (`s` pour *search*) permet de commencer une recherche à partir du curseur en descendant (la commande `C-r` (`r` pour *reverse*) effectue une recherche en remontant). Elle affiche `I-search:` dans le minibuffer pour indiquer qu'on se trouve maintenant en mode de recherche interactive.

En quoi la recherche est-elle interactive ?

Supposons qu'on recherche le mot « toto ». Au moment où l'on tape t, Emacs déplace le curseur sur le premier « `t` » suivant la précédente position du curseur et affiche les autres sur un fond coloré. Lorsqu'on tape o, Emacs déplace le curseur sur la première chaîne de caractères « to » suivant sa position précédente (ou ne bouge pas si l'on y était déjà) et, de même, affiche les autres sur un fond coloré. Et ainsi de suite.

En pratique, pour rechercher un mot, il suffit bien souvent d'en taper les premières lettres et on le trouve sans avoir besoin de le saisir en entier. C'est pratique si l'on cherche « anticonstitutionnellement », il doit suffire de taper « anticon »...

Pour stopper la recherche, on appuie sur la touche Enter. Le curseur reste alors sur le premier mot trouvé.

Il est alors possible de passer au prochain mot correspondant à la même recherche grâce à la commande `C-s C-s`. Un simple `C-s` passe alors au mot suivant et ainsi de suite. La recherche s'arrête en tapant Enter.

Le remplacement s'effectue au moyen de la commande `M-%` (qui utilise donc trois touches du clavier, Meta, Shift ⇧ et %).

Emacs affiche alors `Query replace:` dans le minibuffer, demandant qu'on saisisse le mot à remplacer. Pour prendre un exemple, disons qu'on veut remplacer « toto »

par « tata ». On tape donc le mot à remplacer, suivi de Enter . Emacs demande alors `Query replace toto with:` dans le minibuffer. On tape le mot de remplacement suivi de Enter .

Emacs se place alors sur le premier mot trouvé, qui s'affiche sur un fond coloré afin de le distinguer facilement. On a alors la possibilité de taper :

- `y` pour effectuer le remplacement et passer à l'occurrence suivante ;
- `n` pour ne pas effectuer le remplacement et passer à l'occurrence suivante ;
- `!` pour effectuer tous les remplacements sans poser de question ;
- `q` pour quitter.

En fin de procédure, Emacs affiche le nombre d'occurrences remplacées dans le minibuffer.

> http://www.gnu.org/software/emacs/manual/html_node/emacs/Basic-Isearch.html
> http://www.gnu.org/software/emacs/manual/html_node/emacs/Query-Replace.html

5.12 Saisie de caractères ne figurant pas sur le clavier

Les claviers, quels qu'ils soient, ne permettent pas de saisir directement la totalité des caractères utilisés en français. Le problème est encore plus flagrant lorsqu'il s'agit de rédiger tout ou partie d'un document dans une langue étrangère autre que l'anglais.

Emacs permet de saisir de nombreux caractères absents des claviers au moyen de la commande `C-x 8` (attention, c'est `C-x 8`, pas `C-x C-8`) suivi d'un ou deux caractères (pour les signes figurant dans le codage ISO 8859-1) ou de Enter puis du nom complet du signe voulu (pour les autres).

Par exemple, `C-x 8 / e` permet de saisir « æ » et `C-x 8` Enter `EURO SIGN` permet de saisir « € ».

La liste des combinaisons d'un ou deux caractères pouvant suivre `C-x 8` est accessible au moyen de la commande `C-x 8 C-h`.

Le tableau 5.1 page ci-contre indique les combinaisons les plus utiles en français.

> http://www.gnu.org/software/emacs/manual/html_node/emacs/Unibyte-Mode.html

C-x 8 / e	æ	C-x 8	Enter	LATIN SMALL LIGATURE OE	œ
C-x 8 / E	Æ	C-x 8	Enter	LATIN CAPITAL LIGATURE OE	Œ
C-x 8 <	«	C-x 8	Enter	EURO SIGN	€
C-x 8 >	»	C-x 8 o			°

TABLE 5.1 – *Combinaisons permettant de saisir des caractères usuels en français avec Emacs.*

5.13 Commande d'annulation de la commande en cours de saisie

Nous avons vu précédemment la commande `C-/`, qui permet d'annuler la dernière modification apportée au document. Il existe une autre commande d'annulation, dans un autre contexte.

Il peut en effet être utile d'annuler la saisie d'une commande demandant quelque chose dans le minibuffer. Supposons par exemple qu'on utilise la commande `C-x C-w` par erreur et qu'on souhaite l'annuler. Il ne s'agit pas d'annuler la dernière modification apportée au document, qu'on souhaite conserver tel quel, mais d'annuler la commande en cours, qui est en attente d'informations supplémentaires.

Dans ce cas, on utilise la commande `C-g`. Celle-ci annule toute commande en attente d'informations via le minibuffer et laisse le document en l'état.

5.14 Commandes diverses

Lorsqu'on travaille sur un fichier texte contenant un document écrit dans une langue telle que le français, il arrive fréquemment d'apporter des modifications à un paragraphe rédigé auparavant. Tant et si bien qu'au bout d'un moment, on se retrouve avec des paragraphes contenant des lignes ayant des longueurs disparates, ce qui provoque un effet visuel disgracieux.

La commande `M-q`, tapée n'importe où à l'intérieur d'un paragraphe, permet de le reformater afin d'harmoniser la longueur des lignes, par défaut à `70` caractères au plus. Elle ne fonctionne que dans les documents pour lesquels le mode d'Emacs est `Text`.

Il peut être utile de travailler simultanément sur deux parties d'un même fichier ou sur deux fichiers différents, le tout dans la même fenêtre, afin par exemple de copier du texte de l'un vers l'autre.

La commande `C-x 2` (attention, c'est `C-x 2`, pas `C-x C-2`) permet de séparer la fenêtre en deux parties présentant le même fichier. Il est alors possible de se positionner dans chaque partie à un endroit différent du fichier, voire de charger un autre fichier dans l'une des parties au moyen de la commande `C-x C-f`. On peut passer de l'une à l'autre partie au moyen de la commande `C-x o`. Enfin, la commande `C-x 0` ramène la fenêtre d'Emacs à son état initial.

Signalons qu'il existe aussi une commande `C-x 3`, dont le résultat est laissé à la curiosité du lecteur.

Pour finir, il faut dire quelques mots sur la commande `M-x`. Les commandes abordées jusqu'à présent l'ont été via leur raccourci. Les commandes d'Emacs ont également un nom et elles peuvent être exécutées au moyen de ce nom, introduit par la commande `M-x`. Par exemple, la commande `kill-line` a pour raccourci `C-k` et peut être également exécutée en tapant `M-x`, puis `kill-line` dans le minibuffer.

En pratique, rares sont les commandes d'Emacs disposant d'un raccourci. Seules les plus utilisées en ont un. `M-x` est donc le seul moyen d'exécuter les autres.

Vous pouvez par exemple essayer la commande `M-x auto-fill-mode` dans un fichier texte. Le mode passera de `Text` à `Text Fill`. Ceci permet, lorsqu'on arrive en bout de ligne, de faire en sorte qu'Emacs insère automatiquement un saut de ligne, ce qui évite d'avoir recours à `M-q` lorsqu'on saisit un paragraphe (mais cette commande est néanmoins souvent utile au moment des retouches).

> http://www.gnu.org/software/emacs/manual/html_node/emacs/Quitting.html
> http://www.gnu.org/software/emacs/manual/html_node/emacs/Fill-Commands.html
> http://www.gnu.org/software/emacs/manual/html_node/emacs/Split-Window.html
> http://www.gnu.org/software/emacs/manual/html_node/emacs/M_002dx.html

5.15 Quitter

Comme nous l'avons vu dans la présentation des commandes, la commande `C-x C-c` permet de quitter Emacs. Si le fichier en cours d'édition n'a pas été sauvegardé, Emacs propose de le faire ou de quitter en perdant les modifications effectuées depuis la dernière sauvegarde.

Vu la proximité des touches en question, il est d'ailleurs plus rapide de sauvegarder (`C-x C-s`) puis de quitter (`C-x C-c`) plutôt que de quitter sans sauvegarder et de demander à Emacs de le faire lorsqu'il ne manquera pas de poser la question.

http://www.gnu.org/software/emacs/manual/html_node/emacs/Exiting.html

5.16 Conclusion

Nous avons donc étudié la vingtaine de commandes permettant une utilisation efficace d'Emacs. Celles-ci sont résumées dans le tableau A.1 page 369.

Il ne s'agit que d'un survol très rapide de ce qu'offre Emacs mais permettant de se débrouiller dans la plupart des cas. Le lecteur curieux pourra se reporter aux ouvrages suivants de la bibliographie pour approfondir ses connaissances :

- *GNU Emacs Manual* [8]
- *Learning GNU Emacs* [2]
- *GNU Emacs : UNIX Text Editing and Programming* [6]
- *GNU Emacs Pocket Reference* [1]

6

Les éditeurs de texte `vi` et Vim

L'ÉDITEUR DE TEXTE `vi` (prononcer les lettres à l'anglaise : « vie ail ») est l'un des plus anciens éditeurs de texte sous UNIX. D'approche difficile, il se révèle d'une puissance extraordinaire pour peu qu'on prenne la peine d'en apprendre le fonctionnement. C'est d'ailleurs l'éditeur de texte favori des vrais amateurs d'UNIX [1].

Il souffre néanmoins d'un défaut : il n'évolue plus depuis de nombreuses années et il lui manque certaines caractéristiques bien utiles qu'on retrouve dans tous les éditeurs de texte modernes. Le `vi` d'origine continue pourtant à être utilisé par les ingénieurs système pour sa simplicité et sa sécurité.

Ceux qui ont besoin de plus utilisent généralement Vim (*Vi IMproved*). Il s'agit d'un logiciel inspiré de `vi` et offrant de nombreuses améliorations. Ce document a d'ailleurs été rédigé avec Vim.

Dans la plupart des distributions Linux, Vim est installé à la place de `vi`. C'est le cas à l'ENSTA ParisTech.

Sauf indication particulière, ce que nous allons étudier par la suite concernera `vi` et tous ses dérivés sans distinction, dont Vim. Nous indiquerons néanmoins quelques spécificités intéressantes propres à Vim.

> http://fr.wikipedia.org/wiki/Vi
> http://www.vim.org/
> http://fr.wikipedia.org/wiki/Vim

6.1 Lancement de `vi` et de Vim

`vi` se lance depuis un interpréteur de commandes au moyen de la commande `vi` (étonnant, non ?) suivie du nom du fichier à éditer :

1. D'aucuns diront : des unixiens barbus (pléonasme).

```
$ vi toto.txt Enter
```

Vim, sans surprise, se lance au moyen de la commande `vim` suivie du nom du fichier à éditer :

```
$ vim toto.txt Enter
```

Dans la plupart des distributions Linux, `vi` lance en fait `vim` (c'est le cas à l'ENSTA ParisTech).

Contrairement à Emacs, `vi` n'ouvre pas de fenêtre et reste confiné dans celle de l'interpréteur de commandes. On peut voir dans la figure 6.1 page ci-contre ce qu'affiche Vim sur un fichier de deux lignes.

Quelques indications sont affichées tout en bas :

- le nom du fichier en cours d'édition ;
- le nombre de lignes du fichier ;
- la taille en octets du fichier ;
- le numéro de ligne et le numéro de colonne du curseur (`1,1` au moment du lancement) ;
- la position dans le fichier, sur le même principe qu'Emacs (`All`, `Top`, `Bot` ou un pourcentage).

Remarquez que les lignes situées au-delà de la fin du fichier débutent par ~. Il s'agit d'une indication visuelle permettant de les distinguer d'éventuelles lignes vides en fin de fichier.

6.2 Saisie du texte

Contrairement à Emacs, dans lequel on peut taper du texte dès son lancement, il est nécessaire dans `vi` de passer tout d'abord en mode d'insertion de texte (qu'on appelle plus simplement *mode insertion*).

En effet, `vi` a deux modes de fonctionnement :

- le mode de saisie des commandes (*mode commande*) ;
- le mode d'insertion de texte (*mode insertion*).

Après son lancement, `vi` est en mode commande et il n'existe pas moins de six façons de passer en mode insertion en fonction de l'endroit auquel on souhaite voir apparaître le texte qu'on va saisir. Le passage en mode insertion se fait en tapant un caractère au clavier :

```
Il est joli mon fichier toto.txt et j'aime bien
les cours d'informatique de l'ENSTA ParisTech !
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"toto.txt" 2L, 96C                                1,1           All
```

FIGURE 6.1 – *Vim.*

- `i` (*insert*) pour passer en mode insertion à l'endroit où se trouve le curseur, c'est-à-dire que le texte qu'on saisira par la suite apparaîtra à partir de la position actuelle du curseur ;
- `a` (*after*) pour passer en mode insertion immédiatement après le curseur, c'est-à-dire que le texte qu'on saisira par la suite apparaîtra à droite de la position actuelle du curseur ;
- `I` pour passer en mode insertion au début de la ligne sur laquelle se situe le curseur ;
- `A` pour passer en mode insertion à la fin de la ligne sur laquelle se situe le curseur ;
- `o` (*open*) pour créer une nouvelle ligne au-dessous de celle sur laquelle se situe le curseur (le reste du document est alors décalé d'une ligne vers le bas) et passer en mode insertion au début de cette ligne ;
- `O` pour créer une nouvelle ligne au-dessus de celle sur laquelle se situe le curseur (le reste du document est alors décalé d'une ligne vers le bas) et passer en mode insertion au début de cette ligne ;

Le tableau 6.1 résume les six façons de passer en mode insertion.

au caractère situé sous le curseur	`i`
au caractère situé après le curseur	`a`
en début de ligne	`I`
en fin de ligne	`A`
au-dessous de la ligne actuelle	`o`
au-dessus de la ligne actuelle	`O`

TABLE 6.1 – *Les six façons de passer en mode insertion dans* `vi`.

Pourquoi proposer autant de façons de passer en mode insertion ? En effet, compte tenu des possibilités de déplacement du curseur, seules les deux premières façons sont réellement nécessaires. Dans `vi`, contrairement à Emacs, on ne peut pas déplacer le curseur au-delà du dernier caractère d'une ligne donc `i` ne suffit pas. En revanche, en combinant déplacements, `i`, `a` et le retour à la ligne, on peut émuler `I`, `A`, `o` et `O`.

En observant la façon dont on utilise généralement un éditeur de texte, on se rend compte qu'il arrive fréquemment d'avoir à ajouter du texte en début de ligne ou en fin de ligne et d'avoir à rajouter du texte entre deux lignes existantes. Les six façons de passer en mode insertion permettent de le faire rapidement sans avoir

d'abord à se déplacer, puis à passer en mode insertion (et éventuellement à rajouter une ligne blanche).

C'est là une des caractéristiques de `vi`, qui fait tout son attrait. Il existe de nombreuses commandes superflues mais qui correspondent à un usage répandu et qui permettent une utilisation efficace.

Le passage en mode insertion est matérialisé par l'indication `Insert` en bas à droite de la fenêtre dans `vi` et par l'indication `-- INSERT --` en bas à gauche de la fenêtre dans Vim.

Une fois en mode insertion, la saisie du texte se fait de manière habituelle. La touche [←] efface le dernier caractère saisi, [Enter] permet de passer à la ligne et les touches fléchées de se déplacer.

Afin d'expérimenter les fonctionnalités de `vi`, nous aurons besoin d'un peu de texte. Tapez ce qui vous passe par la tête, au moins trois paragraphes (un paragraphe est séparé du suivant par une ligne blanche) contenant chacun trois lignes de texte, avant d'aller plus loin.

6.3 Les commandes

Lorsqu'on est en mode insertion, il faut appuyer sur la touche [Esc] pour passer en mode commande.

Le passage en mode commande est matérialisé par le remplacement de l'indication `Insert` par `Command` en bas à droite de la fenêtre dans `vi` et par la disparition de l'indication `-- INSERT --` en bas à gauche de la fenêtre dans Vim.

Lorsqu'on utilise `vi`, on alterne donc entre le mode commande et le mode insertion.

Puisque `vi` dispose de deux modes de fonctionnement, là où Emacs n'en a qu'un, il n'est pas nécessaire d'introduire les commandes par une touche particulière. Nous avons d'ailleurs déjà vu que les commandes de passage en mode insertion utilisaient directement des caractères usuels, sans touche d'introduction.

Certaines commandes utilisent un seul caractères et produisent un résultat immédiat (c'est par exemple le cas des commandes de passage en mode insertion), d'autres en utilisent plusieurs. Dans ce cas, au fur et à mesure que l'on en saisit les caractères, Vim les indique au bas de la fenêtre à gauche de la position du curseur alors que `vi` n'en fait rien.

Dans la suite de ce tour d'horizon, nous n'allons pas aborder toutes les commandes de `vi` mais seulement la vingtaine de commandes qui sert le plus. Il est

important d'expérimenter ces commandes sur les quelques lignes de texte saisies plus haut, afin de bien comprendre leur fonctionnement.

6.4 Commandes de déplacement

Il est possible de se déplacer dans un document au moyen des touches fléchées, que ce soit en mode commande ou en mode insertion. À noter que d'anciennes versions de `vi` ne permettaient les déplacements qu'en mode commande.

En mode commande, on peut également utiliser `h`, `j`, `k` et `l` pour se déplacer respectivement vers la gauche, vers le bas, vers le haut et vers la droite.

La commande `0` permet de se déplacer en début de ligne, `$` en fin de ligne (`$` désigne toujours la fin de quelque chose pour `vi`).

Les commandes `W` (pour *word*) et `B` (pour *back*) permettent de se déplacer de mot en mot (respectivement en avant et en arrière). Les commandes `{` et `}` permettent de se déplacer de paragraphe en paragraphe (respectivement en montant et en descendant).

La commande `G` (pour *go*) permet de se déplacer sur la dernière ligne du document. Précédée d'un nombre, elle permet de se déplacer sur la ligne ayant ce numéro. Par exemple, la commande `33G` permet de se déplacer sur la ligne numéro 33.

6.5 Commandes de suppression

La commande `x` permet de supprimer le caractère situé sous le curseur, la commande `dw` (pour *delete word*) la fin du mot à partir de la position du curseur jusqu'à la prochaine espace ou jusqu'au prochain signe de ponctuation.

La commande `D` (pour *delete*) supprime la fin de la ligne à partir de la position du curseur. La commande `dd` supprime totalement la ligne, où que le curseur y soit situé.

6.6 Exécution multiple d'une commande

Il est possible d'exécuter *n* fois la même commande en la préfixant par ce nombre. Ainsi, on peut effacer 5 lignes au moyen de `5dd`.

On peut répéter la dernière commande à l'identique, sans avoir besoin de la retaper, au moyen de `.` (point). C'est l'une des caractéristiques les plus appréciées des amateurs de `vi`.

6.7 Copier-coller

Le copier-coller à la souris s'effectue exactement comme avec Emacs. Il faut néanmoins prendre la précaution de positionner le curseur au bon endroit (c'est en effet à la position du curseur que le texte sera collé et non à celle de la souris) et de passer en mode insertion avant de coller le texte (sinon `vi` interprétera le texte collé comme une suite de commandes, ce qui ne produira pas le résultat attendu).

Il est également possible de copier-coller une ou plusieurs lignes grâce à deux commandes. La commande `yy` (pour *yank*) copie la ligne sur laquelle se situe le curseur. Comme vu au paragraphe 6.6 page ci-contre, on peut préfixer cette commande par un nombre afin de copier autant de lignes. Ainsi, la commande `5yy` copie cinq lignes (celle sur laquelle se situe le curseur et les quatre situées au-dessous). Après s'être déplacé au bon endroit, on utilise la commande `p` (pour *paste*) afin de coller la ou les lignes copiées sous la ligne sur laquelle se situe le curseur (ce qui décale la suite du texte vers le bas).

6.8 Couper-coller

Le couper-coller s'effectue selon le même principe. Il utilise la commande `dd` vue au paragraphe 6.5 page précédente pour couper la ligne sur laquelle se situe le curseur. Comme vu au paragraphe 6.6 page ci-contre, on peut préfixer cette commande par un nombre afin de couper autant de lignes. Ainsi, la commande `5dd` coupe cinq lignes (celle sur laquelle se situe le curseur et les quatre situées au-dessous). Après s'être déplacé au bon endroit, on utilise la commande `p` pour coller la ou les lignes coupées sous la ligne sur laquelle se situe le curseur (ce qui décale la suite du texte vers le bas).

6.9 Commande d'annulation des modifications

L'erreur est humaine. Et il arrive régulièrement de commettre une erreur par mégarde. Il est alors possible d'annuler la dernière modification apportée au document en utilisant la commande `u` (pour *undo*). En répétant cette commande, on annule l'avant-dernière modification et ainsi de suite.

6.10 Commandes de gestion de fichiers

La commande `:w` (pour *write*) permet de sauvegarder le fichier, si celui-ci a déjà un nom. S'il n'en a pas, parce qu'on a lancé `vi` sans le faire suivre d'un nom de fichier, ce qui est mal, on peut utiliser `:w`, suivie d'une espace, du nom à donner au fichier et de Enter .

La commande `:e` (pour *edit*), suivie d'une espace, d'un nom de fichier et de Enter permet de charger le fichier en question dans `vi`.

6.11 Commandes de recherche et de remplacement

La recherche par mot clé s'effectue en tapant `/` immédiatement suivi du mot à rechercher, puis Enter . Elle s'effectue depuis la position du curseur vers la fin du fichier. Pour faire une recherche en remontant, il faut taper `?` immédiatement suivi du mot à rechercher, puis Enter . Pour répéter la recherche précédente, on utilise la commande `n`.

Le remplacement a une syntaxe un peu compliquée de prime abord. Pour remplacer « toto » par « tata » dans tout le fichier, on utilise :

```
:1,$s/toto/tata/g
```

La commande de remplacement, comme toutes les commandes complexes dans `vi`, commence par `:` et se conclut par Enter .

On indique ensuite d'où à où effectuer l'opération sous la forme de deux numéros de ligne, séparés par une virgule. On peut utiliser `$` pour indiquer la fin du fichier. Ici, le remplacement s'effectue donc de la première ligne jusqu'à la fin du fichier.

Vient ensuite `s` (pour *substitute*) qui indique qu'on souhaite effectuer un remplacement.

Suivent le mot à rechercher et celui devant le remplacer, encadrés et séparés par des `/`.

Enfin, on indique `g` (pour *global*) afin d'effectuer un remplacement global. Sans cette indication, en cas de présence de plusieurs occurrences du mot recherché sur une même ligne, le remplacement ne concernerait que la première.

6.12 Saisie de caractères ne figurant pas sur le clavier

Les fonctionnalités décrites dans cette section sont spécifiques à Vim.

Vim permet très facilement de saisir de nombreux caractères absents des claviers. Le principe est, en mode insertion, de taper `CTRL-K` suivi de deux caractères ASCII qu'on peut trouver sur n'importe quel clavier. `CTRL-K` est la notation utilisée dans la documentation de Vim pour représenter l'appui sur la touche Ctrl, qu'on maintient enfoncée en appuyant sur K.

Par exemple, `CTRL-K oe` permet de saisir « œ ».

La liste des combinaisons de caractères pouvant suivre `CTRL-K` est accessible au moyen de la commande :

```
:digraphs
```

qu'on peut abréger en `:dig` pour aller plus vite. La navigation dans cette liste s'effectue de la même façon qu'avec la commande `more` (voir le paragraphe 7.4.2 page 102 pour en avoir l'explication). Force est de constater à la lecture de cette liste que la façon de faire de Vim est bien plus complète et bien plus simple que celle d'Emacs.

Le tableau 6.2 indique les combinaisons les plus utiles en français.

`CTRL-K ae`	æ	`CTRL-K oe`	œ
`CTRL-K AE`	Æ	`CTRL-K OE`	Œ
`CTRL-K <<`	«	`CTRL-K Eu`	€
`CTRL-K >>`	»	`CTRL-K DG`	°

TABLE 6.2 – *Combinaisons permettant de saisir des caractères usuels en français avec Vim.*

http://vimdoc.sourceforge.net/htmldoc/digraph.html

6.13 Commande d'annulation de la commande en cours de saisie

Nous avons vu précédemment la commande u, qui permet d'annuler la dernière modification apportée au document. Comme pour Emacs, il existe une autre forme d'annulation, permettant d'annuler la saisie d'une commande comprenant plusieurs caractères (celles ne consistant que d'un caractère ayant un effet instantané). Il s'agit tout simplement de la touche Esc .

6.14 Commandes diverses

La commande J (pour *join*) permet de joindre deux lignes. La ligne sur laquelle se situe le curseur et la ligne suivante fusionnent pour n'en plus former qu'une seule. Une espace est automatiquement ajoutée entre les deux anciennes lignes.

Spécificité de Vim, il est possible de faire en sorte que, lors de la saisie du texte, un saut de ligne soit inséré automatiquement lorsqu'on dépasse une certaine longueur de ligne. Ceci se fait au moyen de la commande :

```
:set textwidth=65
```

Ici, un saut de ligne sera automatiquement inséré lors de la saisie d'une ligne si celle-ci dépasse 65 caractères. Le saut de ligne sera inséré à la place de la dernière espace précédant cette limite, repoussant le mot en cours de saisie sur la ligne suivante.

Une fois cette limite définie, la commande gqap (toujours spécifique à Vim) permet de reformater un paragraphe afin d'harmoniser la longueur des lignes, au plus à la longueur définie par la commande précédente.

6.15 Quitter

Enfin, il existe plusieurs commandes permettant de quitter vi. La plus simple est la commande :q (pour *quit*). Elle n'accepte de quitter vi que si le fichier en cours d'édition a été sauvegardé. S'il ne l'a pas été, elle affiche un message d'erreur. On peut sauvegarder le fichier puis quitter vi grâce aux commandes :wq (pour *write and quit*), :x (pour *exit*) et ZZ (il y en a trois mais elles donnent exactement le même résultat).

Si l'on souhaite quitter `vi` sans sauvegarder le fichier, on utilise la commande `:q!` (le point d'exclamation force à quitter sans sauvegarder le fichier).

6.16 Conclusion

Nous avons donc étudié la vingtaine de commandes permettant une utilisation efficace de `vi`. Celles-ci sont résumées dans le tableau A.2 page 372.

Il ne s'agit que d'un survol très rapide de ce qu'offre `vi` mais permettant de se débrouiller dans la plupart des cas. Le lecteur curieux pourra se reporter aux ouvrages suivants de la bibliographie pour approfondir ses connaissances :

- *Learning the vi and Vim Editors* [5]
- *Hacking Vim 7.2* [7]
- *Vi IMproved—Vim* [3]
- *vi Editor Pocket Reference* [4]

Troisième partie

L'interpréteur de commandes

7

L'interpréteur de commandes

De nos jours, un ordinateur se manipule principalement avec la souris, au moyen d'une interface graphique présentant à l'utilisateur des fenêtres, des menus, des boutons, etc. Cela n'a pas toujours été le cas. Autrefois, on exprimait ce qu'on voulait faire faire à l'ordinateur par écrit, grâce à un vocabulaire spécifique, les *commandes*, qui étaient comprises par un logiciel particulier appelé *interpréteur de commandes* (*shell* en anglais).

Mais pourquoi diable s'intéresser à un logiciel du passé ? C'est que l'interface graphique, si elle est bien adaptée à des manipulations simples, ne l'est pas du tout dès qu'il s'agit d'effectuer des actions complexes ou répétitives.

Il convient d'insister très lourdement sur ce dernier point. Souvenons-nous que le mot « informatique » est né de la fusion des mots « information » et « automatique ». Or ce dernier aspect est encore largement sous-exploité et il est fréquent de voir des utilisateurs d'ordinateur effectuer des actions répétitives sans que cela les dérange. La faute en revient principalement à l'interface graphique, qui n'est généralement pas adaptée à l'automatisation des tâches répétitives.

L'interpréteur de commandes, en revanche, est construit autour de concepts permettant de simplifier au maximum ces tâches répétitives. Il dispose en effet d'un langage de programmation comprenant variables, tests, boucles, etc., permettant d'enchaîner automatiquement des commandes complexes.

Il peut être utilisé :

- soit de manière interactive, l'utilisateur tapant directement les commandes à exécuter, voire des programmes simples ;
- soit en lisant un fichier contenant les commandes à exécuter, ce qui permet de conserver des enchaînements de commandes pour les réutiliser et de concevoir des programmes complexes.

C'est pour cela que l'interpréteur de commandes est toujours resté populaire auprès des informaticiens purs et durs, qui ont besoin d'un outil de travail efficace, alors même que l'interface graphique le remplaçait vis-à-vis du grand public. Les

années 1990 ont sonné le glas de l'interpréteur de commandes, détrôné par des interfaces graphiques toujours plus intuitives et ergonomiques.

Et puis, bizarrement, certains des plus farouches partisans du tout graphique ont fait marche arrière. Par exemple, Microsoft a sorti en 2006 un interpréteur de commandes nommé *Windows PowerShell*. De diffusion assez confidentielle à l'origine, celui-ci est disponible depuis mars 2009 via le système de mises à jour de Microsoft pour Windows XP et Windows Vista et est distribué d'origine avec Windows 7.

Sous UNIX, en tout cas, l'interpréteur de commandes n'a jamais été en voie de disparition. En pratique, il en existe même plusieurs. Partageant des ancêtres communs, leur syntaxe est relativement proche. Cette partie s'attachera à décrire autant que possible les caractéristiques et syntaxes communes à l'ensemble des interpréteurs de commandes, en indiquant leurs différences lorsque cela sera nécessaire.

Nous étudierons également un certain nombre de commandes, parmi les plus utilisées.

> http://fr.wikipedia.org/wiki/Shell_(informatique)
> http://fr.wikipedia.org/wiki/Shell_Unix
> http://fr.wikipedia.org/wiki/Windows_PowerShell
> http://fr.wikipedia.org/wiki/Commandes_Unix

7.1 Historique des interpréteurs de commandes

Historiquement, le premier interpréteur de commandes pour UNIX fut en 1971 le *Thompson shell*, écrit par Ken Thompson, l'un des créateurs d'UNIX.

Celui-ci n'est plus utilisé aujourd'hui, contrairement à ses six descendants, dont l'arbre généalogique est représenté dans la figure 7.1 page ci-contre.

Les interpréteurs de commandes sont souvent désignés par leur diminutif. Ce sont ces diminutifs qui sont indiqués dans la figure 7.1 page suivante. Le tableau 7.1 page ci-contre détaille, pour chaque interpréteur de commandes, son diminutif ainsi que son nom complet et son créateur.

N'ayant pas encore de bonnes connaissances du fonctionnement des interpréteurs de commandes, nous n'allons pas pouvoir rentrer dans les détails des différences entre ces six logiciels mais nous allons en faire un tour d'horizon rapide.

Le premier successeur du Thompson shell est le Bourne shell (`sh`), créé par Steve Bourne en 1977. Il dispose d'un langage de programmation très complet et toujours utilisé de nos jours. En revanche, son utilisation en mode interactif est

FIGURE 7.1 – *Arbre généalogique des interpréteurs de commandes.*

Diminutif	Nom	Créateur
`sh`	Bourne shell	Steve Bourne
`csh`	C shell	Bill Joy
`tcsh`	TENEX C shell	Ken Greer
`ksh`	Korn shell	David Korn
`bash`	Bourne-again shell	Brian Fox
`zsh`	Z shell	Paul Falstad

TABLE 7.1 – *Diminutifs, noms et créateurs des interpréteurs de commandes.*

particulièrement fastidieuse, puisqu'aucune aide ne permet de simplifier la saisie des commandes.

Autre descendant direct du Thompson shell, le C shell (`csh`), créé par Bill Joy (également auteur de `vi`) en 1978, dispose lui aussi d'un langage de programmation très complet, dont la syntaxe est inspirée de celle du langage de programmation C et incompatible avec celle du Bourne shell. Il a été peu utilisé pour faire de la programmation et est quasiment abandonné pour cela aujourd'hui. En revanche, il dispose d'aides permettant de simplifier la saisie des commandes, aides qui ont été reprises et améliorées par ses successeurs.

Le TENEX C shell (`tcsh`), créé par Ken Greer en 1981, est dérivé du C shell et apporte de nombreuses astuces rendant son utilisation interactive particulièrement agréable.

Extension du Bourne shell, le Korn shell (`ksh`), créé par David Korn en 1982, a

amélioré les possibilités de programmation du Bourne shell. Il a également repris quelques caractéristiques du C shell.

Le Bourne-again shell (`bash`), créé par Brian Fox en 1987, se veut un Bourne shell moderne et est dérivé du Korn shell et du C shell. Il a par ailleurs repris les améliorations apportées par le TENEX C shell a son mode interactif. Il s'agit de l'interpréteur de commandes standard sous Linux et c'est également l'interpréteur de commandes utilisé par défaut à l'ENSTA ParisTech.

Enfin, le Z shell (`zsh`), créé par Paul Falstad en 1990, reprend des caractéristiques du TENEX C shell, du Korn shell et du Bourne-again shell. Sa spécificité principale est de comprendre les syntaxes de programmation du Bourne shell et du C shell.

Aujourd'hui, l'interpréteur de commandes le plus utilisé pour la programmation reste le Bourne shell. En utilisation interactive, on retrouve les TENEX C shell, Bourne-again shell et Z shell.

Le lecteur curieux pourra se reporter aux ouvrages suivants de la bibliographie pour approfondir ses connaissances des différents interpréteurs de commandes :

- *UNIX Shells by Example* [21]
- *The UNIX C Shell Field Guide* [11]
- *Using csh & tcsh* [16]
- *Learning the Korn Shell* [26]
- *Learning the bash Shell* [20]
- *bash Cookbook* [10]
- *bash Quick Reference* [24]

http://en.wikipedia.org/wiki/Thompson_shell
http://en.wikipedia.org/wiki/Bourne_shell
http://en.wikipedia.org/wiki/C_shell
http://www.tcsh.org/Welcome
http://en.wikipedia.org/wiki/Tcsh
http://www.kornshell.com/
http://en.wikipedia.org/wiki/Korn_shell
http://www.gnu.org/software/bash/
http://fr.wikipedia.org/wiki/Bourne-Again_shell
http://zsh.sourceforge.net/
http://fr.wikipedia.org/wiki/Zsh

7.2 Concepts fondamentaux

7.2.1 L'émulateur de terminal

Comme nous l'avons déjà vu dans l'introduction de ce document, le lancement d'un interpréteur de commandes ouvre une fenêtre quasiment vide, hormis sa première ligne.

En réalité, nous avons affaire à deux logiciels :

- l'interpréteur de commandes lui-même, qui est la partie la plus interne de la fenêtre ;
- un *émulateur de terminal*, qui permet d'utiliser l'interpréteur de commandes dans un environnement graphique.

En effet, l'interpréteur de commandes est conçu pour être utilisé au moyen d'un périphérique des années 1970 appelé *terminal*. Un terminal était simplement composé d'un clavier et d'un écran n'affichant que du texte, le tout relié à un système informatique. Les logiciels fonctionnant dans un terminal lisaient du texte, entré au moyen du clavier et affichaient en résultat du texte à l'écran. Souvenez-vous : sous UNIX, on aime bien le texte.

Mais pourquoi utilise-t-on encore au XXIe siècle des logiciels conçus pour des environnements dépassés ? L'interpréteur de commandes n'est en effet pas le seul logiciel à s'attendre à fonctionner dans un terminal, il en existe des centaines d'autres et ils sont d'ailleurs largement majoritaires sous UNIX.

Ceci a l'avantage de la simplicité. Beaucoup de logiciels usuels effectuent des fonctions simples (copier un fichier, déplacer un fichier, effacer un fichier, etc.) qui ne nécessitent en rien l'utilisation d'une interface graphique.

Il est donc commode de conserver un mode de fonctionnement simple pour ces logiciels, à savoir lire et écrire du texte et de déporter la gestion graphique dans l'émulateur de terminal, afin que tous ces logiciels puissent être utilisés par ce biais, plutôt que de tous les modifier pour fonctionner nativement en environnement graphique.

Par ailleurs, ceci permet d'utiliser les fonctions de redirection, fondamentales sous UNIX, qui seront abordées au paragraphe 9.4 page 193.

> http://fr.wikipedia.org/wiki/Terminal_informatique
> http://fr.wikipedia.org/wiki/Émulateur_de_terminal

7.2.2 L'invite

Revenons à l'interpréteur de commandes. À son ouverture, celui-ci affiche une *invite* (*prompt* en anglais) suivie du curseur indiquant l'emplacement du prochain caractère qui sera tapé au clavier.

L'invite est un repère visuel indiquant que l'interpréteur de commandes attend que l'utilisateur saisisse une commande.

L'invite par défaut des Bourne shell et Korn shell est `$`, celle du Bourne-again shell est `bash-x.y$` (où `x.y` est le numéro de version du Bourne-again shell utilisé), celle des C shell, TENEX C shell et Z shell est `%`.

Le contenu de l'invite est configurable et beaucoup d'administrateurs de systèmes informatiques en profitent pour y faire figurer quelques informations utiles.

En particulier, à l'ENSTA ParisTech, l'invite est de la forme (les parties en gras variant d'une personne à une autre) :

```
identifiant@ordinateur:/home/promo/identifiant>
```

On reconnaît :

1. l'identifiant de l'utilisateur ;
2. le caractère `@` en tant que séparateur visuel ;
3. le nom de l'ordinateur sur lequel on est connecté ;
4. le caractère `:` en tant que séparateur visuel ;
5. le chemin d'accès absolu du répertoire courant (nous reviendrons respectivement sur ces concepts aux paragraphes 8.3.6 page 131 et 8.3.4 page 125), contenant la promotion de l'utilisateur et son identifiant ;
6. le caractère `>` en tant que séparateur visuel et dernier composant de l'invite.

Afin de simplifier les exemples qui seront donnés par la suite, nous utiliserons simplement `$` comme invite, sauf dans certains cas particuliers.

http://en.wikipedia.org/wiki/Command-line_interface#Command_prompt

7.2.3 Passé, présent et futur

Tapons une commande simple dans l'interpréteur de commandes (ce que l'utilisateur tape au clavier est figuré en gras dans les exemples) :

```
$ date Enter
Wed Jul 29 12:58:31 CEST 2009
$ ■
```

Puis une autre (la commande `cal` affiche le calendrier du mois) :

```
$ cal Enter
     July 2009
 S  M Tu  W Th  F  S
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31

$ ■
```

On aboutit à une fenêtre de terminal comme celle de la figure 7.2 page suivante, qu'on peut décomposer comme indiqué en figure 7.3 page 95.

Tout ce qui se trouve au-dessus de la dernière invite, celle suivie du curseur, montre le passé. Il s'agit de la trace des commandes saisies précédemment et de leurs résultats. Il n'est pas rare de voir des utilisateurs débutants agiter frénétiquement leur souris en cherchant à modifier quelque chose dans cette zone. C'est bien évidemment impossible, le passé est immuable, en histoire comme en informatique.

La ligne contenant la dernière invite, suivie du curseur, représente le présent. C'est la seule zone sur laquelle on peut agir.

La zone située dessous représente le futur. Elle est bien évidemment vide. Comme la fenêtre de l'émulateur de terminal a une taille finie et qu'il est préférable, afin de pouvoir s'y référer, de montrer un passé certain plutôt qu'un hypothétique futur, cette zone a vocation à diminuer jusqu'à disparaître, laissant l'ensemble de la fenêtre au passé, hormis la dernière ligne dédiée au présent.

```
$ date
Wed Jul 29 12:58:31 CEST 2009
$ cal
     July 2009
 S  M Tu  W Th  F  S
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31

$ ■
```

FIGURE 7.2 – *Quelques commandes.*

7.2.4 Notation des combinaisons de touches

Bien qu'utilisant les mêmes touches du clavier, les combinaisons de touches ne se représentent pas de la même façon en fonction des logiciels dans lesquels on les utilise.

Ainsi, on notera `^A` l'appui sur les touches Ctrl (représentée donc par ^) puis A dans l'interpréteur de commandes.

C'est ce qu'on notait `C-a` avec Emacs.

7.2.5 Sensibilité à la casse

Dans l'interpréteur de commandes, la sensibilité à la *casse*, c'est-à-dire la différence entre majuscules et minuscules, est systématique. On utilise principalement des minuscules, plus faciles à lire et à saisir :

passé

```
$ date
Wed Jul 29 12:58:31 CEST 2009
$ cal
     July 2009
 S  M Tu  W Th  F  S
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

présent

```
$ ■
```

futur

FIGURE 7.3 – *Quelques commandes (décomposition temporelle).*

```
$ date Enter
Sun Aug  9 12:57:00 CEST 2009
$ DATE Enter
bash: DATE: command not found
$ ■
```

http://fr.wikipedia.org/wiki/Sensibilité_à_la_casse

7.2.6 Édition de la ligne de commande et historique

Nous n'avons vu pour le moment que quelques commandes très simples. Nous allons voir par la suite que d'autres peuvent être assez longues à taper. L'erreur étant humaine, il arrive de se tromper et il est agaçant de devoir effacer une partie de la ligne qu'on est en train de taper parce qu'on vient de se rendre compte qu'on a commis une erreur en début de ligne.

Heureusement, les interpréteurs de commandes modernes (TENEX C shell, Korn shell, Bourne-again shell et Z shell) permettent d'éditer la ligne de commande qu'on est en train de saisir. Il est possible de se déplacer au moyens des flèches gauche et droite afin d'aller modifier quelque chose sans devoir effacer toute la fin de la ligne. Cela semble assez évident pour qui a utilisé un éditeur de texte mais cette possibilité n'existe pas dans les interpréteurs de commandes les plus anciens (Bourne shell et C shell), ce qui rend leur utilisation interactive assez frustrante.

Mieux encore, il est possible d'utiliser certaines combinaisons de touches identiques à celles d'Emacs, dans la mesure où elles ont un sens lorsqu'on travaille sur une ligne unique. Parmi celles abordées au chapitre 5 page 57, `^A`, `^E`, `^D` et `^K` (équivalentes à `C-a`, `C-e`, `C-d` et `C-k`) sont utilisables.

Plus fort, les touches fléchées haut et bas permettent de naviguer dans l'*historique* des commandes passées. Un appui sur [↑] rappelle la commande précédente, en positionnant le curseur en fin de ligne. Il est alors possible de l'exécuter à nouveau en appuyant simplement sur [Enter] ou d'utiliser les flèches et les commandes Emacs indiquées ci-dessus afin de la modifier avant de l'exécuter. Il n'est d'ailleurs pas nécessaire que le curseur soit en fin de ligne pour exécuter une commande en appuyant sur [Enter], cela fonctionne n'importe où. Si l'on souhaite rappeler une commande passée pour l'exécuter à l'identique ou la modifier, il suffit de remonter avec [↑] jusqu'au bon endroit. Si l'on est remonté trop haut, on peut utiliser [↓] pour redescendre.

Ceci est particulièrement utile et doit être utilisé dès que possible, pour rappeler une commande récente et l'exécuter à nouveau, telle quelle ou légèrement modifiée.

Il existe d'autres façons d'utiliser l'historique des commandes, qui seront étudiées au paragraphe 9.1 page 173.

7.2.7 Forme des exemples dans la suite du document

À partir de maintenant, afin d'alléger les exemples, ceux-ci n'indiqueront plus la commande à taper en gras mais de façon normale. Par ailleurs, l'appui sur la touche [Enter] ne sera plus indiqué, de même que l'invite et le curseur suivant l'exécution d'une commande.

Ainsi, ce qu'on écrivait :

```
$ date [Enter]
Thu Jul 30 10:08:18 CEST 2009
$ ■
```

devient simplement :

```
$ date
Thu Jul 30 10:08:18 CEST 2009
```

7.3 La commande ls, la syntaxe des commandes

La commande `ls` est probablement la plus utilisée. On reconnaît d'ailleurs les commandes fréquemment utilisées à leur nom composé de deux caractères, afin de pouvoir les saisir le plus vite possible.

La commande `ls` (pour *list*) permet d'afficher la liste de ses fichiers :

```
$ ls
toto.txt
```

On y reconnaît le fichier `toto.txt` créé lors de l'étude des éditeurs de texte.

Il est difficile de s'en rendre compte avec un seul fichier mais, lorsqu'elle en a plusieurs à afficher, la commande `ls` les trie dans l'ordre lexicographique.

http://en.wikipedia.org/wiki/Ls

7.3.1 Les options

Il est souvent nécessaire d'afficher, pour chaque fichier, des informations supplémentaires. Ceci se fait au moyen de l'option `-l` de la commande `ls` :

```
$ ls -l
total 4
-rw-r--r-- 1 babafou prof 334 Jul 30 17:02 toto.txt
```

La possibilité de modifier le comportement d'une commande grâce à une option n'est pas spécifique à `ls`. Il s'agit d'un mécanisme général et, en pratique, quasiment toutes les commandes acceptent une ou plusieurs options.

L'option est composée d'un tiret suivi généralement d'un caractère. Il existe des options composées d'un tiret suivi d'un mot mais cela est plus rare. L'option est

obligatoirement séparée de la commande qui la précède par une espace (on peut en mettre plusieurs mais c'est inutile).

Revenons à l'option `-l` de la commande `ls`. Celle-ci provoque l'affichage des fichiers au format long (`-l` pour *long*). La première ligne du résultat, sans grand intérêt, indique le nombre de blocs occupés sur disque par l'ensemble des fichiers. La suite est plus intéressante. On y retrouve nos fichiers, un par ligne, dont le nom est indiqué en fin de ligne. Auparavant, des informations supplémentaires sont alignées en colonnes. C'est une forme d'affichage très utilisée sous UNIX et que nous retrouverons par la suite.

Ces colonnes indiquent respectivement :

1. le type et les droits d'accès du fichier ;
2. le nombre de liens sur le fichier ;
3. le propriétaire du fichier ;
4. le groupe du fichier ;
5. la taille du fichiers en octets ;
6. la date de dernière modification du fichier (les trois colonnes n'en forment qu'une seule) ;
7. enfin, le nom du fichier.

La première colonne doit être lue comme un caractère, indiquant le type du fichier, suivi de trois groupes de trois caractères, indiquant les droits d'accès du fichier.

Le type du fichier peut être l'un de ceux indiqués dans le tableau 7.2.

`-`	fichier ordinaire
`b`	fichier spécial de type bloc
`c`	fichier spécial de type caractère
`d`	répertoire (*directory* en anglais)
`l`	lien symbolique
`p`	tuyau nommé (*pipe* en anglais)
`s`	prise réseau (*socket* en anglais)

TABLE 7.2 – *Types de fichiers.*

En pratique, les fichiers qu'on rencontre le plus souvent sont les fichiers ordinaires. Ceux-ci sont principalement des fichiers de données, comme notre fichier `toto.txt`, mais il peut également s'agir de fichiers exécutables (c'est-à-dire

de programmes). On rencontre également beaucoup de répertoires (voir le paragraphe 8.3 page 117) ainsi que quelques liens symboliques (voir le paragraphe 8.1.4 page 111), leur usage étant assez limité. Nous discuterons des fichiers spéciaux au paragraphe 9.4.1 page 193. Nous n'aborderons pas les autres types de fichiers.

Les droits d'accès seront étudiés au paragraphe 8.5 page 139.

La signification de la deuxième colonne sera abordée au paragraphe 8.1.4 page 111.

La troisième colonne indique l'utilisateur qui est propriétaire du fichier et la quatrième le groupe du fichier. Ces notions sont à rapprocher des droits d'accès et seront abordées en même temps.

La cinquième colonne indique, pour les fichiers ordinaires, leur taille en octets. Nous verrons le moment donné la signification de cette colonne pour les répertoires et les liens symboliques.

La sixième colonne indique la date de dernière modification du fichier. Il s'agit du mois, du jour et, si le fichier a été modifié pour la dernière fois il y a moins de six mois, de l'heure et de la minute, sinon de l'année.

La commande `ls` est très souvent utilisée avec l'option `-l`. C'est pourquoi beaucoup de sites créent un *alias* pour `ls -l`. Un alias est un diminutif pour une commande plus longue fréquemment utilisée. À l'ENSTA ParisTech, l'alias pour `ls -l` est `ll`.

Une autre option utile de `ls` est l'option `-a` :

```
$ ls -a
.               .bash_history  .bashrc     .mailrc
..              .bash_logout   .cshrc      toto.txt
.Xdefaults      .bash_profile  .elm
```

L'option `-a` (pour *all*) permet d'afficher tous les fichiers, y compris ceux dont le nom commence par un point, ce que `ls` ne fait pas par défaut. Il ne s'agit pas de fichiers cachés comme il peut en exister dans d'autres systèmes d'exploitation, tout est transparent sous UNIX. En revanche, par convention, `ls` n'affiche simplement pas ces fichiers.

Les fichiers dont le nom commence par un point sont traditionnellement des fichiers de configuration pour divers logiciels (leur nom se termine généralement par `rc` [1]) ou des fichiers permettant aux logiciels de sauvegarder des informations lors de leur fermeture. Ces fichiers doivent exister mais il n'est pas nécessaire de les

1. Le suffixe `rc` vient de RUNCOM, un lointain ancêtre des interpréteurs de commandes.

avoir toujours sous les yeux. C'est pour cela que `ls` ne les affiche pas si l'on n'utilise pas l'option `-a`.

Il est possible de combiner les options `-a` et `-l` :

```
$ ls -a -l
total 108
drwxr-xr-x 6 babafou prof  4096 Jul 30 17:02 .
drwxr-xr-x 3 root    root  4096 Jun 26 10:16 ..
-rw-r--r-- 1 babafou prof   576 Sep  3  2007 .Xdefaults
-rw------- 1 babafou prof   167 Jul 10 12:39 .bash_history
-rw-r--r-- 1 babafou prof    24 Sep  3  2007 .bash_logout
-rw-r--r-- 1 babafou prof   191 Feb 18 14:22 .bash_profile
-rw-r--r-- 1 babafou prof   176 Sep  3  2007 .bashrc
-rw-r--r-- 1 babafou prof    48 Sep  3  2007 .cshrc
drwxr-xr-x 2 babafou prof  4096 Sep  3  2007 .elm
-rw-r--r-- 1 babafou prof    26 Sep  3  2007 .mailrc
-rw-r--r-- 1 babafou prof   334 Jul 30 17:02 toto.txt
```

On a alors un affichage long de tous les fichiers.

Ceci est valable pour la plupart des commandes. Dans la mesure où elles ne sont pas antagonistes, il est possible de combiner les options afin d'en cumuler les effets. L'ordre des options est sans importance, donc `ls -a -l` et `ls -l -a` sont équivalentes et on peut les contracter en `ls -al` et `ls -la`. Tout ceci aboutit au même résultat.

Il existe de nombreuses autres options pour `ls`, que nous n'aborderons pas.

> http://en.wikipedia.org/wiki/Command-line_interface#Command-line_option

7.3.2 Les arguments

Si l'on souhaite obtenir l'affichage long des informations concernant un seul fichier, sans avoir une fenêtre polluée par les informations concernant d'autres fichiers, il est possible de l'indiquer en *argument* de la commande `ls` :

```
$ ls -l toto.txt
-rw-r--r-- 1 babafou prof   334 Jul 30 17:02 toto.txt
```

Petit point de vocabulaire, `-l` est une option (nous venons de voir qu'il peut y en avoir plusieurs) et `toto.txt` un argument (il peut également y en avoir plusieurs).

Commande, option(s) et argument(s) doivent être séparés les uns des autres par des espaces.

Autre exemple, si l'on souhaite afficher les informations concernant `toto.txt` et `.bashrc`, on peut taper :

```
$ ls -l toto.txt .bashrc
-rw-r--r-- 1 babafou prof 176 Sep  3  2007 .bashrc
-rw-r--r-- 1 babafou prof 334 Jul 30 17:02 toto.txt
```

Notez qu'il n'est pas nécessaire d'indiquer l'option `-a` dans la mesure où le nom du fichier commençant par un point est donné en argument. Par ailleurs, l'ordre des arguments est sans effet sur l'affichage de `ls`, qui est toujours trié dans l'ordre lexicographique.

En résumé, les options modifient le comportement de la commande, les arguments lui indiquent ce sur quoi elle doit travailler.

> http://en.wikipedia.org/wiki/Command-line_interface#Arguments

7.4 Les commandes d'affichage : `cat`, `more` et `less`

Il est souvent utile d'afficher un fichier texte à l'écran, afin de lire son contenu sans pour autant vouloir le modifier. Utiliser un éditeur de texte pour cela n'est donc pas la bonne solution.

7.4.1 La commande `cat`

La commande `cat` (pour *catenate*) permet justement d'afficher le contenu du ou des fichiers texte passés en arguments :

```
$ cat tata.txt
Ceci est le contenu du fichier tata.txt.
Il est bien court mais tata.txt est un joli fichier.
```

Si plusieurs noms de fichiers sont passés en arguments, leur contenu est affiché dans le même ordre, sans séparation particulière entre les fichiers :

```
$ cat tata.txt titi.txt tutu.txt
Ceci est le contenu du fichier tata.txt.
Il est bien court mais tata.txt est un joli fichier.
Ceci est le contenu du fichier titi.txt.
Il est bien court mais titi.txt est un joli fichier.
Ceci est le contenu du fichier tutu.txt.
Il est bien court mais tutu.txt est un joli fichier.
```

Attention, `cat` ne permet d'afficher que des fichiers texte. Il n'est pas question de l'utiliser pour visualiser des images. Lorsqu'elle lit le contenu d'un fichier, la commande `cat` l'affiche tel quel. En pratique, c'est l'émulateur de terminal qui traduit la valeur de chaque octet envoyé par `cat` pour afficher le glyphe correspondant à chaque octet ou à chaque groupe d'octets, conformément au codage utilisé (voir le paragraphe 4.1 page 45).

Il convient donc d'être attentif au codage des fichiers car afficher un fichier codé en ISO 8859-15 dans un émulateur de terminal configuré en UTF-8 (ou l'inverse) risque d'afficher des choses étranges à la place des caractères dont le code ne fait pas partie de l'ASCII.

http://fr.wikipedia.org/wiki/Cat_(Unix)

7.4.2 La commande `more`

La commande `cat` a un petit défaut. Elle affiche, elle affiche, elle affiche et elle ne s'arrête que quand elle a fini. Lorsqu'on affiche un fichier relativement long, tout son contenu défile et l'on n'en voit finalement que la dernière page. Il est bien possible de remonter grâce à l'ascenseur de l'émulateur de terminal mais celui-ci a une capacité limitée et le début des fichiers vraiment longs a généralement déjà disparu.

Pour pouvoir lire des fichiers texte un peu longs dans de bonnes conditions, il est préférable d'utiliser la commande `more` :

```
$ more toto.txt
```

La commande `more` prend alors le contrôle de la totalité de la fenêtre de l'émulateur de terminal afin d'y afficher le contenu du fichier `toto.txt`.

Il est possible d'y naviguer au moyen des touches :

- Enter, pour descendre d'une ligne ;
- ␣, pour descendre d'une page (la hauteur d'une page est la hauteur de la fenêtre) ;
- b, pour remonter d'une page ;
- ↑, ↓, Page↑, Page↓ fonctionnent également comme attendu ;
- enfin, la touche q permet de quitter `more`.

La commande `more` tire son nom de ce qu'elle affiche en bas de page afin d'indiquer la position actuelle dans le fichier :

```
--More--(n%)
```

http://fr.wikipedia.org/wiki/More_(Unix)

7.4.3 La commande `less`

La commande `less` — nommée ainsi par jeu de mots avec `more` — s'utilise de la même façon. Elle a néanmoins un avantage sur `more`. À force de faire défiler le contenu du fichier, la commande `more`, lorsqu'elle affiche sa dernière ligne, se termine automatiquement et l'on revient à l'invite de l'interpréteur de commandes. Ceci peut être gênant si l'on est allé trop vite et qu'on souhaite revenir en arrière. Il faut alors relancer la commande. En revanche, `less` n'a pas ce défaut. Arrivée en fin de fichier, cette commande ne se termine pas et il est possible soit de remonter soit de la quitter explicitement.

Il est donc souvent plus pratique d'utiliser `less` plutôt que `more`.

À noter que `more` est une commande standard, fournie avec tout système UNIX, ce qui n'est pas le cas de `less`. Cette dernière est néanmoins très répandue et, en particulier, est fournie avec tous les systèmes UNIX libres (BSD, Linux).

http://www.gnu.org/software/less/
http://www.greenwoodsoftware.com/less/
http://en.wikipedia.org/wiki/Less_(Unix)

7.5 La commande `man`

Nous n'avons abordé au paragraphe 7.3 page 97 que deux des nombreuses options de la commande `ls`. Le lecteur curieux qui souhaite en apprendre davantage

et l'utilisateur qui a oublié le fonctionnement exact de telle commande ou de telle option peuvent se reporter au manuel en ligne disponible sur tout système UNIX grâce à la commande `man` (pour *manual*). Elle s'utilise en lui passant en argument le nom de la commande dont on souhaite consulter le manuel :

```
$ man ls
```

Ceci affiche le manuel de la commande `ls` au moyen de `more` ou `less`.

Toutes les commandes utilisables dans l'interpréteur de commandes ont systématiquement un manuel en ligne, y compris `man` :

```
$ man man
```

http://fr.wikipedia.org/wiki/Man_(Unix)

7.6 La commande `clear`

La commande `clear` permet d'effacer l'écran de l'interpréteur de commandes :

```
$ date
Mon Mar  1 17:56:08 CET 2010
$ ls -l
-rw-r--r-- 1 babafou prof   334 Jul 30 17:02 toto.txt
$ clear
```

Après avoir appuyé sur la touche Enter, on obtient un écran vide ne contenant plus, tout en haut, que l'invite et le curseur :

```
$
```

L'historique (voir les paragraphes 7.2.6 page 95 et 9.1 page 173), en revanche, n'est pas effacé donc il est toujours possible de naviguer dans les commandes précédentes au moyen des touches fléchées.

http://fr.wikipedia.org/wiki/Clear_(Unix)

7.7 La commande exit

La commande exit permet de quitter l'interpréteur de commandes :

```
$ exit
```

http://fr.wikipedia.org/wiki/Exit_(Unix)

8

Gestion, organisation et traitement des fichiers

LES FICHIERS ont une très grande importance sous UNIX. Ils forment la base du stockage de l'information et de nombreux concepts manipulés par le système d'exploitation peuvent être assimiliés à des fichiers. Nous étudierons les commandes fondamentales permettant de gérer des fichiers, puis de les organiser. Nous étudierons également plusieurs mécanismes permettant de traiter des fichiers efficacement ainsi que celui permettant d'en contrôler l'accès. Enfin, nous étudierons comment conserver des données n'ayant plus d'utilité immédiate, en occupant le moins de volume possible.

8.1 Gestion des fichiers

Nous allons aborder ici les commandes usuelles de gestion des fichiers, qui en permettent la copie, le renommage, la suppression ainsi que la création de liens.

8.1.1 Copie de fichiers : la commande `cp`

La commande `cp` (pour *copy*) permet de créer une copie d'un fichier :

```
$ cp toto.txt tata.txt
```

La commande `cp` prend deux noms de fichiers en arguments :

1. le fichier d'origine, qui doit exister préalablement, ici `toto.txt` ;
2. le fichier que l'on souhaite créer comme une copie du premier, ici `tata.txt`.

Afin de bien comprendre le fonctionnement de `cp`, il est utile de contrôler la liste des fichiers avant et après l'opération, comme ceci :

```
$ ll
total 4
-rw-r--r-- 1 babafou prof 334 Jul 30 17:02 toto.txt
$ cp toto.txt tata.txt
$ ll
total 8
-rw-r--r-- 1 babafou prof 334 Jul 31 15:11 tata.txt
-rw-r--r-- 1 babafou prof 334 Jul 30 17:02 toto.txt
```

Remarquez que le fichier `tata.txt` est daté du moment où l'on a effectué la copie. Un fichier copié est donc toujours plus récent que le fichier d'origine.

La commande `cp` n'affiche rien si elle se déroule correctement. C'est tout à fait normal. La plupart des commandes que nous allons étudier sont destinées à réaliser une fonction précise, pas à afficher quelque chose.

Si un fichier `tata.txt` existait avant d'effectuer la copie, il est écrasé sans demander confirmation. Si l'on souhaite être prudent et éviter d'écraser un fichier par mégarde, on peut utiliser l'option `-i` (pour *interactive*) :

```
$ cp -i toto.txt tata.txt
cp: overwrite 'tata.txt'? ■
```

On peut alors répondre `y` si l'on souhaite écraser le fichier `tata.txt` ou `n` pour annuler l'opération.

> Dans l'environnement informatique de l'ENSTA ParisTech, la commande `cp` utilisée est en réalité un alias sur `cp -i`. Elle demande donc toujours confirmation en cas d'écrasement.

La copie peut échouer, en particulier si fichier d'origine n'existe pas. En cas d'erreur, un message approprié est affiché :

```
$ cp tutu.txt tata.txt
cp: cannot stat 'tutu.txt': No such file or directory
```

http://en.wikipedia.org/wiki/Cp_(Unix)

8.1.2 Renommage de fichiers : la commande mv

La commande mv (pour *move*) permet de renommer un fichier :

```
$ mv toto.txt tata.txt
```

La commande mv prend deux noms de fichiers en arguments :

1. le fichier d'origine, qui doit exister préalablement, ici toto.txt ;
2. le nouveau nom que l'on souhaite donner à ce fichier, ici tata.txt.

À la différence de cp, on a donc toujours un fichier unique avant et après l'opération. Il a seulement changé de nom entre temps :

```
$ ll
total 4
-rw-r--r-- 1 babafou prof 334 Jul 30 17:02 toto.txt
$ mv toto.txt tata.txt
$ ll
total 4
-rw-r--r-- 1 babafou prof 334 Jul 30 17:02 tata.txt
```

La commande mv n'affiche rien si elle se déroule correctement.

Si un fichier tata.txt existait avant d'effectuer le renommage, il est écrasé sans demander confirmation. Si l'on souhaite être prudent et éviter d'écraser un fichier par mégarde, on peut utiliser l'option -i (comme pour cp) :

```
$ mv -i toto.txt tata.txt
mv: overwrite 'tata.txt'? ■
```

On peut alors répondre y si l'on souhaite écraser le fichier tata.txt ou n pour annuler l'opération.

Dans l'environnement informatique de l'ENSTA ParisTech, la commande mv utilisée est en réalité un alias sur mv -i. Elle demande donc toujours confirmation en cas d'écrasement.

Le renommage peut échouer, en particulier si le fichier d'origine n'existe pas. En cas d'erreur, un message approprié est affiché :

```
$ mv tutu.txt tata.txt
mv: cannot stat 'tutu.txt': No such file or directory
```

http://en.wikipedia.org/wiki/Mv

8.1.3 Suppression de fichiers : la commande `rm`

La commande `rm` (pour *remove*) permet de supprimer un ou plusieurs fichiers, dont les noms sont passés en arguments :

```
$ rm toto.txt
$ rm toto1.txt toto2.txt toto3.txt
```

La commande `rm` n'affiche rien si elle se déroule correctement.

Il n'est demandé aucune confirmation avant de supprimer un fichier. Si l'on souhaite être prudent et éviter de supprimer un fichier par mégarde, on peut utiliser l'option `-i` :

```
$ rm -i toto1.txt toto2.txt toto3.txt
rm: remove regular file 'toto1.txt'? y
rm: remove regular file 'toto2.txt'? n
rm: remove regular file 'toto3.txt'? ■
```

Si l'un des fichiers à supprimer n'existe pas, `rm` affiche un message d'erreur mais supprime correctement les fichiers qui existent :

```
$ rm toto.txt tutu.txt tata.txt
rm: cannot remove 'tutu.txt': No such file or directory
```

Lorsqu'un fichier a été supprimé, il n'y a aucun moyen simple pour le récupérér. D'où l'importance d'effectuer des sauvegardes régulières de ses données.

http://fr.wikipedia.org/wiki/Rm_(Unix)

8.1.4 Création de liens : la commande `ln`

La commande `ln` (pour *link*) permet de créer un *lien* (*link* en anglais) supplémentaire sur un fichier.

Pour comprendre son fonctionnement, il faut se pencher sur la façon dont les fichiers sont stockés sur disque.

Un fichier est composé de trois éléments :

1. un ou plusieurs noms (nous allons voir qu'un même fichier peut avoir plusieurs noms, c'est tout l'intérêt de la commande `ln`) ;
2. un *i-nœud* (*inode* en anglais) ;
3. son contenu proprement dit.

Le nom d'un fichier ainsi que son contenu vous sont familiers. L'i-nœud, en revanche, est un concept nouveau, bien que déjà abordé sans le savoir. L'i-nœud est une structure du système de fichiers permettant de stocker, pour un fichier donné :

- ses droits d'accès ;
- son nombre de liens ;
- son propriétaire ;
- son groupe ;
- sa taille ;
- la date de dernière modification de son contenu ;
- la date de dernière modification de l'i-nœud lui-même ;
- sa date de dernier accès ;
- et enfin un pointeur vers la localisation de son contenu sur le disque.

En pratique, la majorité de ces informations se retouve dans l'affichage de `ls -l`.

Un numéro unique est associé à l'i-nœud, permettant de le localiser sur le disque. Ce numéro est appelé *index* et son initiale à donné son « i » au terme « i-nœud ».

Le nom du fichier fait référence au numéro de son i-nœud. Cette relation est ce qu'on appelle un *lien*. L'i-nœud est lui aussi rattaché à son contenu. Le tout est schématisé dans la figure 8.1 page suivante. Ce chaînage permet de recoller tous les morceaux d'un fichier.

Enfin, on peut faire afficher le numéro d'i-nœud associé à un fichier au moyen de l'option `-i` de la commande `ls` :

FIGURE 8.1 – *Structure d'un fichier.*

```
$ ls -i
68782057 toto.txt
```

Combinée avec l'option `-l`, elle fait apparaître le numéro d'i-nœud en première colonne :

```
$ ls -il
total 4
68782057 -rw-r--r-- 1 babafou prof 334 Jul 30 17:02 toto.txt
```

Cette digression théorique nous ayant permis de préparer le terrain, revenons à la commande `ln`. Celle-ci permet donc de créer un lien supplémentaire sur un fichier existant, c'est-à-dire un nom supplémentaire :

```
$ ls -il
total 4
68782057 -rw-r--r-- 1 babafou prof 334 Jul 30 17:02 toto.txt
$ ln toto.txt tata.txt
$ ls -il
total 8
68782057 -rw-r--r-- 2 babafou prof 334 Jul 30 17:02 tata.txt
68782057 -rw-r--r-- 2 babafou prof 334 Jul 30 17:02 toto.txt
```

La commande `ln` a créé un second nom, `tata.txt`, pour le fichier ordinaire `toto.txt`, comme schématisé dans la figure 8.2 page suivante. Il ne s'agit pas d'une copie car les deux noms correspondent au même numéro d'i-nœud. Celui-ci, ainsi que le contenu du fichier, n'ont donc pas été dupliqués, ce qui fait que la création

d'un lien ne prend quasiment pas de place sur disque. On peut également remarquer que la valeur de la colonne indiquant le nombre de liens a augmenté. Celle-ci indique donc le nombre de noms dont un fichier dispose.

FIGURE 8.2 – *Structure d'un fichier ayant deux noms.*

Supprimer l'un des deux noms ne supprime pas le fichier en tant que tel (i-nœud et contenu), il supprime le lien correspondant. Tant qu'il existe au moins un lien sur un fichier, celui-ci reste accessible.

En revanche, supprimer l'un des deux noms modifie le nombre de liens du fichier :

```
$ ls -il
total 8
68782057 -rw-r--r-- 2 babafou prof 334 Jul 30 17:02 tata.txt
68782057 -rw-r--r-- 2 babafou prof 334 Jul 30 17:02 toto.txt
$ rm tata.txt
$ ls -il
total 4
68782057 -rw-r--r-- 1 babafou prof 334 Jul 30 17:02 toto.txt
```

Comme pour `cp` et `mv`, si un fichier `tata.txt` existait, il est écrasé lors de la création du lien. Là encore, l'option `-i` de `ln` permet de demander confirmation en cas d'écrasement :

```
$ ln -i toto.txt tata.txt
ln: replace 'tata.txt'? ■
```

Il est possible de créer autant de liens d'un fichier qu'on le souhaite.

Pour le moment, pouvoir disposer de plusieurs noms pour le même fichier peut paraître inutile. Nous y reviendrons au paragraphe 8.3.3 page 122.

Toujours est-il qu'il est difficile de distinguer les liens correspondant à un fichier particulier si l'on travaille avec un grand nombre de fichiers. Pour cela, il existe un autre type de lien, le *lien symbolique*.

L'option `-s` de `ln` permet de créer un lien symbolique :

```
$ ll
total 4
-rw-r--r-- 1 babafou prof 334 Jul 30 17:02 toto.txt
$ ln -s toto.txt tata.txt
$ ll
total 4
lrwxrwxrwx 1 babafou prof   8 Aug  3  2009 tata.txt -> toto.txt
-rw-r--r-- 1 babafou prof 334 Jul 30 17:02 toto.txt
```

Remarquez que :

- il est clairement indiqué que `tata.txt` est un lien symbolique vers `toto.txt` ;
- un lien symbolique n'est pas un fichier ordinaire, comme l'indique le caractère `l` en première colonne, il s'agit d'un type de fichier particulier ;
- le nombre de liens sur le fichier `toto.txt` n'a pas changé.

Le lien symbolique est en fait un type de fichier particulier contenant le nom d'un autre fichier, comme schématisé dans la figure 8.3. Le contenu de `tata.txt` est donc `toto.txt`, d'où sa taille de 8 octets et le fait que `ls -l` puisse afficher `tata.txt -> toto.txt`. Lorsqu'on tente de lire le contenu de `tata.txt`, le système d'exploitation remarque qu'il ne s'agit pas d'un fichier ordinaire mais d'un lien symbolique et accède en réalité au contenu de `toto.txt`.

FIGURE 8.3 – *Principe du lien symbolique.*

L'avantage par rapport au lien vu précédemment est qu'il est clairement indiqué dans l'affichage de `ls -l`, sans avoir à comparer les numéros d'i-nœud, que `tata.txt` est un lien symbolique vers `toto.txt`. L'inconvénient est que la relation entre les deux noms n'est pas symétrique. En effet, dans la mesure où le lien symbolique contient le nom du fichier de référence, le premier cessera de fonctionner si le second est supprimé ou renommé.

Néanmoins, le simple fait de pouvoir afficher la relation entre les deux fichiers fait que les liens symboliques sont de loin les plus utilisés.

> http://en.wikipedia.org/wiki/Ln_(Unix)
> http://fr.wikipedia.org/wiki/Inode

8.2 Le complètement

Vous l'aurez certainement remarqué, travailler avec l'interpréteur de commandes nécessite l'utilisation constante de nombreux noms de fichiers. Nous reviendrons au paragraphe 8.6 page 148 sur la bonne façon de nommer ses fichiers mais on peut, sans trop déflorer le sujet, deviner qu'un bon nom de fichier est un nom qui permet d'en identifier le contenu sans ambiguïté. C'est donc un nom relativement long et précis. Or taper de longs noms de fichiers en entier peut vite s'avérer fastidieux pour l'utilisateur.

C'est pourquoi les interpréteurs de commandes modernes (TENEX C shell, Korn shell, Bourne-again shell et Z shell), en plus d'avoir une fonction d'édition de la ligne de commande, qui fait déjà gagner beaucoup de temps, proposent aussi le *complètement* [1] (*completion* en anglais) des noms de fichiers.

L'idée est de taper le minimum de choses et de laisser l'interpréteur de commande compléter automatiquement les noms des fichiers, plutôt que de les taper en entier.

Voyons tout de suite un exemple. Le principe du complètement est de taper le début du nom de fichier et de laisser l'interpréteur de commandes le terminer tout seul en appuyant sur la touche tabulation ⇆ :

```
$ ll to ⇆
```

Dans la mesure où `toto.txt` est le seul fichier commençant par `to`, l'intepréteur de commandes complète le nom du fichier, le fait suivre d'une espace (au cas où l'on souhaite taper d'autres noms de fichiers) et positionne le curseur à la suite :

1. On utilise également l'anglicisme *complétion*, qui est vraiment laid.

```
$ ll toto.txt ■
```

On peut alors saisir un autre nom de fichier ou simplement valider la commande avec [Enter] (l'espace en fin de ligne est alors ignorée, il est inutile de la supprimer).

Dans le cas où plusieurs fichiers commencent de la même façon, l'interpréteur de commandes va compléter ce qu'il peut, puis demander à l'utilisateur de poursuivre manuellement. Commençons par préparer le terrain :

```
$ cp toto.txt toto1.txt
$ cp toto.txt toto2.txt
$ cp toto.txt toto3.txt
```

Nous avons donc maintenant quatre fichiers commençant par `to`.

```
$ ll to [⇆]
```

L'interpréteur de commandes regarde quels sont les fichiers dont le nom commence par `to`. Ils ont tous en commun la propriété de commencer par `toto` donc l'interpréteur de commandes est capable de compléter le nom de fichier jusqu'à ce point :

```
$ ll toto■
```

Remarquez que le curseur est collé à `toto`, sans espace comme nous avions dans le précédent exemple. Ceci est un repère visuel indiquant que le nom du fichier n'est pas complet et que l'interpréteur de commandes a besoin que l'utilisateur tape quelque chose pour l'aider à faire la différence entre les noms des fichiers commençant par `toto`.

Si l'on n'a plus en mémoire les noms des fichiers, on peut les faire afficher en tapant deux fois sur la touche [⇆] :

```
$ ll toto [⇆] [⇆]
```

L'interpréteur de commandes affiche alors les noms des fichiers commençant par `toto` et réaffiche la ligne de commande :

```
$ ll toto
toto.txt   toto1.txt  toto2.txt  toto3.txt
$ ll toto
```

Si le fichier désiré est `toto1.txt`, il suffit de taper `1` puis une nouvelle fois ⇆ pour que l'interpréteur de commandes termine le nom du fichier :

```
$ ll toto1 ⇆
```

Ce qui donne :

```
$ ll toto1.txt
```

Avec des noms de fichiers aussi courts, il est presque aussi rapide de les saisir en entier. Néanmoins, avec des noms de fichiers plus longs, l'utilisation du complètement fait vraiment gagner beaucoup de temps.

Le rappel des commandes précédentes et l'édition de la ligne de commande, abordés au paragraphe 7.2.6 page 95, ainsi que le complètement rendent l'utilisation de l'interpréteur de commandes la plus rapide possible. Il est essentiel d'utiliser ces fonctions dès que possible afin de taper le minimum de choses et donc de travailler efficacement.

http://en.wikipedia.org/wiki/Command-line_completion

8.3 Organisation des fichiers

Lorsqu'on travaille avec de nombreux fichiers, il est utile de pouvoir les organiser, les classer, les regrouper.

L'objet permettant d'organiser les fichiers s'appelle un *répertoire* (*directory* en anglais). On rencontre également les termes *dossier* (*folder* en anglais) et *catalogue* (*catalog* en anglais).

http://fr.wikipedia.org/wiki/Répertoire_(informatique)

8.3.1 Création de répertoires : la commande `mkdir`

La commande `mkdir` (pour *make directories*) permet de créer un répertoire, dont le nom lui est passé en argument :

```
$ ll
total 4
-rw-r--r-- 1 babafou prof 334 Jul 30 17:02 toto.txt
$ mkdir rep
$ ll
total 8
drwxr-xr-x 2 babafou prof 4096 Aug  4 15:56 rep
-rw-r--r-- 1 babafou prof  334 Jul 30 17:02 toto.txt
```

On reconnaît que `rep` est un répertoire au `d` présent en début de ligne de l'affichage de `ls`.

On peut afficher le contenu du répertoire (qui est pour l'instant vide) grâce à `ls` :

```
$ ls rep
```

Si l'on souhaite en revanche obtenir l'affichage long des informations du répertoire lui-même et non de son contenu, il faut utiliser l'option `-d` de `ls` (qui n'a d'intérêt que couplée à l'option `-l`) :

```
$ ll -d rep
drwxr-xr-x 2 babafou prof 4096 Aug  4 15:56 rep
```

Si l'on souhaite créer plusieurs répertoires en même temps, `mkdir` accepte plusieurs arguments :

```
$ mkdir rep1 rep2 rep3
```

http://fr.wikipedia.org/wiki/Mkdir

8.3.2 Chemin d'accès et structure arborescente

Nous n'avons pour le moment qu'un répertoire `rep` vide mais supposons qu'il contienne un fichier `toto`.

Pour désigner ce fichier, il est nécessaire d'indiquer le nom du répertoire qui le contient, à savoir `rep`, immédiatement suivi du caractère `/`, immédiatement suivi du nom du fichier, ce qui donne :

```
rep/toto
```

C'est ce qu'on appelle le *chemin d'accès* (*path* en anglais) au fichier `toto`. Le terme est imagé et décrit bien la façon de le construire : on parcourt le chemin à suivre pour accéder au fichier en indiquant le nom du répertoire contenant le fichier, le séparateur `/` puis le nom du fichier.

Donc, par exemple, pour afficher les informations concernant le fichier `toto`, on peut utiliser la commande :

```
$ ll rep/toto
```

Pour supprimer le fichier :

```
$ rm rep/toto
```

Et ainsi de suite.

À noter que le complètement fonctionne également avec les noms de répertoires. On peut donc faire :

```
$ ll r ⇆
```

Ce qui donnera :

```
$ ll rep/■
```

On peut alors appuyer sur Enter si l'on souhaite simplement afficher le contenu du répertoire (le `/` en fin de ligne est ignoré) ou poursuivre :

```
$ ll rep/t ⇆
```

voire simplement :

```
$ ll rep/ ⇆
```

dans la mesure où le répertoire `rep` ne contient qu'un seul fichier.

On aura au bout du compte :

```
$ ll rep/toto ■
```

Supposons maintenant que le répertoire `rep` contienne deux fichiers, `toto` et `tata`. Leurs chemins d'accès sont donc respectivement `rep/toto` et `rep/tata`.

On peut représenter graphiquement la structure du répertoire `rep` comme indiqué dans la figure 8.4.

FIGURE 8.4 – *Structure du répertoire `rep`.*

Supposons maintenant que l'on crée un répertoire `rep2` dans `rep`. La commande permettant ceci est :

```
$ mkdir rep/rep2
```

Supposons que `rep2` contienne deux fichiers, `titi` et `tutu`. Leurs chemins d'accès sont donc respectivement `rep/rep2/titi` et `rep/rep2/tutu`.

Là encore, on peut représenter graphiquement la structure du répertoire `rep` contenant `rep2` comme indiqué dans la figure 8.5 page suivante.

Vous imaginez bien qu'on peut développer cet exemple quasiment à l'infini, par exemple avec une structure comme celle schématisée dans la figure 8.6 page ci-contre (qui n'indique que les répertoires, pas les fichiers).

FIGURE 8.5 – *Structure du répertoire* `rep` *contenant* `rep2`.

FIGURE 8.6 – *Arborescence imaginaire.*

Une structure de cette forme est ce qu'on appelle un *arbre*. Chaque fichier correspond à l'une des extrémités de la structure, qu'on appelle une *feuille* (les répertoires vides, qui n'ont d'ailleurs aucun intérêt, sont également des feuilles). Chaque répertoire est un embranchement qu'on appelle un *nœud*. Chaque nœud peut être raccordé à d'autres nœuds et à des feuilles. En revanche, un nœud donné n'a qu'un seul nœud de niveau supérieur (voire aucun une fois qu'on se trouve tout en haut de la structure).

Il s'agit exactement de la structure d'un arbre, avec ses branches, ses ramifications et ses feuilles, sauf que l'habitude en informatique est de le représenter à l'envers. Au lieu de pousser vers le ciel, comme les arbres végétaux, les arbres informatiques poussent vers le sol.

C'est pour cette raison que le système de fichiers sous UNIX est souvent qualifié d'*arborescence*. Cette structure arborescente du système de fichiers a été introduite pour la première fois dans Multics, popularisée par UNIX puis reprise par tous les

systèmes d'exploitation qui ont suivi.

Terminons par une remarque de vocabulaire. Lorsqu'on considère un répertoire, on utilise généralement le terme *sous-répertoire* pour désigner un répertoire contenu dans le premier. Ainsi, dans l'exemple de la figure 8.6 page précédente, le répertoire `rep01` a trois sous-répertoires, `rep02`, `rep03` et `rep04`. À son tour, le répertoire `rep02` a trois sous-répertoires, `rep05`, `rep06` et `rep07`. La notion de sous-répertoire est donc relative. Par extension, on peut également qualifier de sous-répertoire un répertoire situé plus profondément dans l'arborescence. Toujours dans l'exemple de la figure 8.6 page précédente, le répertoire `rep19` peut donc être considéré comme un sous-répertoire du répertoire `rep01`.

> http://fr.wikipedia.org/wiki/Arbre_enraciné
> http://fr.wikipedia.org/wiki/Sous-répertoire

8.3.3 Retour sur les commandes de gestion des fichiers

Maintenant que nous avons un répertoire, comment y ranger des fichiers ? Outre le fait de créer directement des fichiers dans ce répertoire, on peut y copier ou y déplacer des fichiers existants.

Pour cela, nous pouvons utiliser les commandes `cp`, `mv` et `ln` abordées respectivement aux paragraphes 8.1.1 page 107, 8.1.2 page 109 et 8.1.4 page 111.

8.3.3.1 La commande `cp` et les répertoires

Reprenons notre exemple habituel et copions le fichier `toto.txt` dans le répertoire `rep` :

```
$ ll
total 8
drwxr-xr-x 2 babafou prof 4096 Aug  4 15:56 rep
-rw-r--r-- 1 babafou prof  334 Jul 30 17:02 toto.txt
$ cp toto.txt rep
$ ll rep
total 4
-rw-r--r-- 1 babafou prof  334 Aug  4 17:45 toto.txt
$ ll
total 8
drwxr-xr-x 2 babafou prof 4096 Aug  4 15:56 rep
-rw-r--r-- 1 babafou prof  334 Jul 30 17:02 toto.txt
```

Lorsque son second argument est un répertoire, la commande `cp` y copie le fichier dont le nom est passé en premier argument.

Nous avons donc maintenant deux fichiers du même nom : `toto.txt`. Comment les différencier ? C'est très simple, ils n'ont pas le même chemin d'accès. Le fichier d'origine a simplement pour chemin d'accès `toto.txt` alors que sa copie a pour chemin d'accès `rep/toto.txt`.

Il est en effet tout à fait possible d'avoir deux fichiers du même nom tant que leurs chemins d'accès sont différents.

La commande de copie précédente est donc équivalente à :

```
$ cp toto.txt rep/toto.txt
```

Il est possible de nommer la copie différemment du fichier d'origine en indiquant explicitement son nom :

```
$ cp toto.txt rep/tata.txt
```

Enfin, si l'on désire copier un ensemble de fichiers sous le même nom dans un répertoire, il est possible de faire comme ceci :

```
$ cp fichier1 fichier2 fichier3 répertoire
```

8.3.3.2 La commande mv et les répertoires

De la même façon, on peut déplacer un fichier dans un répertoire :

```
$ ll
total 8
drwxr-xr-x 2 babafou prof 4096 Aug  4 15:56 rep
-rw-r--r-- 1 babafou prof  334 Jul 30 17:02 toto.txt
$ mv toto.txt rep
$ ll rep
total 4
-rw-r--r-- 1 babafou prof  334 Jul 30 17:02 toto.txt
$ ll
total 4
drwxr-xr-x 2 babafou prof 4096 Aug  4 15:56 rep
```

C'est d'ailleurs la fonction principale de la commande mv, comme son nom peut le laisser supposer. Le renommage n'est qu'une forme particulière de déplacement.

On peut également renommer un fichier tout en le déplaçant :

```
$ mv toto.txt rep/tata.txt
```

Et, comme pour cp, on peut déplacer un ensemble de fichiers dans un répertoire :

```
$ mv fichier1 fichier2 fichier3 répertoire
```

8.3.3.3 La commande ln et les répertoires

L'utilisation de liens prend tout son sens avec les répertoires. Il semble en effet inutile de disposer de plusieurs noms pour le même fichiers sauf si ce fichier doit exister de manière identique à des endroits différents.

La syntaxe d'utilisation de ln avec des répertoires est strictement identique à celle de cp et de mv qui vient d'être indiquée. Nous n'allons pas la rappeler une troisième fois mais donner un exemple de la façon dont on peut utiliser les liens.

Considérons un laboratoire dont l'équipe, composée de quatre personnes, doit analyser des fichiers volumineux. Ces fichiers sont stockés dans un répertoire commun, accessible à tous, et ne doivent pas être déplacés. Or, pour faciliter l'utilisation de

ces fichiers, il serait pratique que chaque personne ait les fichiers utiles dans son répertoire de travail (`p1` pour la première personne, `p2` pour la deuxième et ainsi de suite). Copier les fichiers serait possible mais cela gaspillerait de l'espace de stockage puisque les fichiers présents dans les répertoires de travail des personne seraient en double par rapport au répertoire `commun`. Une solution élégante est, dans chaque répertoire de travail, de créer des liens ou des liens symboliques vers les fichiers à analyser dans le répertoire `commun`, comme schématisé dans la figure 8.7.

FIGURE 8.7 – *Exemple d'utilisation de* `ln`.

Ainsi chacun dispose dans son répertoire des fichiers qui lui ont été attribués sans augmenter le volume des données stockées (un lien ou un lien symbolique ne prennent quasiment pas de place sur disque).

8.3.4 Se déplacer dans un répertoire : la commande `cd`

Nous avons vu au paragraphe 8.3.2 page 119 comment désigner un fichier stocké dans un répertoire, éventuellement sur plusieurs niveaux de profondeur, en préfixant son nom par ceux de la suite de répertoires à traverser, séparés par des `/`.

Cela convient si l'on n'a que quelques commandes à taper mais le fait de préfixer les noms de fichiers peut vite s'avérer fastidieux. Il serait plus utile de pouvoir se déplacer dans le répertoire qui contient les fichiers sur lesquels on veut travailler afin de pouvoir les désigner directement par leur nom.

Ceci se fait au moyen de la commande `cd` (pour *change directory*) :

```
$ cd rep
```

On se retrouve alors dans le répertoire `rep` et l'on peut désigner les fichiers qui s'y trouvent directement par leur nom.

Le répertoire `rep` est maintenant le *répertoire courant* (*working directory* en anglais), c'est-à-dire celui dans lequel on se trouve et où l'on peut désigner les fichiers sans les préfixer.

Un effet de bord intéressant de la commande `cd` est qu'elle modifie l'invite de l'interpréteur de commandes :

```
babafou@babasse:/home/prof/babafou>cd rep
babafou@babasse:/home/prof/babafou/rep>
```

Au paragraphe 7.2.2 page 92, nous n'avions pas décrit la partie de l'invite se situant entre : et >. Nous ne pouvons pas encore la décrire totalement mais vous devez maintenant reconnaître des choses. Cela ressemble quelque part à un chemin d'accès. C'est en effet le cas, c'est le chemin d'accès absolu (nous reviendrons sur cette notion au paragraphe 8.3.6 page 131) du répertoire courant.

> Le fait d'indiquer le chemin d'accès absolu du répertoire courant dans l'invite est spécifique à la configuration de l'ENSTA ParisTech mais c'est une pratique assez répandue parce qu'elle permet de savoir en un coup d'œil dans quel répertoire on se trouve.

À défaut, on peut utiliser la commande `pwd` (pour *print working directory*), qui affiche le chemin d'accès absolu du répertoire courant :

```
babafou@babasse:/home/prof/babafou/rep>pwd
/home/prof/babafou/rep
```

Profitons-en pour faire le tour du propriétaire :

```
$ ll -a
total 12
drwxr-xr-x 2 babafou prof 4096 Aug  5 16:00 .
drwxr-xr-x 6 babafou prof 4096 Jul 30 17:02 ..
-rw-r--r-- 1 babafou prof  334 Aug  4 17:45 toto.txt
```

Alors que nous venons tout juste de créer le répertoire `rep` et de recopier un fichier dedans, voilà déjà que deux autres répertoires le squattent ! Mais d'où viennent-ils ? Et pourquoi ont-ils ces noms étranges ?

Si vous avez fait attention, nous les avons déjà aperçus dans les exemples d'affichage de `ls -a`.

Les répertoires `.` et `..` font partie de la structure du système de fichiers. Ils existent dans chaque répertoire et sont créés automatiquement dans tout nouveau répertoire.

Le répertoire `.` est un lien sur le répertoire qui le contient. Il est en effet utile, nous le verrons par la suite, de pouvoir désigner « ici » de manière simple. En conséquence, la commande suivante est sans effet :

```
babafou@babasse:/home/prof/babafou/rep>cd .
babafou@babasse:/home/prof/babafou/rep>
```

Le répertoire `..` est un lien sur le répertoire parent de celui qui le contient dans l'arborescence du système de fichiers, c'est-à-dire « au-dessus ». Nous avons en effet vu au paragraphe 8.3.2 page 119 qu'un nœud donné de l'arborescence du système de fichiers n'a qu'un seul nœud de niveau supérieur. En conséquence, la commande suivante permet de remonter d'un répertoire :

```
babafou@babasse:/home/prof/babafou/rep>cd ..
babafou@babasse:/home/prof/babafou>
```

En résumé, on peut donc dire que `.` est « ici » et que `..` est « au-dessus ».

> http://fr.wikipedia.org/wiki/Cd_(Commande)
> http://fr.wikipedia.org/wiki/Pwd

8.3.5 Promenade dans le système de fichiers

Qui a dit que l'informatique était ennuyeuse ? Au contraire, c'est l'occasion de faire une promenade dans des contrées sauvages...

Puisqu'il est maintenant possible de se déplacer dans le système de fichiers, nous pouvons envisager d'explorer les répertoires qui nous entourent. Il est assez clair au vu de l'invite ou de l'affichage de la commande `pwd` que nous sommes dans un répertoire `/home/prof/babafou`. Pour les élèves, il s'agit plutôt d'un répertoire `/home/<promo>/<identifiant>` (remplacez `<promo>` par votre promotion et `<identifiant>` par votre identifiant).

Qu'y a-t-il au-dessus ?

```
babafou@babasse:/home/prof/babafou>cd ..
babafou@babasse:/home/prof>
```

Faites de même et affichez le contenu du répertoire avec `ls`. Vous y verrez les répertoires personnels de tous les membres de votre promotion.

Chaque utilisateur d'un système UNIX possède en effet un *répertoire personnel* (*home directory* en anglais), qui est celui dans lequel il est placé en début de session et qui contient tous ses fichiers.

Pour vous, il s'agit de notre fameux `/home/<promo>/<identifiant>`.

Qu'y a-t-il au-dessus ?

```
babafou@babasse:/home/prof>cd ..
babafou@babasse:/home>
```

> On retrouve ici les différentes catégories d'utilisateurs de l'ENSTA ParisTech : élèves par promotion, administratifs, assoces, professeurs, laboratoires, etc. Le nombre d'utilisateurs du système informatique de l'ENSTA ParisTech est en effet suffisamment important pour qu'il soit intéressant de hiérarchiser les répertoires personnels plutôt que de les stocker tous au même niveau.

Qu'y a-t-il au-dessus ?

```
babafou@babasse:/home>cd ..
babafou@babasse:/>
```

Ce qu'on trouve ici est plus difficile à décrire. Nous ne le ferons sommairement dans un instant.

Qu'y a-t-il au-dessus ?

```
babafou@babasse:/>cd ..
babafou@babasse:/>
```

Tiens, on ne bouge plus. Le répertoire courant dans l'invite n'a visiblement pas changé et il est facile de vérifier avec `ls` qu'on voit toujours les mêmes fichiers qu'auparavant.

Nous sommes en effet dans le *répertoire racine* (*root directory* en anglais) du système de fichiers. Le répertoire racine est le point de départ du système de fichiers et on le désigne sous le nom `/`. Le même caractère sert donc à désigner le répertoire racine et à séparer les composants d'un chemin d'accès.

Le répertoire racine est un répertoire particulier, pour lequel `.` et `..` sont identiques. C'est même le seul répertoire ayant cette particularité.

Revenons sur le contenu du répertoire racine. Il serait fastidieux de le décrire en détail, d'autant plus qu'il existe des différences entre UNIX utilisés mais on peut dégager certaines constantes. Nous en profiterons d'ailleurs pour explorer l'arborescence plus en profondeur.

FIGURE 8.8 – *Structure du système de fichiers sous UNIX.*

L'arborescence classique d'un système de fichiers UNIX est schématisée dans la figure 8.8.

Le répertoire racine contient :

- un répertoire `bin` (pour *binaries*), qui contient beaucoup des commandes que nous avons utilisées ;
- un répertoire `dev` (pour *devices*), qui contient les fichiers spéciaux (dont nous discuterons au paragraphe 9.4.1 page 193) ;
- un répertoire `etc` (pour *et cetera*), qui contient les fichiers de configuration du système ;
- un répertoire `home`, qui contient, comme nous l'avons déjà vu, les répertoires personnels des utilisateurs ;
- un répertoire `lib` (pour *libraries*), qui contient des bibliothèques de code utilisées par l'ensemble du système ;
- un répertoire `sbin`, qui contient des commandes utilisées pour la maintenance du système ;
- un répertoire `tmp`, qui sert à stocker des fichiers temporaires ;

- un répertoire `usr`, que nous allons décrire plus en détail ;
- un répertoire `var`, qui contient des données dont le volume peut varier (documents en attente d'impression, par exemple).

Le répertoire `/usr` contient :

- un répertoire `X11R6`, qui contient les logiciels et fichiers de l'interface graphique ;
- un répertoire `include`, qui contient les fichiers d'en-tête du compilateur C ;
- un répertoire `local`, dont nous allons discuter juste après ;
- un répertoire `share`, qui contient des fichiers partageables entre différents systèmes, principalement de la documentation.

Le répertoire `/usr` contient également des répertoires `bin`, `lib` et `sbin`, dont le principe est identique à ceux qu'on trouve dans le répertoire racine. La différence est que ceux du répertoire racine contiennent des fichiers indispensables au démarrage du système. Historiquement, `/usr` était placé sur un disque différent du disque de démarrage, voire sur un serveur de fichiers, et n'était donc pas accessible pendant le démarrage du système. On y stockait donc des fichiers uniquement utiles une fois le démarrage terminé et donc destinés aux utilisateurs (d'où le nom `usr`).

Le répertoire `/usr/local` contient des répertoires `bin`, `include`, `lib`, `sbin` et `share`, sur le même principe que `/usr`, à la différence qu'ils sont locaux à chaque ordinateur ou à chaque site. Les fichiers spécifiques de la configuration de l'ENSTA ParisTech, par exemple, sont donc quelque part sous ce répertoire.

Vous pouvez utiliser la commande `man hier` pour avoir une description plus précise de l'arborescence du système de fichiers de l'UNIX que vous utilisez.

Une fois la promenade terminée, chacun peut rentrer chez soi au moyen de la commande `cd`, sans argument :

```
babafou@babasse:/>cd
babafou@babasse:/home/prof/babafou>
```

Où qu'on soit dans le système de fichiers, `cd` sans argument ramène dans son répertoire personnel.

> http://fr.wikipedia.org/wiki/Répertoire_racine
> http://fr.wikipedia.org/wiki/Filesystem_Hierarchy_Standard

8.3.6 Chemin d'accès absolu et chemin d'accès relatif

Après ce périple épique, nous avons bien mérité de nous reposer en discutant de vocabulaire. Il faut en effet approfondir le concept de *chemin d'accès*.

Un chemin d'accès *absolu* est un chemin d'accès qui part du répertoire racine du système de fichiers et parcourt celui-ci pour aboutir au fichier à désigner. Un chemin d'accès absolu commence donc par `/`. Par exemple, `/home/prof/babafou/toto` est un chemin d'accès absolu.

Un chemin d'accès *relatif* est un chemin d'accès qui ne part pas du répertoire racine du système de fichiers. Un chemin d'accès relatif ne commence donc pas par `/`. Par exemple, `rep/toto` est un chemin d'accès relatif.

Un chemin d'accès absolu est non ambigu, il désigne toujours le même fichier, indépendamment de l'endroit où l'on se trouve. Un chemin d'accès relatif, en revanche, est relatif à l'endroit où l'on se trouve.

Considérons par exemple l'arborescence représentée dans la figure 8.9.

FIGURE 8.9 – *Chemin d'accès relatif (exemple 1).*

Si l'on se trouve dans le répertoire `rep1`, le fichier `toto` de gauche a pour chemin d'accès relatif `rep/toto`. Celui de droite a pour chemin d'accès relatif `../rep2/rep/toto`.

Si l'on se trouve dans le répertoire `rep2`, le fichier `toto` de gauche a pour chemin d'accès relatif `../rep1/rep/toto`. Celui de droite a pour chemin d'accès relatif `rep/toto`.

Le répertoire courant conditionne donc les chemins d'accès relatifs. Ce sont néanmoins ceux qu'on utilise le plus souvent parce qu'ils sont plus courts que les chemins d'accès absolus et donc plus pratiques à utiliser.

Il est important de bien appréhender leur construction, qui est en fait un parcours d'arbre très simple. Le départ s'effectue du répertoire courant. Quand on

remonte un niveau, on met `..`, quand on descend, on met le nom du répertoire qu'on traverse. Et on sépare le tout avec des `/`.

Considérons maintenant l'arborescence un peu plus complexe représentée dans la figure 8.10.

FIGURE 8.10 – *Chemin d'accès relatif (exemple 2).*

Le tableau 8.1 indique, lorsqu'on se trouve dans chacun des répertoires, les chemins d'accès relatifs des fichiers `toto1` et `toto2`.

	toto1	toto2
rep1	rep3/toto1	rep4/toto2
rep2	../rep1/rep3/toto1	../rep1/rep4/toto2
rep3	toto1	../rep4/toto2
rep4	../rep3/toto1	toto2
rep5	../../rep1/rep3/toto1	../../rep1/rep4/toto2
rep6	../../rep1/rep3/toto1	../../rep1/rep4/toto2

TABLE 8.1 – *Chemin d'accès relatif (exemple 2), `toto1` et `toto2`.*

Essayez, sans regarder le tableau 8.2 page ci-contre qui contient la solution, de faire la même chose pour `toto3` et `toto4`.

Pour terminer, signalons une syntaxe particulière :

```
$ cd ~identifiant
```

Ici, l'interpréteur de commandes va remplacer `~identifiant` par le chemin d'accès absolu du répertoire personnel de l'utilisateur indiqué. Ceci fonctionne bien entendu avec n'importe quelle commande utilisant un chemin d'accès :

	toto3	toto4
rep1	../rep2/rep5/toto3	../rep2/rep6/toto4
rep2	rep5/toto3	rep6/toto4
rep3	../../rep2/rep5/toto3	../../rep2/rep6/toto4
rep4	../../rep2/rep5/toto3	../../rep2/rep6/toto4
rep5	toto3	../rep6/toto4
rep6	../rep5/toto3	toto4

TABLE 8.2 – *Chemin d'accès relatif (exemple 2), `toto3` et `toto4`.*

```
$ more ~identifiant/rep/toto.txt
```

Si l'identifiant n'est pas indiqué, ~ fait référence à son propre répertoire personnel :

```
$ more ~/rep/toto.txt
```

http://fr.wikipedia.org/wiki/Chemin_d'accès

8.3.7 Suppression de répertoires

8.3.7.1 La commande `rmdir`

La commande `rmdir` (pour *remove directory*) permet de supprimer un ou plusieurs répertoires vides (c'est-à-dire ne contenant plus rien d'autre que `.` et `..`) :

```
$ rmdir rep
$ rmdir rep1 rep2 rep3
```

Si le répertoire à supprimer n'est pas vide, on obtient un message d'erreur :

```
$ rmdir rep
rmdir: rep: Directory not empty
```

> http://en.wikipedia.org/wiki/Rmdir

8.3.7.2 L'option `-r` de la commande `rm`

Si l'on souhaite en revanche supprimer un ou plusieurs répertoires et tout leur contenu, il faut utiliser la commande `rm` avec l'option `-r` :

```
$ rm -r rep
$ rm -r rep1 rep2 rep3
```

8.3.8 Quelques conseils pour organiser vos fichiers

Maintenant que vous savez travailler avec des répertoires, vous pouvez créer, par exemple, un répertoire `sii` pour y ranger tous les fichiers relatifs à ce cours au lieu de les amonceler directement dans votre répertoire personnel.

Par la suite, vous pourrez créer un répertoire par cours et, au besoin, ramifier ce répertoire si nécessaire, par exemple en créant dedans un répertoire par petite classe pour y stocker les exercices de chaque séance.

8.4 Les métacaractères de génération de noms de fichiers

Nous avons vu précédemment que certaines commandes acceptaient un nombre quelconque d'arguments. Il n'est ainsi pas rare de traiter plusieurs fichiers en une seule commande plutôt que un par un.

Par exemple, on utilisera :

```
$ rm toto1 toto2 toto3
```

plutôt que :

```
$ rm toto1
$ rm toto2
$ rm toto3
```

Dans cet exemple, il n'y a que trois fichiers mais imaginez qu'il faille supprimer les fichiers `toto00` jusqu'à `toto99`. Même avec l'aide du complètement, taper la commande de suppression serait fastidieux et répétitif. Or nous avons vu que l'un des objectifs de l'interpréteur de commandes était justement de fournir des outils permettant de simplifier les tâches répétitives.

C'est le but des *métacaractères de génération de noms de fichiers* (qu'on appellera plus simplement métacaractères par la suite). Leur but est d'exprimer un ensemble de noms de fichiers de la manière la plus compacte possible.

http://fr.wikipedia.org/wiki/Métacaractère

8.4.1 Le métacaractère ?

Revenons à notre premier exemple. Si l'on veut supprimer les fichiers `toto1`, `toto2` et `toto3`, on peut donc utiliser la commande :

```
$ rm toto1 toto2 toto3
```

Mais on aura plus vite fait ainsi :

```
$ rm toto?
```

Le caractère `?` est en effet un métacaractère remplaçant exactement un caractère quelconque.

Comment cela fonctionne-t-il ? Une fois qu'on a tapé sur Enter, l'interpréteur de commandes va analyser la ligne à la recherche de caractères ayant une signification particulière pour lui. C'est le cas de `?`. Il considère donc l'expression contenant ce caractère, à savoir `toto?`. Cela signifie : les noms de fichiers commençant par `toto`, suivi d'exactement un caractère quelconque. L'interpréteur de commandes effectue alors l'équivalent d'un `ls`, sélectionne les fichiers dont le nom correspond à ce critère, remplace `toto?` dans la ligne de commandes par la liste de ces fichiers et, finalement, exécute la commande.

On peut visualiser ce processus en utilisant la commande `echo`, dont le but est simplement d'afficher la liste de ses arguments :

```
$ echo toto?
toto1 toto2 toto3
```

La commande `echo` n'a donc pas reçu `toto?` en argument mais la liste explicite composée de `toto1`, `toto2` et `toto3`. C'est l'interpréteur de commandes qui a effectué cette substitution.

Lorsqu'on commence à utiliser les métacaractères ou qu'on souhaite simplement vérifier une commande avant de l'exécuter, il est intéressant d'utiliser `echo`. Cela permet de vérifier que l'expression contenant les métacaractères est correcte sans risquer de perdre des fichiers. Dans les exemples suivants, nous utiliserons donc `echo`.

Attention, s'il existe des fichiers `toto4`, `toto5`, etc. ou des fichiers `totoa`, `totob`, etc., ils seront pris en compte par l'expression `toto?`. Nous verrons des métacaractères plus restrictifs dans la section suivante.

Il faut également bien comprendre que les expressions contenant des métacaractères vont générer uniquement les noms des fichiers qui existent déjà. Cela ne peut servir à créer de nouveaux fichiers. Dans les exemples suivants, nous supposerons que les fichiers mentionnés existent bien.

Il est bien entendu possible d'utiliser `?` n'importe où dans un nom de fichier. Par exemple l'expression suivante affichera tous les fichiers dont le nom commence par `toto` suivi d'exactement un caractère suivi de `.txt` :

```
$ echo toto?.txt
```

On peut même utiliser des expressions contenant plusieurs `?` si cela est nécessaire. Ainsi l'expression suivante affichera les fichiers `toto00` jusqu'à `toto99` :

```
$ echo toto??
```

8.4.2 Les métacaractères `[` et `]`

Le métacaractère `?` remplace exactement un caractère quelconque. Mais il est parfois utile d'être plus précis. Ainsi, s'il existe des fichiers `toto1` jusqu'à `toto9` mais qu'on ne souhaite exprimer que `toto1` jusqu'à `toto3`, on peut utiliser :

```
$ echo toto[123]
```

L'expression `[123]` représente un et un seul caractère parmi ceux indiqués, à savoir `1`, `2` ou `3`.

On peut également indiquer un intervalle. Ici, `[1-5]` désigne les chiffres de `1` à `5` :

```
$ echo toto[1-5]
```

On peut également utiliser des lettres, minuscules ou majuscules :

```
$ echo toto[a-z]
$ echo toto[A-Z]
```

Attention à ne pas mélanger chiffres, minuscules et majuscules, il faut avoir le même type de caractère de part et d'autre du tiret.

8.4.3 Le métacaractère *

Le meilleur est pour la fin. Le métacaractère * remplace zéro, un ou plusieurs caractères. C'est le métacaractère ultime. Avec lui, plus rien ne dépasse !

Par exemple, l'expression suivante affichera tous les fichiers dont le nom commence par `toto` (suivi de rien ou de quelque chose) :

```
$ echo toto*
```

Celle-ci affichera tous les fichiers dont le nom se termine par `.txt` :

```
$ echo *.txt
```

Et ceci n'est ni plus ni moins que le `ls` du pauvre :

```
$ echo *
```

Vous pouvez imaginer les ravages que peut produire une commande telle que :

```
$ rm *
```

Il faut d'ailleurs être prudent lorsqu'on utilise * avec des commandes destructrices. Supposons que l'on souhaite supprimer tous les fichiers dont le nom commence par `toto` et qu'on fasse :

```
$ rm toto *
```

Oui, il y a bien une funeste erreur de frappe : une espace entre `toto` et `*`. Ceci va donc supprimer `toto` et tous les fichiers se trouvant dans le répertoire courant. Les fichiers que l'on souhaitait supprimer l'auront bien été mais avec quelques dommages collatéraux. Prudence, donc.

8.4.4 Utilisation avancée

Indiquons quelques exemples plus compliqués d'utilisation des métacaractères.

On peut utiliser les métacaractères dans des chemins d'accès :

```
$ echo rep/toto*
```

Les métacaractères génèrent indifféremment noms de fichiers et noms de répertoires :

```
$ echo */toto.txt
```

Et on peut bien entendu tout mélanger :

```
$ echo toto-200[0-4]-??-??-*
```

8.4.5 Gestion des erreurs

On ne le répétera jamais assez : les métacaractères ne permettent de générer que des noms de fichiers qui existent. Les utiliser dans des expressions ne permettant de générer aucun nom de fichier peut causer des surprises.

En effet, le comportement des métacaractères appliqués à des noms de fichiers qui n'existent pas est différent suivant les interpréteurs de commandes. On peut le tester avec la commande `echo`.

Supposons qu'il existe dans le répertoire courant trois fichiers `toto1`, `toto2`, `toto3` et c'est tout.

Les interpréteurs de commandes `sh`, `ksh` et `bash`, confrontés à une expression contenant un métacaractère et ne correspondant à aucun fichier existant, laissent cette expression telle quelle :

```
$ echo toto? tata?
toto1 toto2 toto3 tata?
```

Pour leur part, `csh` et `tcsh` suppriment l'expression fautive :

```
% echo toto? tata?
toto1 toto2 toto3
```

Enfin, `zsh` adopte le seul comportement qu'on puisse considérer comme sensé, il n'exécute pas la commande et affiche un message d'erreur :

```
% echo toto? tata?
zsh: no matches found: tata?
```

Aucun de ces comportement ne pose de problème mais il est préférable d'être conscient de ces différences de comportement. Il est encore meilleur d'éviter d'utiliser des expressions contenant un métacaractère et ne permettant de générer aucun nom de fichier.

8.5 Les droits d'accès

UNIX a toujours été un système d'exploitation *multi-utilisateur*, c'est-à-dire qu'il permet à plusieurs personnes d'utiliser le même ordinateur simultanément. Cela nécessite certaines mesures de sécurité, en particulier en ce qui concerne l'accès aux fichiers. On peut en effet vouloir que certains utilisateurs puissent accéder à certains de ses fichiers mais pas à d'autres, que d'autres utilisateurs ne puissent accéder à aucun de ses fichiers, etc.

C'est l'objet des *droits d'accès*.

http://fr.wikipedia.org/wiki/Permissions_Unix

8.5.1 Principe des droits d'accès

Chaque fichier dispose de droits d'accès qui sont stockés dans son i-nœud et affichés en première colonne par `ls -l` :

```
$ ls -l toto.txt
-rw-r--r-- 1 babafou prof   334 Jul 30 17:02 toto.txt
```

Les droits d'accès suivent immédiatement le premier caractère de la première colonne, qui indique le type du fichier. Les droits d'accès du fichier `toto.txt` sont donc :

```
rw-r--r--
```

Il faut lire ceci comme trois ensembles de trois caractères :

```
rw- r-- r--
```

Dans chaque ensemble de trois caractères, on peut avoir :

1. en première position, un `r` (pour *read*) si le droit de lire le fichier est accordé ou un `-` s'il ne l'est pas ;
2. en deuxième position, un `w` (pour *write*) si le droit d'écrire dans le fichier est accordé ou un `-` s'il ne l'est pas ;
3. en troisième position, un `x` (pour *execute*) si le droit d'exécuter [2] le fichier est accordé ou un `-` s'il ne l'est pas ; ce droit n'a de sens que pour les logiciels, pas pour les fichiers de données.

Chaque ensemble de droits concerne des utilisateurs particuliers du système informatique :

1. le premier ensemble s'applique au propriétaire du fichier ;
2. le deuxième ensemble s'applique aux membres du groupe du fichier ;
3. le troisième ensemble s'applique à tous les autres utilisateurs.

Détaillons tout cela avec notre fichier `toto.txt` :

```
$ ls -l toto.txt
-rw-r--r-- 1 babafou prof   334 Jul 30 17:02 toto.txt
```

2. Exécuter au sens de « lancer un programme », pas de « fusiller »...

Le premier ensemble de droits, `rw-`, s'applique donc au propriétaire du fichier, `babafou` (qui est indiqué dans la troisième colonne de l'affichage de `ls -l`). Celui-ci a (`r`) le droit de lire son fichier. Il a également (`w`) le droit d'écrire dans son fichier. Tout cela semble raisonnable. En revanche, il n'a pas (`-`) le droit d'exécuter ce fichier. Puisqu'il ne s'agit pas d'un logiciel mais d'un fichier de données, c'est tout à fait compréhensible.

Le deuxième ensemble de droits, `r--`, s'applique aux membres du groupe du fichier. Celui-ci est indiqué dans la quatrième colonne de l'affichage de `ls -l`. Il s'agit ici de `prof`. À l'ENSTA ParisTech, tous les enseignants font partie de ce groupe. Ils peuvent donc lire le fichier `toto.txt` et c'est tout. Les élèves, quant à eux, sont regroupés par promotion. Les droits d'accès de groupe de leurs fichiers concernent donc leur promotion.

Le troisième ensemble de droits, `r--`, s'applique à tous les autres utilisateurs du système informatique, c'est-à-dire à ceux qui ne sont ni l'utilisateur `babafou` ni les membres du groupe `prof`.

Le tableau 8.3 résume tout ceci.

	propriétaire			groupe			autres		
présence du droit	r	w	x	r	w	x	r	w	x
absence du droit	-	-	-	-	-	-	-	-	-

TABLE 8.3 – *Droits d'accès.*

Il est logique que le propriétaire d'un fichier dispose de plus ou d'au moins autant de droits que les membres de son groupe. De même, il est logique que ceux-ci disposent de plus ou d'au moins autant de droits que les autres utilisateurs.

Des droits d'accès tels que ceux-ci, par exemple, n'ont donc aucune logique et ne devraient être rencontrés sur aucun système UNIX :

```
--xr-xrwx
```

Les droits d'accès sur les répertoires ont une signification semblable si l'on considère un répertoire comme étant la liste des noms des fichiers qu'il contient :

1. le droit `r` donne la possibilité d'afficher la liste des noms des fichiers contenus dans le répertoire, donc de faire `ls` dessus ;
2. le droit `w` donne la possibilité d'écrire dans la liste des noms des fichiers contenus dans le répertoire, donc de créer des fichiers dans le répertoire,

même si l'on n'est pas propriétaire de celui-ci, de renommer et de supprimer des fichiers dans le répertoire, même si l'on n'est pas propriétaire de ces fichiers ;

3. le droit `x` donne la possibilité de traverser le répertoire, c'est-à-dire de faire `cd` dans le répertoire ou d'accéder au contenu de ses fichiers (sous réserve que les droits d'accès aux fichiers eux-mêmes le permettent).

Il convient d'insister sur le danger d'avoir sur ses propres répertoires un `w` ailleurs que pour soi. Un `w` pour le groupe fait que tous les membres de celui-ci peuvent en particulier supprimer les fichiers contenus dans le répertoire. Un `w` pour les autres est évidemment bien plus dangereux.

Les droits d'accès `r` et `x` sur les répertoires, quant à eux, vont quasiment toujours ensemble. Il semble en effet peu logique de pouvoir afficher le contenu d'un répertoire si l'on ne peut s'y déplacer et *a contrario* de ne pas avoir la liste des fichiers qu'il contient si l'on peut aller dedans.

Imaginez que le répertoire est une pâtisserie. Le droit `r` est le droit de regarder dans la vitrine tous ces babas au rhum, ces éclairs, ces meringues, ces millefeuilles, ces oranais, ces paris-brest, ces religieuses... Le droit `x` est le droit d'entrer dans la pâtisserie pour céder à la tentation... Ça ne sert à rien de baver devant la vitrine si l'on ne peut pas rentrer dans la boutique (`r--`). De même, y rentrer sans voir ce qu'on y vend est frustrant (`--x`).

Enfin, les plus perspicaces auront remarqué que les liens symboliques avaient des droits d'accès inquiétants :

```
lrwxrwxrwx 1 babafou prof   8 Aug  3  2009 tata.txt -> toto.txt
```

En réalité, les droits d'accès des liens symboliques sont ignorés, ce sont toujours les droits d'accès du fichier vers lequel pointe le lien symbolique qui sont pris en compte.

8.5.2 Représentation numérique des droits d'accès

Comme toujours en informatique, il est nécessaire de représenter les droits d'accès sous forme numérique plutôt que sous la forme d'une chaîne de caractères telle que `rw-r--r--`. Comment faire ?

La représentation des droits d'accès sous la forme d'une chaîne de caractères a deux caractéristiques remarquables :

- chaque droit d'accès a une position bien précise ;

– chaque droit d'accès n'a que deux états possibles, accordé ou refusé.

On peut donc envisager de représenter des droits d'accès sous la forme d'un nombre binaire (voir le paragraphe 2.1.3.1 page 25) comprenant neuf chiffres, un par droit d'accès, chaque chiffre correspondant au droit situé à la même position dans la représentation des droits d'accès sous la forme d'une chaîne de caractères et, pour chaque chiffre, 1 représentant l'accord d'un droit et `0` son refus. Ainsi, `rw-r--r--` peut être représenté par 110100100.

Cette représentation est malheureusement loin d'être pratique et il serait intéressant de pouvoir la simplifier. Par définition, le nombre 110100100 se décompose en puissances de 2 ainsi :

$$\begin{aligned}110100100 = {} & 1\times 2^8 + 1\times 2^7 + 0\times 2^6 + \\ & 1\times 2^5 + 0\times 2^4 + 0\times 2^3 + \\ & 1\times 2^2 + 0\times 2^1 + 0\times 2^0\end{aligned}$$

soit, en factorisant la puissance de 2 la plus petite de chaque ligne :

$$\begin{aligned}110100100 = {} & (1\times 2^2 + 1\times 2^1 + 0\times 2^0)\times 2^6 + \\ & (1\times 2^2 + 0\times 2^1 + 0\times 2^0)\times 2^3 + \\ & (1\times 2^2 + 0\times 2^1 + 0\times 2^0)\times 2^0\end{aligned}$$

or $2^0 = 8^0$, $2^3 = 8^1$ et $2^6 = 8^2$:

$$\begin{aligned}110100100 = {} & (1\times 2^2 + 1\times 2^1 + 0\times 2^0)\times 8^2 + \\ & (1\times 2^2 + 0\times 2^1 + 0\times 2^0)\times 8^1 + \\ & (1\times 2^2 + 0\times 2^1 + 0\times 2^0)\times 8^0\end{aligned}$$

et donc, en calculant les expressions entre parenthèses :

$$110100100 = 6\times 8^2 + 4\times 8^1 + 4\times 8^0$$

Le nombre binaire 110100100 peut donc également être représenté par le nombre 644 en base 8. Attention à ne pas dire « six cent quarante-quatre » car il ne s'agit pas de 644 en base 10. En l'occurrence, on ne peut pas vraiment dire mieux que « six quatre quatre »...

Vous conviendrez que la démonstration précédente est généralisable à n'importe quel nombre binaire dont on groupe les chiffres trois par trois. C'est d'ailleurs une relation générale entre une base de numération et ses puissances.

On peut donc représenter les droits d'accès sous forme de nombres en base 8, c'est-à-dire en octal (voir le paragraphe 2.1.3.2 page 25). Chaque chiffre octal correspondant à un ensemble de trois droits d'accès. Les correspondances entre droits d'accès, binaire et octal sont résumées dans le tableau 8.4.

	r	w	x	binaire	octal
---	0	0	0	000	0
--x	0	0	1	001	1
-w-	0	1	0	010	2
-wx	0	1	1	011	3
r--	1	0	0	100	4
r-x	1	0	1	101	5
rw-	1	1	0	110	6
rwx	1	1	1	111	7

TABLE 8.4 – *Droits d'accès en binaire et en octal.*

D'ailleurs, si l'on retient que `r` correspond à $2^2 = 4$, que `w` correspond à $2^1 = 2$ et que `x` correspond à $2^0 = 1$, on peut n'utiliser que ces trois valeurs pour transformer des droits d'accès en nombre octal et *vice versa* plutôt que d'utiliser les huit valeurs du tableau 8.4.

Donc `rw-r--r--` donne `rw-` pour le premier chiffre, soit $4+2=6$, `r--` pour les deuxième et troisième chiffres, soit 4, c'est-à-dire 644 au final. La décomposition inverse se fait sans ambiguïté.

Pour terminer, indiquons que l'ordre dans lequel les droits d'accès sont indiqués (`r` puis `w` puis `x`) est purement arbitraire mais obéit à une certaine logique. Il est en effet plus utile de pouvoir lire le contenu d'un fichier que de pouvoir écrire dedans (en pratique, on consulte beaucoup plus souvent ses fichiers qu'on ne les modifie). De même, il est plus utile de pouvoir écrire dans un fichier que de pouvoir l'exécuter (en pratique, il y a beaucoup plus de fichiers de données que de logiciels). On attribue donc la plus haute valeur numérique à `r`, la suivante à `w` et la plus faible à `x`. De cette façon, comme déjà indiqué, il est facile de vérifier numériquement que le propriétaire d'un fichier possède des droits d'accès supérieurs ou égaux à ceux de son groupe, eux-mêmes supérieurs ou égaux à ceux des autres.

Après ce petit intermède mathématique, revenons à quelque chose de plus informatique.

8.5.3 La commande `chmod`

La commande `chmod` (pour *change mode*) permet de modifier les droits d'accès d'un ensemble de fichiers ou de répertoires.

Sa syntaxe est inhabituelle parce que son premier argument (ce n'est pas une option) indique quels droits d'accès modifier dans les fichiers dont les noms figurent dans les argument suivants.

Prenons un exemple :

```
$ chmod o-r toto
```

Cette commande va supprimer le droit en lecture pour les autres sur le fichier `toto`.

Le premier argument de `chmod` contient en effet trois parties :

1. la première partie indique à qui appliquer la modification des droits d'accès :
   - `u` (pour *user*) s'il s'agit du propriétaire du fichier,
   - `g` (pour *group*) s'il s'agit du groupe du fichier,
   - `o` (pour *others*) s'il s'agit des autres,
   - on peut regrouper ces caractères pour appliquer la modification à plusieurs catégories : `ug`, `uo`, `go`, voire `ugo` qu'on peut également exprimer par `a` (pour *all*) ;
2. la deuxième partie indique si l'on veut ajouter (`+`) ou supprimer (`-`) des droits d'accès ;
3. la troisième partie indique lesquels, sous la forme habituelle (`r`, `w` et `x`).

Donc la commande suivante ajoute les droits en écriture pour le groupe et les autres sur le fichier `toto` :

```
$ chmod go+w toto
```

Il est également possible d'utiliser des métacaractères :

```
$ chmod go+r toto*
```

Néanmoins, l'utilisation de `chmod` pour modifier les droits d'accès de plusieurs fichiers de cette façon se heurte à un problème.

Si l'on a des fichiers ayant des droits d'accès hétéroclites, comment les remettre tous identiques, par exemple `rw-r--r--` ? On peut toujours faire :

```
$ chmod a-rwx toto*
$ chmod a+r toto*
$ chmod u+w toto*
```

pour supprimer les droits d'accès existants, rajouter les droits en lecture pour tous puis le droit en écriture pour le propriétaire mais faire la chose en trois commandes n'est pas élégant.

Dans ce cas, utiliser la représentation numérique des droits d'accès à appliquer aux fichiers est la bonne façon de faire :

```
$ chmod 644 toto*
```

Ici, les droits d'accès sont fixés à 644 (`rw-r--r--`) indépendamment des droits d'accès initiaux des fichiers.

Bien entendu, cela fonctionne également avec un seul fichier :

```
$ chmod 644 toto
```

La commande `chmod`, utilisée avec l'option `-R`, permet également de modifier les droits d'accès récursivement sur le contenu des répertoires rencontrés.

Par exemple, la commande suivante ne modifie que les droits d'accès du répertoire `rep` lui-même sans toucher aux droits d'accès des fichiers et sous-répertoires qu'il contient :

```
$ chmod g+w rep
```

Si l'on ajoute l'option `-R`, les droits d'accès des fichiers et des sous-répertoires contenus dans `rep` (et récursivement à l'intérieur des sous-répertoires) seront également modifiés :

```
$ chmod -R g+w rep
```

http://fr.wikipedia.org/wiki/Chmod

8.5.4 La commande `umask`

La commande `chmod` permet de modifier les droits d'accès de fichiers qui existent déjà. Si l'on souhaite faire en sorte que les fichiers que l'on va créer aient des droits d'accès particuliers, sans avoir à les modifier après leur création, il faut utiliser la commande `umask`.

Utilisée sans argument, elle affiche la valeur (en octal) du masque de création des fichiers (le premier `0` indique simplement que la valeur qui suit est en octal) :

```
$ umask
0022
```

`022` est une valeur classique de masque de création des fichiers sous UNIX.

Les fichiers sont créés par défaut avec 666 comme droits d'accès, moins la valeur du masque de création. $666 - 22 = 644$. C'est pour cela que les fichiers sont créés avec `rw-r--r--` comme droits d'accès.

De même, les répertoires sont créés par défaut avec 777 comme droits d'accès, moins la valeur du masque de création. $777 - 22 = 755$. C'est pour cela que les répertoires sont créés avec `rwxr-xr-x` comme droits d'accès.

Si l'on souhaite que ses fichiers soient créés avec des droits d'accès plus restrictifs, la meilleure méthode est de commencer avec les répertoires.

Par exemple, pour que les répertoires soient créés avec `rwx------`, c'est-à-dire `700`, comme droits d'accès, il faut un masque de création valant 77 (puisqu'il faut soustraire 77 à 777 pour obtenir `700`).

On fait cela en donnant un argument à `umask` :

```
$ umask 77
```

À noter que la soustraction du masque de création se fait sans retenue. Donc, en ce qui concerne les fichiers, ceux-ci seront créés avec les droits d'accès $666 - 77 = 600$.

C'est pour cela qu'il faut calculer le masque de création à partir des droits d'accès que l'on souhaite pour les répertoires. Si l'on était parti des fichiers, on aurait calculé un masque de création valant 66 (puisqu'il faut soustraire 66 à 666 pour obtenir 600). Mais, dans ce cas, les répertoires auraient été créés avec 711 ($777 - 66$) comme droits d'accès, ce qui n'est pas la valeur souhaitée.

Le masque de création des fichiers est indépendant dans chaque interpréteur de commandes. Donc si plusieurs fenêtres d'émulateur de terminal sont ouvertes,

toute modification du masque de création dans l'interpréteur de commandes de l'une d'elles n'a pas d'incidence sur les autres.

http://fr.wikipedia.org/wiki/Umask

8.6 Les noms des fichiers

Nous avons jusqu'à maintenant utilisé des noms de fichiers sans définir exactement quelles règles ils doivent respecter.

En théorie, un nom de fichier sous UNIX peut contenir n'importe quoi sauf `/`, puisque ce caractère sert à séparer les composants d'un chemin d'accès, et l'octet nul (code ASCII `0`), qui sert sous UNIX à marquer la fin d'une chaîne de caractères.

Les autres caractères sont utilisables. Néanmoins, nous avons vu que certains ont une signification particulière pour l'interpréteur de commandes. Et il en existe d'autres, qui n'ont pas encore été étudiés. D'ailleurs, quasiment tous les signes de ponctuation ont une signification particulière pour l'interpréteur de commandes. Il convient donc de les éviter, sauf le point (`.`), le tiret (`-`) et le tiret bas (`_`).

Si l'on tient absolument à utiliser des caractères ayant une signification particulière pour l'interpréteur de commandes, il est nécessaire de prendre des précautions. Supposons que vous avez un fichier malhabilement appelé `*` et que vous voulez le supprimer. Bien entendu, on peut faire :

```
$ rm *
```

Ceci effacera le fichier `*` et un certain nombre d'autres, comme nous l'avons vu au paragraphe 8.4.3 page 137.

Pour n'effacer que le fichier `*`, la bonne façon de faire est d'indiquer à l'interpréteur de commandes de ne pas tenir compte de la signification particulière de ce caractère en le faisant précéder d'une barre oblique inversée `\` :

```
$ rm \*
```

Si un nom de fichier contient plusieurs caractères spéciaux, il faut les faire précéder chacun d'un `\`. Par exemple, pour effacer le fichier `[1]` :

```
$ rm toto\[1\]
```

On peut également entourer l'ensemble du nom de fichier d'une paire de guillemets ou d'apostrophes (les apostrophes permettent d'inhiber plus de caractères spéciaux que les guillemets, voir le paragraphe 12.5 page 261) :

```
$ rm "[1]"
$ rm '[1]'
```

Il en va de même avec les espaces :

```
$ rm nom\ de\ fichier\ avec\ espaces
$ rm "nom de fichier avec espaces"
$ rm 'nom de fichier avec espaces'
```

Il est préférable de ne pas utiliser de caractères accentués dans les noms de fichiers et de s'en tenir aux seuls caractères du code ASCII. En effet, si l'on souhaite transmettre des fichiers d'un ordinateur à un autre, l'ASCII est la seule partie commune de tous les systèmes de codages et un nom de fichier ne contenant que des caractères ASCII sera toujours bien transmis.

De manière plus générale, il faut concevoir ses noms de fichiers de façon à les rendre efficacement manipulables au moyen des métacaractères, c'est-à-dire prévoir les noms des fichiers pour pouvoir désigner au moyen des métacaractères ceux qui auront à être traités ensemble.

Pour cela, on peut par exemple privilégier le format de date ISO 8601, qui indique d'abord l'année, puis le mois, puis le jour (c'est-à-dire qu'on indique les éléments d'une date du plus significatif au moins significatif). Ainsi, 1789-07-14 désigne le 14 juillet 1789.

On peut par exemple traiter tous ses fichiers de 2009 ainsi, quelle que soit leur extension :

```
$ commande fichier-2009-*
```

Pour ne traiter que les fichiers texte :

```
$ commande fichier-2009-*.txt
```

Le format de date habituellement utilisé en France, outre son ambiguïté avec d'autres formats nationaux et le fait qu'on ne puisse pas utiliser de `/` dans un nom de fichier, oblige généralement à utiliser des expressions plus compliquées. Ainsi le premier exemple devient :

```
$ commande fichier-*-2009.*
```

Enfin, il est préférable de ne pas utiliser de tiret comme premier caractère d'un nom de fichier. Supposons que vous avez un fichier `-i` et que vous souhaitez l'effacer :

```
$ rm -i
usage: rm [-f|-i] [-dPRrvW] file ...
```

Ici, `rm` considère `-i` comme une option, puisqu'il s'agit d'un mot commençant par un tiret, et pas comme un argument. Puisqu'il lui manque son argument, il se plaint en affichant la syntaxe permettant de l'utiliser.

Utiliser les guillemets, les apostrophes ou `\` ne fonctionnera pas dans la mesure ou le tiret n'est pas un caractère spécial pour l'interpréteur de commandes.

La bonne façon de faire est d'utiliser un chemin d'accès ne commençant pas par un tiret. Le plus simple est :

```
$ rm ./-i
```

Et le tour est joué !

> http://fr.wikipedia.org/wiki/ISO_8601
> http://www.cl.cam.ac.uk/~mgk25/iso-time.html

8.7 Gestion de l'espace de stockage, compression et archivage

De plus en plus de données sont stockées sous forme de fichiers informatiques. Selon leur nature et leur intérêt, ceux-ci doivent être conservés sur des périodes variables, allant de quelques jours à plusieurs dizaines d'années. Même si les capacités de stockage ne cessent d'augmenter, elles ont néanmoins un coût. Il est donc important de connaître et d'optimiser le volume occupé par ses fichiers.

Lorsqu'ils ne sont plus fréquemment utilisés mais qu'on souhaite les conserver, les fichiers ou ensembles de fichiers peuvent être comprimés (il s'agit de réduire la taille de chaque fichier afin qu'il occupe moins de place tout en préservant son contenu) ou archivés (il s'agit de rassembler un ensemble de fichiers dans un unique fichier, qu'on appelle une archive, et qui est lui-même souvent comprimé).

8.7.1 Gestion de l'espace de stockage

Plus on travaille sur un système informatique, plus on accumule de fichiers. Comment retrouver certains fichiers dont on a oublié l'emplacement exact ? Comment savoir quel est l'espace total occupé par l'ensemble de ses fichiers ? Comment savoir quel est l'espace occupé par chaque répertoire ? Comment connaître l'occupation des disques locaux et des serveurs de fichiers ?

8.7.1.1 Retrouver des fichiers : la commande `find`

La commande `find` permet de retrouver, selon certains critères, des fichiers dont on a oublié l'emplacement exact.

Elle s'utilise suivie d'un ou de plusieurs noms de répertoires dans lesquels effectuer les recherches et d'options indiquant les critères de recherche. Elle parcourt alors récursivement ces répertoires et affiche les noms des fichiers s'y trouvant et correspondant aux critères de recherche indiqués.

Pour les exemples qui suivent, nous supposerons être dans un répertoire courant ayant la structure représentée dans la figure 8.11. `rep1`, `rep2` et `rep3` sont des répertoires, `fich1`, `fich2` et `fich3` sont des fichiers ordinaires.

FIGURE 8.11 – *Arborescence utilisée pour illustrer la commande `find`.*

La façon la plus simple d'utiliser la commande `find` est de ne pas indiquer de critère de recherche. Dans ce cas, elle se contente de parcourir récursivement les répertoires indiqués en arguments en affichant répertoires et fichiers rencontrés :

```
$ find rep1 rep2/rep3
rep1
rep1/fich1
rep2/rep3
rep2/rep3/fich3
```

Lorsqu'elle parcourt un répertoire, la commande `find` affiche répertoires et fichiers rencontrés dans un ordre qui semble aléatoire :

```
1 $ find .
2 .
3 ./rep2
4 ./rep2/rep3
5 ./rep2/rep3/fich3
6 ./rep2/fich2
7 ./rep1
8 ./rep1/fich1
```

Comme on peut le voir, l'affichage n'est pas trié comme l'est, par exemple, celui de la commande `ls`. En effet, dans le répertoire courant, le répertoire `rep2` (lignes 3 à 6) est parcouru avant le répertoire `rep1` (lignes 7 et 8) et, dans le répertoire `rep2`, le fichier `fich2` (ligne 6) apparaît après le répertoire `rep3` (lignes 4 et 5). En réalité, cela n'a rien d'aléatoire. Chaque répertoire est simplement parcouru dans l'ordre selon lequel son contenu est stocké dans le système de fichiers.

Par ailleurs, lorsqu'un sous-répertoire est rencontré, il est parcouru en totalité avant de poursuivre le parcours du répertoire qui le contient. On peut le voir de la ligne 3 à la ligne 6 de l'exemple précédent. On entre dans le répertoire `rep2` à la ligne 3, où l'on trouve un répertoire `rep3` à la ligne 4. Plutôt que de poursuivre le parcours du répertoire `rep2`, on explore `rep3` à la ligne 5 puis on reprend le parcours du répertoire `rep2` à la ligne 6. C'est ce qu'on appelle un parcours en *profondeur d'abord*. L'autre méthode de parcours d'un arbre est le parcours en *largeur d'abord*, dans lequel chaque répertoire est exploré en totalité avant de descendre dans ses sous-répertoires.

Les critères de recherche sont indiqués sous la forme d'options qui doivent être placées après les noms des répertoires dans lesquels effectuer les recherches.

L'option `-name` de la commande `find` permet d'indiquer tout ou partie du nom du fichier recherché. Ici, on recherche, à partir du répertoire courant, les fichiers (il peut y en avoir plusieurs, dans des répertoires différents) qui s'appellent exactement `fich1` :

```
$ find . -name fich1
./rep1/fich1
```

Il est possible d'utiliser des métacaractères (voir le paragraphe 8.4 page 134) en prenant la précaution de les inhiber (voir le paragraphe 8.6 page 148) afin qu'ils ne soient pas pris en compte par l'interpréteur de commandes mais soient passés tels quels à la commande `find`. Ainsi, la commande suivante recherche, à partir du répertoire courant, les fichiers dont le nom commence par `fi` :

```
$ find . -name fi\*
./rep2/rep3/fich3
./rep2/fich2
./rep1/fich1
```

L'option `-mtime` de la commande `find`, suivie d'un nombre n, permet de trouver les fichiers ayant été modifiés depuis moins de n jours :

```
$ find . -mtime 1
./rep2/fich2
```

Il existe de nombreux autres critères de recherche. Le lecteur curieux pourra se reporter au manuel de la commande `find` pour approfondir ses connaissances.

Enfin, il convient de terminer par une mise en garde. Il arrive de voir des exemples d'utilisation de la commande `find` effectuant leur recherche à partir du répertoire racine `/` et tout administrateur système a déjà été confronté à des utilisateurs faisant de même, à la recherche d'un de leurs fichiers :

```
$ find / -name toto
```

Il faut être conscient qu'une telle recherche implique de parcourir l'ensemble du système de fichiers, qui peut atteindre un volume conséquent si l'ordinateur utilisé est raccordé à des serveurs de fichiers de grande capacité — ce qui est fréquent sur des ordinateurs autres que des postes purement personnels — tels que des serveurs de calcul. Et, plus le système de fichiers à parcourir est volumineux, plus cette recherche durera longtemps. Par ailleurs, une telle recherche utilise des ressources non négligeables au niveau du microprocesseur. Il est de toute façon généralement inutile d'effectuer une recherche à partir du répertoire racine, étant donné que les fichiers des utilisateurs ne peuvent se trouver que dans leurs répertoires personnels respectifs et, éventuellement, dans certains répertoires partagés, dont les utilisateurs concernés ont nécessairement connaissance puisqu'ils y ont sciemment placé des fichiers. Il est donc indispensable, lorsqu'on cherche un fichier, de se limiter aux seuls répertoires où il peut se trouver, afin de ne pas gaspiller les ressources informatiques mises à disposition.

> http://fr.wikipedia.org/wiki/Find

8.7.1.2 La commande `quota`

Sur certains systèmes informatiques, les utilisateurs se voient imposer une limitation de l'espace de stockage alloué à leurs fichiers. Une telle limitation s'appelle un *quota*.

Il existe deux types de quotas :

- un quota sur le volume occupé par les fichiers ;
- un quota sur le nombre de fichiers.

Les quotas ont pour but d'éviter qu'un utilisateur ne remplisse à lui seul la partie restante d'un espace de stockage mutualisé, au détriment des autres utilisateurs. À cet effet, il convient que l'administrateur du système s'assure que la somme des quotas des utilisateurs est inférieure à la taille de l'espace de stockage mutualisé.

> À l'ENSTA ParisTech, des quotas sont imposés aux élèves, qui sont limités à :
> - 200 Mo en volume et 8 000 fichiers pour les élèves de première année ;
> - 300 Mo en volume et 8 000 fichiers pour les élèves de deuxième année ;
> - 500 Mo en volume et 8 000 fichiers pour les élèves de troisième année et de mastère spécialisé.

La commande `quota` permet d'afficher ses quotas :

```
1  Disk quotas for user babafou (uid 1664):
2       Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
3  se2.ensta.fr:/home
4                    60296  295000  300000            1029    8000    9000
```

On y retrouve un habituel affichage en colonnes, dont la ligne 2 indique la signification de chaque colonne des lignes suivantes.

Les lignes 3 et 4 de l'exemple constituent en fait la même ligne logique, qui a été scindée en deux en raison de la largeur de la première colonne.

Chaque ligne à partir de la ligne 3 indique les quotas de l'utilisateur sur les disques locaux et sur les serveurs de fichiers sur lesquels il dispose d'un espace de stockage. Dans l'exemple, l'utilisateur ne dispose que d'un seul espace de stockage.

La colonne `Filesystem` indique l'espace de stockage concerné. Les disques locaux sont indiqués sous la forme d'un chemin d'accès absolu commençant par `/dev` (il s'agit du fichier spécial permettant d'accéder au disque) et les serveurs de fichiers sont indiqués par leur nom suivi de deux-points puis du chemin d'accès absolu du répertoire contenant les espaces de travail des utilisateurs (c'est le cas dans notre exemple).

Les quatre colonnes suivantes concernent le quota sur le volume occupé par les fichiers.

La colonne `blocks` indique le volume total occupé, en kilooctets.

La première colonne `quota` indique le quota sur le volume alloué à l'utilisateur, en kilooctets.

La première colonne `limit` indique la limite sur le volume allouée à l'utilisateur, en kilooctets.

La différence entre quota et limite est la suivante. L'utilisateur ne peut en aucun cas dépasser la limite. Ce contrôle est effectué à chaque écriture de fichier par le système d'exploitation. En revanche, il peut dépasser le quota (mais en restant sous la limite) à condition qu'il repasse dessous en moins d'une semaine. Faute de le faire, l'utilisateur ne pourra alors ni modifier ses fichiers existants ni créer de nouveaux fichiers et devra impérativement repasser sous le quota. Afin d'éviter ce désagrément, il convient donc de vérifier régulièrement l'état de ses quotas et de ne pas tarder à supprimer les fichiers inutiles ou à archiver (en comprimant l'archive) ceux qui n'ont plus d'utilité immédiate.

La première colonne `grace` indique, dans le cas où l'utilisateur a dépassé le quota sur le volume, le temps qu'il lui reste pour repasser dessous.

Les quatre colonnes suivantes concernent le quota sur le nombre de fichiers.

La colonne `files` indique le nombre de fichiers.

La seconde colonne `quota` indique le quota sur le nombre de fichiers alloué à l'utilisateur.

La seconde colonne `limit` indique la limite sur le nombre de fichiers allouée à l'utilisateur.

La signification du quota et de la limite sur le nombre de fichiers est la même que pour le quota et la limite sur le volume.

La seconde colonne `grace` indique, dans le cas où l'utilisateur a dépassé le quota en nombre de fichiers, le temps qu'il lui reste pour repasser dessous.

http://en.wikipedia.org/wiki/Disk_quota

8.7.1.3 La commande `du`

La commande `du` (pour *disk usage*) permet d'afficher le volume occupé par l'ensemble du contenu d'un répertoire et de chacun de ses sous-répertoires.

Selon l'UNIX utilisé, les résultats sont affichés soit en demi-kilooctets soit en kilooctets. Afin d'éviter de se poser des questions et d'avoir des résultats plus parlants, il est préférable d'utiliser la commande `du` avec l'option `-k`, qui permet l'affichage des résultats en kilooctets.

Pour les exemples qui suivent, nous supposerons être dans un répertoire courant ayant la structure représentée dans la figure 8.12. Les répertoires indiqués contiennent quelques fichiers.

FIGURE 8.12 – *Arborescence utilisée pour illustrer la commande* `du`.

Utilisée sans argument, la commande `du` affiche le volume occupé par chacun des sous-répertoires du répertoire courant puis par le répertoire courant lui-même :

```
1 $ du -k
2 546     ./rep1
3 2866    ./rep2
4 338     ./rep3/rep4
5 34      ./rep3/rep5
6 390     ./rep3
7 7294    .
```

Pour un répertoire donné, le volume indiqué en première colonne est le volume total occupé par l'ensemble du contenu du répertoire, fichiers et sous-répertoires compris.

Ainsi, dans l'exemple précédent, les lignes 2 à 5 indiquent les volumes occupés par les fichiers contenus respectivement dans les répertoires `rep1`, `rep2`, `rep3/rep4` et `rep3/rep5`, qui n'ont aucun sous-répertoire. La ligne 6 indique le volume total occupé par le répertoire `rep3`, à savoir le volume occupé par les fichiers qu'il contient directement ainsi que ceux des répertoires `rep3/rep4` et `rep3/rep5`. la ligne 7 indique le volume total occupé par le répertoire courant, ses fichiers, tous ses sous-répertoires et leurs fichiers.

Utilisée avec un ou plusieurs noms de répertoires en arguments, la commande `du` affiche, pour chaque répertoire en argument, le volume occupé par chacun de ses sous-répertoires puis par le répertoire lui-même :

```
$ du -k rep1 rep3
546     rep1
338     rep3/rep4
34      rep3/rep5
390     rep3
```

Si le volume occupé par les sous-répertoire n'a pas d'intérêt et qu'on ne souhaite afficher que le volume total, on peut utiliser la commande `du` avec l'option `-s` (pour *sum*) :

```
$ du -ks
7294    .
$ du -ks rep1 rep3
546     rep1
390     rep3
```

La commande `du` ne permet pas d'elle-même le tri de son affichage. Il serait pourtant utile que les répertoires puissent être triés selon le volume total occupé par chacun d'eux. Cela est possible en raccordant la commande `du` au moyen d'un tuyau à la commande `sort` (voir le paragraphe 9.5.5 page 216).

http://fr.wikipedia.org/wiki/Du_(Unix)

8.7.1.4 La commande `df`

La commande `df` (pour *disk free*) permet d'afficher l'occupation des disques locaux et des serveurs de fichiers constituant le système de fichiers.

Selon l'UNIX utilisé, les résultats sont affichés soit en demi-kilooctets soit en kilooctets. Afin d'éviter de se poser des questions et d'avoir des résultats plus parlants, il est préférable d'utiliser la commande `df` avec l'option `-k`, qui permet l'affichage des résultats en kilooctets :

```
$ df -k
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/sda3             73590752  44378456  25413728  64% /
/dev/sda1               101086     20293     75574  22% /boot
tmpfs                   516664         0    516664   0% /dev/shm
se2.ensta.fr:/home   1117077480 125568680 933849112  12% /auto/se2
se4.ensta.fr:/opt    1805969176 392989256 1321241944  23% /auto/se4
```

Malheureusement, les capacités actuelles des serveurs de fichiers, qui se comptent au minimum en téraoctets, font que les nombres sont illisibles tellement ils sont importants et que l'affichage en colonnes s'en retrouve décalé. De nombreux UNIX fournissent donc une commande `df` avec une option `-h` (pour *human-readable*), qui permet d'afficher des résultats plus clairs :

```
1 $ df -h
2 Filesystem            Size  Used Avail Use% Mounted on
3 /dev/sda3              71G   43G   25G  64% /
4 /dev/sda1              99M   20M   74M  22% /boot
5 tmpfs                 505M     0  505M   0% /dev/shm
6 se2.ensta.fr:/home    1.1T  120G  891G  12% /auto/se2
7 se4.ensta.fr:/opt     1.7T  375G  1.3T  23% /auto/se4
```

Les tailles indiquées sont suivies d'un `M` s'il s'agit de mégaoctets, d'un `G` s'il s'agit de gigaoctets et d'un `T` s'il s'agit de téraoctets.

On retrouve un habituel affichage en colonnes, dont la ligne 2 indique la signification de chaque colonne des lignes suivantes.

Chaque ligne à partir de la ligne 3 indique l'occupation d'un disque local ou d'un serveur de fichiers constituant le système de fichiers.

La colonne `Filesystem` indique le disque local ou le serveur de fichiers concerné. Les disques locaux sont indiqués sous la forme d'un chemin d'accès absolu commençant par `/dev` (il s'agit du fichier spécial permettant d'accéder au disque), ce qui est le cas des lignes 3 et 4 de l'exemple précédent, et les serveurs de fichiers sont indiqués par leur nom suivi de deux-points puis du chemin d'accès absolu du répertoire qu'ils partagent, ce qui est le cas des lignes 6 et 7 de l'exemple précédent. La ligne 5, quant à elle, correspond à un espace de stockage situé en mémoire vive, c'est une particularité de certains UNIX (Linux dans notre exemple).

La colonne `Size` indique la taille du disque local ou du serveur de fichiers.

La colonne `Used` indique l'espace occupé sur le disque local ou le serveur de fichiers.

La colonne `Avail` indique l'espace libre sur le disque local ou le serveur de fichiers.

La colonne `Use%` indique le taux d'occupation du disque local ou du serveur de fichiers.

La colonne `Mounted on` indique où se situe le disque local ou le serveur de fichiers dans l'arborescence du système de fichiers (c'est ce qu'on appelle le *point de montage*).

Plutôt utilisée par l'administrateur du système, la commande `df` peut néanmoins se révéler utile pour un utilisateur. Par exemple, si l'on dispose d'un espace de stockage sans quotas et qu'on doit générer ou transférer un gros volume de données, il est préférable de vérifier que l'espace libre est suffisant avant de procéder à l'opération.

http://fr.wikipedia.org/wiki/Df_(Unix)

8.7.2 Compression

La *compression* consiste à transformer, au moyen d'un algorithme, une suite de données (par exemple le contenu d'un fichier) en une autre suite de données plus courte mais contenant les mêmes informations. La *décompression* est l'opération inverse.

Sans rentrer dans les détails, il faut savoir qu'il existe deux types de compressions :

- la compression dite *sans perte* lorsque la décompression restitue exactement les données d'origine ;
- la compression dite *avec pertes* lorsque la décompression aboutit à des données légèrement différentes des données d'origine.

La compression avec pertes ne peut s'appliquer qu'à certains types de données bien particuliers. Elle est par exemple utilisée par le format d'image JPEG. Très grossièrement, dans une image JPEG, une suite de pixels de couleurs proches, dont les différences sont indiscernables par l'œil humain, sera codée sous la forme du nombre de ces pixels suivi de leur couleur moyenne. Le stockage des grandes zones homogènes des images en est donc considérablement réduit. La compression avec pertes est également utilisée par le format audio MP3 dans lequel la précision du codage de certains sons peu audibles est réduite.

En revanche, nous n'utiliserons pour notre part que de la compression sans perte afin de pouvoir restaurer les données d'origine sans aucune altération.

Très schématiquement, la compression sans perte consiste à découper les données à comprimer en blocs. Chaque bloc est analysé de façon à identifier les redondances qu'il contient. Ces redondances sont alors stockées une seule fois, en indiquant à quels endroits du bloc elles se trouvent. Les fichiers contenant de nombreuses redondances (typiquement, les fichiers texte) seront donc plus comprimés que des fichiers en contenant moins.

Les fichiers contenant peu de redondances n'ont donc pas intérêt à être comprimés de la sorte. C'est en particulier le cas pour les fichiers déjà comprimés d'une autre façon, comme les images JPEG ou les fichiers audio MP3.

Nous allons maintenant étudier plusieurs commandes permettant la compression et la décompression de fichiers, dans l'ordre croissant de leur efficacité (qui correspond également à leur ordre chronologique). Chaque commande de compression sera appliquée à un fichier texte témoin de 100 ko afin de juger de son taux de compression.

> http://fr.wikipedia.org/wiki/Compression_de_données
> http://fr.wikipedia.org/wiki/JPEG
> http://fr.wikipedia.org/wiki/MPEG-1/2_Audio_Layer_3

8.7.2.1 Les commandes `compress` et `uncompress`

La plus ancienne des commandes de compression est `compress`. C'est aussi la moins efficace. Elle prend un ou plusieurs noms de fichiers en arguments. Chacun de ces fichiers sera comprimé et remplacé par un fichier du même nom auquel sera rajoutée l'extension `.Z` (en majuscule, on reconnaît à cette extension un fichier comprimé avec `compress`) :

```
$ ll toto.txt
-rw-r--r--  1 babafou prof 102400 Dec  8 14:30 toto.txt
$ compress toto.txt
$ ll toto.txt.Z
-rw-r--r--  1 babafou prof  45513 Dec  8 14:30 toto.txt.Z
```

Les droits d'accès ainsi que la date de dernière modification du fichier d'origine sont conservés dans le fichier comprimé.

Sur le fichier témoin, `compress` génère un fichier comprimé tout de même plus de deux fois plus petit.

Pour décomprimer un ou plusieurs fichiers comprimés avec `compress`, on utilise la commande `uncompress` :

```
$ uncompress toto.txt.Z
$ ll toto.txt
-rw-r--r--  1 babafou prof 102400 Dec  8 14:30 toto.txt
```

Les droits d'accès ainsi que la date de dernière modification du fichier d'origine sont restaurés dans le fichier décomprimé.

> http://fr.wikipedia.org/wiki/Compress

8.7.2.2 Les commandes `gzip` et `gunzip`

La plus répandue des commandes de compression est `gzip`. Elle prend un ou plusieurs noms de fichiers en arguments. Chacun de ces fichiers sera comprimé

et remplacé par un fichier du même nom auquel sera rajoutée l'extension `.gz` (on reconnaît à cette extension un fichier comprimé avec `gzip`) :

```
$ ll toto.txt
-rw-r--r--  1 babafou prof 102400 Dec  8 14:30 toto.txt
$ gzip toto.txt
$ ll toto.txt.gz
-rw-r--r--  1 babafou prof  32188 Dec  8 14:30 toto.txt.gz
```

Les droits d'accès ainsi que la date de dernière modification du fichier d'origine sont conservés dans le fichier comprimé.

Sur le fichier témoin, `gzip` génère un fichier comprimé de moins du tiers de sa taille d'origine.

Il est possible de contrôler le niveau de compression de `gzip` au moyen des options `-1` à `-9`. L'option `-1` donne le temps d'exécution le plus rapide au prix d'une compression moins efficace. L'option `-9` donne la compression la plus efficace au prix d'un temps d'exécution plus important. Les autres options permettent d'obtenir les niveaux de compression intermédiaires. Le niveau par défaut (sans option) correspond à l'option `-6`. Sur notre fichier témoin, avec l'option `-9`, `gzip` génère effectivement un fichier plus petit :

```
$ ll toto.txt
-rw-r--r--  1 babafou prof 102400 Dec  8 14:30 toto.txt
$ gzip -9 toto.txt
$ ll toto.txt.gz
-rw-r--r--  1 babafou prof  32143 Dec  8 14:30 toto.txt.gz
```

Compte tenu des performances des ordinateurs actuels, on peut utiliser systématiquement l'option `-9`.

Pour décomprimer un ou plusieurs fichiers comprimés avec `gzip`, on utilise la commande `gunzip` :

```
$ gunzip toto.txt.gz
$ ll toto.txt
-rw-r--r--  1 babafou prof 102400 Dec  8 14:30 toto.txt
```

Les droits d'accès ainsi que la date de dernière modification du fichier d'origine sont restaurés dans le fichier décomprimé.

> http://www.gzip.org/
> http://fr.wikipedia.org/wiki/Gzip

8.7.2.3 Les commandes `bzip2` et `bunzip2`

Une autre commande de compression très répandue est `bzip2`. Elle prend un ou plusieurs noms de fichiers en arguments. Chacun de ces fichiers sera comprimé et remplacé par un fichier du même nom auquel sera rajoutée l'extension `.bz2` (on reconnaît à cette extension un fichier comprimé avec `bzip2`) :

```
$ ll toto.txt
-rw-r--r--  1 babafou prof 102400 Dec  8 14:30 toto.txt
$ bzip2 toto.txt
$ ll toto.txt.bz2
-rw-r--r--  1 babafou prof  29289 Dec  8 14:30 toto.txt.bz2
```

Les droits d'accès ainsi que la date de dernière modification du fichier d'origine sont conservés dans le fichier comprimé.

Sur le fichier témoin, `bzip2` génère un fichier comprimé un peu plus petit que ce que donne `gzip`.

Pour décomprimer un ou plusieurs fichiers comprimés avec `bzip2`, on utilise la commande `bunzip2` :

```
$ bunzip2 toto.txt.bz2
$ ll toto.txt
-rw-r--r--  1 babafou prof 102400 Dec  8 14:30 toto.txt
```

Les droits d'accès ainsi que la date de dernière modification du fichier d'origine sont restaurés dans le fichier décomprimé.

> http://www.bzip.org/
> http://fr.wikipedia.org/wiki/Bzip2

8.7.2.4 Les commandes `lzma` et `unlzma`

Une autre commande de compression est `lzma`. Elle prend un ou plusieurs noms de fichiers en arguments. Chacun de ces fichiers sera comprimé et remplacé par un fichier du même nom auquel sera rajoutée l'extension `.lzma` (on reconnaît à cette extension un fichier comprimé avec `lzma`) :

```
$ ll toto.txt
-rw-r--r--  1 babafou prof 102400 Dec  8 14:30 toto.txt
$ lzma toto.txt
$ ll toto.txt.lzma
-rw-r--r--  1 babafou prof  29251 Dec  8 14:30 toto.txt.lzma
```

Les droits d'accès ainsi que la date de dernière modification du fichier d'origine sont conservés dans le fichier comprimé.

Sur le fichier témoin, `lzma` génère un fichier comprimé de taille équivalente à ce que donne `bzip2`.

Pour décomprimer un ou plusieurs fichiers comprimés avec `lzma`, on utilise la commande `unlzma` :

```
$ unlzma toto.txt.lzma
$ ll toto.txt
-rw-r--r--  1 babafou prof 102400 Dec  8 14:30 toto.txt
```

Les droits d'accès ainsi que la date de dernière modification du fichier d'origine sont restaurés dans le fichier décomprimé.

> http://tukaani.org/lzma/
> http://fr.wikipedia.org/wiki/LZMA

8.7.2.5 Les commandes `xz` et `unxz`

> Les commandes `xz` et `unxz` ne sont pas encore disponibles dans l'environnement informatique de l'ENSTA ParisTech.

La plus récente des commandes de compression, qui utilise une évolution de l'algorithme de `lzma`, est `xz`. Elle prend un ou plusieurs noms de fichiers en arguments. Chacun de ces fichiers sera comprimé et remplacé par un fichier du même

nom auquel sera rajoutée l'extension `.xz` (on reconnaît à cette extension un fichier comprimé avec `xz`) :

```
$ ll toto.txt
-rw-r--r--  1 babafou prof 102400 Dec  8 14:30 toto.txt
$ xz toto.txt
$ ll toto.txt.xz
-rw-r--r--  1 babafou prof  29296 Dec  8 14:30 toto.txt.xz
```

Les droits d'accès ainsi que la date de dernière modification du fichier d'origine sont conservés dans le fichier comprimé.

Sur le fichier témoin, `xz` génère un fichier comprimé de taille équivalente à ce que donnent `lzma` et `bzip2`.

Pour décomprimer un ou plusieurs fichiers comprimés avec `xz`, on utilise la commande `unxz` :

```
$ unxz toto.txt.xz
$ ll toto.txt
-rw-r--r--  1 babafou prof 102400 Dec  8 14:30 toto.txt
```

Les droits d'accès ainsi que la date de dernière modification du fichier d'origine sont restaurés dans le fichier décomprimé.

> http://tukaani.org/xz/
> http://fr.wikipedia.org/wiki/XZ_(format_de_fichier)

8.7.3 Archivage

L'*archivage* consiste à rassembler une arborescence de répertoires et de fichiers dans un unique fichier, qu'on appelle une *archive*, et qui contient toutes les informations nécessaires à la recréation de l'arborescence d'origine.

Lorsqu'un ensemble de fichiers n'a plus d'utilité immédiate, on peut générer une archive permettant de recréer ces fichiers le jour venu. Afin d'occuper le moins de volume de stockage possible, cette archive est souvent comprimée, sauf si elle contient un grand nombre de fichiers déjà comprimés, comme des images JPEG, des fichiers audio MP3 ou des fichiers comprimés avec les commandes qui viennent d'être étudiées.

Une archive (comprimée au besoin) peut ainsi facilement être stockée sur un support externe (clé USB, bande magnétique, etc.) ou transférée grâce au réseau informatique.

http://en.wikipedia.org/wiki/List_of_archive_formats

8.7.3.1 La commande `tar`

Sous UNIX, la commande de prédilection pour manipuler des archives est la commande `tar` (pour *tape archiver*). Comme son nom l'indique, elle trouve son origine dans l'archivage sur bande magnétique mais elle peut tout aussi bien gérer des archives sous forme de fichiers.

La commande `tar` s'utilise obligatoirement avec au moins une option. Dans ces conditions, l'utilisation du tiret pour introduire les options n'est pas nécessaire.

http://fr.wikipedia.org/wiki/Tar_(informatique)

8.7.3.1.1 Créer une archive

Pour créer une archive d'un répertoire nommé `rep` et contenant trois fichiers, `toto1`, `toto2` et `toto3`, on utilise :

```
$ tar cvf rep.tar rep
rep/
rep/toto1
rep/toto2
rep/toto3
```

Cette commande crée une archive `rep.tar` du répertoire `rep`. Il est d'usage de nommer les archives du même nom que le répertoire à archiver, suivi de l'extension `.tar`.

La signification des options de la commande `tar` est la suivante :

- l'option `-c` (pour *create*) permet de créer une archive ;
- l'option `-v` (pour *verbose*) permet d'afficher les chemins d'accès relatifs des fichiers archivés ;
- l'option `-f` (pour *file*) permet de créer une archive dans le fichier dont le nom suit plutôt que sur bande magnétique.

L'option `-v` provoque l'affichage des noms des répertoires et fichiers au fur et à mesure qu'ils sont intégrés dans l'archive. Cette option n'est pas indispensable mais elle permet de visualiser l'avancement de la commande. L'archivage de répertoires volumineux peut en effet être long et il est toujours utile de voir que les choses progressent.

L'ordre dans lequel on indique les options de la commande `tar` est sans importance.

Dans l'exemple précédent, nous avons archivé un unique répertoire et son contenu. La commande `tar` permet d'archiver des fichiers seuls, des répertoires et toute combinaison des deux :

```
$ tar cvf archive.tar toto1 toto2 rep1 rep2
[...]
```

Néanmoins, il est préférable de s'en tenir à l'archivage d'un unique répertoire (quitte à regrouper dedans ce qu'on souhaite archiver). En effet, ceci permet de nommer l'archive selon le nom du répertoire archivé en lui rajoutant l'extension `.tar`, comme nous venons de le voir. Ceci permet également d'assurer la personne qui va extraire cette archive que cette opération va créer un unique répertoire dont on connaît déjà le nom (puisqu'il s'agit du nom de l'archive, sans l'extension `.tar`). Il est en effet particulièrement désagréable d'extraire une archive contenant des dizaines de fichiers sans qu'ils soient contenus dans un répertoire, tant et si bien qu'ils vont polluer le répertoire courant et, dans le pire des cas, écraser des fichiers préexistants ayant des noms identiques à ceux de certains fichiers contenus dans l'archive.

La commande `tar` laisse les répertoires et fichiers archivés en place. Une fois l'archive créée, on peut donc supprimer le répertoire ayant été archivé :

```
$ rm -r rep
```

8.7.3.1.2 Extraire une archive

Pour extraire une archive, c'est-à-dire recréer l'arborescence d'origine, on utilise :

```
$ tar xvf rep.tar
rep/
rep/toto1
rep/toto2
rep/toto3
```

L'option `-x` (pour *extract*) de la commande `tar` permet d'extraire l'archive. Les autres options ont la même signification que pour la création d'une archive.

Si l'archive n'a plus d'utilité, on peut alors la supprimer.

8.7.3.1.3 Afficher le contenu d'une archive

Il peut être utile d'afficher le contenu d'une archive sans pour autant l'extraire. Pour cela, on utilise l'option `-t` (pour *list*) de la commande `tar` :

```
$ tar tf rep.tar
rep/
rep/toto1
rep/toto2
rep/toto3
```

8.7.3.1.4 Compression et décompression d'une archive

Afin d'occuper le moins de place possible, il est intéressant de comprimer une archive nouvellement créée :

```
$ tar cvf rep.tar rep
[...]
$ bzip2 rep.tar
```

L'utilisation de la commande `bzip2` va donc générer un fichier `rep.tar.bz2`.

Pour décomprimer le fichier et extraire l'archive, on procède dans l'ordre inverse :

```
$ bunzip2 rep.tar.bz2
$ tar xvf rep.tar
[...]
```

On peut bien entendu utiliser les commandes de compression et de décompression de son choix.

Néanmoins, comme l'utilisation de la compression en conjonction avec l'archivage est quasiment systématique, les versions modernes de la commande `tar` savent la gérer directement, au moins pour les commandes `compress`, `gzip`, `bzip2` et les commandes de décompression associées. Pour les autres commandes, il faut procéder en deux étapes comme indiqué précédemment.

Ainsi les options suivantes de la commande `tar` permettent d'utiliser automatiquement une commande de compression (si elles sont utilisées avec l'option `-c`) ou de décompression (si elles sont utilisées avec les options `-x` ou `-t`) :

- l'option `-Z` (majuscule) pour `compress` et `uncompress` ;
- l'option `-z` (minuscule) pour `gzip` et `gunzip` ;
- l'option `-j` pour `bzip2` et `bunzip2`.

Par exemple :

```
$ tar cvjf rep.tar.bz2 rep
[...]
$ tar xvjf rep.tar.bz2
[...]
```

Lors de la création de l'archive, il est nécessaire d'indiquer l'extension correcte (`.Z` pour l'option `-Z`, `.gz` pour l'option `-z` et `.bz2` pour l'option `-j`) de l'archive comprimée.

8.7.4 Échanges avec d'autres systèmes d'exploitation

Si, sous UNIX, l'utilisation de la commande `tar` pour l'archivage, en conjonction avec une commande de compression, est incontournable, d'autres systèmes d'exploitation utilisent des logiciels qui leur sont propres.

Ces logiciels réalisent généralement ensemble les fonctions d'archivage et de compression, ce qui est probablement plus simple pour l'utilisateur novice mais empêche de choisir l'algorithme de compression.

8.7.4.1 Les commandes `zip` et `unzip`

Le format ZIP est le format natif d'archive comprimée sous Windows. De ce fait, il est très répandu. Les fichiers ZIP se reconnaissent à leur extension `.zip`

(éventuellement en majuscules, Windows ne faisant pas la différence entre majuscules et minuscules).

L'algorithme de compression utilisé par le format ZIP est le même que celui utilisé par la commande `gzip`. Une archive ZIP aura donc quasiment la même taille qu'une archive créée avec `tar` puis comprimée avec `gzip`.

La commande `zip`, utilisée avec l'option `-r` permet de créer une archive :

```
$ zip -r rep.zip rep
  adding: rep/ (stored 0%)
  adding: rep/toto1 (deflated 72%)
  adding: rep/toto2 (deflated 72%)
  adding: rep/toto3 (deflated 72%)
```

La commande `unzip` permet d'extraire une archive :

```
$ unzip rep.zip
Archive:  rep.zip
   creating: rep/
  inflating: rep/toto1
  inflating: rep/toto2
  inflating: rep/toto3
```

L'option `-l` de la commande `unzip` permet d'afficher le contenu d'une archive sans l'extraire :

```
$ unzip -l rep.zip
Archive:  rep.zip
  Length     Date   Time    Name
 --------    ----   ----    ----
        0  12-10-09 18:39   rep/
   420211  12-10-09 18:41   rep/toto1
   420211  12-10-09 18:41   rep/toto2
   420211  12-10-09 18:41   rep/toto3
 --------                   -------
  1260633                   4 files
```

http://fr.wikipedia.org/wiki/ZIP_(format_de_fichier)

8.7.4.2 La commande `unrar`

Le format RAR, quoique moins utilisé que le format ZIP, est pourtant relativement répandu sous Windows. Les fichiers RAR se reconnaissent à leur extension `.rar` (éventuellement en majuscules, Windows ne faisant pas la différence entre majuscules et minuscules).

La création d'archives au format RAR n'est possible qu'avec des logiciels commerciaux sous licence de l'auteur du format.

En revanche, l'extraction d'archives au format RAR est possible sous UNIX au moyen de la commande `unrar`.

La commande `unrar`, utilisée avec `e` (pour *extract*) comme argument, suivi du nom du fichier d'archive, permet d'extraire celle-ci :

```
$ unrar e rep.rar

UNRAR 3.90 beta 4 freeware      Copyright (c) 1993-2009 Alexander Roshal


Extracting from rep.rar

Extracting  toto1                                                     OK
Extracting  toto2                                                     OK
Extracting  toto3                                                     OK
All OK
```

La commande `unrar`, utilisée avec `l` (pour *list*) comme argument, suivi du nom du fichier d'archive, permet d'en afficher le contenu :

```
$ unrar l rep.rar

UNRAR 3.90 beta 4 freeware      Copyright (c) 1993-2009 Alexander Roshal

Archive rep.rar

 Name             Size   Packed Ratio  Date   Time     Attr      CRC   Meth Ver
-------------------------------------------------------------------------------
 toto1          421361   118036  28% 10-12-09 19:58  .....A.   496B92BD m3g 2.9
 toto2          421361   118036  28% 10-12-09 19:58  .....A.   496B92BD m3g 2.9
 toto3          421361   118036  28% 10-12-09 19:58  .....A.   496B92BD m3g 2.9
 rep                 0        0   0% 10-12-09 19:58  .D.....   00000000 m0  2.0
-------------------------------------------------------------------------------
    4          1264083   354108  28%
```

http://fr.wikipedia.org/wiki/RAR_(format_de_fichier)

9

Commandes, processus et redirections

LA FONCTION PREMIÈRE d'un ordinateur n'est pas le stockage des fichiers, qui est d'ailleurs réalisé par des périphériques, mais bel et bien l'exécution de logiciels qui, elle, est réalisée par le cerveau de l'ordinateur, c'est-à-dire le microprocesseur. À cet effet, l'interpréteur de commandes permet de contrôler le lancement, l'exécution et l'arrêt des logiciels. Il offre également des outils permettant d'en conserver les résultats et de faire dialoguer les logiciels entre eux.

9.1 L'historique des commandes

Comme nous l'avons vu au paragraphe 7.2.6 page 95, il est possible d'utiliser les touches fléchées pour naviguer dans l'*historique* des commandes. Il est également possible de l'utiliser au travers d'un ensemble d'expressions.

http://en.wikipedia.org/wiki/Command_history

9.1.1 La commande `history`

L'historique des commandes peut être affiché grâce à la commande `history` :

```
$ history
    1  date
    2  ls -l toto.txt
    3  chmod 644 toto.txt
    4  cp tata.txt tutu.txt
    5  history
```

Les commandes sont affichées dans l'ordre chronologique et sont précédées de leur numéro.

Certains interpréteurs de commandes (C shell, TENEX C shell et Bourne-again shell) indiquent la commande `history` qui vient d'être exécutée comme dernière commande de l'historique (c'est le cas dans l'exemple précédent) alors que d'autres (Korn shell et Z shell) s'arrêtent à la commande précédente. La commande `history` n'existe pas en Bourne shell.

Lorsque l'historique est un peu long et qu'on ne souhaite afficher que les dernières commandes, on en indique le nombre en argument de la commande `history` :

```
$ history 3
    3  chmod 644 toto.txt
    4  cp tata.txt tutu.txt
    5  history 3
```

L'historique a une taille finie. Il contient généralement quelques dizaines ou quelques centaines de commandes.

http://en.wikipedia.org/wiki/History_(Unix)

9.1.2 Au sujet des exemples qui suivent

Afin de conserver le même référentiel et d'éviter de rappeler systématiquement l'historique ou de le reconstruire mentalement, les exemples qui suivent seront supposés être saisis immédiatement après cet historique, c'est-à-dire à l'endroit où se trouve le curseur :

```
$ history
    1  date
    2  ls -l toto.txt
    3  chmod 644 toto.txt
    4  cp tata.txt tutu.txt
    5  history
$ ■
```

9.1.3 Exécuter à nouveau une commande dans son intégralité

Il est possible d'exécuter à nouveau une commande en saisissant un point d'exclamation suivi de son numéro dans l'historique. Ainsi, la commande `ls` de l'exemple précédent peut être exécutée à nouveau en tapant `!2` :

```
$ !2
ls -l toto.txt
-rw-r--r--  1 babafou prof 554923 Mar  1 11:44 toto.txt
```

L'interpréteur de commandes affiche la commande qu'il exécute avant de le faire.

On peut obtenir le même résultat en faisant suivre le point d'exclamation d'un nombre négatif indiquant la commande à exécuter. Ici, la commande `ls` est la quatrième commande en remontant dans l'historique donc on peut taper `!-4` :

```
$ !-4
ls -l toto.txt
-rw-r--r--  1 babafou prof 554923 Mar  1 11:44 toto.txt
```

La commande précédente peut être exécutée à nouveau en tapant simplement `!!`, qui est donc l'équivalent de `!-1`.

On peut également exécuter à nouveau la dernière commande commençant par une chaîne de caractères en saisissant un point d'exclamation suivi de cette chaîne de caractères. Ainsi, `!da` exécutera à nouveau la dernière commande commençant par `da` :

```
$ !da
date
Mon Mar  1 10:47:01 CET 2010
```

De même, on peut exécuter à nouveau la dernière commande contenant une chaîne de caractères en saisissant un point d'exclamation suivi d'un point d'interrogation et de cette chaîne de caractères. Ainsi, `!?tot` exécutera à nouveau la dernière commande contenant `tot` (dans son nom, ses options ou ses arguments) :

```
$ !?tot
ls -l toto.txt
-rw-r--r--  1 babafou prof 554923 Mar  1 11:44 toto.txt
```

9.1.4 Utiliser certains mots d'une commande précédente

Plutôt que d'exécuter à nouveau une commande précédente dans sa totalité, on peut également en extraire certains mots en utilisant les expressions que nous venons d'étudier suivies d'un suffixe introduit par un deux-points.

Ainsi, deux-points suivi d'un nombre désigne le mot correspondant (en commençant par `0`).

```
$ ls -l !4:1
ls -l tata.txt
-rw-r--r--  1 babafou prof 556643 Mar  2 10:32 tata.txt
```

Ici, l'expression `!4:1` fait référence au deuxième (puisqu'on commence à zéro) mot de la quatrième commande de l'historique.

On peut également indiquer un intervalle et utiliser des éléments provenant de différentes commandes de l'historique :

```
$ !ls:0-1 !4:1
ls -l tata.txt
-rw-r--r--  1 babafou prof 556643 Mar  2 10:32 tata.txt
```

Ici, l'expression `!ls:0-1` fait référence aux premier et deuxième mots de la dernière commande commençant par `ls`.

Le caractère `$` peut être utilisé dans un intervalle pour désigner le dernier mot ou être utilisé seul :

```
$ rm !ls:$
rm toto.txt
```

Pour extraire un ou plusieurs mots d'une commande contenant une chaîne de caractères, il faut ajouter un point d'interrogation immédiatement avant le deux-points du suffixe :

```
$ rm !?tot?:2
rm toto.txt
```

Pour réutiliser un mot de la commande précédente, on peut utiliser plus simplement `!:0`, `!:1`, etc., jusqu'à `!:$` qu'on peut simplifier en `!$`.

Il est assez courant d'exécuter à la suite plusieurs commandes ayant le même argument. C'est là que `!$` se révèle très utile :

```
$ ls -l toto.txt
-rw-r--r--  1 babafou prof 554923 Mar  1 11:44 toto.txt
$ rm !$
rm toto.txt
```

Enfin, il est possible de répéter la commande précédente en remplaçant une chaîne de caractères par une autre. Pour cela, on saisit un accent circonflexe suivi de la chaîne de caractères à remplacer suivie d'un autre accent circonflexe suivi de la chaîne de caractères de remplacement (le tout sans espaces, sauf, au besoin, dans les chaînes de caractères) :

```
$ ls -l toto.txt
-rw-r--r--  1 babafou prof 554923 Mar  1 11:44 toto.txt
$ ^toto^tata
ls -l tata.txt
-rw-r--r--  1 babafou prof 556643 Mar  2 10:32 tata.txt
```

9.2 Les chemins de recherche des commandes

9.2.1 Commandes externes et commandes internes

Que se passe-t-il lorsqu'on exécute une commande ? Nous avons vu au paragraphe 8.3.5 page 127 que certains répertoires (en l'occurrence `/bin`, `/sbin`, `/usr/bin`, `/usr/sbin`, `/usr/local/bin` et `/usr/local/sbin`) contiennent des fichiers exécutables dont les noms sont identiques à certaines commandes que nous avons étudiées.

Ce n'est pas une coïncidence. En effet, quasiment toutes les commandes sont des fichiers exécutables qui sont stockés quelque part dans l'arborescence du système de fichiers. Ce sont des *commandes externes*.

Par opposition, il existe quelques *commandes internes*, qui sont directement traitées par l'interpréteur de commandes. Les commandes `cd`, `exit`, `type` (que nous allons étudier plus loin) et `umask` sont des commandes internes et il en existe d'autres.

Il existe également quelques très rares commandes qui sont des commandes internes et qui existent également en tant que commandes externes. C'est par exemple le cas des commandes `kill` (que nous allons étudier au paragraphe 9.3.5 page 188), `echo` et `which` (que nous allons étudier plus loin).

Revenons à la question initiale. Que se passe-t-il lorsqu'on exécute une commande ? Si celle-ci est une commande interne, l'interpréteur de commandes reconnaît son nom et il la traite directement. Sinon, il s'agit d'une commande externe.

Si le nom de cette commande contient une barre oblique `/`, l'interpréteur de commandes essaie d'exécuter le fichier correspondant, considéré comme un chemin d'accès relatif si le nom de la commande ne commence pas par `/` ou comme un chemin d'accès absolu si le nom de la commande commence par `/`. Par exemple, si l'on tape la commande suivante :

```
$ rep/toto
```

l'interpréteur de commandes essaiera d'exécuter le fichier `toto` (qui doit donc être exécutable) situé dans le répertoire `rep`.

Sinon (c'est le cas général), l'interpréteur de commandes doit déterminer où se trouve le fichier exécutable de la commande dans l'arborescence du système de fichiers. Pour cela, il utilise la variable d'environnement `PATH`.

9.2.2 La variable d'environnement `PATH`

Une *variable d'environnement* est un paramètre de configuration de l'interpréteur de commandes, dont la valeur est fixée par l'administrateur du système ou par l'utilisateur.

On peut accéder à la valeur d'une variable d'environnement en faisant précéder son nom d'un symbole dollar `$`. Ainsi, on peut afficher le contenu d'une variable d'environnement en utilisant la commande `echo` :

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
```

La variable d'environnement `PATH` contient une liste de chemins d'accès de répertoires séparés par des deux-points. C'est ce qu'on appelle les chemins de recherche des commandes.

En effet, lors de son démarrage, l'interpréteur de commandes va parcourir dans l'ordre les répertoires indiqués dans la variable d'environnement `PATH` et conserver

en mémoire une liste des commandes (c'est-à-dire des fichiers exécutables) qu'ils contiennent.

Lorsqu'on exécute une commande externe dont le nom ne contient pas de barre oblique `/`, donc, l'interpréteur de commandes va parcourir cette liste à la recherche de la commande et va exécuter la première qu'il trouve (l'ordre des répertoires dans la variable d'environnement `PATH` a donc son importance) ou afficher un message d'erreur si la commande n'y figure pas.

> http://en.wikipedia.org/wiki/PATH_(variable)

9.2.3 Les commandes `type` et `which`

Si l'on souhaite afficher le chemin d'accès absolu d'une commande en utilisant les informations contenues dans la variable d'environnement `PATH`, on peut utiliser les commandes `type` ou `which` :

```
$ type ls
ls is /bin/ls
$ which ls
/bin/ls
```

> http://en.wikipedia.org/wiki/Type_(Unix)
> http://en.wikipedia.org/wiki/Which_(Unix)

9.2.4 Mise en garde

Il arrive fréquemment de voir le répertoire `.` (répertoire courant) dans le contenu de la variable d'environnement `PATH` :

```
$ echo $PATH
/usr/bin:/bin:.
```

Ceci peut être pratique pour les personnes qui font beaucoup de programmation. En effet, quand on programme, on va générer un fichier exécutable dans le répertoire courant. Comment faire pour l'exécuter ? Si le répertoire courant ne figure pas dans le contenu de la variable d'environnement `PATH`, il faut exécuter son programme (qu'on appellera `test`) de la sorte :

```
$ ./test
```

On est donc tenté d'ajouter `.` quelque part dans le contenu de la variable d'environnement `PATH` pour s'épargner de taper les deux caractères `./` :

```
$ test
```

Malheureusement, ceci pose deux problèmes.

Si `.` figure vers la fin du contenu de la variable d'environnement `PATH` (en tout cas, après `/bin`), on risque d'avoir une surprise. Supposons que notre programme `test` soit conçu pour afficher des informations dès son lancement :

```
$ test
$ ■
```

Rien n'est affiché... On peut passer longtemps à chercher pourquoi dans le programme parce que la source du problème n'y est pas. En effet il existe une commande `test` dans le répertoire `/bin` (quand elle n'est pas une commande interne de l'interpréteur de commandes). Comme `/bin` figure avant `.` dans le contenu de la variable d'environnement `PATH`, c'est `/bin/test` qui est exécuté quand on tape simplement `test`.

Qu'à cela ne tienne, il suffit donc de mettre `.` au début du contenu de la variable d'environnement `PATH` pour que ce soit notre `test` qui soit exécuté. Le problème inverse se posera alors parce qu'il sera impossible d'exécuter les commandes ayant le même nom qu'un de ses programmes autrement que par son chemin d'accès absolu (ce que l'utilisation de la variable d'environnement `PATH` est censée éviter).

La morale de cette histoire est double. Tout d'abord, mettre `.` dans le contenu de la variable d'environnement `PATH` pose des problèmes, où qu'il soit. On aura beau prendre garde à ne pas nommer ses programmes comme des commandes externes, il y en a tellement qu'on aura forcément un problème un jour. Ensuite, quand on programme, il est fortement recommandé d'exécuter ses programmes en faisant systématiquement précéder leur nom de `./` afin d'être toujours certain d'exécuter le bon programme :

```
$ ./mon_programme
```

9.3 Gestion des processus

UNIX étant un système d'exploitation multitâche, il est possible de faire fonctionner plusieurs logiciels en même temps. Plus précisément, on parle de *processus* (*process* en anglais) pour désigner un logiciel en cours d'exécution.

http://fr.wikipedia.org/wiki/Processus_(informatique)

9.3.1 L'ordonnancement

Mais les processus fonctionnent-ils réellement en même temps ? Un microprocesseur mono-cœur — nous discuterons des microprocesseurs multi-cœurs plus loin — ne peut exécuter qu'une seule instruction à un instant donné. Il ne peut donc exécuter qu'un seul processus à la fois. L'illusion du multitâche provient du fait que chaque processus fonctionne pendant un laps de temps donné puis laisse sa place à un autre processus et ainsi de suite. Ce laps de temps, de l'ordre de quelques dizaines de millisecondes, est suffisamment long à l'échelle du microprocesseur, qui est cadencé à plusieurs gigahertz, pour laisser au processus un temps d'exécution raisonnable et suffisamment court à notre échelle pour nous donner l'illusion d'une exécution simultanée des processus.

Plus techniquement, un processus se voit attribuer l'utilisation du microprocesseur pendant un certain temps, qu'on appelle un *quantum*, par l'*ordonnanceur* (*scheduler* en anglais), qui est le composant du système d'exploitation responsable de l'attribution du microprocesseur aux différents processus. Le mécanisme d'attribution s'appelle l'*ordonnancement* (*scheduling* en anglais). À la fin de son quantum, l'ordonnanceur retire l'utilisation du microprocesseur au processus qui en disposait, choisit un nouveau processus (qui peut parfois être le même) et lui confie l'utilisation du microprocesseur pendant un nouveau quantum (qui peut être d'une durée différente du précédent).

La figure 9.1 page suivante représente l'ordonnancement de trois processus, indiqués sur l'axe des ordonnées par les repères 1, 2 et 3. Le temps est indiqué par l'axe des abscisses.

Le processus 1 utilise le microprocesseur du temps t_0 au temps t_1. À cet instant, l'utilisation du microprocesseur lui est retirée par l'ordonnanceur, qui choisit un nouveau processus à qui confier l'utilisation du microprocesseur, à savoir le processus 2. Ceci dure un temps ε_1. Pendant ce temps, aucun processus ne s'exécute puisque l'ordonnanceur utilise le microprocesseur. L'exécution de l'ordonnanceur

FIGURE 9.1 – *Ordonnancement.*

est représentée par les deux traits interrompus verticaux, le premier en t_1 et le second en $t_1 + \varepsilon_1$ (non indiqué sur l'axe des abscisses).

Il en va de même pour le processus 2 et ainsi de suite.

Les quanta, représentés dans la figure 9.1 par les lignes horizontales figurant en face de chaque processus, ne sont pas nécessairement de la même durée. Les processus ne sont pas non plus nécessairement ordonnancés de manière régulière (1 puis 2 puis 3 puis 1 puis 2 puis 3, etc.). Tout ceci dépend de l'algorithme de choix de l'ordonnanceur et de la priorité des processus (voir le paragraphe 9.6.2 page 222).

Un fonctionnement multitâche où l'ordonnanceur retire l'utilisation du microprocesseur aux processus est dit *préemptif*. C'est aujourd'hui le mode de fonctionnement de tout système d'exploitation sérieux. Par le passé, certains systèmes d'exploitation (dont Windows avant Windows 95, qui n'était d'ailleurs pas totalement préemptif, et Mac OS avant Mac OS X) utilisaient un fonctionnement multitâche coopératif, dans lequel les processus devaient d'eux-mêmes rendre l'utilisation du microprocesseur à l'ordonnanceur, ce qu'ils ne faisaient pas toujours...

Avec un microprocesseur multi-cœur, plusieurs processus (autant que de cœurs) peuvent réellement s'exécuter simultanément. L'ordonnanceur utilise l'un des cœurs, préalablement retiré au processus qui l'utilisait, afin de choisir un autre processus auquel l'attribuer. Pendant ce temps, les processus utilisant les autres cœurs continuent à fonctionner.

L'ordonnancement est étudié plus en détail dans le cours de deuxième année IN201, *Systèmes d'exploitation*.

> http://fr.wikipedia.org/wiki/Ordonnancement_dans_les_systèmes_d'exploitation
>
> http://fr.wikipedia.org/wiki/Multitâche_préemptif

http://fr.wikipedia.org/wiki/Microprocesseur_multi-cœur

9.3.2 Lancement d'un processus en arrière-plan

Nous n'avons pour le moment utilisé des commandes que séquentiellement. Ceci peut d'ailleurs poser problème avec certaines d'entre elles. En effet, par défaut, l'interpréteur de commandes attend la fin de chaque commande qu'il exécute avant d'afficher une nouvelle invite. Cela est raisonnable pour la plupart des commandes que nous avons étudiées, qui se déroulent très rapidement mais d'autres peuvent durer beaucoup plus longtemps.

Par exemple, lorsqu'on lance Emacs, l'interpréteur de commandes attend qu'on ait quitté cet éditeur de texte avant d'afficher une nouvelle invite. Pendant ce temps, l'interpréteur de commandes depuis lequel Emacs a été lancé n'est pas utilisable. Cela est d'autant plus regrettable qu'Emacs ouvre sa propre fenêtre et n'a ensuite plus aucune interaction avec l'utilisateur par le biais de l'interpréteur de commandes. C'est d'ailleurs le cas de quasiment toutes les commandes ouvrant leur propre fenêtre.

Il serait plus utile de pouvoir lancer ce type de commande en indiquant à l'interpréteur de commandes qu'il peut ne pas attendre sa fin et réafficher une nouvelle invite tout de suite.

Ceci est possible grâce au caractère `&` :

```
$ emacs toto.txt &
[1] 1664
$ ■
```

Celui-ci indique à l'interpréteur de commandes qu'il doit lancer la commande en *arrière-plan* (*background* en anglais) — on parle également de *tâche de fond* —, c'est-à-dire sans attendre qu'elle se termine.

Ce caractère doit figurer en dernier sur la ligne de commande. Il n'est pas nécessaire de le faire précéder d'une espace.

L'interpréteur de commandes affiche donc tout de suite une nouvelle invite après avoir indiqué deux informations :

1. entre crochets, le numéro de tâche de la commande qui vient d'être lancée (nous ne reviendrons pas sur ce concept) ;
2. l'*identifiant de processus* (nous reviendrons sur ce concept au paragraphe 9.3.3 page 185) de la commande qui vient d'être lancée.

Il est ensuite possible d'utiliser l'interpréteur de commandes pendant qu'Emacs s'exécute en arrière-plan.

Lorsque qu'on quitte l'éditeur de texte, l'interpréteur de commandes l'indique juste avant d'afficher sa prochaine invite :

```
[1]+  Done                    emacs toto.txt
```

Ceci peut être utile lorsqu'on lance en arrière-plan, par exemple, une commande qui se contente d'effectuer des traitements (calcul scientifique, simulation, etc.) sans ouvrir sa propre fenêtre ou afficher quoi que ce soit dans le terminal. On est alors averti de la fin de cette commande.

Pour finir, que faire si l'on a lancé une commande en oubliant de le faire en arrière-plan ? On se retrouve avec un interpréteur de commande inutilisable, ce qui est dommage. On peut généralement quitter le logiciel qu'on vient de lancer pour le relancer en arrière plan mais il y a une façon plus élégante de s'en sortir.

Dans l'interpréteur de commandes, il suffit de taper `^Z` :

```
$ emacs toto.txt
^Z
[1]+  Stopped                 emacs toto.txt
$ ■
```

Cela va suspendre l'exécution du processus et provoquer l'affichage de l'invite. Il faut alors indiquer à l'interpréteur de commandes qu'il doit reprendre l'exécution du processus mais en arrière plan au moyen de la commande `bg` (pour *background*) :

```
$ bg
[1]+ emacs toto.txt &
$ ■
```

Et la boulette est oubliée !

http://fr.wikipedia.org/wiki/Bg_(Unix)

9.3.3 La commande ps

La commande ps (pour *process status*) affiche la liste des processus en activité sur l'ordinateur.

Son affichage n'est pas intéressant sans options. À cet égard, ps est une commande inhabituelle car, dans la mesure où elle n'accepte que des options et aucun argument, il n'est pas nécessaire de faire précéder ses options par le tiret que nous avons utilisé jusque là puisqu'aucune ambiguïté n'est possible. Par souci d'économie, nous n'utiliserons donc pas de tiret dans les exemples pour les options de ps.

L'option -x permet d'afficher l'ensemble de ses processus :

```
$ ps x
  PID TTY      STAT   TIME COMMAND
18230 ?        S      0:00 sshd
18231 pts/1    Ss     0:00 -tcsh
18641 pts/1    R+     0:00 ps x
```

On retrouve un habituel affichage en colonnes. Chaque ligne, hormis la première, correspond à un processus.

La signification de chaque colonne est la suivante :

1. la colonne PID indique l'*identifiant de processus* (*process identifier* en anglais), qui est un nombre unique attribué à chaque commande lors de son lancement ;
2. la colonne TTY (pour *teletype*) indique le terminal depuis lequel la commande a été lancée ;
3. la colonne STAT (pour *status*) indique l'état du processus : R (pour *runnable*) s'il est prêt à être exécuté, S (pour *sleeping*) s'il est en sommeil ; il existe d'autres états que nous ne mentionnerons pas ;
4. la colonne TIME indique la durée cumulée d'occupation du microprocesseur ;
5. la colonne COMMAND indique la commande telle qu'elle a été lancée depuis l'interpréteur de commandes ou telle qu'elle a décidé d'apparaître dans l'affichage de ps (le libellé de cette colonne est en effet paramétrable par chaque commande).

L'option -u permet d'afficher quelques informations (donc quelques colonnes) supplémentaires :

```
$ ps ux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
babafou  18230  0.0  0.1  10968  2084 ?        S    17:19   0:00 sshd
babafou  18231  1.5  0.1   8740  2728 pts/1    Ss   17:19   0:00 -tcsh
babafou  27371  0.0  0.0   8056  1888 pts/1    R+   17:28   0:00 ps ux
```

- la colonne `USER` indique le propriétaire du processus ;
- la colonne `%CPU`[1] indique le pourcentage du microprocesseur utilisé par le processus ;
- la colonne `%MEM` indique le pourcentage de mémoire utilisé par le processus ;
- la colonne `VSZ` (pour *virtual size*) indique (en kilooctets) la taille totale du processus ;
- la colonne `RSS` (pour *resident set size*) indique (en kilooctets) le volume de mémoire vive occupé par le processus ;
- la colonne `START` indique l'heure de lancement du processus.

L'option `-a` (pour *all*) permet d'afficher l'ensemble des processus en activité sur l'ordinateur, pas seulement les siens :

```
$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   2112   636 ?        Ss   00:12   0:02 init [5]
[...]
```

La colonne `COMMAND` étant tronquée pour que l'affichage des lignes ne dépasse pas 80 caractères, il est possible de forcer un affichage complet grâce à l'option `-w` (pour *wide*) :

```
$ ps auxw
[...]
```

> http://fr.wikipedia.org/wiki/Ps_(Unix)
> http://fr.wikipedia.org/wiki/Identifiant_de_processus

1. CPU signifie *central processing unit*.

9.3.4 La commande `top`

La commande `ps` donne une vision statique des processus. Il est parfois utile d'en avoir une vision plus vivante afin de suivre leur fonctionnement en temps réel.

La commande `top` affiche, outre certaines informations globales, la liste des processus les plus actifs, c'est-à-dire de ceux qui utilisent le plus le microprocesseur.

C'est particulièrement utile lorsqu'on gère un système informatique, afin de savoir quels sont les plus gros consommateurs des ressources de calcul et de détecter d'éventuels abus. Lorsqu'on met au point des programmes de calcul, il est également intéressant d'en étudier le comportement de cette façon afin de suivre leur consommation au niveau du microprocesseur et de la mémoire.

La commande `top` a un fonctionnement particulier par rapport aux commandes vues jusqu'à présent puisqu'elle prend, durant le temps de son exécution, le contrôle de tout l'émulateur de terminal (un peu comme `vi`, `more` ou `less`) et met à jour son affichage toutes les trois secondes :

```
top - 19:38:28 up 19:25,  1 user,  load average: 0.00, 0.02, 0.00
Tasks: 122 total,   1 running, 121 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.4%us,  0.9%sy,  0.0%ni, 98.4%id,  0.2%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   2040636k total,   957472k used,  1083164k free,   348932k buffers
Swap:  2048276k total,        0k used,  2048276k free,   286348k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2920 gdm       20   0 13616 4020 3320 S    2  0.2   0:29.34 at-spi-registry
    1 root      20   0  2112  636  544 S    0  0.0   0:02.47 init
    2 root      15  -5     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S    0  0.0   0:00.41 migration/0
    4 root      15  -5     0    0    0 S    0  0.0   0:03.61 ksoftirqd/0
    5 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/0
    6 root      RT  -5     0    0    0 S    0  0.0   0:00.81 migration/1
    7 root      15  -5     0    0    0 S    0  0.0   0:02.49 ksoftirqd/1
    8 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/1
    9 root      15  -5     0    0    0 S    0  0.0   0:00.26 events/0
   10 root      15  -5     0    0    0 S    0  0.0   0:00.20 events/1
   11 root      15  -5     0    0    0 S    0  0.0   0:00.00 khelper
   63 root      15  -5     0    0    0 S    0  0.0   0:00.02 kblockd/0
   64 root      15  -5     0    0    0 S    0  0.0   0:00.00 kblockd/1
   66 root      15  -5     0    0    0 S    0  0.0   0:00.00 kacpid
   67 root      15  -5     0    0    0 S    0  0.0   0:00.00 kacpi_notify
  136 root      15  -5     0    0    0 S    0  0.0   0:00.00 cqueue
```

Les cinq premières lignes donnent différents indicateurs globaux sur l'état du système d'exploitation (nous ne rentrerons pas dans les détails).

Le reste reprend un affichage en colonnes semblable à celui de `ps`. On y retrouve les colonnes les plus intéressantes : `PID`, `USER`, `%CPU`, `%MEM` et `COMMAND`. La signification de la colonne `NI` sera étudiée au paragraphe 9.6.2 page 222. Les processus sont classés par ordre décroissant d'occupation du microprocesseur (colonne `%CPU`).

Pour quitter `top`, il faut taper `q`.

> http://www.unixtop.org/
> http://en.wikipedia.org/wiki/Top_(software)

9.3.5 La commande `kill`

Le nom de cette commande est fort mal choisi. Certes elle permet de tuer un processus mais ce n'est là qu'un des aspects de son fonctionnement. De manière plus générale `kill` sert à envoyer un *signal* à un processus. Un signal est une information atomique envoyée d'un processus à un autre ou du système d'exploitation à un processus. Les signaux sont étudiés plus en détail dans le cours de deuxième année IN201, *Systèmes d'exploitation*.

La commande `kill` ne doit être utilisée pour tuer un processus qu'en dernier recours, lorsqu'on ne peut plus quitter le logiciel correspondant par les voies naturelles.

Pour les exemples, nous allons utiliser une petite application sympathique appelée `xeyes`, que nous allons lancer en arrière-plan :

```
$ xeyes &
[1] 1664
$ ■
```

Celle-ci ouvre une fenêtre, placez-là bien en vue, elle permettra, en disparaissant, de vérifier que le processus a bien été tué.

La commande `kill` prend en argument le ou les identifiants des processus à tuer. Ici, celui de `xeyes` est 1664 donc il faut faire :

```
$ kill 1664
```

Il existe un ensemble fini de signaux, dont on peut avoir la liste au moyen de l'option `-l` (pour *list*) de `kill` :

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Chaque signal a un nom et un numéro. Par défaut, le signal envoyé par `kill` est `SIGTERM` (pour *terminate*). Si le processus à éliminer est récalcitrant (certains processus peuvent résister à l'envoi du signal `SIGTERM`), il est possible d'envoyer le signal `SIGKILL`, dont on ne réchappe pas. On indique à `kill` le signal à envoyer en mettant en option son nom, sans le `SIG` initial, ou son numéro. Il s'agit d'un cas où les options sont des mots complets et pas des caractères individuels.

Donc pour envoyer le signal `SIGKILL` à un processus, il faut utiliser l'option `-KILL` (par nom) ou l'option `-9` (par numéro) :

```
$ kill -KILL 1664
$ kill -9 1664
```

> http://fr.wikipedia.org/wiki/Kill_(Unix)
> http://fr.wikipedia.org/wiki/Signal_(informatique)

9.3.6 Le statut de sortie d'un processus

Lorsqu'il se termine, un processus envoie un nombre entier au processus qui l'a créé. Ce nombre est le *statut de sortie* du processus. Il indique s'il a pu effectuer correctement la tâche qui lui incombait ou s'il a rencontré un problème. Le statut de sortie vaut :

- `0` si le processus s'est correctement déroulé ;
- une valeur non nulle (souvent 1) en cas d'erreur.

http://en.wikipedia.org/wiki/Exit_status

9.3.6.1 Les variables spéciales `?` et `status`

Une *variable spéciale* est une variable dont la valeur est fixée par l'interpréteur de commandes.

Le statut de sortie de la dernière commande est stocké dans une variable spéciale :

- `?` pour les Bourne shell (`sh`), Korn shell (`ksh`), Bourne-again shell (`bash`) et Z shell (`zsh`) (le TENEX C shell (`tcsh`) reconnaît également cette variable) ;
- `status` pour les C shell (`csh`) et TENEX C shell (`tcsh`) (le Z shell (`zsh`) reconnaît également cette variable).

Ainsi, avec `sh`, `ksh`, `bash` ou `zsh` :

```
$ ls
tata toto
$ echo $?
0
$ rm titi
rm: titi: No such file or directory
$ echo $?
1
```

Avec `csh` ou `tcsh` :

```
% ls
tata toto
% echo $status
0
% rm titi
rm: titi: No such file or directory
% echo $status
1
```

Il faut bien comprendre que les variables `?` et `status` contiennent le statut de sortie de la dernière commande. Considérons l'exemple suivant :

```
$ commande1
[...]
$ commande2
[...]
$ echo $?
0
```

Ici, le statut de sortie affiché correspond à celui de `commande2`. N'ayant pas utilisé `?` ou `status` immédiatement après la fin de `commande1`, son statut de sortie est perdu.

9.3.6.2 Succession de commandes

Il est possible d'utiliser le statut de sortie d'une commande pour exécuter une autre commande sous condition.

Considérons l'opérateur `&&` :

```
$ commande1 && commande2
```

La commande `commande2` ne sera exécutée que si le statut de sortie de la commande `commande1` est nul, c'est-à-dire si elle s'est correctement terminée. Donc, si `commande1` se termine correctement, `commande2` sera exécutée. Si `commande1` ne se termine pas correctement, `commande2` ne sera pas exécutée. Pour cette raison, l'opérateur `&&` est appelé opérateur « et ».

Les espaces autour de l'opérateur `&&` ne sont pas indispensables.

Considérons maintenant l'opérateur `||` :

```
$ commande1 || commande2
```

La commande `commande2` ne sera exécutée que si le statut de sortie de la commande `commande1` est non nul, c'est-à-dire si elle ne s'est pas correctement terminée. Donc, si `commande1` se termine correctement, `commande2` ne sera pas exécutée. Si `commande1` ne se termine pas correctement, `commande2` sera exécutée. Pour cette raison, l'opérateur `||` est appelé opérateur « ou ».

Les espaces autour de l'opérateur `||` ne sont pas indispensables.

Enfin, pour être tout à fait complet, signalons également l'existence du séparateur `;` :

```
$ commande1 ; commande2
```

Le séparateur `;` permet d'exécuter une séquence de commandes, sans condition. Ici, `commande1` sera exécutée, suivie de `commande2`.

Les espaces autour du séparateur `;` ne sont pas indispensables.

9.3.6.3 Les commandes `true` et `false`

Signalons l'existence des commandes `true`, dont le statut de sortie est toujours vrai, et `false`, dont le statut de sortie est toujours faux :

```
$ true
$ echo $?
0
$ false
$ echo $?
1
```

http://en.wikipedia.org/wiki/True_and_false_(Unix)

9.4 Redirections

9.4.1 Sous UNIX, tout est fichier

L'un des choix de conception d'UNIX est particulièrement élégant et ouvre de nouveaux horizons. Il s'agit de ramener le plus de choses, en particulier les périphériques, à des fichiers, dans la mesure où cela est possible, c'est-à-dire si la communication avec eux se fait en lisant ou en écrivant une suite d'octets (comme on lit ou on écrit une suite d'octets dans un fichier).

Une imprimante, par exemple, est un périphérique qui s'attend à recevoir une suite d'octets décrivant les données à imprimer. On peut donc, sous UNIX, accéder à l'imprimante au moyen d'un fichier spécial `/dev/lp`, `/dev/lpt0` ou `/dev/lp0` (cela varie en fonction de l'UNIX utilisé).

Un *fichier spécial* est un fichier qui, lorsqu'on lit ou écrit dedans, transfère les données depuis ou vers le périphérique qui lui est associé. Les fichiers spéciaux sont tous regroupés dans le répertoire `/dev`.

De même, le terminal peut être considéré comme trois fichiers :

- l'*entrée standard*, correspondant au clavier ;
- la *sortie standard*, correspondant à l'écran ;
- la *sortie d'erreur standard*, correspondant également à l'écran.

Il y a deux sorties standards pour pouvoir au besoin faire la différence entre le flux de sortie normal et les messages d'erreur.

L'entrée standard est accessible via le fichier spécial `/dev/stdin`, la sortie standard est accessible via le fichier spécial `/dev/stdout` et la sortie d'erreur standard est accessible via le fichier spécial `/dev/stderr`.

En pratique, les commandes utilisées dans l'interpréteur de commandes lisent leur entrée standard et écrivent sur leur sortie standard et leur sortie d'erreur standard via ces fichiers spéciaux comme représenté dans la figure 9.2. Les fonctions de lecture et d'écriture dans un fichier leur servent donc également à accéder indirectement au clavier et à l'écran du terminal.

FIGURE 9.2 – *Entrée, sortie et sortie d'erreur standards.*

Historiquement, cela a permis de simplifier considérablement l'accès des logiciels aux périphériques. Ils devaient auparavant gérer une grande partie des communications eux-mêmes et connaître, par exemple, les spécificités des différents types de terminaux susceptibles d'être utilisés. Grâce aux fichiers spéciaux, UNIX s'occupe seul de gérer l'accès aux périphériques pour tous les logiciels, ceux-ci se contentant de lire et d'écrire dans des fichiers.

Nous allons voir que, grâce à cette façon de s'abstraire du terminal, il est possible de faire des choses très intéressantes.

> http://fr.wikipedia.org/wiki/Flux_standard
> http://en.wikipedia.org/wiki/Redirection_(computing)

9.4.2 Redirection de la sortie standard

La plupart des commandes que nous avons étudiées n'affichent rien du tout. D'autres, en revanche, savent être plus loquaces. C'est le cas, par exemple, de `ps`, dont le but est justement d'afficher quelque chose.

Il peut être utile de conserver dans un fichier ce qu'une commande affiche. Comment faire ? On pourrait envisager d'utiliser un éditeur de texte afin d'y copier-coller ce que la commande a affiché. Néanmoins, s'il y a plusieurs écrans à faire défiler, ceci est répétitif (pas bon...) et il est toujours possible de commettre des erreurs d'inattention (oubli d'une ligne ou copie en double). Si l'affichage est suffisamment volumineux, le début peut même avoir disparu de la mémoire de l'émulateur de terminal. Bref, ce n'est pas la bonne solution.

Puisque l'affichage se fait en écrivant sur la sortie standard, par défaut raccordée à l'écran, pourquoi ne pas plutôt la raccorder à un fichier qui permettrait de conserver l'affichage de la commande, comme représenté dans la figure 9.3 ?

FIGURE 9.3 – *Redirection de la sortie standard.*

C'est possible en redirigeant la sortie standard vers un fichier :

```
$ ps aux > sortie
$ ■
```

Remarquez que rien n'a été affiché à l'écran. En revanche, un fichier `sortie` a été créé, qui contient ce que `ps aux` aurait affiché sans la redirection (vous pouvez le vérifier avec `cat`, `more` ou `less`).

Le caractère > a donc pour effet de rediriger la sortie standard de la commande qui le précède vers le fichier dont le nom est indiqué après lui.

Si le fichier existait auparavant, il est écrasé.

Si l'on souhaite accumuler dans un fichier les affichages successifs de plusieurs commandes, on peut utiliser une autre forme de redirection de la sortie standard :

```
$ commande1 >> sortie
$ commande2 >> sortie
$ commande3 >> sortie
```

L'expression >> redirige la sortie standard de la commande qui la précède vers le fichier dont le nom est indiqué après elle.

Si le fichier n'existait pas auparavant, il est créé. S'il existait, le résultat de la redirection est placé après le contenu du fichier.

Pour revenir à notre exemple, le fichier `sortie` (dont on supposera qu'il n'existait pas auparavant) contient donc l'affichage de `commande1`, suivi de celui de `commande2`, suivi de celui de `commande3`.

À noter que, pour une fois, les espaces autour de > et >> ne sont pas nécessaires.

On souhaite parfois utiliser une commande loquace pour ses effets de bord, sans avoir besoin de ce qu'elle affiche à l'écran. Dans ce cas, il est possible de rediriger sa sortie standard vers le fichier spécial `/dev/null`, qui se comporte comme un trou noir :

```
$ commande > /dev/null
$ ■
```

http://fr.wikipedia.org/wiki//dev/null

FIGURE 9.4 – *Redirection de la sortie d'erreur standard.*

9.4.3 Redirection de la sortie d'erreur standard

Si l'on souhaite rediriger vers un fichier la sortie d'erreur standard, comme représenté dans la figure 9.4, il faut utiliser l'expression `2>` :

```
$ ls fichier_qui_n_existe_pas 2> erreur
$ cat erreur
ls: fichier_qui_n_existe_pas: No such file or directory
```

L'espace précédent `2>` est obligatoire. Celle suivant cette expression ne l'est pas.

FIGURE 9.5 – *Redirection des sortie et sortie d'erreur standards.*

On peut combiner les deux formes de redirection des sorties pour rediriger la sortie standard et la sortie d'erreur standard vers des fichiers, comme représenté dans la figure 9.5 :

```
$ commande > sortie 2> erreur
```

9.4.4 Redirection de l'entrée standard

Il est également possible de rediriger l'entrée standard, c'est-à-dire de faire en sorte qu'une commande, au lieu de lire au clavier, lise le contenu d'un fichier, comme représenté dans la figure 9.6.

FIGURE 9.6 – *Redirection de l'entrée standard.*

Pour tester ceci, il convient d'introduire la commande `bc` (pour *basic calculator*). Celle-ci exécute les opérations arithmétiques reçues sur son entrée standard :

```
$ bc
24+27
51
13*128
1664
quit
$ ■
```

Nous allons donc rediriger l'entrée standard de `bc`. Pour cela, nous avons besoin d'un fichier d'entrée. Saisissez dans un fichier `entree`, au moyen de votre éditeur de texte favori, quelques lignes contenant des opérations arithmétiques, par exemple :

```
24+27
13*128
```

Il est inutile d'indiquer `quit`, `bc` se terminera en atteignant la fin du fichier (ce qui fermera l'entrée standard).

Sauvez le fichier et quittez l'éditeur de texte.

Pour rediriger l'entrée standard depuis un fichier, il faut utiliser le caractère `<` :

```
$ bc < entree
51
1664
$ ■
```

Les espaces autour de < ne sont pas nécessaires.

On peut bien entendu associer toutes les formes de redirection pour que certaines commandes, dont l'exécution peut être longue, vivent leur vie sans déranger personne. On en profite également pour les lancer en arrière-plan :

```
$ commande < entree > sortie 2> erreur &
```

9.4.5 Les tuyaux

Nous allons réutiliser la commande amusante `xeyes` afin de voir comment il est possible d'en automatiser l'assassinat :

```
$ xeyes &
[1] 1664
$ ■
```

Faisons comme si nous n'avions pas vu l'identifiant de processus affiché lors du lancement de la commande en arrière-plan. Pour l'obtenir, dans le but d'utiliser `kill`, on peut utiliser la commande `ps` :

```
$ ps ux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
[...]
babafou   1664  0.0  0.1   7456  2636 ttyq5    SN   14:18   0:00 xeyes
[...]
```

Il serait intéressant de pouvoir isoler la ligne qui concerne `xeyes`. Il existe justement une commande permettant d'afficher les lignes d'un fichier contenant un motif textuel : la commande `grep` (qui sera étudiée au paragraphe 9.5.2 page 205). Nous allons donc rediriger la sortie standard de `ps` afin de pouvoir utiliser `grep` dessus :

```
$ ps ux > sortie-ps
$ grep xeyes sortie-ps
babafou   1664  0.0  0.1   7456  2636 ttyq5    SN   14:18   0:00 xeyes
```

La commande `grep` recherche le motif qui lui est passé en premier argument dans les fichiers dont les noms lui sont passés dans les arguments suivants et affiche les lignes de ces fichiers contenant ce motif.

Le plus gros du travail est fait. Pour terminer, il serait intéressant d'extraire l'identifiant de processus de la sortie standard de `grep`. La commande `awk` (qui sera étudiée au paragraphe 9.5.3 page 209) fait cela :

```
$ grep xeyes sortie-ps > sortie-grep
$ awk '{print $2}' sortie-grep
1664
```

Ici, on demande à `awk` d'extraire la deuxième colonne (les colonnes étant séparées par des espaces) de chaque ligne du fichier `sortie-grep`. Sa syntaxe est un peu compliquée. En effet, `awk` est un outil de traitement de données très puissant reposant sur un vrai langage de programmation.

Nous avons donc notre résultat et nous pouvons fièrement faire :

```
$ kill 1664
```

Nous pouvons aussi honteusement faire :

```
$ rm sortie-ps sortie-grep
```

voire :

```
$ rm sortie-*
```

si ce sont les deux seuls fichiers dont le nom commence par `sortie-`.

En effet, il est honteux d'avoir recours de la sorte à des fichiers temporaires, qu'il faut effacer dans les instants qui suivent leur création. C'est parfaitement inefficace.

Retraçons la suite d'opérations ayant permis d'obtenir l'identifiant du processus `xeyes` :

```
$ ps ux > sortie-ps
$ grep xeyes sortie-ps > sortie-grep
$ awk '{print $2}' sortie-grep
1664
```

On peut schématiser l'enchaînement des commandes comme représenté dans la figure 9.7.

FIGURE 9.7 – *Obtention de l'identifiant de processus avec des fichiers temporaires.*

Il s'avère que `grep` est une commande particulière, qu'on appelle *filtre*. Un filtre est une commande réalisant une certaine action sur les fichiers dont les noms lui sont passés en arguments ou, en leur absence, sur son entrée standard.

On peut facilement vérifier ces deux comportements :

```
$ grep xeyes sortie-ps
babafou   1664  0.0  0.1   7456  2636 ttyq5    SN   14:18   0:00 xeyes
$ grep xeyes < sortie-ps
babafou   1664  0.0  0.1   7456  2636 ttyq5    SN   14:18   0:00 xeyes
```

Dans le premier cas, on donne le fichier `sortie-ps` en argument, dans le second, on redirige l'entrée standard de `grep` depuis ce même fichier. Dans les deux cas, le résultat est le même.

Il est donc tentant d'essayer de se passer du fichier temporaire `sortie-ps` en connectant directement la sortie standard de `ps` à l'entrée standard de `grep`.

Ceci est possible grâce à un *tuyau* (*pipe* en anglais). Le terme *tube* est également employé. Un tuyau permet d'interconnecter la sortie standard d'une commande

à l'entrée standard d'une autre et de faire transiter les données de l'une à l'autre directement en mémoire, sans passer par un fichier temporaire.

On crée un tuyau en intercalant une barre verticale | (il n'est pas nécessaire de l'entourer d'espaces) entre les deux commandes à relier :

```
$ ps ux | grep xeyes
babafou   1664  0.0  0.1   7456  2636 pts/1    Ss   14:18   0:00 xeyes
babafou   3351  0.0  0.1   2868  1136 pts/1    S+   22:30   0:00 grep xeyes
```

Ceci se comprend ainsi : faire la liste des processus et n'afficher que les lignes contenant `xeyes`.

Bizarrement, on a un résultat différent de celui utilisant un fichier temporaire. La différence est que maintenant, `ps` et `grep` fonctionnent simultanément. Dans la mesure où l'argument de `grep` est le motif recherché, la ligne concernant `grep` apparaît également. Il faudrait pouvoir, dans la sortie standard de `grep`, supprimer cette ligne. C'est possible grâce à l'option `-v` de `grep`, qui inverse son fonctionnement. Au lieu d'afficher les lignes contenant un motif, `grep` n'affiche que celles ne le contenant pas :

```
$ ps ux | grep xeyes | grep -v grep
babafou   1664  0.0  0.1   7456  2636 pts/1    Ss   14:18   0:00 xeyes
```

Ceci se comprend ainsi : faire la liste des processus, n'afficher que les lignes contenant `xeyes` mais pas celles contenant `grep`.

Par chance (ou peut-être l'exemple est-il habilement choisi...), `awk` est également un filtre. On peut donc faire :

```
$ ps ux | grep xeyes | grep -v grep | awk '{print $2}'
1664
```

Ceci se comprend ainsi : faire la liste des processus, n'afficher que les lignes contenant `xeyes` mais pas celles contenant `grep` et en extraire la deuxième colonne.

On peut schématiser l'enchaînement des commandes comme représenté dans la figure 9.8 page suivante.

Cette fois, nous pouvons enfin fièrement faire :

```
$ kill 1664
```

FIGURE 9.8 – *Obtention de l'identifiant de processus avec des tuyaux.*

Fièrement parce que les tuyaux sont au cœur de la philosophie d'UNIX et il faut les utiliser dès que possible. Dès qu'on repère un fichier temporaire généré par la redirection de la sortie standard d'une commande et pris comme argument ou entrée standard d'une autre, ainsi que représenté dans la figure 9.7 page 200.

De nombreuses commandes sont d'ailleurs des filtres. Certaines d'entre elles sont étudiées au paragraphe 9.5 page suivante.

http://fr.wikipedia.org/wiki/Tube_(shell)

9.4.6 Substitution de commande

Ceci n'est pas une redirection à proprement parler mais un mécanisme relativement proche et qui nous manque pour terminer notre exemple d'assassinat de xeyes en beauté.

Grâce aux tuyaux, nous aboutissons donc à :

```
$ ps ux | grep xeyes | grep -v grep | awk '{print $2}'
1664
$ kill 1664
```

Mais il reste toujours une étape manuelle dans ce traitement...

Les plus hardis ont certainement envie de faire :

```
$ ps ux | grep xeyes | grep -v grep | awk '{print $2}' | kill
```

Mais cela ne peut pas fonctionner. En effet, `kill` n'est pas un filtre et ne sait donc pas lire son entrée standard en l'absence d'argument.

Néanmoins, il existe une syntaxe adaptée à ce cas :

```
$ kill `ps ux | grep xeyes | grep -v grep | awk '{print $2}'`
```

Elle utilise une paire d'apostrophes inverses ` (c'est un caractère qui est toujours utilisé par paire).

Lorsqu'il rencontre une paire d'apostrophes inverses contenant une commande simple ou un enchaînement avec des tuyaux, l'interpréteur de commandes substitue sa sortie standard à l'ensemble `...` comme représenté dans la figure 9.9.

FIGURE 9.9 – *Substitution de commande.*

Si la sortie standard de la commande ou de l'enchaînement entre apostrophes inverses est composée de plusieurs lignes, celles-ci sont concaténées en les séparant par des espaces.

http://en.wikipedia.org/wiki/Command_substitution

9.5 Quelques filtres usuels

Un *filtre* est une commande réalisant une certaine action sur les fichiers dont les noms lui sont passés en arguments ou, en leur absence, sur son entrée standard, très souvent interconnectée avec la sortie standard d'une autre commande au moyen d'un tuyau (voir le paragraphe 9.4.5 page 198).

Les tuyaux occupant une place centrale dans la philosophie d'UNIX, toutes les commandes qui s'y prêtent sont en fait des filtres, afin de pouvoir les utiliser comme tels.

http://en.wikipedia.org/wiki/Filter_(Unix)

9.5.1 Les commandes `more` et `less`

Certaines commandes génèrent un affichage qui peut se révéler parfois long, par exemple `ps aux` ou `ls` dans un répertoire contenant de nombreux fichiers. Lorsque l'affichage déborde de la fenêtre, il n'est pas toujours confortable de remonter en arrière grâce à l'ascenseur de la fenêtre de l'émulateur de terminal. On peut alors rediriger la sortie standard de la commande vers un fichier, pour pouvoir le consulter confortablement avec `more` ou `less`. Ce fichier étant un fichier temporaire, cela devrait déclencher tout de suite une alarme : mais où sont passés les tuyaux !

En effet, `more` et `less` sont des filtres, parmi les plus utilisés. En présence d'une commande ayant un affichage généreux, on peut donc faire :

```
$ commande | more
```

ou :

```
$ commande | less
```

Il vaut d'ailleurs mieux utiliser `less` dans ce cas. En effet, certaines versions de `more` ne permettent pas de revenir en arrière lorsqu'elles sont utilisées comme filtre. Par ailleurs, le problème mentionné au paragraphe 7.4.3 page 103 subsiste : arrivé en fin d'affichage, `more` se termine, contrairement à `less`. Là où il n'est pas vraiment gênant de relancer `more` sur un fichier, cela l'est beaucoup plus lorsqu'il s'agit de relancer un enchaînement de commandes, qui peut utiliser des ressources non négligeables.

Pour terminer sur le sujet, essayez, si vous ne l'avez pas déjà fait, d'utiliser `more` ou `less` derrière un tuyau :

```
$ ps aux | more
$ ps aux | less
```

Vous pouvez constater qu'on peut bien naviguer dans l'affichage au moyen des touches du clavier qu'on utilise habituellement avec ces commandes. Mais, puisque l'entrée standard de `more` ou `less` est raccordée à la sortie standard de `ps` grâce au tuyau, il est normalement impossible à ces deux commande d'accéder au clavier...

Pas par leur entrée standard, en tout cas. Mais `more` et `less` ont plus d'un tour dans leur sac. Elles accèdent en fait au clavier grâce au fichier spécial `/dev/tty` qui est toujours raccordé au terminal, redirection ou pas. Lire dans ce fichier spécial permet d'accéder au clavier, écrire dedans à l'écran.

9.5.2 La commande `grep`

La commande `grep` (pour *global regular expression print*) recherche le motif qui lui est passé en premier argument dans les fichiers dont les noms lui sont passés dans les arguments suivants ou, en leur absence, sur l'entrée standard (puisqu'il s'agit d'un filtre) et, par défaut, affiche chaque ligne contenant ce motif.

Le motif doit être fourni sous la forme d'une *expression rationnelle*. Les expressions rationnelles sont très souvent utilisées sous UNIX pour le traitement des chaînes de caractères. Elles seront étudiées au paragraphe 9.5.2.1. Le paragraphe 9.5.2.3 page 208 présentera ensuite quelques options de la commande `grep`.

Le lecteur curieux pourra se reporter à l'ouvrage suivant de la bibliographie pour approfondir ses connaissances :

– *grep Pocket Reference* [12]

> http://fr.wikipedia.org/wiki/Grep

9.5.2.1 Les expressions rationnelles

Une *expression rationnelle* (*regular expression* en anglais) est une chaîne de caractères permettant de décrire un ensemble de chaînes de caractères.

Pour cela, on utilise un langage donnant une signification particulière à certains caractères. Ceux-ci sont indiqués dans le tableau 9.1 page suivante.

Les expressions rationnelles les plus simples sont celles n'utilisant aucun des caractères du tableau 9.1 page suivante. Elles représentent la chaîne de caractère exacte qu'elle contiennent. Ainsi, l'expression rationnelle `toto` correspond à la chaîne de caractères « toto » et à elle seule.

Le point `.` permet de désigner n'importe quel caractère. Ainsi, l'expression rationnelle `e.cu.er` correspond à toute chaîne de caractères commençant par un « e » suivi d'un caractère quelconque suivi de « cu » suivi d'un caractère quelconque suivi de « er », comme par exemple « excuser ».

Si l'on souhaite quelque chose de plus précis, il est possible de se restreindre à une liste ou à un intervalle (dans l'ordre alphabétique) de caractères en l'indiquant entre crochets `[ ... ]`. L'expression entre crochets correspond à un et exactement un

Expressions élémentaires	
`.`	n'importe quel caractère
`[ ... ]`	liste ou intervalle de caractères
`[^ ... ]`	négation d'une liste ou d'un intervalle de caractères
`^`	début de ligne
`$`	fin de ligne
Opérateurs de répétition (s'appliquent à l'expression précédente)	
`?`	zéro ou une fois
`*`	zéro fois ou plus
`+`	une fois ou plus
`{n}`	exactement *n* fois
`{n,}`	*n* fois ou plus
`{n,m}`	entre *n* et *m* fois
Choix et groupement	
`|`	choix entre plusieurs éléments
`( ... )`	groupement

TABLE 9.1 – *Caractères utilisables dans une expression rationnelle.*

caractère figurant dans la liste ou l'intervalle. Ainsi, l'expression rationnelle `pi[pr]e` correspond aux chaînes de caractères « pipe » et « pire » et l'expression rationnelle `ra[r-t]e` correspond aux chaînes de caractères « rare », « rase » et « rate ».

La signification d'une expression entre crochets est inversée lorsque le premier caractère de cette expression est un accent circonflexe `^`. Dans ce cas, l'expression correspond à un et exactement un caractère ne figurant pas dans la liste ou l'intervalle. Ainsi, l'expression rationnelle `[^ms]otte` correspond à la chaîne de caractères « botte », entre autres, mais pas aux chaînes de caractères « motte » et « sotte ».

Pour faire figurer un crochet fermant `]` dans une expression entre crochets, il faut le placer en premier. De même, pour faire figurer un accent circonflexe `^`, il faut le placer n'importe où sauf en premier. Enfin, pour faire figurer un tiret `-`, il faut le placer en dernier.

Il est possible d'indiquer que l'expression rationnelle doit figurer en début de ligne, grâce à l'accent circonflexe `^`, qui doit être le premier caractère de l'expression rationnelle, ou qu'elle doit figurer en fin de ligne, grâce au symbole dollar `$`, qui doit être le dernier caractère de l'expression rationnelle. Ainsi, l'expression rationnelle

`^toto` corespond à la chaîne de caractères « toto » uniquement si elle se trouve en début de ligne et l'expression rationnelle `toto$` correspond à la chaîne de caractères « toto » uniquement si elle se trouve en fin de ligne.

Des opérateurs de répétition peuvent s'appliquer à l'élément qui les précède. Ainsi, le point d'interrogation `?` indique que l'élément précédent peut figurer zéro ou une fois, l'astérisque `*` indique que l'élément précédent peut figurer zéro fois ou plus et le signe plus `+` indique que l'élément précédent peut figurer une fois ou plus. Il est possible de contrôler plus finement le nombre de répétitions grâce à une expression entre accolades. Par exemple, l'expression rationnelle `coch?on` correspond aux chaînes de caractères « cocon » et « cochon », l'expression rationnelle `[acgt]*` correspond à n'importe quelle séquence d'ADN, éventuellement vide, l'expression rationnelle `[acgt]+` correspond à n'importe quelle séquence d'ADN contenant au moins un caractère et l'expression rationnelle `[acgu]{3}` correspond à un codon d'ARN messager.

La barre verticale `|` propose le choix entre plusieurs éléments. Ainsi, l'expression rationnelle `fromage|dessert|rien` correspond aux chaînes de caractères « fromage », « dessert » et « rien ».

Il est parfois nécessaire d'utiliser des parenthèses `( ... )` afin de grouper certains éléments d'une expression rationnelle, souvent dans le but de leur appliquer un opérateur de répétition. Ainsi, l'expression rationnelle `(GA|BU|ZO|MEU)+` permet de parler Shadok.

Enfin, il est possible d'inhiber la signification particulière des caractères du tableau 9.1 page précédente en les faisant précéder d'une barre oblique inversée `\`. Ainsi, l'expression rationnelle `\[1\]` correspond à la chaîne de caractères « [1] ».

Le lecteur curieux pourra se reporter aux ouvrages suivants de la bibliographie pour approfondir ses connaissances :

- *Mastering Regular Expressions* [17]
- *Regular Expressions Cookbook* [18]
- *Regular Expression Pocket Reference* [28]

http://fr.wikipedia.org/wiki/Expression_rationnelle

9.5.2.2 Utilisation d'une expression rationnelle avec la commande `grep`

Certains des caractères utilisables dans une expression rationnelle ayant également une signification particulière pour l'interpréteur de commandes, il convient de les inhiber pour ce dernier en faisant précéder chacun d'eux d'une barre oblique inversée ou en entourant l'expression rationnelle d'une paire d'apostrophes (les

guillemets ne conviennent pas dans ce cas) comme vu au paragraphe 8.6 page 148. L'utilisation d'une paire d'apostrophes étant plus claire, cette méthode est vivement recommandée :

```
$ grep '^[acgt]*$' fichier
```

Par ailleurs, la syntaxe des expressions rationnelles décrite au paragraphe 9.5.2.1 page 205 correspond aux expressions rationnelles dites *étendues*. Historiquement, les premières expressions rationnelles, dites *simples*, avaient une syntaxe plus lourde, dans laquelle on utilisait `\?`, `\+`, `\{`, `\|`, `\(`, `\)` au lieu respectivement de `?`, `+`, `{`, `|`, `(` et `)`.

Par défaut, la commande `grep` s'attend à ce que son premier argument soit une expression rationnelle simple. S'il s'agit d'une expression rationnelle étendue, il faut utiliser l'option `-E` de la commande `grep` :

```
$ grep -E '^[acgt]+$' fichier
```

ou, ce qui revient au même, la commande `egrep` (pour *extended* `grep`) :

```
$ egrep '^[acgt]+$' fichier
```

9.5.2.3 Quelques options de la commande `grep`

Hormis l'option `-E`, la commande `grep` dispose de nombreuses options. Nous allons étudier les plus utiles.

L'option `-i` de la commande `grep` permet d'ignorer la différence entre majuscules et minuscules lors de la comparaison entre l'expression rationnelle et le contenu du fichier :

```
$ grep -i toto fichier
le mot toto en minuscules
le mot TOTO en majuscules
```

L'option `-l` de la commande `grep` permet d'afficher non pas les lignes correspondant à l'expression rationnelle mais les noms des fichiers les contenant :

```
$ grep -l toto fichier1 fichier2 fichier3
fichier2
```

L'option `-n` de la commande `grep` préfixe chaque ligne correspondant à l'expression rationnelle de son numéro suivi d'un deux-points :

```
$ grep -n toto fichier
2:le mot toto dans le fichier
4:encore le mot toto dans le fichier
```

L'option `-v` de la commande `grep` inverse la signification de la comparaison avec l'expression rationnelle. On affiche donc les lignes ne lui correspondant pas :

```
$ grep -v toto fichier
tout le fichier
sauf les lignes contenant TOTO (en minuscules)
```

9.5.3 La commande `awk`

Le langage de programmation AWK a été conçu pour le traitement de données textuelles. Il tire son nom des patronymes de ses auteurs, Alfred Aho, Peter Weinberger et Brian Kernighan.

La commande `awk` permet d'utiliser le langage de programmation AWK (la commande s'écrit en minuscules, le langage de programmation en majuscules).

Son premier argument est une chaîne de caractères contenant le programme à exécuter sur les fichiers dont les noms lui sont passés dans les arguments suivants ou, en leur absence, sur l'entrée standard (puisqu'il s'agit d'un filtre).

Un programme en langage AWK est une suite d'instructions de la forme :

```
condition {action}
```

La condition peut utiliser des tests de comparaison ou une expression rationnelle, qu'on écrit alors entre deux barres obliques : `/expression rationnelle/`.

Par ailleurs, il existe deux conditions particulières :

- `BEGIN`, dont l'action correspondante est exécutée au début du programme ;

– `END`, dont l'action correspondante est exécutée à la fin du programme.

La forme générale d'un programme en langage AWK est donc :

```
BEGIN {action} condition {action} ... END {action}
```

La commande `awk` commence par exécuter l'action associée à la condition `BEGIN`, si elle existe. Puis elle lit les fichiers dont les noms lui sont passés en arguments ou l'entrée standard, ligne par ligne. Si la première condition du programme est vérifiée pour la ligne qui vient d'être lue, l'action correspondante est exécutée. Puis on passe à la condition suivante et ainsi de suite. Enfin, l'action associée à la condition `END`, si elle existe, est exécutée.

Il est possible d'avoir dans le programme des actions sans conditions. Dans ce cas, ces actions sont systématiquement exécutées pour chaque ligne.

Les accolades ayant une signification particulière pour l'interpréteur de commandes, de même que certains des caractères utilisés dans les expressions rationnelles et dans les actions, il convient d'entourer systématiquement les programmes en langage AWK d'une paire d'apostrophes.

Pour illustrer l'utilisation de la commande `awk`, nous allons reprendre l'exemple du paragraphe 9.4.5 page 198 visant à déterminer l'identifiant du processus `xeyes` mais en utilisant exclusivement `awk`.

En première approche, on peut utiliser `/xeyes/` comme condition afin de trouver les lignes contenant cette expression rationnelle dans la sortie standard de `ps ux` et les afficher en utilisant l'action `print`, qui affiche la ligne en cours de traitement (si la condition est remplie) :

```
$ ps ux | awk '/xeyes/ {print}'
babafou   1664  0.0  0.1   7456  2636 pts/1    Ss   14:18   0:00 xeyes
babafou   3351  0.0  0.1   2868  1136 pts/1    S+   22:30   0:00 awk xeyes {print}
```

Comme précédemment, il faut éliminer la ligne correspondant à la commande `awk`. On peut utiliser pour cela une condition qui sera vérifiée pour cette ligne, à savoir l'expression rationnelle `/awk/` et lui associer l'action `next`, qui permet d'ignorer les autres conditions et de passer au traitement de la ligne suivante :

```
$ ps ux | awk '/awk/ {next} /xeyes/ {print}'
babafou   1664  0.0  0.1   7456  2636 pts/1    Ss   14:18   0:00 xeyes
```

L'ordre des instructions est important. L'instruction `/awk/ {next}` doit se trouver avant l'instruction `/xeyes/ {print}` pour ignorer la ligne correspondant à la commande `awk`.

Plutôt que d'afficher toute la ligne, on peut se contenter du seul identifiant de processus. En effet, la commande `awk` est capable de découper une ligne de texte en champs (les champs étant séparés par des espaces) et de permettre l'accès à chaque champ depuis le programme. Ainsi, `$1` représente le premier champ, `$2` le deuxième champ, etc.

L'identifiant de processus correspondant au deuxième champ, on peut donc faire :

```
$ ps ux | awk '/awk/ {next} /xeyes/ {print $2}'
1664
```

Il est également possible de limiter l'application de l'expression rationnelle à un champ particulier. Le nom de la commande étant le onzième champ de la sortie standard de la commande `ps`, on peut également faire :

```
$ ps ux | awk '$11 ~ /xeyes/ {print $2}'
1664
```

L'opérateur ~ permet de restreindre la comparaison de l'expression rationnelle qui le suit au champ qui le précède. Ceci permet de se passer de l'élimination de la ligne correspondant à la commande `awk`.

Enfin, dans la mesure où l'expression rationnelle n'apporte rien de particulier, puisqu'il ne s'agit que d'une chaîne de caractères fixe, on peut même faire simplement un test d'égalité entre le onzième champ et la chaîne de caractères « xeyes » :

```
$ ps ux | awk '$11 == "xeyes" {print $2}'
1664
```

Lorsque les programmes en langage AWK deviennent volumineux, il est plus simple de les saisir dans un fichier et de demander à la commande `awk` d'exécuter le programme contenu dans ce fichier.

Considérons le fichier de données `cave.txt` suivant :

```
Alsace    Riesling     2 2005 blanc Joseph Gsell
Alsace    Riesling     4 2002 blanc Victor Hertz
Bordeaux  Cadillac     6 2004 blanc Château la Clyde
Bordeaux  Médoc        3 2000 rouge Château d'Escot
Bordeaux  Pomerol      2 2003 rouge Château Pierhem
Bourgogne Aloxe-Corton 4 1998 rouge Dubois Bernard & fils
Bourgogne Montagny     6 2007 blanc Jean-Pierre Berthenet
Bourgogne Volnay       6 2005 rouge Pascal Laboureau
Languedoc Cabardès     4 2002 rouge Château Ventenac
Loire     Sancerre     1 2007 blanc Domaine Gérard Fiou
Sud-Ouest Gaillac      1 2005 rouge Château Adélaïde
Sud-Ouest Madiran      3 2004 rouge Cru du Paradis
Sud-Ouest Monbazillac  7 2003 blanc Château le Fagé
```

Il représente l'inventaire d'une cave à vin. Chaque ligne correspond à un ensemble de bouteilles identiques dont les champs sont :

1. le vignoble ;
2. l'appellation ;
3. le nombre de bouteilles ;
4. le millésime ;
5. la couleur ;
6. (et champs suivants) le producteur.

Le programme en langage AWK représenté dans le listage 9.1 page suivante permet d'analyser ce fichier afin d'en extraire certaines informations, à savoir le nombre de bouteilles par vignoble et par couleur. On pourrait bien entendu envisager d'autres traitements sur ce fichier.

Les instructions des lignes 6 à 10 permettent d'établir le nombre de bouteilles par vignoble. On utilise à cet effet une variable correspondant au vignoble éponyme et, en cas de correspondance, on lui ajoute le troisième champ, c'est-à-dire le nombre de bouteilles (les variables sont créées automatiquement et initialisées à zéro).

Les instructions des lignes 12 et 13 font de même pour le nombre de bouteilles par couleur.

Enfin, les instructions des lignes 16 à 25, correspondant à la condition `END`, permettent d'afficher les informations recueillies. Remarquez que, lorsque l'action contient plusieurs instructions, celles-ci doivent être terminées par un point-virgule.

```
BEGIN {
   print "Ma cave" ;
   print "" ;
}

/^Alsace/       {alsace    += $3}
/^Bordeaux/     {bordeaux  += $3}
/^Bourgogne/    {bourgogne += $3}
/^Languedoc/    {languedoc += $3}
/^Loire/        {loire     += $3}
/^Sud-Ouest/    {sud_ouest += $3}

/blanc/         {blanc += $3}
/rouge/         {rouge += $3}

END {
   print "Nombre de bouteilles par vignoble" ;
   print "Alsace    " alsace ;
   print "Bordeaux  " bordeaux ;
   print "Bourgogne " bourgogne ;
   print "Languedoc " languedoc ;
   print "Loire     " loire ;
   print "Sud-Ouest " sud_ouest ;
   print "" ;
   print "Nombre de bouteilles par couleur" ;
   print "blanc " blanc ;
   print "rouge " rouge ;
}
```

LISTAGE 9.1 – *Exemple de programme en langage AWK.*

L'option `-f` de la commande `awk`, suivie d'un nom de fichier, permet d'exécuter le programme en langage AWK contenu dans ce fichier.

Si le programme du listage 9.1 page précédente est contenu dans le fichier `cave.awk`, on peut donc l'appliquer à l'inventaire de la cave contenu dans le fichier `cave.txt` ainsi :

```
$ awk -f cave.awk cave.txt
Ma cave

Nombre de bouteilles par vignoble
Alsace    6
Bordeaux  11
Bourgogne 16
Languedoc 4
Loire     1
Sud-Ouest 11

Nombre de bouteilles par couleur
blanc 26
rouge 23
```

Nous n'irons pas plus loin dans la description du langage de programmation AWK, dont l'étude détaillée dépasse le cadre de ce cours. Le lecteur curieux pourra se reporter aux ouvrages suivants de la bibliographie pour approfondir ses connaissances :

- *The AWK Programming Language* [9]
- *Effective awk Programming* [22]
- *sed & awk* [15]
- *sed & awk Pocket Reference* [23]

> http://fr.wikipedia.org/wiki/Awk
> http://en.wikibooks.org/wiki/An_Awk_Primer

9.5.4 La commande sed

Un éditeur de texte, tel que ceux étudiés dans les chapitres 5 page 57 et 6 page 73, est un logiciel interactif, qui crée ou modifie un fichier texte en fonction des actions successives de l'utilisateur.

Lorsque les modifications à apporter à un fichier texte sont connues à l'avance et doivent être appliquées de manière répétitive ou lorsque des modifications identiques doivent être appliquées à plusieurs fichiers texte, l'utilisation d'un éditeur de texte interactif peut s'avérer fastidieuse. Il est alors préférable d'utiliser un éditeur de texte non interactif.

La commande `sed` (pour *stream editor*) est un éditeur de texte non interactif, utilisable en tant que filtre. Elle lit ligne par ligne les fichiers dont les noms lui sont passés en arguments ou, en leur absence, son entrée standard et, pour chaque ligne, lui applique une ou plusieurs actions. Le résultat est affiché au fur et à mesure sur la sortie standard, qu'on peut rediriger vers un fichier pour le conserver, ou fournir à une autre commande au moyen d'un tuyau pour lui appliquer des traitements supplémentaires.

L'utilisation la plus courante de la commande `sed` est le remplacement de mots dans un texte. Ainsi, la commande suivante remplace tous les `toto` du fichier `entree` par des `tata` et le résultat est redirigé vers le fichier `sortie` :

```
$ sed s/toto/tata/g entree > sortie
```

Le premier argument de la commande `sed` indique, lorsqu'il n'y en a qu'une, l'action à réaliser. Les arguments suivants sont les noms des fichiers à traiter. En leur absence, la commande `sed` lit son entrée standard.

Dans l'exemple précédent, l'action à réaliser est `s/toto/tata/g`. Ceci signifie qu'il faut remplacer (`s` pour *substitute*) chaque mot `toto` par `tata` de manière globale (en raison du `g` à la fin de l'action). Sans ce `g`, seul le premier `toto` de chaque ligne serait remplacé par `tata`. Les composants de l'action (`s`, `toto`, `tata` et `g`) sont séparés les uns des autres par des barres obliques `/` (sans espaces).

Le mot à remplacer n'est d'ailleurs pas un mot tout simple mais une expression rationnelle (voir le paragraphe 9.5.2.1 page 205). Comme indiqué au paragraphe 9.5.2.2 page 207, il est nécessaire de prendre des précautions lorsqu'on utilise une expression rationnelle contenant des caractères ayant une signification particulière pour l'interpréteur de commandes. Dans ce cas, on entoure habituellement l'action d'une paire d'apostrophes. Par ailleurs, la commande `sed` utilise par défaut une expression rationnelle simple. S'il s'agit d'une expression rationnelle étendue, il faut utiliser l'option `-E`. Par exemple, pour remplacer les `toto` ou `titi` par `tata`, on peut faire :

```
$ sed -E 's/toto|titi/tata/g' entree > sortie
```

Lorsque plusieurs actions sont à appliquer au texte, chacune doit suivre l'option `-e` de la commande `sed`. Ainsi, la commande précédente peut également être décomposée en deux actions :

```
$ sed -e s/toto/tata/g -e s/titi/tata/g entree > sortie
```

Si de nombreuses actions sont à appliquer au texte ou si l'on souhaite les conserver afin de les réutiliser, il est possible de les spécifier dans un fichier, ce qui est plus pratique que de les indiquer sur la ligne de commande.

Considérons le fichier `actions.sed` suivant, qui contient les actions à effectuer, une action par ligne :

```
s/toto/tata/g
s/titi/tata/g
```

La commande `sed` peut être pilotée par celui-ci en utilisant l'option `-f` suivie du nom du fichier :

```
$ sed -f actions.sed entree > sortie
```

Nous n'irons pas plus loin dans la description de la commande `sed`, dont l'étude détaillée dépasse le cadre de ce cours. Le lecteur curieux pourra se reporter aux ouvrages suivants de la bibliographie pour approfondir ses connaissances :

- *sed & awk* [15]
- *sed & awk Pocket Reference* [23]

http://fr.wikipedia.org/wiki/Stream_Editor

9.5.5 La commande `sort`

La commande `sort` permet de trier selon divers critères les lignes d'un texte organisé en colonnes. Elle est utilisable en tant que filtre. Elle lit les lignes de texte provenant des fichiers dont les noms lui sont passés en arguments ou, en leur absence, de son entrée standard et affiche sur sa sortie standard le résultat du tri de ces lignes.

Utilisée sans option, le critère de tri de la commande `sort` est l'ordre lexicographique de la première colonne de chaque ligne, les colonnes étant délimitées par des espaces :

```
$ cat toto
cc bb
bb aa
aa zz
$ sort toto
aa zz
bb aa
cc bb
```

Si l'on souhaite que le tri s'effectue par rapport à une autre colonne, il faut utiliser l'option `-k` (pour *key*, ceci correspond à la colonne selon laquelle s'effectue le tri) suivie du numéro de cette colonne. Ainsi, pour trier le fichier de l'exemple précédent en fonction de sa deuxième colonne, on peut faire :

```
$ sort -k 2 toto
bb aa
cc bb
aa zz
```

L'ordre lexicographique place les lettres majuscules avant les lettres minuscules :

```
$ cat toto
zz
MM
BB
aa
$ sort toto
BB
MM
aa
zz
```

L'option `-f` permet d'ignorer cette distinction :

```
$ sort -f toto
aa
BB
MM
zz
```

L'option `-n` (pour *numeric*) permet d'indiquer que le tri doit être fait selon l'ordre numérique :

```
$ cat toto
51
1664
33
$ sort -n toto
33
51
1664
```

Le tri du même fichier selon l'ordre lexicographique donne :

```
$ sort toto
1664
33
51
```

Par défaut, les colonnes composant le texte sont censées être séparées par des espaces. L'option `-t`, suivie d'un caractère, permet d'utiliser ce caractère comme séparateur. L'option `-t` est généralement utilisée en conjonction avec l'option `-k` :

```
$ cat toto
cc:bb
bb:aa
aa:zz
$ sort -k 2 -t : toto
bb:aa
cc:bb
aa:zz
```

Enfin, l'option `-r` (pour *reverse*) permet d'inverser l'ordre du tri. Ainsi, en reprenant le fichier de l'exemple précédent, on peut faire :

```
$ sort -r -k 2 -t : toto
aa:zz
cc:bb
bb:aa
```

Une utilisation classique de la commande `sort` est de trier l'affichage d'une autre commande, qui n'offre aucune possibilité de tri, au moyen d'un tuyau. Par exemple, l'affichage de la commande `du`, page 157, peut être trié ainsi selon le volume total occupé par l'ensemble du contenu de chaque répertoire (tri numérique au moyen de l'option `-n` par rapport à la première colonne puisque l'option `-k` n'est pas utilisée), les répertoires les plus volumineux étant donc clairement visibles en fin d'affichage :

```
$ du -k | sort -n
34      ./rep3/rep5
338     ./rep3/rep4
390     ./rep3
546     ./rep1
2866    ./rep2
7294    .
```

http://en.wikipedia.org/wiki/Sort_(Unix)

9.5.6 La commande `wc`

La commande `wc` (pour *word count*) permet d'afficher le nombre de lignes, de mots (les mots étant séparés les uns des autres par des espaces) et d'octets des fichiers dont les noms lui sont passés en arguments ou, en leur absence, de son entrée standard (puisqu'il s'agit d'un filtre) :

```
$ wc toto
   16687   79550  610080 toto
```

La commande `wc` affiche dans l'ordre :

1. le nombre de lignes du fichier (ici, 16687) ;
2. le nombre de mots du fichier (ici, 79550) ;
3. le nombre d'octets du fichier (ici, 610080), dont la valeur est identique à la taille du fichier telle qu'affichée par `ls -l`.

Lorsque plusieurs noms de fichiers lui sont passés en arguments, la commande `wc` affiche ces informations pour chacun des fichiers, suivies de leur cumul :

```
$ wc tata titi tutu
      15      29     261 tata
      16      13      67 titi
       5      25     187 tutu
      36      67     515 total
```

Utilisée en dernière commande d'un tuyau, la commande `wc` n'affiche que les informations, sans nom de fichier (qui, dans ce contexte, n'existe pas) :

```
$ ps aux | wc
      50     589    3944
```

Bien souvent, seul le nombre de lignes a un intérêt et l'option `-l` de la commande `wc` permet de n'afficher que lui :

```
$ ps aux | wc -l
      50
```

Dans la mesure où de nombreuses commandes ont un affichage indiquant un résultat par ligne (un fichier, un processus, etc.), on utilise souvent `wc -l` pour compter ces lignes. Ainsi, la commande de l'exemple précédent permet de compter le nombre de processus (auquel il ne faut pas oublier de retrancher 1 en raison de la première ligne de l'affichage de `ps` indiquant la signification de chaque colonne).

De même, l'option `-w` de la commande `wc` permet d'afficher le nombre de mots.

Enfin, l'option `-c` de la commande `wc` permet d'afficher le nombre d'octets. Avec de nombreux codages du texte, il s'agit également du nombre de caractères mais pas avec le codage UTF-8 (voir le paragraphe 4.3.3 page 50). L'option `-m` de la commande `wc` permet alors d'afficher le nombre de caractères. Par exemple, avec un fichier `toto` codé en UTF-8 :

```
$ cat toto
pâté
$ wc -c toto
       7 toto
$ wc -m toto
       5 toto
```

Le caractère supplémentaire (« pâté » compte quatre caractères et six octets lorsqu'il est codé en UTF-8) est le caractère de retour à la ligne, qui n'est pas visible mais néanmoins présent dans le fichier et comptabilisé par la commande `wc`.

Toute ces options peuvent être combinées, à l'exception des options `-c` et `-m`, qui ne peuvent pas être utilisées ensemble. La combinaison `-lwc` correspond au comportement sans option de la commande `wc`.

> http://en.wikipedia.org/wiki/Wc_(Unix)

9.6 Gestion des longs processus

Les commandes étudiées jusqu'à présent s'exécutent en un temps très court mais il existe également des commandes — qui réalisent généralement de lourds traitements de données tels que des calculs scientifiques — qui s'exécutent en plusieurs heures, plusieurs jours, voire plusieurs semaines.

Leur gestion demande des traitements particuliers afin de permettre leur déroulement dans les meilleures conditions.

9.6.1 La commande `nohup`

Lorsqu'on quitte l'interpréteur de commandes au moyen de la commande `exit` ou qu'on termine la session graphique, tous les processus lancés par l'utilisateur sont arrêtés par le système d'exploitation.

Si l'on doit lancer une commande ayant un long temps d'exécution, cela pose problème car il faut donc rester connecté sur l'ordinateur sur lequel on l'a lancée.

La commande `nohup` (pour *no hang-up*) permet de lancer un processus en l'émancipant, c'est-à-dire en le rendant totalement indépendant de l'interpréteur de commandes ou de la session qui ont contribué à son lancement :

```
$ nohup commande &
```

La commande `nohup` est suivie de la commande à lancer. Dans l'exemple précédent, celle-ci est représentée par `commande`, avec éventuellement ses options et ses arguments (qui ne sont pas indiqués dans l'exemple afin de ne pas l'alourdir). Le tout est lancé en arrière-plan pour pouvoir récupérer la main dans l'interpréteur de commandes.

Si elles ne sont pas redirigées explicitement, la sortie standard et la sortie d'erreur standard sont redirigées par `nohup` vers le fichier `nohup.out`.

Si l'on souhaite les rediriger vers deux fichiers différents ou rediriger l'entrée standard depuis un fichier, on peut utiliser tous les mécanismes étudiés au paragraphe 9.4 page 193 :

```
$ nohup commande < entree > sortie 2> erreur &
```

Pour émanciper un processus, il faut absolument le lancer avec la commande `nohup`. Il n'existe pas de moyen d'émanciper un processus s'il a été lancé autrement.

http://fr.wikipedia.org/wiki/Nohup

9.6.2 Les commandes `nice` et `renice`

Sous UNIX, il est possible de modifier la *priorité* d'un processus. Celle-ci est définie au moyen d'un nombre compris entre −20 et +19. C'est celui qu'affiche la commande `top` (voir le paragraphe 9.3.4 page 187) dans la colonne `NI`. La priorité par défaut d'un processus est de `0`. Plus la priorité est négative, plus le processus s'exécutera rapidement. Plus la priorité est positive, plus le processus s'exécutera lentement.

La priorité d'un processus influe sur la façon dont il est traité par l'ordonnanceur (voir le paragraphe 9.3.1 page 181). Selon l'algorithme utilisé par l'ordonnanceur, un processus plus prioritaire, c'est-à-dire ayant une priorité négative, disposera de quanta plus longs ou sera ordonnancé plus souvent et ce d'autant plus que sa priorité sera négative. À l'inverse, un processus moins prioritaire, c'est-à-dire ayant une priorité positive, disposera de quanta moins longs ou sera ordonnancé moins souvent et ce d'autant moins que sa priorité sera positive.

Évidemment, la priorité n'a d'influence sur la durée d'exécution d'un processus que s'il n'est pas le seul à utiliser le microprocesseur, auquel cas il sera ordonnancé

systématiquement. Pour que la priorité joue son rôle, il faut qu'au minimum deux processus utilisent régulièrement le microprocesseur.

Afin d'éviter qu'un utilisateur n'abuse de cette possibilité en lançant tous ses processus à la priorité −20, seul l'administrateur du système peut utiliser les priorités négatives. Les utilisateurs ont donc seulement accès aux priorités positives, c'est-à-dire qu'ils ne peuvent que ralentir leurs processus.

Quel est l'intérêt de ralentir ses processus ? Comme le nom de la commande `nice` l'indique, il s'agit d'être gentil avec les autres utilisateurs d'un système informatique. En temps normal, on cherche à obtenir les résultats de ses processus le plus rapidement possible mais il existe des cas où l'on peut se permettre de les ralentir. Par exemple, lorsqu'on s'absente pour une durée plus ou moins longue (déplacement, congés, etc.), on peut ralentir les processus dont on sait qu'ils vont se terminer avant son retour ou lancer avec une priorité plus faible de nouveaux processus. Un système informatique partagé par plusieurs catégories d'utilisateurs peut également imposer des priorités différentes pour leurs calculs (les étudiants ayant une priorité plus faible que celle des chercheurs, par exemple).

La commande `nice` permet de lancer un processus en modifiant sa priorité :

```
$ nice -n 5 commande &
```

La commande `nice` s'utilise avec l'option `-n` suivie de la priorité à donner au processus et de la commande à lancer, avec éventuellement ses options et ses arguments (qui ne sont pas indiqués dans l'exemple afin de ne pas l'alourdir). Le tout est lancé en arrière-plan pour pouvoir récupérer la main dans l'interpréteur de commandes.

La commande `nice` ayant pour but de lancer des processus ayant un long temps d'exécution (puisque réduire la priorité d'un processus se terminant rapidement n'a pas de sens), on l'utilise typiquement couplée avec la commande `nohup` (voir le paragraphe 9.6.1 page 221) et en utilisant les mécanismes de redirection étudiés au paragraphe 9.4 page 193 :

```
$ nohup nice -n 5 commande < entree > sortie 2> erreur &
```

La commande `renice` permet de modifier la priorité d'un ou de plusieurs processus en cours d'exécution. Elle prend en arguments la priorité à affecter et le ou les identifiants des processus concernés :

```
$ renice 5 1664
1664: old priority 0, new priority 5
```

Une fois la priorité diminuée, il n'est plus possible de l'augmenter (seul l'administrateur de système le peut) :

```
$ renice 0 1664
renice: 1664: setpriority: Permission denied
```

En revanche, il est possible de la diminuer davantage, si besoin est.

http://fr.wikipedia.org/wiki/Nice_(Unix)

9.6.3 La commande `screen`

La commande `nohup` a un défaut, elle ne permet pas d'émanciper un processus en pouvant continuer d'interagir directement avec lui par la suite. La redirection des entrée et sortie standards est la seule forme d'interaction possible avec le processus.

La commande `screen` permet de lancer un interpréteur de commandes, d'interagir avec lui, puis de l'émanciper lorsque nécessaire, avec toute commande lancée depuis celui-ci, et d'en reprendre le contrôle plus tard.

Un exemple d'utilisation de la commande `screen` permettra de mieux en comprendre le fonctionnement :

```
$ screen
```

Une fois lancée, la commande `screen` exécute un nouvel interpréteur de commandes, qu'on peut utiliser normalement mais aussi pour lancer une commande ayant un long temps d'exécution.

À tout moment, la combinaison de touches `C-a C-d` (la documentation de la commande `screen` utilise le même principe de notation qu'Emacs) permet de détacher l'ensemble formé par la commande `screen`, l'interpréteur de commandes qu'elle a lancé et tout processus exécuté depuis celui-ci du terminal, ce qui équivaut à l'émanciper. On revient alors à l'interpréteur de commandes qui a servi à exécuter la commande `screen`.

Cette dernière continue à fonctionner de manière émancipée mais son affichage est pour le moment inaccessible. On peut alors fermer sa session ou simplement faire autre chose.

Pour reprendre le contrôle de la commande `screen`, il faut, si on l'a quitté, se reconnecter sur l'ordinateur où on l'a lancée (physiquement ou à distance, voir le paragraphe 11.2 page 249) et utiliser la commande `screen` avec l'option `-r` (pour *resume*) :

```
$ screen -r
```

On se retrouve alors dans l'interpréteur de commandes lancé précédemment par la commande `screen` et l'on peut interagir avec lui normalement. On pourra par la suite détacher à nouveau la commande `screen` au moyen de la combinaison de touches `C-a C-d` puis en reprendre le contrôle avec `screen -r` et ainsi de suite.

Il est possible de lancer plusieurs commandes `screen`. Si plus de deux d'entre elles sont détachées, `screen -r` en affiche la liste :

```
$ screen -r
There are several suitable screens on:
        1664.pts-1.cafe01       (Detached)
        3351.pts-1.cafe01       (Detached)
Type "screen [-d] -r [pid.]tty.host" to resume one of them.
```

Il faut ensuite indiquer l'identifiant de processus de la commande `screen` dont on souhaite reprendre le contrôle :

```
$ screen -r 1664
```

Savoir quel identifiant de processus correspond à quelle commande `screen` n'est pas évident. Aussi est-il possible d'attribuer un nom à chaque session lancée par la commande `screen` au moyen de l'option `-S` (pour *session*) suivie du nom choisi. Au lieu de lancer la commande `screen` tout simplement, on fait donc :

```
$ screen -S session1
```

On peut alors travailler normalement puis détacher l'ensemble. On peut ensuite lancer d'autres commandes `screen` sur le même principe puis les détacher. Les noms attribués à chaque session figurent maintenant dans l'affichage de `screen -r` :

```
$ screen -r
There are several suitable screens on:
        5088.session1   (Detached)
        5613.session2   (Detached)
        5351.session3   (Detached)
Type "screen [-d] -r [pid.]tty.host" to resume one of them.
```

et l'on peut reprendre le contrôle d'une session particulière en indiquant son nom plutôt que son identifiant de processus :

```
$ screen -r session2
```

Quitter l'interpréteur de commandes lancé par la commande `screen` au moyen de la commande `exit` permet de quitter la commande `screen` elle-même.

La faculté de pouvoir lancer plusieurs interpréteurs de commandes depuis le même terminal fait que la commande `screen` constitue un multiplexeur de terminaux.

> http://www.gnu.org/software/screen/
> http://fr.wikipedia.org/wiki/GNU_Screen
> http://fr.wikipedia.org/wiki/Multiplexage

9.6.4 La commande `tmux`

La commande `tmux` est également un multiplexeur de terminaux et s'utilise dans le même esprit que la commande `screen`. Moins répandue que cette dernière, elle est en particulier fournie avec l'installation de base d'OpenBSD et est disponible sur d'autres UNIX.

```
$ tmux
```

Comme la commande `screen`, la commande `tmux` exécute un nouvel interpréteur de commandes.

Il est possible de détacher l'ensemble grâce à la combinaison de touches `C-b d` ou en exécutant la commande `tmux` avec l'argument `detach` :

```
$ tmux detach
```

Pour reprendre le contrôe de la commande `tmux`, on l'utilise avec l'argument `attach` :

```
$ tmux attach
```

Il est possible d'afficher la liste des commandes `tmux` en cours d'exécution en faisant `tmux ls` :

```
$ tmux ls
0: 1 windows (created Thu Mar 17 10:49:42 2011) [80x23]
1: 1 windows (created Thu Mar 17 14:59:31 2011) [80x23]
```

Chaque ligne correspond à une commande `tmux` en cours d'exécution. Chacune est repérée par un *identifiant de session* qui figure en début de ligne (il s'agit du nombre avant le deux-points).

Pour reprendre le contrôle d'une commande `tmux` particulière, il faut indiquer son identifiant de session au moyen de l'option `-t` (pour *target*) :

```
$ tmux attach -t 0
```

Savoir quel identifiant de processus correspond à quelle commande `tmux` n'est pas évident. Aussi est-il possible d'attribuer un nom à chaque session lancée par la commande `tmux` au moyen de l'argument `new` suivi de l'option `-s` (pour *session*) et du nom choisi. Au lieu de lancer la commande `tmux` tout simplement, on fait donc :

```
$ tmux new -s session1
```

On peut alors travailler normalement puis détacher l'ensemble. On peut ensuite lancer d'autres commandes `tmux` sur le même principe puis les détacher. Les noms attribués à chaque session figurent maintenant dans l'affichage de `tmux ls` :

```
$ tmux ls
session1: 1 windows (created Tue May 24 15:42:53 2011) [80x23]
session2: 1 windows (created Tue May 24 15:43:13 2011) [80x23]
session3: 1 windows (created Tue May 24 15:43:15 2011) [80x23]
```

et l'on peut reprendre le contrôle d'une session particulière en indiquant son nom plutôt que son identifiant de processus :

```
$ tmux attach -t session2
```

Quitter l'interpréteur de commandes lancé par la commande `tmux` au moyen de la commande `exit` permet de quitter la commande `tmux` elle-même.

> http://tmux.sourceforge.net/
> http://fr.wikipedia.org/wiki/Multiplexage

10

Impression

IMPRIMER, c'est mal. C'est mal parce que ça se fait sur des restes d'arbres morts. C'est un peu moins mal lorsque le papier sur lequel on imprime est recyclé. Mais c'est tout de même mal parce que, bien souvent, ce qu'on imprime finit très rapidement dans une poubelle (rarement jaune).

Malgré la tendance au tout numérique, on n'a jamais consommé autant de papier qu'aujourd'hui. Et s'il faut réprimer ses envies d'imprimer, il est des situations où l'impression est un mal nécessaire. Nous allons étudier comment imprimer de manière responsable, en vérifiant l'état de l'imprimante avant toute impression, en supprimant d'éventuels travaux d'impression inutiles et en optimisant l'utilisation du papier.

10.1 Nommage des imprimantes

Sous UNIX, chaque imprimante est identifiée par un ou plusieurs noms, qui lui sont attribués par l'administrateur du système informatique. Lorsqu'une imprimante dispose de plusieurs noms, chacun d'eux permet d'exploiter une caractéristique particulière de l'imprimante, comme l'impression en recto seul, recto-verso ou encore le choix du format du papier.

> À l'ENSTA ParisTech, les trois imprimantes accessibles aux élèves sont situées au deuxième étage, dans le couloir des salles informatiques, pièce 273. Les deux imprimantes noir et blanc se nomment `speedy` et `speedo` pour l'impression en recto-verso, `speedy1` et `speedo1` pour l'impression en recto seul. L'imprimante couleur se nomme `pcolor` pour l'impression en recto-verso, `pcolor1` pour l'impression en recto seul.

10.2 Avant d'imprimer

L'époque où chacun disposait d'une imprimante à portée de main est révolue. Aujourd'hui, la tendance est à la mutualisation. L'imprimante commune — qui est de plus en plus souvent un photocopieur faisant également office d'imprimante — est généralement placée dans un local technique ou un couloir, afin d'être facilement accessible par tous.

Il est possible que cette imprimante soit hors service, bien souvent par manque de papier. Il est donc important de vérifier son état avant de lancer une impression. Si jamais on a lancé une impression sur une imprimante qui ne fonctionne pas, il est nécessaire de l'annuler.

10.2.1 La commande `lpq`

La commande `lpq` (pour *line printer queue*) permet d'afficher la file d'attente d'une imprimante. Elle s'utilise avec l'option `-P` (majuscule) suivie du nom de l'imprimante concernée :

```
$ lpq -P speedy
Printer: speedy@se1 (destination speedy@speedy)
 Queue: pas de travail imprimable dans la file d'attente
 Server: pas de serveur actif
 Status: travail 'eleve@cafe01+666' sauvé at 09:42:59.122
 Rank   Owner/ID              Pr/Class Job Files                 Size Time
done   eleve@cafe01+666            A   666 toto.ps            386371 09:42:58
No entries
```

Le plus important est indiqué en dernière ligne : `No entries` signifie que la file d'attente de l'imprimante est vide. On peut donc supposer qu'elle fonctionne correctement et il est alors possible de lancer une impression.

Lorsque l'imprimante est hors service, généralement par manque de papier, l'affichage peut être plus long :

```
1  $ lpq -P speedy
2  Printer: speedy@se1 (destination speedo@speedo)
3   Queue: 3 travail imprimables
4   Server: pid 23030 actif
5   Unspooler: pid 23031  actif
6   Status: waiting for subserver to exit at 17:38:43.797
7   Rank   Owner/ID               Pr/Class Job Files                 Size Time
8  stalled(344sec) eleve@cafe01+666    A   666 toto.ps           18477382 17:34:42
9  2      eleve@cafe01+667             A   667 tata.ps           24567517 17:38:34
10 3      eleve@cafe01+700             A   700 titi.ps           13198435 17:38:43
11 4      eleve@cafe01+534             A   534 tutu.ps           17369403 17:38:53
12 Active: cafe01 printing via LPD
```

Les lignes 8 à 11 indiquent les travaux d'impression en attente, sous la forme d'un affichage en colonnes. Pour chaque travail d'impression en attente sont indiqués :

- sa position dans la file d'attente (colonne `Rank`) ;
- l'identifiant de son propriétaire (colonne `Owner/ID`) ;
- un nombre unique permettant de l'identifier (colonne `Job`) ;
- sa taille en octets (colonne `Size`) ;
- l'heure de sa soumission (colonne `Time`).

Si l'on veut économiser un caractère, on peut se dispenser de l'espace entre l'option `-P` et le nom de l'imprimante :

```
$ lpq -Pspeedy
[...]
```

http://en.wikipedia.org/wiki/Berkeley_printing_system

10.2.2 La commande `lprm`

Lorsqu'une imprimante est hors service et qu'on a un ou plusieurs travaux d'impression dans sa file d'attente, il est préférable de les en supprimer avant d'utiliser une autre imprimante. En effet, on ne sait pas toujours quand l'imprimante défectueuse sera remise en service et il est gênant de laisser des travaux d'impression en attente. En effet, il est toujours désagréable pour la personne qui redémarre une imprimante d'en voir sortir immédiatement des dizaines, voire des centaines de pages, qui finissent bien souvent à la poubelle.

La commande `lprm` (pour *line printer remove*) permet de supprimer un ou plusieurs travaux d'impression en attente. Elle s'utilise avec l'option `-P` (majuscule)

suivie du nom de l'imprimante concernée et avec le ou les numéros (qui sont indiqués par la commande `lpq` dans la colonne `Job`) des travaux d'impression à supprimer comme arguments :

```
$ lprm -P speedy 666 667
Imprimante speedy@se1:
  vérification des autorisations 'eleve@cafe01+666'
  enlevé de la file d'attente 'eleve@cafe01+666'
  vérification des autorisations 'eleve@cafe01+667'
  enlevé de la file d'attente 'eleve@cafe01+667'
```

On peut également utiliser un `-` (tiret) comme argument pour demander la suppression du premier de ses travaux d'impression sans avoir à indiquer son numéro exact :

```
$ lprm -P speedy -
[...]
```

Si l'on veut économiser un caractère, on peut se dispenser de l'espace entre l'option `-P` et le nom de l'imprimante :

```
$ lprm -Pspeedy 666
[...]
```

http://en.wikipedia.org/wiki/Berkeley_printing_system

10.3 Le format PostScript

Un document quelconque ne peut pas être imprimé tel quel. En effet, les imprimantes ne savent pas traiter la multitude de formats de documents qui existent. Pour imprimer un document, il est nécessaire de le convertir au format *PostScript*, qui est le format standard reconnu par toutes les imprimantes.

Le format PostScript a été conçu par la société Adobe au début des années 1980. Il permet de décrire les éléments graphiques (ligne, arc de cercle, courbe, etc.) ainsi que le texte d'un document de manière vectorielle alors qu'ils étaient auparavant décrits avec une matrice de points. La description vectorielle permet de nombreuses

manipulations sans perte de qualité lors de l'impression et rend les documents indépendants des capacités des imprimantes. Le format PostScript permet également de piloter les imprimantes afin d'en sélectionner certaines caractérisques (impression en recto-verso ou en recto seul, choix du bac à papier, etc.).

En pratique, un fichier PostScript est un fichier texte. Le listage 10.1 (les mots clés correspondant aux actions à effectuer y sont indiqués en gras) contient un exemple de fichier PostScript, dont le rendu graphique est indiqué dans la figure 10.1 page suivante.

```
%!PS-Adobe-3.0

1 setlinewidth

newpath
50 600 moveto
10 60 rlineto
40 0 rlineto
-10 -60 rlineto
-40 0 rlineto
stroke

newpath
95 630 10 0 360 arc
stroke

newpath
95 630 translate
20 rotate
2 1 scale
0 0 17 0 360 arc
stroke

showpage
```

LISTAGE 10.1 – *Exemple de fichier PostScript.*

Nous n'étudierons pas davantage le format PostScript. Néanmoins, il est utile de savoir reconnaître un fichier PostScript. Sa première ligne est caractéristique

FIGURE 10.1 – *Rendu graphique du fichier PostScript du listage 10.1.*

(le numéro de version peut varier en fonction du logiciel qui a produit le fichier PostScript) :

```
%!PS-Adobe-3.0
```

Les fichiers PostScript ont généralement une extension `.ps`.

Avant d'imprimer (ou au lieu d'imprimer), on peut visualiser un fichier PostScript sur écran au moyen de la commande `gv` :

```
$ gv fichier.ps &
```

La commande `gv` ouvre une fenêtre montrant la première page du fichier PostScript et permettant de naviguer à l'intérieur du document, d'en changer l'orientation, le grossissement, etc.

Les seuls fichiers pouvant être imprimés directement sont les fichiers PostScript. Nous allons voir comment procéder au paragraphe 10.4. Ensuite, nous étudierons les cas particuliers des fichiers texte au paragraphe 10.5 page 237 et des fichiers PDF au paragraphe 10.6 page 242. Pour imprimer d'autres types de documents, il faut :

- soit utiliser le logiciel qui les a conçus s'il dispose d'une fonction directe d'impression ou de la possibilité de générer un fichier PostScript ;
- soit utiliser un convertisseur permettant de générer un fichier PostScript à partir du document en question.

> http://fr.wikipedia.org/wiki/PostScript
> http://www.gnu.org/software/gv/

10.4 Impression de fichiers PostScript

La nature vectorielle du format PostScript le rend apte à de nombreuses manipulations — en particulier des similitudes — tout en conservant la même qualité

d'impression. Nous allons étudier comment imprimer puis comment manipuler un fichier PostScript avant impression.

10.4.1 La commande `lpr`

La commande `lpr` (pour *line print*) permet d'imprimer un fichier PostScript. Elle s'utilise avec l'option `-P` (majuscule) suivie du nom de l'imprimante et avec le nom du fichier à imprimer comme argument :

```
$ lpr -P speedy toto.ps
```

Si l'on veut économiser un caractère, on peut se dispenser de l'espace entre l'option `-P` et le nom de l'imprimante :

```
$ lpr -Pspeedy toto.ps
```

La commande `lpr` est un filtre. Elle peut donc être utilisée (sans argument) en raccordant son entrée standard au moyen d'un tuyau à la sortie standard d'une commande affichant du PostScript :

```
$ commande_affichant_du_PostScript | lpr -P speedy
```

> http://en.wikipedia.org/wiki/Lpr
> http://en.wikipedia.org/wiki/Berkeley_printing_system

10.4.2 Les utilitaires PSUtils

Les utilitaires PSUtils sont des commandes permettant de manipuler des fichiers PostScript. Nous allons étudier les deux plus utiles de ces commandes. Celles-ci sont des filtres et affichent leur résultat sur leur sortie standard. Elles peuvent donc être raccordées pour l'impression à la commande `lpr` au moyen d'un tuyau.

> http://www.tardis.ed.ac.uk/~ajcd/psutils/

10.4.2.1 La commande `psselect`

La commande `psselect` permet de sélectionner certaines pages d'un document en vue de n'imprimer que celles-ci. Il est en effet regrettable d'imprimer un document dans son intégralité lorsqu'on n'a besoin que de quelques pages.

Les pages à sélectionner sont indiquées grâce à l'option `-p` immédiatement suivie (sans espace) d'une expression pouvant contenir des numéros de pages ou des intervalles de numéros de pages, le tout séparé par des virgules. Le résultat est affiché sur la sortie standard et peut donc être soit redirigé dans un fichier soit passé à une autre commande (par exemple `lpr`) au moyen d'un tuyau.

Ainsi, l'exemple suivant imprimera les pages 3, 6 à 9 et 15 du fichier PostScript `toto.ps` :

```
$ psselect -p3,6-9,15 toto.ps | lpr -P speedy
[3] [6] [7] [8] [9] [15] Wrote 6 pages, 44409 bytes
```

http://www.tardis.ed.ac.uk/~ajcd/psutils/psselect.html

10.4.2.2 La commande `psnup`

La commande `psnup` permet de réarranger un document PostScript pour imprimer plusieurs pages par feuille de papier.

L'option `-2` permet d'imprimer deux pages par feuille de papier, au format paysage :

```
$ psnup -2 toto.ps | lpr -P speedy
[1] [2] [3] [4] [5] Wrote 5 pages, 42626 bytes
```

L'option `-4` permet d'imprimer quatre pages par feuille de papier, au format portrait :

```
$ psnup -4 toto.ps | lpr -P speedy
[1] [2] [3] Wrote 3 pages, 43308 bytes
```

On peut également combiner `psselect` et `psnup` pour sélectionner quelques pages d'un document et en imprimer plusieurs sur chaque feuille de papier :

```
$ psselect -p1-4 toto.ps | psnup -4 | lpr -P speedy
[1] [2] [3] [4] Wrote 4 pages, 45291 bytes
[1] Wrote 1 pages, 48249 bytes
```

http://www.tardis.ed.ac.uk/~ajcd/psutils/psnup.html

10.5 Impression de fichiers texte

Étant donné la prépondérance des fichiers texte sous UNIX, il existe des commandes dédiées permettant d'imprimer ces fichiers. Ces commandes peuvent également générer les fichiers PostScript correspondants au lieu de les imprimer.

10.5.1 La commande a2ps

La commande a2ps permet d'imprimer un fichier texte ou de le convertir en PostScript.

Pour imprimer un fichier texte, la commande a2ps s'utilise avec l'option -P (majuscule) suivie du nom de l'imprimante et avec le ou les fichiers à imprimer comme arguments :

```
$ a2ps -P speedy toto1.txt toto2.txt
[toto1.txt (standard) : 10 pages sur 5 feuilles]
[toto2.txt (standard) : 1 page sur 1 feuille]
l'identifiant de requête est speedy-90 ( 1 fichier(s) )
[Total : 11 pages sur 6 feuilles] envoyé à l'imprimante « speedy »
[7 lignes enroulées]
```

Si l'on veut économiser un caractère, on peut se dispenser de l'espace entre l'option -P et le nom de l'imprimante :

```
$ a2ps -Pspeedy toto.ps
[...]
```

Par souci d'économie, la commande a2ps imprime par défaut deux pages par feuille de papier, au format paysage, comme on peut le voir dans la figure 10.2 page suivante.

Printed by Marc Baudoin

Dec 11, 09 22:16 **mickeysoft.c** Page 1/2

```
#include <nonsense.h>
#include <lies.h>
#include <spyware.h> /* Microsoft Network Connectivity library */
#include <process.h> /* For the court of law */
#define say(x) lie(x)
#define computeruser ALL_WANT_TO_BUY_OUR_BUGWARE
#define next_year soon
#define the_product_is_ready_to_ship   another_beta_version

void main()

{
  if (latest_window_version>one_month_old)
  {
    if (there_are_still_bugs)
      market(bugfix);
    if (sales_drop_below_certain_point)
      raise(RUMOURS_ABOUT_A_NEW_BUGLESS_VERSION);
  }
  while(everyone_chats_about_new_version)
  {
    make_false_promise(it_will_be_multitasking); /* Standard Call, in
                                                    lie.h */
    if (rumours_grow_wilder)
      make_false_promise(it_will_be_plug_n_play);

    if (rumours_grow_even_wilder)
    {
      market_time=ripe;
      say("It will be ready in one month");
      order(programmers, stop_fixing_bugs_in_old_version);
      order(programmers, start_brainstorm_about_new_version);
      order(marketingstaff, permission_to_spread_nonsense);
      vapourware=TRUE;
      break;
    }
  }
  switch (nasty_questions_of_the_worldpress)
  {
     case WHEN_WILL_IT_BE_READY:
       say("It will be ready in", today+30_days," we're just testing");
       break;
     case WILL_THIS_PLUG_AND_PLAY_THING_WORK:
       say("Yes it will work");
       ask(programmers, why_does_it_not_work);
       pretend(there_is_no_problem);
       break;
     case WHAT_ARE_MINIMAL_HARDWARE_REQUIREMENTS:
       say("It will run on a 8086 with lightning speed due to"
           " the 32 bits architecture");
       inform(INTEL, "Pentium sales will rise skyhigh");
       inform(SAMSUNG, "Start a new memorychip plant"
              "cos all those customers will need at least 32 megs");
       inform(QUANTUM, "Thanks to our fatware your sales will triple");
       get_big_bonus(INTEL, SAMSUNG, QUANTUM);
       break;
     case DOES_MICROSOFT_GET_TOO_MUCH_INFLUENCE:
       say("Oh no, we are just here to make a better world for
     everyone");
       register(journalist, Big_Bill_Book);
       when(time_is_ripe)
       {
```

Dec 11, 09 22:16 **mickeysoft.c** Page 2/2

```
         arrest(journalist);
         brainwash(journalist);
         when(journalist_says_windows95_is_bugfree)
         {
           order(journalist, "write a nice objective article");
           release (journalist);
         }
       }
       break;
  }
  while (vapourware)
  {
     introduction_date++; /* Delay */
     if (no_one_believes_anymore_there_will_be_a_release)
       break;
     say("It will be ready in",today+ONE_MONTH);
  }
  release(beta_version)
  while (everyone_is_dumb_enough_to_buy_our_bugware)
  {
    bills_bank_account += 150*megabucks;
    release(new_and_even_better_beta_version);
    introduce(more_memory_requirements);
    if (customers_report_installation_problems)
    {
     say("that is a hardware problem, not a software problem");
      if (smart_customer_says_but_you_promised_plug_and_play)
      {
        ignore(customer);
        order(microsoft_intelligence_agency, "Keep an eye on this
                    bastard");
      }
    }
    if (there_is_another_company)
    {
      steal(their_ideas);
      accuse(compagny, stealing_our_ideas);
      hire(a_lot_of_lawyers); /* in process.h */
      wait(until_other_company_cannot_afford_another_lawsuit);
      buy_out(other_company);
    }
  }
  /* Now everyone realizes that we sell bugware and they are all angry at
     us */
  order(plastic_surgeon, make_bill_look_like_poor_bastard);
  buy(nice_little_island); hire(harem);
  laugh_at(everyone,
for_having_the_patience_year_after_year_for_another_unfinished_version);
}

void bugfix(void)
{
  charge (a_lot_of_money);
  if (customer_says_he_does_not_want_to_pay_for_bugfix)
    say("It is not a bugfix but a new version");
  if (still_complaints)
  {
    ignore(customer);
    register(customer, Big_Bill_Book);
    /* We'll get him when everyone uses Billware!!*/
  }
}
```

Friday December 11, 2009 /home/babafou/divers/humour/mickeysoft.c 1/1

FIGURE 10.2 – *Fichier imprimé avec la commande* `a2ps`.

Si l'on souhaite faire encore mieux et imprimer quatre pages par feuille de papier, au format portrait, on peut utiliser l'option `-4` :

```
$ a2ps -4 -P speedy toto.txt
[...]
```

Si, en revanche, on ne souhaite imprimer qu'une seule page par feuille de papier, on peut utiliser l'option `-1` :

```
$ a2ps -1 -P speedy toto.txt
[...]
```

Le comportement par défaut de la commande `a2ps` correspond à l'option `-2` :

```
$ a2ps -2 -P speedy toto.txt
[...]
```

En regardant de plus près la figure 10.2 page ci-contre, qui est le résultat de l'impression avec la commande `a2ps` d'un programme en langage C, on peut remarquer que certaines parties du texte sont en gras et d'autres en italique. En effet, `a2ps` sait reconnaître certains types de fichiers texte, dont de nombreux langages de programmation, et les présenter en différenciant leurs éléments syntaxiques de cette façon, ce qui facilite la lecture.

Si l'on utilise une imprimante capable d'imprimer en recto-verso mais qui n'est pas configurée pour le faire par défaut, l'option `-s`, suivie de la valeur `2`, permet d'activer l'impression en recto-verso :

```
$ a2ps -s 2 -P speedy toto.txt
[...]
```

Pour générer un fichier PostScript à partir d'un fichier texte sans l'imprimer, la commande `a2ps` s'utilise avec l'option `-o` (pour *output*) suivie du nom du fichier PostScript à générer et avec le ou les fichiers texte comme arguments :

```
$ a2ps -o toto.ps toto1.txt toto2.txt
[toto1.txt (standard) : 10 pages sur 5 feuilles]
[toto2.txt (standard) : 1 page sur 1 feuille]
[Total : 11 pages sur 6 feuilles] enregistré dans le fichier « toto.ps »
[7 lignes enroulées]
```

La commande `a2ps` est un filtre. Elle peut donc être utilisée (sans argument) en raccordant son entrée standard au moyen d'un tuyau à la sortie standard d'une commande affichant du texte :

```
$ commande_affichant_du_texte | a2ps -P speedy
```

http://www.gnu.org/software/a2ps/

10.5.2 La commande `enscript`

La commande `enscript` permet d'imprimer un fichier texte ou de le convertir en PostScript.

Pour imprimer un fichier texte, la commande `enscript` s'utilise avec l'option `-P` (majuscule) suivie du nom de l'imprimante et avec le ou les fichiers à imprimer comme arguments :

```
$ enscript -P speedy toto1.txt toto2.txt
[ 11 pages x 1 copies ]expédié à «speedy
4 lignes étaient enroulées
```

Si l'on veut économiser un caractère, on peut se dispenser de l'espace entre l'option `-P` et le nom de l'imprimante :

```
$ enscript -Pspeedy toto.ps
[...]
```

La figure 10.3 page suivante montre un exemple de fichier imprimé avec la commande `enscript`.

Contrairement à `a2ps`, la commande `enscript` imprime par défaut une seule page par feuille de papier. L'option `-U` (majuscule) suivie de `2` ou `4` permet d'imprimer deux ou quatre pages par feuille de papier :

```
$ enscript -U 2 -P speedy toto.txt
$ enscript -U 4 -P speedy toto.txt
```

La commande `enscript` sait également différencier les éléments syntaxiques de certains types de fichiers texte en les composant en gras ou en italique. Pour cela, il faut utiliser l'option `-E` :

```
$ enscript -E -P speedy toto.txt
[...]
```

Si l'on utilise une imprimante capable d'imprimer en recto-verso mais qui n'est pas configurée pour le faire par défaut, l'option `-D`, suivie de la valeur `Duplex:true`, permet d'activer l'impression en recto-verso :

```
/home/babafou/ensta/in201/poly/babafou/exemples/fflush.c       Sat Aug 21 13:14:51 1999
#include <sys/types.h>

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define LECTURE  0
#define ECRITURE 1

int main ( int argc , char **argv )
{
   int    tuyau[2] , a , b , c ;
   FILE   *mon_tuyau ;

   if ( pipe ( tuyau ) == -1 )
   {
      perror ( "Erreur dans pipe()" ) ;
      exit ( EXIT_FAILURE ) ;
   }

   switch ( fork ( ) )
   {
   case -1 :   /* erreur */
      perror ( "Erreur dans fork()" ) ;
      exit ( EXIT_FAILURE ) ;
   case 0 :   /* processus fils, lecteur */
      close ( tuyau[ECRITURE] ) ;
      mon_tuyau = fdopen ( tuyau[LECTURE] , "r" ) ;
      if ( mon_tuyau == NULL )
      {
         perror ( "Erreur dans fdopen()" ) ;
         exit ( EXIT_FAILURE ) ;
      }
      fscanf ( mon_tuyau , "%d%d%d" , &a , &b , &c ) ;
      printf ( "[fils] Mon père a écrit %d %d %d\n" ,
               a , b , c ) ;
      fclose ( mon_tuyau ) ;
      exit ( EXIT_SUCCESS ) ;
   default :   /* processus père, écrivain */
      close ( tuyau[LECTURE] ) ;
      mon_tuyau = fdopen ( tuyau[ECRITURE] , "w" ) ;
      if ( mon_tuyau == NULL )
      {
         perror ( "Erreur dans fdopen()" ) ;
         exit ( EXIT_FAILURE ) ;
      }
      fprintf ( mon_tuyau , "1 2 3\n" ) ;
      printf ( "[père] Je viens d'écrire dans le tuyau,\n" ) ;
      printf ( "       mais les données sont encore\n" ) ;
      printf ( "       dans la zone de mémoire tampon.\n" ) ;
      sleep ( 5 ) ;
      fflush ( mon_tuyau ) ;
      printf ( "[père] Je viens de forcer l'écriture\n" ) ;
      printf ( "       des données de la mémoire tampon\n" ) ;
      printf ( "       vers le tuyau.\n" ) ;
      printf ( "       J'attends 5 secondes.\n" ) ;
      sleep ( 5 ) ;
      fclose ( mon_tuyau ) ;
      exit ( EXIT_SUCCESS ) ;
   }
}
```

FIGURE 10.3 – *Fichier imprimé avec la commande* `enscript`.

```
$ enscript -D Duplex:true -P speedy toto.txt
[...]
```

Pour générer un fichier PostScript à partir d'un fichier texte sans l'imprimer, la commande `enscript` s'utilise avec l'option `-o` (pour *output*) suivie du nom du fichier PostScript à générer et avec le ou les fichiers texte comme arguments :

```
$ enscript -o toto.ps toto1.txt toto2.txt
[ 11 pages x 1 copies ] placées dans toto.ps
4 lignes étaient enroulées
```

La commande `enscript` est un filtre. Elle peut donc être utilisée (sans argument) en raccordant son entrée standard au moyen d'un tuyau à la sortie standard d'une commande affichant du texte :

```
$ commande_affichant_du_texte | enscript -P speedy
```

> http://www.gnu.org/software/enscript/
> http://en.wikipedia.org/wiki/Enscript

10.5.3 Impression de fichiers texte codés en UTF-8

De nombreuses imprimantes ne gèrent pas correctement les fichiers texte codés en UTF-8. Il est alors nécessaire de les convertir en ISO 8859-15 au moyen de la commande `iconv` avant de les imprimer.

On peut procéder ainsi avec la commande `a2ps` :

```
$ iconv -f utf-8 -t iso-8859-15 toto.txt | a2ps -P speedy
```

ou ainsi avec la commande `enscript` :

```
$ iconv -f utf-8 -t iso-8859-15 toto.txt | enscript -P speedy
```

> http://en.wikipedia.org/wiki/Iconv

10.6 Impression de fichiers PDF

La commande `pdftops` permet de convertir un fichier PDF en fichier PostScript :

```
$ pdftops toto.pdf
```

Utilisée avec un seul nom de fichier en argument (qui doit être un fichier au format PDF), la commande `pdftops` génère un fichier du même nom en remplaçant l'extension `.pdf` par l'extension `.ps`.

On peut choisir le nom du fichier PostScript généré en l'indiquant en second argument :

```
$ pdftops toto.pdf postscript.ps
```

Si le second argument est un tiret, le fichier PostScript ne sera pas généré mais sera affiché sur la sortie standard, ce qui permet d'utiliser un tuyau, par exemple pour l'imprimer sans créer de fichier temporaire :

```
$ pdftops toto.pdf - | lpr -P speedy
```

Si l'on utilise une imprimante capable d'imprimer en recto-verso mais qui n'est pas configurée pour le faire par défaut, l'option `-duplex` permet de générer un fichier PostScript contenant la directive d'activation de l'impression en recto-verso :

```
$ pdftops -duplex toto.pdf
```

Enfin, il faut signaler l'existence d'une commande `pdf2ps` qui convertit également un fichier PDF en fichier PostScript mais qui génère des fichiers PostScript plus volumineux et de moins bonne qualité que `pdftops`.

11

Connexion à distance

UN SYSTÈME D'EXPLOITATION *multi-utilisateur* — ce qui est le cas d'UNIX depuis sa première version — permet à plusieurs personnes d'utiliser le même ordinateur simultanément, généralement au moyen d'un interpréteur de commandes. Cela est possible en raccordant plusieurs terminaux à l'ordinateur, en s'y connectant via le réseau téléphonique s'il dispose d'un modem ou encore à travers un réseau informatique tel que l'Internet. C'est cette dernière méthode qui est de loin la plus utilisée aujourd'hui.

La connexion à distance à travers un réseau informatique pose le problème de la confidentialité des communications. Il est en effet possible de les espionner soit en mesurant le champ électromagnétique généré par le courant circulant dans les câbles électriques [1] soit en interceptant directement les communications au niveau des équipements électroniques d'interconnexion du réseau informatique.

Autrefois, on utilisait pour se connecter à un ordinateur distant et y lancer un interpréteur de commandes la commande `telnet` (pour *teletype network*). Malheureusement, les informations transmises et reçues par `telnet` le sont en clair, c'est-à-dire sans aucun traitement permettant d'assurer un minimum de confidentialité. Son utilisation est donc tombée en désuétude, sauf pour certains tests qui ne nécessitent pas la transmission d'informations confidentielles.

De même, la commande `ftp` (pour *file transfer protocol*) était utilisée, comme son nom l'indique, pour transférer des fichiers d'un ordinateur à un autre. Elle l'est toujours aujourd'hui mais uniquement pour accéder à des serveurs de fichiers anonymes, dont l'utilisation ne nécessite pas la transmission d'informations confidentielles.

De nos jours, tout type de communication avec un système informatique distant implique l'utilisation du chiffrement dès lors qu'un minimum de confidentialité est nécessaire.

1. Ce qui n'est pas possible avec des câbles en fibre optique.

> http://fr.wikipedia.org/wiki/Telnet
> http://en.wikipedia.org/wiki/Ftp
> http://en.wikipedia.org/wiki/Ftp#Anonymous_FTP

11.1 Le chiffrement

Le *chiffrement* a pour but de protéger des données lors de leur transmission par un canal susceptible d'être espionné. Les données à transmettre, qu'on appelle *message clair*, sont tranformées en données incompréhensibles, qu'on appelle *message chiffré*, au moyen d'un algorithme.

On notera :
- m le message clair ;
- c le message chiffré ;
- f l'algorithme de chiffrement ;
- C_f la fonction de chiffrement associée à l'algorithme f ;
- D_f la fonction de déchiffrement associée à l'algorithme f.

On a donc :

$$\mathrm{C}_f(m) = c$$

Le message chiffré peut alors être transmis à son destinataire. Pour retrouver le message clair, celui-ci appliquera au message chiffré la fonction de déchiffrement :

$$\mathrm{D}_f(c) = m$$

À propos du chiffrement, on entend souvent les termes « cryptage », « crypter », voire « encryption » ou « décryption ». Ce sont tous des termes incorrects, qui n'existent pas en français. On parle de chiffrement pour l'action de chiffrer un message clair et de déchiffrement pour l'action de déchiffrer un message chiffré. En revanche, les termes « décryptage » et « décrypter » existent et correspondent au fait de retrouver tout ou partie du message clair à partir du message chiffré sans utiliser la fonction de déchiffrement. Donc le destinataire légitime d'un message chiffré le déchiffre alors qu'un espion le décrypte.

L'algorithme seul ne suffit pas à assurer la sécurité du chiffrement. Si c'était le cas, il faudrait qu'il soit secret, uniquement connu des personnes ayant à l'utiliser. Or l'expérience prouve que les algorithmes de chiffrement secrets ne le restent pas longtemps. Les bons algorithmes de chiffrement sont donc publics, ce qui permet aux spécialistes de les étudier et de confirmer ou d'infirmer leur qualité.

Il est donc nécessaire d'introduire de la variabilité dans les algorithmes. Pour cela, on utilise une *clé*, qui est un paramètre de l'algorithme. Il s'agit d'une longue chaîne de caractères ou d'un nombre très grand. La fonction de chiffrement sera donc appliquée au message clair et à la clé. Le chiffrement d'un même message clair avec deux clés différentes donnera deux messages chiffrés totalement différents, compliquant un éventuel décryptage.

Il existe deux grandes familles d'algorithmes de chiffrement, selon la nature de la clé utilisée : le chiffrement symétrique et le chiffrement asymétrique.

Le lecteur curieux pourra se reporter aux ouvrages suivants de la bibliographie pour approfondir ses connaissances :

- *Applied Cryptography* [58]
- *Handbook of Applied Cryptography* [57]

> http://fr.wikipedia.org/wiki/Chiffrement
> http://www.bortzmeyer.org/cryptage-n-existe-pas.html

11.1.1 Chiffrement symétrique

L'algorithme de chiffrement symétrique s utilise la même clé k pour la fonction de chiffrement C_s et la fonction de déchiffrement D_s :

$$C_s(m,k) = c \qquad D_s(c,k) = m$$

L'expéditeur et le destinataire du message chiffré doivent donc tous deux disposer de cette clé. Le problème initial de la transmission du message se retrouve alors transposé à celui de la transmission de la clé. Néanmoins, une même clé peut être utilisée pour chiffrer et déchiffrer de nombreux messages.

Les algorithmes symétriques utilisent des clés de petite taille (on considère généralement qu'une clé d'au moins 128 bits est sûre compte tenu des moyens de calcul actuels) et sont plutôt rapides.

> http://fr.wikipedia.org/wiki/Cryptographie_symétrique

11.1.2 Chiffrement asymétrique

L'algorithme de chiffrement asymétrique a utilise un couple de clés. Ces clés sont générées ensemble (elles doivent l'être car elles entretiennent des relations mathématiques entre elles) par le destinataire du message.

La *clé publique* k_c peut, comme son nom l'indique, être diffusée publiquement sans risque de sécurité. La clé publique du destinataire sert au chiffrement du message par son expéditeur :

$$\mathrm{C}_a(m, k_c) = c$$

La *clé privée* k_d doit, au contraire, rester seulement connue de son propriétaire, le destinataire du message, et sert à son déchiffrement :

$$\mathrm{D}_a(c, k_d) = m$$

Le chiffrement asymétrique résoud le problème de transmission de la clé intrinsèque au chiffrement symétrique. Néanmoins, les algorithmes asymétriques utilisent des clés de grande taille (on considère généralement qu'une clé d'au moins 1 024 bits est sûre compte tenu des moyens de calcul actuels) et sont sensiblement plus lents que les algorithmes symétriques.

Il faut par ailleurs que l'expéditeur s'assure au préalable que la clé publique k_c est bien celle du destinataire (et non celle d'un imposteur qui pourrait alors déchiffrer les messages). À cet effet, on utilise généralement l'*empreinte* (*fingerprint* en anglais) de la clé publique. Une empreinte est une chaîne de caractères assez courte obtenue à partir de données plus volumineuses et permettant d'identifier celles-ci. Il est alors facile de vérifier l'empreinte de la clé publique k_c auprès de son propriétaire.

http://fr.wikipedia.org/wiki/Cryptographie_asymétrique

11.1.3 Chiffrement hybride

En pratique, les logiciels de chiffrement utilisent un mélange de chiffrement symétrique et de chiffrement asymétrique.

Le destinataire du message dispose d'un couple de clés asymétriques (k_c, k_d). L'expéditeur génère une clé symétrique aléatoire k et chiffre le message m avec :

$$\mathrm{C}_s(m, k) = c$$

Il chiffre également la clé symétrique avec la clé publique du destinataire :

$$\mathrm{C}_a(k, k_c) = k'$$

L'ensemble (c, k') est envoyé au destinataire. Celui-ci commence par déchiffrer la clé symétrique au moyen de sa clé privée :

$$D_a(k', k_d) = k$$

Ce qui lui permet finalement de déchiffrer le message :

$$D_s(c, k) = m$$

Cette façon de combiner chiffrements symétrique et asymétrique permet donc de profiter de leurs avantages respectifs, à savoir la rapidité de traitement du chiffrement symétrique et la possibilité de diffusion de la clé publique du chiffrement asymétrique. Ce dernier n'étant utilisé que pour chiffrer la clé symétrique, sa lenteur n'est pas un problème.

http://fr.wikipedia.org/wiki/Cryptographie_hybride

11.2 La commande ssh

La commande `ssh` (pour *secure shell*) permet de lancer un interpréteur de commandes sur un ordinateur distant depuis l'ordinateur sur lequel on est physiquement connecté. La communication entre les deux ordinateurs se fait au moyen d'un système de chiffrement hybride.

Le lecteur curieux pourra se reporter aux ouvrages suivants de la bibliographie pour approfondir ses connaissances :

- *SSH, The Secure Shell : The Definitive Guide* [13]
- *Pro OpenSSH* [27]
- *SSH Mastery* [19]

http://www.openssh.com/
http://fr.wikipedia.org/wiki/Secure_Shell

11.2.1 Connexion simple

Afin de clarifier les exemples qui suivent, il y sera fait usage de l'invite exacte de l'environnement informatique de l'ENSTA ParisTech au lieu de l'invite simplifiée `$`. Dans ces exemples, nous nous connecterons à l'ordinateur `olive01` depuis l'ordinateur `cafe01`. Par ailleurs, les commandes et interactions devant être saisies par l'utilisateur à chaque étape seront indiquées en gras.

Pour se connecter à un ordinateur distant, il faut indiquer son nom en argument de la commande `ssh` :

```
babafou@cafe01:/home/prof/babafou>ssh olive01
The authenticity of host 'olive01 (147.250.9.11)' can't be established.
RSA key fingerprint is 2e:b2:82:fd:db:11:f1:35:f4:c0:5e:ac:8d:7c:f4:e8.
Are you sure you want to continue connecting (yes/no)? ■
```

Lorsqu'on se connecte à un ordinateur distant pour la première fois, `ssh` affiche l'empreinte de sa clé publique (voir le paragraphe 11.1.2 page 247). Afin de vérifier la validité de cette empreinte, on peut se connecter physiquement sur l'ordinateur distant, si cela est possible, et utiliser la commande `ssh-keygen`, dont l'option `-l` permet d'afficher l'empreinte de la clé publique dont le chemin d'accès est indiqué par l'option `-f` (ce chemin d'accès peut varier en fonction du système UNIX utilisé) :

```
babafou@olive01:/home/prof/babafou>ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key
2048 2e:b2:82:fd:db:11:f1:35:f4:c0:5e:ac:8d:7c:f4:e8 /etc/ssh/ssh_host_rsa_key.pub
```

On peut constater que l'empreinte affichée par `ssh-keygen` est bien la même que celle affichée par `ssh`.

Néanmoins, bien souvent, il n'est pas possible ou pas pratique de se connecter physiquement sur un ordinateur dans le seul but d'afficher l'empreinte de sa clé publique. Si l'on ne dispose pas d'une autre méthode pour vérifier cette empreinte (par exemple, une liste d'empreintes fournie par l'administrateur du système), on fait généralement l'impasse sur cette vérification.

Revenons à notre connexion à distance. Pour poursuivre la procédure après l'affichage par `ssh` de l'empreinte de la clé publique de l'ordinateur distant, il faut taper `yes` en toutes lettres (ou `no` si l'on souhaite l'interrompre) :

```
babafou@cafe01:/home/prof/babafou>ssh olive01
The authenticity of host 'olive01 (147.250.9.11)' can't be established.
RSA key fingerprint is 2e:b2:82:fd:db:11:f1:35:f4:c0:5e:ac:8d:7c:f4:e8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'olive01,147.250.9.11' (RSA) to the list of known hosts.
babafou@olive01's password: ■
```

Cette étape n'est nécessaire que lors de la première connexion à un ordinateur distant. En effet, `ssh` conserve les clés publiques des ordinateurs sur lesquels on s'est déjà connecté dans le fichier `~/.ssh/known_hosts` et ne demande pas confirmation

si la clé publique de l'ordinateur auquel on se connecte figure déjà dans ce fichier. En revanche, si jamais elle changeait (en raison d'une réinstallation du système, par exemple), il serait demandé à nouveau confirmation.

Après saisie du mot de passe (qui ne s'affiche pas), on se retrouve avec un interpréteur de commande fonctionnant sur l'ordinateur distant, comme le prouve son invite :

```
babafou@cafe01:/home/prof/babafou>ssh olive01
The authenticity of host 'olive01 (147.250.9.11)' can't be established.
RSA key fingerprint is 2e:b2:82:fd:db:11:f1:35:f4:c0:5e:ac:8d:7c:f4:e8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'olive01,147.250.9.11' (RSA) to the list of known hosts.
babafou@olive01's password: mot_de_passe
babafou@olive01:/home/prof/babafou>
```

Si l'invite de l'interpréteur de commandes n'affiche pas le nom de l'ordinateur sur lequel il fonctionne, ou simplement si l'on souhaite afficher celui-ci, on peut utiliser la commande `hostname`, qui affiche le nom de l'ordinateur sur lequel on se trouve :

```
babafou@olive01:/home/prof/babafou>hostname
olive01
```

On peut ensuite utiliser sur l'ordinateur distant toutes les commandes utilisables dans un interpréteur de commandes (à l'exception des logiciels graphiques, qui seront abordés au pragraphe 11.2.2).

Pour quitter l'interpréteur de commandes sur l'ordinateur distant, on utilise la commande `exit` :

```
babafou@olive01:/home/prof/babafou>exit
logout
Connection to olive01 closed.
babafou@cafe01:/home/prof/babafou>
```

11.2.2 Connexion avec affichage graphique déporté

Il est possible grâce à la commande `ssh` d'exécuter des logiciels graphiques sur un ordinateur distant en les faisant s'afficher sur son écran. Pour cela, il faut utiliser `ssh` avec l'option `-X` lors de la connexion à l'ordinateur distant :

```
$ ssh -X ordinateur
```

Ceci lancera un interpréteur de commandes sur l'ordinateur distant après saisie du mot de passe, comme vu précédemment. L'option `-X` permet en plus d'utiliser la connexion chiffrée entre les deux ordinateurs pour déporter l'affichage des applications graphiques.

Cela peut ne pas fonctionner correctement avec certains ordinateurs. Dans ce cas, on peut utiliser plutôt l'option `-Y`

```
$ ssh -Y ordinateur
```

11.2.3 Connexion avec un identifiant différent

Il peut arriver d'avoir à se connecter sur un ordinateur distant avec un identifiant différent de celui utilisé sur l'ordinateur de départ, parce que les règles d'attribution des identifiants sont différentes ici et là-bas ou tout simplement parce qu'on emprunte le compte de quelqu'un pour se connecter ailleurs.

Dans ce cas, il faut utiliser la commande `ssh` avec l'option `-l` suivie de l'identifiant sur l'ordinateur distant :

```
$ ssh -l identifiant ordinateur
```

On peut aussi utiliser, sans l'option `-l`, une syntaxe ressemblant à une adresse électronique :

```
$ ssh identifiant@ordinateur
```

11.2.4 Connexion depuis l'extérieur de l'ENSTA ParisTech

La connexion depuis l'extérieur de l'ENSTA ParisTech vers un ordinateur de son réseau interne nécessite l'utilisation d'une procédure particulière, en raison d'un dispositif de sécurité contrôlant les connexions en entrée de l'école.

Pour cela, on utilise la commande `ssh` :

1. avec l'option `-t` (dont l'explication est trop technique pour un cours d'introduction à l'informatique) ;
2. avec l'option `-l` suivie de son identifiant dans l'environnement informatique de l'ENSTA ParisTech ;
3. en indiquant `relais.ensta.fr` comme premier argument (cet ordinateur est le dispositif de sécurité indiqué auparavant) ;
4. en indiquant le nom de l'ordinateur sur lequel on souhaite se connecter comme second argument.

Ce qui donne cette commande :

```
$ ssh -t -l identifiant relais.ensta.fr ordinateur
```

Le mot de passe de l'utilisateur sera demandé à deux reprises, la première fois par `relais.ensta.fr`, la seconde par l'ordinateur sur lequel on souhaite se connecter.

11.3 La commande scp

La commande `scp` (pour *secure copy*) permet de copier un fichier d'un ordinateur à un autre de manière chiffrée, en utilisant le même canal de communication que la commande `ssh`.

Pour copier un fichier `fichier` sur un ordinateur distant `ordinateur`, on utilise :

```
$ scp fichier ordinateur:
```

Le deux-points collé au nom de l'ordinateur distant est indispensable et permet d'identifier lequel des deux arguments de la commande `scp` est l'ordinateur distant.

Si rien n'est indiqué derrière ce deux-points, le fichier `fichier` est copié sous le même nom dans le répertoire personnel de l'utilisateur sur l'ordinateur distant. On peut indiquer après le deux-points le chemin d'accès sous lequel copier le fichier :

```
$ scp fichier ordinateur:rep1/rep2
```

En sens inverse, on peut copier un fichier depuis l'ordinateur distant :

```
$ scp ordinateur:fichier .
```

Pour copier un fichier vers ou depuis un ordinateur sur lequel on dispose d'un identifiant différent, il faut indiquer cet identifiant devant le nom de l'ordinateur, en intercalant une arrobe `@` entre les deux (l'option `-l` de `ssh` a une signification différente pour `scp`) :

```
$ scp fichier identifiant@ordinateur:
```

http://fr.wikipedia.org/wiki/Secure_copy

11.4 Savoir qui est connecté sur un ordinateur : les commandes `who` et `w`

La commande `who` permet d'afficher la liste des utilisateurs qui sont connectés à l'ordinateur sur lequel on travaille :

```
$ who
eleve    tty7         2010-09-21 13:41 (:0)
babafou  pts/1        2010-09-21 15:57 (cafe01.ensta.fr)
```

Dans l'exemple précédent, l'utilisateur `eleve` est physiquement connecté à l'ordinateur (ce qu'indique le `:0` entre parenthèses dans la dernière colonne) alors que l'utilisateur `babafou` est connecté depuis la machine `cafe01.ensta.fr`, probablement au moyen de la commande `ssh`.

Hormis pour la brièveté de son nom, on lui préfère généralement la commande `w`, dont l'affichage est plus complet. Ainsi, dans les mêmes conditions que pour l'exemple précédent, la commande `w` affiche :

```
$ w
 15:58:53 up  2:37,  2 users,  load average: 0,60, 0,44, 0,21
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
eleve    tty7     :0               13:41    1.00s  1:12   0.28s bash
babafou  pts/1    cafe01.ensta.fr  15:57    0.00s  0.25s  0.00s w
```

On retrouve à partir de la deuxième ligne un habituel affichage en colonnes.

> http://fr.wikipedia.org/wiki/Who_(Unix)
> http://en.wikipedia.org/wiki/W_(Unix)

12 Programmation

L'INTERPRÉTEUR DE COMMANDES ne permet pas seulement l'exécution de commandes isolées, aussi complexes soient-elles, il dispose également d'un langage de programmation complet permettant d'enchaîner des commandes au moyen de tests et de boucles. Les programmes peuvent être saisis soit en ligne de commande lorsqu'ils sont suffisamments courts soit dans un fichier lorsqu'ils sont plus longs ou qu'on souhaite les conserver pour les réutiliser.

Il n'existe pas un seul langage de programmation mais deux. La syntaxe de celui du Bourne shell (`sh`) a été reprise par les Korn shell (`ksh`) et Bourne-again shell (`bash`). Celle introduite par le C shell (`csh`) est différente et a été reprise par le TENEX C shell (`tcsh`). Quant au Z shell (`zsh`), il accepte indifféremment les deux syntaxes. Le paragraphe 7.1 page 88 vous rappellera l'historique des interpréteurs de commandes et les relations généalogiques qu'ils entretiennent entre eux.

Nous étudierons par la suite le langage de programmation du Bourne shell, qui est de loin le plus utilisé.

Le lecteur curieux pourra se reporter aux ouvrages suivants de la bibliographie pour approfondir ses connaissances :

- *Programmation shell sous Unix/Linux* [14]
- *Classic Shell Scripting* [25]
- *Sams Teach Yourself Shell Programming in 24 Hours* [29]

12.1 Pourquoi programmer ?

Il existe des situations dans lesquelles les différents outils syntaxiques étudiés jusqu'ici ne sont pas suffisants pour exprimer l'action qu'on souhaite accomplir. Considérons l'exemple suivant : renommer dix fichiers nommés `toto0` jusqu'à `toto9` (qui existent) en `tata0` jusqu'à `tata9` (qui n'existent pas).

Compte tenu des similitudes entre les noms des fichiers `toto0` à `toto9` et `tata0` à `tata9`, on pense immédiatement à utiliser un métacaractère. La commande suivante conviendrait-elle ?

```
$ mv toto? tata?
```

Malheureusement non car les métacaractères ne permettent de générer que des noms de fichiers qui existent et les fichiers dont le nom commence par `tata` n'existent pas encore. Qui plus est, la syntaxe de la commande `mv` permet soit de renommer un seul fichier soit de déplacer plusieurs fichiers mais elle ne permet pas de renommer plusieurs fichiers en même temps.

Faut-il pour autant se résoudre à utiliser dix fois la commande `mv` pour renomer les fichiers un par un ? Oui, c'est la seule façon de faire. Mais il existe une méthode plus efficace que de taper dix commandes, c'est d'utiliser un programme. Nous verrons la solution à ce problème au paragraphe 12.7.2.1 page 271.

12.2 Comment programmer ?

Comme M. Jourdain disait de la prose sans qu'il le sût, nous programmons depuis le début sans le savoir. En effet, que ce soit une simple commande sans même une option ou un argument ou bien encore un enchaînement de commandes reliées par des tuyaux, tout cela est déjà un programme.

Plus généralement, un programme est constitué d'une suite de commandes et d'instructions permettant d'en contrôler l'exécution (que nous étudierons au paragraphe 12.7 page 265).

12.2.1 Les scripts

Néanmoins, un programme est rarement saisi en ligne de commande, sauf s'il est très court et n'a pas besoin d'être réutilisé. Le plus souvent, un programme est saisi dans un fichier qu'on appelle un *script*. Par métonymie, on appelle également script le programme contenu dans ce fichier.

Les scripts en Bourne shell sont habituellement stockés dans des fichiers ayant une extension `.sh`.

12.2.2 Exécution d'un script

Considérons le script suivant, stocké dans le fichier `toto.sh` :

```
cal
date
```

On y retrouve deux commandes bien connues, chacune sur une ligne. La commande `cal` sera exécutée en premier, puis viendra le tour de la commande `date`. Un script est en effet exécuté ligne par ligne, de haut en bas. La fin de ligne dans le fichier fait office d'appui sur la touche Enter .

Pour exécuter ce script, on peut l'indiquer en argument de la commande `sh` :

```
$ sh toto.sh
```

Il existe également deux autres syntaxes spécifiques à certains interpréteurs de commandes.

Avec `bash`, `csh`, `tcsh` et `zsh`, on peut faire :

```
$ source toto.sh
```

Avec `bash` uniquement, on peut faire :

```
$ . toto.sh
```

Mais la façon la plus commune de faire est de rajouter une ligne en tête du script :

```
#! /bin/sh
```

et de le rendre exécutable :

```
$ chmod 755 toto.sh
```

Notre script `toto.sh` devient donc :

```
#! /bin/sh

cal
date
```

La ligne blanche suivant la première ligne n'est pas nécessaire mais permet de bien la séparer de la suite du script.

Le script peut alors être exécuté, comme expliqué au paragraphe 9.2.4 page 179, de cette façon :

```
$ ./toto.sh
```

La présence des deux caractères `#!` au début du script (et qui doivent absolument être les deux premiers caractères du script) indique à UNIX que, lorsqu'on l'exécute, il doit en fait exécuter la commande dont le chemin d'accès absolu suit ces deux caractères, soit ici `/bin/sh`, en lui passant le nom du script en argument, soit ici `toto.sh`, ce qui revient à la première méthode indiquée précédemment pour exécuter le script (puisque le chemin d'accès absolu de `sh` est `/bin/sh`, ce qui est facile à vérifier grâce aux commandes `type` ou `which`).

L'espace entre `#!` et `/bin/sh` n'est pas indispensable mais permet là encore de bien séparer les choses.

Libre à chacun d'exécuter ses scripts comme bon lui semble. Afin de ne pas les encombrer inutilement, les prochains exemples ne contiendront pas la ligne :

```
#! /bin/sh
```

Il faudra donc la rajouter (et ne pas oublier de rendre le script exécutable) si l'on souhaite utiliser cette méthode.

http://en.wikipedia.org/wiki/Shebang_(Unix)

12.3 Continuation de ligne

Un script est constitué d'une suite de commandes et d'instructions de contrôle (que nous étudierons au paragraphe 12.7 page 265). Celles-ci figurent généralement sur une seule ligne mais peuvent s'étendre sur plusieurs lignes consécutives au besoin. Pour cela, on utilise une barre oblique inversée `\` en fin de ligne afin d'indiquer que celle-ci se poursuit sur la ligne suivante.

Ainsi, la commande :

```
ps ux | grep xeyes | grep -v grep | awk '{print $2}'
```

peut être scindée en plusieurs lignes comme ceci :

```
ps ux | grep xeyes \
      | grep -v grep \
      | awk '{print $2}'
```

12.4 Les commentaires

Il peut être utile d'inclure dans un script des indications n'en faisant pas partie mais destinées à éclairer les personnes qui seront amenées à le lire (c'est souvent le cas de son auteur lorsqu'il le relit après un certain temps et qu'il se demande ce qu'il a bien pu essayer de faire). C'est ce qu'on appelle des *commentaires*.

Les commentaires commencent par un croisillon # et s'étendent jusqu'à la fin de la ligne. Ils peuvent figurer seuls sur une ligne ou en fin de ligne.

Ainsi, l'exemple précédent peut être commenté de la sorte :

```
# obtention de la liste des identifiants des processus xeyes

ps ux                       # on établit la liste des processus
      | grep xeyes \        # on y recherche xeyes
      | grep -v grep \      # on en supprime grep
      | awk '{print $2}'    # et on extrait la deuxième colonne
```

Les continuations de lignes, autrement peu utiles parce que l'enchaînement complet est suffisamment lisible sur une seule ligne, permettent ici de placer chaque commande de l'enchaînement sur sa propre ligne afin de lui affecter un commentaire dédié. Par ailleurs, les commentaires sont alignés verticalement pour en faciliter la lecture.

12.5 Les variables

Une *variable* permet de stocker une valeur afin de pouvoir l'utiliser ultérieurement.

Là où certains langages de programmation distinguent plusieurs types de variables en fonction des valeurs qu'elles peuvent stocker (nombres entiers, nombres à

virgule flottante, chaînes de caractères, etc.), l'interpréteur de commandes n'a qu'un seul type de variable, qui permet de stocker des chaînes de caractères.

Il existe trois catégories de variables :

- les variables spéciales, dont le nom est constitué d'un symbole, d'un chiffre ou d'une chaîne de caractères alphanumériques et dont la valeur est fixée par l'interpréteur de commandes (nous avons déjà étudié les variables `?` et `status` au paragraphe 9.3.6.1 page 190, nous en étudierons d'autres au paragraphe 12.8 page 273) ;
- les variables d'environnement, dont le nom est traditionnellement constitué de lettres majuscules et dont la valeur est soit fixée dans la configuration de l'interpréteur de commandes soit héritée de son père (nous avons par exemple étudié la variable d'environnement `PATH` au paragraphe 9.2.2 page 178) ;
- les variables définies par l'utilisateur, qui sont celles que nous allons étudier maintenant.

Les noms des variables définies par l'utilisateur peuvent contenir des lettres (on fait la différence entre majuscules et minuscules), des chiffres ou des tirets bas. Ils ne peuvent pas commencer par un chiffre. Les lettres majuscules étant traditionnellement réservées aux variables d'environnement, on ne les utilise généralement pas.

Une variable est créée en lui affectant une valeur, selon la syntaxe suivante :

```
nom=valeur
```

Il ne faut mettre absolument aucune espace autour du signe égal.

Si la valeur contient des espaces, chacune d'elle doit être précédée d'une barre oblique inversée `\` ou la valeur doit être entourée par une paire de guillemets ou d'apostrophes :

```
nom=valeur\ avec\ espaces
nom="valeur avec espaces"
nom='valeur avec espaces'
```

On peut donc faire, par exemple :

```
var1=toto
var2="titi tutu"
```

On accède à la valeur d'une variable en faisant précéder son nom d'un symbole dollar `$`. Ainsi, on peut afficher le contenu d'une variable en utilisant la commande `echo` :

```
echo $variable
```

Un exemple plus complexe combine plusieurs commandes reliées par des tuyaux et une substitution :

```
pid=‘ps ux | grep xeyes | grep -v grep | awk ’{print $2}’‘
kill $pid
```

La substitution est très utile pour affecter à une variable la sortie standard d'une commande ou d'un enchaînement de commandes.

Un autre exemple utilise deux variables pour constituer la valeur d'une autre :

```
pid1=33
pid2=51
pid="$pid1 $pid2"
```

La valeur de la variable `pid` est alors :

```
33 51
```

Les choses auraient été différentes si l'on avait utilisé des apostrophes à la place des guillemets :

```
pid1=33
pid2=51
pid=’$pid1 $pid2’
```

La valeur de la variable `pid` serait alors :

```
$pid1 $pid2
```

En effet, dans une expression entre guillemets, les variables précédées d'un symbole dollar sont remplacées par leur valeur. C'est d'ailleurs le cas général, sauf dans une expression entre apostrophes, qui n'est pas soumise à ce traitement et reste en l'état.

C'est en particulier pour cela qu'on fait suivre la commande `awk` d'une expression entre apostrophes. Celle-ci contenant un `$2` qui est destiné à `awk` et pas à l'interpréteur de commandes, il est nécessaire d'empêcher celui-ci de remplacer `$2` par autre chose, voire de générer une erreur si cette variable n'existe pas pour lui.

12.6 Le calcul entier

L'interpréteur de commandes permet de réaliser des calculs sur des valeurs entières. L'expression à calculer doit être encadrée par `$((` et `))` :

```
a=10
b=$(( 166 * a + 4 ))
echo $b
```

Une expression encadrée par `$((` et `))` peut utiliser les opérateurs indiqués dans le tableau 12.1.

`+`	addition
`-`	soustraction
`*`	multiplication
`/`	quotient de la division euclidienne
`%`	reste de la division euclidienne

TABLE 12.1 – *Opérateurs utilisables dans une expression encadrée par `$((` et `))`.*

Les opérateurs `*`, `/` et `%` sont plus prioritaires que les opérateurs `+` et `-`. Les multiplications et les calculs des quotients et restes de divisions euclidiennes sont donc effectués avant les additions et soustractions. À priorité égale, les calculs sont effectués de gauche à droite.

Il n'est pas nécessaire de faire précéder d'un symbole dollar les noms des variables apparaissant dans une expression encadrée par `$((` et `))` mais on peut le faire si on le souhaite.

Une expression encadrée par `$((` et `))` peut contenir des espaces. Leur utilisation est recommandée car elle permet de bien séparer les différents éléments d'une expression et facilite donc sa lecture.

12.7 Les instructions de contrôle

Les instructions de contrôle permettent d'exécuter des commandes en fonction d'une condition ou de les répéter plusieurs fois.

12.7.1 Les tests

Les tests permettent d'exécuter différentes commandes en fonction d'une condition.

12.7.1.1 La commande `test`

La commande `test` permet d'effectuer de nombreux types de tests sur des fichiers, des chaînes de caractères ou des entiers. Son statut de sortie est `0` si le test est réussi et 1 dans le cas contraire.

La commande `test` s'utilise suivie d'une expression indiquant le test à effectuer et ce à quoi l'appliquer, par exemple :

```
$ test 5 -eq 3
$ echo $?
1
$ test 5 -gt 3
$ echo $?
0
```

Le tableau 12.2 page suivante indique les expressions les plus utiles pouvant suivre la commande `test`.

On peut nier le sens d'une expression en la faisant précéder d'un point d'exclamation `!` :

Expression	Condition de réussite
`-d` *`fichier`*	si *`fichier`* existe et est un répertoire
`-e` *`fichier`*	si *`fichier`* existe (quel que soit son type)
`-f` *`fichier`*	si *`fichier`* existe et est un fichier ordinaire
`-h` *`fichier`*	si *`fichier`* existe et est un lien symbolique
`c1` `=` *`c2`*	si les chaînes de caractères *`c1`* et *`c2`* sont identiques
`c1` `!=` *`c2`*	si les chaînes de caractères *`c1`* et *`c2`* sont différentes
`n1` `-eq` *`n2`*	si les entiers *`n1`* et *`n2`* sont égaux
`n1` `-ne` *`n2`*	si les entiers *`n1`* et *`n2`* sont différents
`n1` `-gt` *`n2`*	si l'entier *`n1`* est strictement supérieur à l'entier *`n2`*
`n1` `-ge` *`n2`*	si l'entier *`n1`* est supérieur ou égal à l'entier *`n2`*
`n1` `-lt` *`n2`*	si l'entier *`n1`* est strictement inférieur à l'entier *`n2`*
`n1` `-le` *`n2`*	si l'entier *`n1`* est inférieur ou égal à l'entier *`n2`*

TABLE 12.2 – *Expressions les plus utiles pouvant suivre la commande* `test`.

```
$ test 5 -eq 3
$ echo $?
1
$ test ! 5 -eq 3
$ echo $?
0
```

On peut également combiner deux expressions avec `-a` (pour *and*, la combinaison est vraie si les deux expressions sont vraies et fausse dans les autres cas) ou `-o` (pour *or*, la combinaison est vraie si l'une des deux expression est vraie et fausse si les deux sont fausses) :

```
$ test 5 -gt 3 -a 2 -lt 7
$ echo $?
0
$ test 5 -gt 3 -a 2 -gt 7
$ echo $?
1
```

```
$ test 5 -gt 3 -o 2 -gt 7
$ echo $?
0
$ test 3 -gt 5 -o 2 -gt 7
$ echo $?
1
```

Enfin, la commande `test` peut également être utilisée sous une forme particulière, en n'indiquant que l'expression entre crochets. Ainsi, le premier exemple peut également s'écrire ainsi :

```
$ [ 5 -eq 3 ]
$ echo $?
1
$ [ 5 -gt 3 ]
$ echo $?
0
```

L'espace après le crochet ouvrant et l'espace avant le crochet fermant sont obligatoires.

Cette façon d'écrire les tests est la plus utilisée dans les scripts.

http://fr.wikipedia.org/wiki/Test_(Unix)

12.7.1.2 Le test `if`

Le test `if` permet, si le statut de sortie d'une commande qui sert de condition est nul, d'exécuter un bloc de commandes :

```
if [ -f toto ]
then
   echo Le fichier toto existe, on le supprime.
   rm toto
fi
```

L'instruction `if` est suivie de la commande à tester. Si son statut de sortie est nul, le bloc de commandes compris entre les mots clés `then` et `fi` est exécuté.

Remarquez que le bloc de commandes est indenté. L'*indentation* consiste à décaler chaque ligne vers la droite en ajoutant quelques espaces (combien ? c'est une affaire de goût personnel mais trois espaces représentent une indentation raisonnable) au début afin de faire apparaître la structure du script. L'indentation est syntaxiquement inutile pour l'interpréteur de commandes mais indispensable au confort de lecture et à une compréhension rapide de la structure du script par des personnes. Une indentation systématique et cohérente est donc plus que recommandée.

Lorsque la condition n'est pas remplie, il est possible d'exécuter un autre bloc de commandes, qu'on ajoute à la suite du premier, précédé de l'instruction `else`. L'instruction `fi` termine l'ensemble :

```
if [ -f toto ]
then
   echo Le fichier toto existe, on le supprime.
   rm toto
else
   echo Le fichier toto n\'existe pas.
fi
```

Enfin, si l'on souhaite tester plusieurs conditions à la suite et exécuter un bloc de commandes pour la première qui sera vérifiée, on peut utiliser l'instruction `elif` (qui est la contraction de `else` et de `if`) :

```
if [ -f toto ]
then
   echo Le fichier toto existe, on le supprime.
   rm toto
elif [ -f tata ]
then
   echo Le fichier tata existe, on le supprime.
   rm tata
elif [ -f titi ]
then
   echo Le fichier titi existe, on le supprime.
   rm titi
else
   echo Aucun des fichiers toto, tata et titi n\'existe.
fi
```

La présence de l'instruction `else` et du bloc correspondant est facultative.

12.7.1.3 Le test `case`

Lorsqu'une chaîne de caractères est à tester pour savoir si elle appartient à un ensemble de valeurs, on peut bien entendu utiliser un test `if` suivi de plusieurs `elif` mais il existe un test `case` dont l'utilisation est plus élégante dans ce cas particulier :

```
case $variable in
   toto )
      echo la variable contient toto
      ;;
   tata )
      echo la variable contient tata
      ;;
   titi )
      echo la variable contient titi
      ;;
esac
```

L'instruction `case` est suivie de la chaîne de caractères à tester (qui est souvent la valeur d'une variable mais qui peut également être le résultat d'une substitution de commande, voir le paragraphe 9.4.6 page 202) puis du mot clé `in`.

On indique ensuite la première valeur à tester par rapport à cette chaîne de caractères, suivie d'une parenthèse fermante `)` (l'espace précédent la parenthèse n'est pas obligatoire). Si la valeur correspond, les commandes qui suivent sont exécutées, jusqu'au double point-virgule `;;` qui les termine. Ceci est répété pour chaque chaîne de caractères à tester. L'indentation n'est pas obligatoire mais néanmoins recommandée.

Enfin, le tout se termine par le mot clé `esac`.

Si les mêmes commandes sont à exécuter pour plusieurs valeurs, on peut regrouper celles-ci au moyen du caractère `|` :

```
case $variable in
   toto | tata )
      echo la variable contient toto ou tata
      ;;
   titi )
      echo la variable contient titi
      ;;
esac
```

Enfin, il est possible d'utiliser les mêmes métacaractères que ceux étudiés au paragraphe 8.4 page 134 pour indiquer les valeurs à tester, à la différence près que ceux-ci s'appliquent alors à des chaînes de caractères et pas à des noms de fichiers :

```
case $variable in
   [0-9]* )
      echo la variable commence par un chiffre
      ;;
   [A-Z]* )
      echo la variable commence par une lettre majuscule
      ;;
   [a-z]* )
      echo la variable commence par une lettre minuscule
      ;;
esac
```

12.7.2 Les boucles

Les boucles permettent de répéter des commandes plusieurs fois.

12.7.2.1 La boucle `for`

La boucle `for` permet d'attribuer successivement chaque valeur d'une liste à une variable et, pour chaque valeur, d'exécuter un bloc de commandes.

C'est ainsi que nous allons résoudre le problème que nous nous étions posé au paragraphe 12.1 page 257 :

```
for i in 0 1 2 3 4 5 6 7 8 9
do
   mv toto$i tata$i
done
```

L'instruction `for` est suivie du nom de la variable à utiliser, qui est créée à cette occasion, puis du mot clé `in` et de la liste de valeurs à attribuer successivement à la variable.

Le bloc de commandes à exécuter pour chaque valeur de la liste est compris entre les mots clés `do` et `done`. L'indentation n'est pas obligatoire mais néanmoins recommandée.

Dans notre exemple, la variable `i` va prendre successivement les valeurs de `0` à `9`. Compte tenu de l'utilisation de la valeur de cette variable dans le bloc de la boucle `for`, la commande suivante sera exécutée lors de la première itération (`i` ayant la valeur `0`) :

```
mv toto0 tata0
```

Et ainsi de suite. Si le problème posé au paragraphe 12.1 page 257 avait concerné le renommage des fichiers `toto00` jusqu'à `toto99` en `tata00` jusqu'à `tata99`, on aurait utilisé deux boucles imbriquées :

```
for i in 0 1 2 3 4 5 6 7 8 9
do
   for j in 0 1 2 3 4 5 6 7 8 9
   do
      mv toto$i$j tata$i$j
   done
done
```

La liste de valeurs peut également être générée grâce à des métacaractères :

```
for f in *.ps
do
   psnup -4 $f | lpr -P imprimante
done
```

ou une substitution de commande :

```
for p in `ps ux | grep xeyes | grep -v grep | awk '{print $2}'`
do
   echo on tue le processus $p
   kill $p
done
```

12.7.2.2 Les boucles `while` et `until`

La boucle `while` permet d'exécuter un bloc de commandes tant que le statut de sortie d'une commande est nul :

```
a=0
while [ $a -lt 5 ]
do
   echo $a
   a=$(( a + 1 ))
done
```

L'instruction `while` est suivie de la commande à tester. Si son statut de sortie est nul, le bloc de commandes compris entre les mots clés `do` et `done` est exécuté. L'indentation n'est pas obligatoire mais néanmoins recommandée.

Le script précédent affichera donc :

```
0
1
2
3
4
```

Il est important de vérifier formellement que la commande à tester, qu'on appelle *condition d'arrêt*, finira par avoir un statut de sortie non nul, permettant d'interrompre la boucle `while`. Dans le cas contraire, on bouclera à l'infini.

La boucle `until` permet d'exécuter un bloc de commandes jusqu'à ce que le statut de sortie d'une commande soit nul. La condition d'arrêt est donc inversée par rapport à la boucle `while`. L'exemple précédent peut s'écrire ainsi avec une boucle `until` :

```
a=0
until [ $a -ge 5 ]
do
   echo $a
   a=$(( a + 1 ))
done
```

12.8 Les arguments

Il est possible d'exécuter un script en lui indiquant des arguments :

```
$ ./toto.sh arg1 arg2 arg3
```

Dans le script, le nombre d'arguments est contenu dans la variable spéciale `#` et la liste des arguments est contenue dans la variable spéciale `@` :

```
echo le script a $# arguments :

for a in "$@"
do
   echo $a
done
```

Il est important de toujours utiliser la valeur de la variable spéciale `@` entre guillemets. Sans cela, un argument contenant des espaces sera décomposé en la suite de mots qu'il contient.

Un exemple sera plus parlant. Le script précédent, exécuté avec trois arguments dont l'un contient une espace, affiche ceci :

```
$ ./toto.sh arg1 'arg2 arg2bis' arg3
le script a 3 arguments :
arg1
arg2 arg2bis
arg3
```

Le deuxième argument du script, qui contient deux mots séparés par une espace, a bien été traité dans sa totalité.

Si l'on supprime les guillemets autour de `$@` dans le script, on aura ceci en exécutant le script avec les mêmes arguments :

```
$ ./toto.sh arg1 'arg2 arg2bis' arg3
le script a 3 arguments :
arg1
arg2
arg2bis
arg3
```

Le nombre d'arguments est correct car la valeur de la variable spéciale `#` n'est bien entendu pas impactée par la présence ou l'absence de guillemets autour de `$@`. En revanche, les deux mots du deuxième argument du script ont été séparés et traités indépendamment dans la boucle `for`.

12.9 Les fonctions

Une fonction permet d'écrire un morceau de script dans le but de l'utiliser à plusieurs reprises.

La définition d'une fonction est de la forme :

```
fonction ()
{
   corps de la fonction
}
```

La première ligne indique le nom de la fonction, qui doit être suivi d'un couple de parenthèses.

Suit le corps de la fonction, qui est entouré d'un couple d'accolades. Le corps de la fonction est constitué des commandes et des instructions permettant de réaliser l'objectif de la fonction. L'indentation n'est pas obligatoire mais néanmoins recommandée.

Une fonction doit être définie avant de pouvoir être utilisée. C'est pourquoi on regroupe généralement les définitions de fonctions en début de script.

Pour exécuter une fonction, c'est-à-dire exécuter les commandes et instructions constituant son corps, il suffit d'utiliser le nom de la fonction comme on le ferait avec une commande.

Voici un exemple très simple de script définissant puis utilisant une fonction :

```
toto ()
{
   echo nous sommes dans la fonction toto
}

toto
```

Il est possible d'exécuter une fonction en lui indiquant des arguments, de la même façon qu'avec une commande. À l'intérieur de la fonction, le nombre et la liste des arguments sont contenus dans les mêmes variables spéciales que celles abordées au paragraphe 12.8 page 273 :

```
toto ()
{
   echo la fonction a $# arguments :

   for a in "$@"
   do
      echo $a
   done
}

toto arg1 arg2 arg3
```

Ces variables spéciales ne contiennent les nombre et liste des arguments de la fonction qu'à l'intérieur de celle-ci. En dehors de la fonction, elles contiennent les nombre et liste des arguments du script.

Une fonction a également un statut de sortie, qu'on peut envoyer au moyen de l'instruction `return` :

```
toto ()
{
   echo nous sommes dans la fonction toto
   return 0
}

toto
echo $?
```

L'instruction `return` provoque la sortie immédiate de la fonction en envoyant le statut de sortie qui la suit.

L'instruction `return` utilisée seule envoie un statut de sortie nul.

Si le statut de sortie de la fonction n'est pas explicitement envoyé grâce à l'instruction `return`, il sera égal à celui de la dernière commande exécutée par la fonction.

12.10 Statut de sortie

Le statut de sortie d'un script est envoyé au moyen de la commande `exit` :

```
echo ceci est un script qui se termine bien
exit 0
```

La commande `exit` provoque la sortie immédiate du script en envoyant le statut de sortie indiqué en argument.

Si le statut de sortie du script n'est pas explicitement envoyé grâce à la commande `exit` ou si celle-ci est utilisée sans argument, le statut de sortie du script sera égal à celui de la dernière commande exécutée par le script.

12.11 Quelques conseils généraux sur la programmation

Pour terminer cette rapide introduction à la programmation avec l'interpréteur de commandes, il convient d'indiquer quelques conseils, qui sont d'ailleurs valables quel que soit le langage de programmation utilisé.

La lisibilité d'un programme est essentielle car elle facilite sa lecture et sa compréhension. À cet effet, il est indispensable d'utiliser une indentation systématique pour faire apparaître la structure du programme. Des lignes blanches peuvent également être utilisées pour séparer visuellement les différentes parties d'un programme.

Les commentaires sont importants pour éclaircir certaines parties du programme dont la compréhension n'est pas évidente. Il ne faut pas hésiter à en utiliser.

Le problème se pose souvent de savoir quand tester un programme. Faut-il écrire le programme en entier et le tester une fois terminé (avec en hors d'œuvre une farandole d'erreurs de syntaxe) ou écrire le programme petit à petit et le tester au fur et à mesure ?

La bonne méthode pour concevoir un programme est bien entendu la seconde. De manière plus détaillée, la démarche à suivre est la suivante :

1. écrire un petit morceau de programme qui se suffit à lui-même ;
2. tester le programme (et donc corriger les erreurs de syntaxe) et vérifier s'il fonctionne correctement ; si ce n'est pas le cas, le corriger jusqu'à obtention du résultat souhaité puis reprendre à l'étape 1.

De cette façon, vous serez certain d'avoir, à tout moment, un programme qui fonctionne.

Enfin, pour finir sur une note philosophique, lorsqu'on programme, il faut toujours avoir à l'esprit ce principe fondamental :

> *L'ordinateur fait ce que vous lui dites de faire,*
> *pas ce que vous voulez qu'il fasse.*

Vous pouvez prendre un moment pour méditer...

Quatrième partie

Création d'un site Web

13

Formats de documents et historique de la toile

Le Web (de son nom complet *World Wide Web*, en français *toile d'araignée mondiale*, *toile mondiale* ou tout simplement *toile*) est l'application emblématique de l'Internet, à tel point que beaucoup confondent ce seul service avec le réseau informatique mondial (qui en offre bien plus).

Source fantastique d'information mais aussi de désinformation, le meilleur y côtoie souvent le pire. Certaines entreprises en restreignent d'ailleurs l'accès à leurs employés afin d'éviter que ceux-ci ne passent trop de temps sur des sites sans rapport avec leur activité professionnelle. Toujours est-il que la toile est aujourd'hui un outil indispensable et qu'on peut se demander comment on ferait pour vivre sans.

Dans ce cours, la toile sera le prétexte à la réalisation d'un petit site de quelques pages, permettant la sensibilisation des élèves aux formats de documents, à la séparation de la structure et de l'aspect ainsi qu'au respect des normes.

Les sites Web utilisent en effet deux formats de fichiers spécifiques :

- HTML (*Hypertext Markup Language*), qui décrit la structure logique et le contenu textuel des pages ;
- CSS (*Cascading Style Sheets*), qui décrit l'aspect visuel des pages.

Chaque page d'un site Web correspond à un fichier au format HTML (on parle de *page HTML*). L'aspect visuel de l'ensemble des pages d'un site est contrôlé par un fichier au format CSS (on parle de *feuille de style*) auquel toutes les pages HTML font référence.

En pratique, pages HTML et feuille de style sont simplement des fichiers texte dont la syntaxe sera explicitée dans les chapitres suivants.

Mais auparavant, il convient de s'attarder sur un problème fondamental, celui des formats de documents.

13.1 Des formats de documents

Nous l'avons rapidement vu au paragraphe 4.2 page 46, le *format* d'un fichier est la façon dont sont organisées les données qu'il contient.

http://fr.wikipedia.org/wiki/Format_de_données

13.1.1 Formats ouverts, formats fermés

Certains formats sont dits *ouverts*. Un format ouvert est un format dont les spécifications techniques sont publiques et sans restriction d'accès ni de mise en œuvre [1]. Il est donc possible à quiconque le souhaite de comprendre le fonctionnement d'un tel format et d'écrire un logiciel l'utilisant.

Par opposition, les formats *fermés* (appelés également formats propriétaires) sont des formats pour lesquels il n'existe aucune spécification technique accessible publiquement. Chaque format fermé est généralement lié à un logiciel particulier, qui est souvent le seul à pouvoir l'exploiter. On peut donc légitimement s'interroger sur la pérennité des données confiées à des logiciels utilisant des formats fermés.

En effet, la tendance actuelle est à la dématérialisation, que ce soit pour les entreprises ou pour les particuliers, ce qui implique une utilisation de plus en plus importante de fichiers numériques. Ceci pose deux problèmes :

- celui du support de stockage des données (pourrai-je relire mon CD-ROM, ma clé USB ou ma carte SD dans dix ans ?) ;
- celui du format des données (pourrai-je relire mon fichier dans dix ans ?).

Le problème du support de stockage n'en est pas vraiment un compte tenu de l'augmentation constante des capacités de stockage. Il suffit de recopier régulièrement ses fichiers sur un nouveau support. On peut par exemple faire tenir le contenu de 25 CD-ROM sur une clé USB de 16 Go, qui coûte aujourd'hui quelques dizaines d'euros. En conservant ses fichiers en double ou triple exemplaire à des endroits différents, on est également quasiment assuré de ne pas les perdre en cas de défaillance d'un des supports ou de sinistre.

Le vrai problème est celui du format des données, lorsqu'on utilise un format fermé. Mon logiciel de dans dix ans, s'il existe encore, saura-t-il relire mon fichier d'aujourd'hui sans perte ou distorsion de contenu ? Faut-il envisager, comme pour le support, des conversions régulières avec une version plus récente du logiciel (avec le même risque de perte ou de distorsion de contenu) ? Les logiciels en question permettant rarement de convertir automatiquement un ensemble de fichiers (avec les doutes qu'on peut légitimement avoir sur la qualité de la conversion), il peut s'agir d'un travail considérable s'il faut le faire à la main.

1. Il s'agit de la formulation exacte figurant dans le texte de la loi n° 2004-575 du 21 juin 2004 pour la confiance dans l'économie numérique.

Ce problème se pose beaucoup moins avec les formats ouverts, qui ne sont pas dépendants d'un logiciel particulier. Par ailleurs, dans la mesure où leurs spécifications techniques sont publiques, il sera toujours possible d'en interpréter le contenu à l'avenir, sans perte de données.

> http://fr.wikipedia.org/wiki/Format_ouvert
> http://fr.wikipedia.org/wiki/Format_propriétaire

13.1.2 Un exemple de format ouvert : SGML

SGML (*Standard Generalized Markup Language*) est un cadre général permettant de concevoir des documents structurés.

Un exemple de fichier SGML permettra de mieux comprendre. Celui indiqué dans le listage 13.1 page suivante décrit la recette du planteur.

Un document SGML est un mélange de texte et de balises, qui servent à indiquer la structure du document. Une *balise* (*tag* en anglais) est composée d'une chaîne de caractères, appelée *élément*, encadrée par deux chevrons < et > qui permettent de distinguer la balise du texte du document. Les balises vont généralement par paire, `<BALISE>` pour indiquer le début de quelque chose et `</BALISE>` pour indiquer sa fin. On emploie également le terme *élément* pour désigner l'ensemble comprenant la balise de début, la balise de fin correspondante et leur contenu.

Ainsi l'ensemble du document figure entre `<COCKTAIL>` et `</COCKTAIL>`. À l'intérieur, `<NOM>` indique le nom du cocktail. Suivent les ingrédients ainsi que les étapes de la préparation.

Les balises de fin peuvent être omises lorsque le contexte le permet. Ainsi, dans notre exemple, l'utilisation de la balise `<QUANTITE>` dans la liste des ingrédients implique la fin de la balise `<INGREDIENT>` qui la précède.

Un document SGML est généralement destiné à être converti en document visualisable à l'écran ou imprimable. Notez qu'il n'est indiqué nulle part quelle doit être la présentation du document, si les ingrédients doivent figurer sous forme de liste ou de tableau, etc. Il faut bien insister sur ce point : un document SGML est défini en termes de structure, pas de présentation. C'est ensuite au moment de la conversion qu'est spécifiée la mise en page et l'on peut tout à fait adopter plusieurs présentations pour un même document SGML. La forme peut changer, pas le fond.

SGML ne définit pas d'éléments particuliers comme ceux de l'exemple mais permet de créer des ensembles d'éléments et d'indiquer la façon de les agencer.

Ainsi, dans notre exemple, l'ensemble du document doit être inclus entre des balises `<COCKTAIL>` et `</COCKTAIL>`. Doit suivre le nom du cocktail, puis la liste

```
<!DOCTYPE COCKTAIL SYSTEM "cocktail.dtd">

<COCKTAIL>

<NOM>Planteur

<INGREDIENTS>
   <INGREDIENT>jus d'orange
   <QUANTITE>5 dl

   <INGREDIENT>jus de goyave
   <QUANTITE>5 dl

   <INGREDIENT>jus d'ananas
   <QUANTITE>5 dl

   <INGREDIENT>rhum blanc
   <QUANTITE>3 dl

   <INGREDIENT>sirop de sucre de canne
   <QUANTITE>3 dl

   <INGREDIENT>cannelle
   <QUANTITE>1 bâton

   <INGREDIENT>vanille
   <QUANTITE>1 gousse
</INGREDIENTS>

<PREPARATION>
   <ETAPE>mélanger les liquides
   <ETAPE>fendre la gousse de vanille en deux, gratter
          l'intérieur et ajouter le tout à la préparation
          avec la cannelle
   <ETAPE>laisser macérer une nuit
   <ETAPE>servir frais
</PREPARATION>

</COCKTAIL>
```


LISTAGE 13.1 – *Exemple de fichier SGML.*

des ingrédients, puis la préparation. On ne peut pas, par exemple, faire figurer un élément `INGREDIENT` en dehors de la liste des ingrédients figurant dans l'élément `INGREDIENTS`.

La liste des éléments utilisables dans un document ainsi que la façon de les agencer sont définies dans la DTD (*Document Type Definition*). La DTD est stockée dans un fichier texte, ici `cocktail.dtd`, dont le nom est indiqué par l'élément `DOCTYPE`. La forme de sa balise est différente des balises habituelles et elle indique également quel est l'élément principal du document (ici, `COCKTAIL`, qui contient l'ensemble du document).

Confronté à un besoin documentaire, on peut donc utiliser une DTD permettant de concevoir des documents adéquats. Si une DTD convenable n'existe pas, il est toujours possible d'en créer une.

> http://fr.wikipedia.org/wiki/Standard_Generalized_Markup_Language
> http://fr.wikipedia.org/wiki/Document_Type_Definition

13.1.3 L'évolution de SGML : XML

SGML n'a été adopté que par de grandes entreprises ayant d'importants besoins de documentation. Sa mise en œuvre est en effet assez lourde.

XML (*Extensible Markup Language*) reprend les caractéristiques de SGML et vise à les rendre plus simples. Les différences visibles entre SGML et XML sont principalement que :

- les éléments doivent être écrits en minuscules ;
- toute balise ouvrante doit être explicitement fermée ;
- les balises doivent être correctement imbriquées.

La fermeture explicite des balises vise à simplifier le travail des outils de traitement des fichiers XML. En SGML, la fermeture des balises n'est pas obligatoire si elle peut être inférée par le contexte. Ceci complique le travail des outils de traitement et peut parfois être ambigu.

L'imbrication des balises a également pour objectifs la simplicité et la disparition des ambiguïtés. Ainsi, cette imbrication, autorisée en SGML :

```
<balise1> <balise2> ··· </balise1> </balise2>
```

est interdite en XML. Les balises doivent être fermées dans l'ordre inverse de leur ouverture :

```
<balise1> <balise2> ··· </balise2> </balise1>
```

Ainsi, l'exemple de cocktail, écrit en XML, donne ce qui est indiqué dans le listage 13.2 page suivante.

> http://www.w3.org/XML/
> http://fr.wikipedia.org/wiki/Extensible_Markup_Language

13.2 Historique de la toile

Le World Wide Web a été créé en 1989 par Tim Berners-Lee, informaticien à l'Organisation européenne pour la recherche nucléaire (CERN), rapidement rejoint par l'un de ses collègues, Robert Cailliau.

Son but était de partager des documents au sein du CERN sur le principe de l'*hypertexte*, popularisé par le logiciel HyperCard sur Macintosh. Dans un document hypertexte, certains mots ou groupes de mots sont rendus actifs et le fait de les sélectionner mène vers un autre document. Un tel mot ou groupe de mots est ce qu'on appelle un *hyperlien*.

Pour décrire ces documents, un format ouvert spécifique, HTML (*Hypertext Markup Language*), dérivé de SGML, est inventé. Conçu pour être indépendant de l'environnement d'affichage des documents (système d'exploitation, logiciel, écran ou autre périphérique de sortie), HTML est encore très dépouillé et ne contient que quelques éléments de structure ainsi que celui permettant de relier les documents entre eux.

Nous utiliserons le terme *butineur* (*browser* en anglais) pour désigner le logiciel d'affichage des pages HTML. On utilise également *navigateur* mais il faut reconnaître que butineur est bien plus bucolique et traduit mieux la promenade qu'on peut faire, au gré de sa curiosité, d'un site à un autre.

En 1991, le premier serveur Web hors du CERN est créé. En 1993, le butineur NCSA Mosaic, disponible d'abord pour UNIX, puis pour Windows et Macintosh, commence à populariser la toile. NCSA Mosaic étend le langage HTML pour permettre d'inclure des images dans les documents. Il rajoutera plus tard la gestion des formulaires.

En 1994, est fondé le World Wide Web Consortium (W3C), destiné à piloter l'évolution des technologies utilisées sur la toile, notamment HTML.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE cocktail SYSTEM "cocktail.dtd">

<cocktail>

<nom>Planteur</nom>

<ingredients>
   <ingredient>jus d'orange</ingredient>
   <quantite>5 dl</quantite>

   <ingredient>jus de goyave</ingredient>
   <quantite>5 dl</quantite>

   <ingredient>jus d'ananas</ingredient>
   <quantite>5 dl</quantite>

   <ingredient>rhum blanc</ingredient>
   <quantite>3 dl</quantite>

   <ingredient>sirop de sucre de canne</ingredient>
   <quantite>3 dl</quantite>

   <ingredient>cannelle</ingredient>
   <quantite>1 bâton</quantite>

   <ingredient>vanille</ingredient>
   <quantite>1 gousse</quantite>
</ingredients>

<preparation>
   <etape>mélanger les liquides</etape>
   <etape>fendre la gousse de vanille en deux, gratter
          l'intérieur et ajouter le tout à la préparation
          avec la cannelle</etape>
   <etape>laisser macérer une nuit</etape>
   <etape>servir frais</etape>
</preparation>
```



```
</cocktail>
```

LISTAGE 13.2 – *Exemple de fichier XML.*

Toujours en 1994, une partie de l'équipe ayant travaillé sur NCSA Mosaic crée l'entreprise Mosaic Communications Corporation, renommée par la suite Netscape Communications Corporation, afin de diffuser le butineur Mosaic Netscape, renommé lui aussi Netscape Navigator, et disponible pour divers UNIX, Windows et Macintosh.

Année bien chargée, 1994 marque également le début de l'accès à l'Internet pour le grand public.

En 1995, Microsoft, qui s'est laissé distancer sur ce qui apparaît déjà comme un secteur stratégique, lance Internet Explorer. Il s'agit tout d'abord d'une extension payante de Windows 95. Il sera ensuite téléchargeable gratuitement, intégré aux mises à jour de Windows 95 puis finalement intégré d'origine à Windows 98, valant à Microsoft de nombreuses condamnations pour abus de position dominante.

Débuta alors ce qu'on a appelé la guerre des butineurs (*the browser wars*) entre Internet Explorer et Netscape Navigator. Chaque logiciel ajoutant ses propres éléments à HTML et copiant ceux de l'autre. Le tout sans aucun respect pour le principe de séparation de la structure et de l'aspect, ainsi que pour le processus de standardisation du W3C. Il n'était pas rare de voir en page d'accueil des sites Web une mention indiquant : « ce site est optimisé pour telle version de tel butineur », baffouant ainsi le principe d'indépendance de HTML par rapport au butineur. Cette escalade de la violence aboutira, entre autres, à l'invention de l'infâme élément `BLINK` pour faire clignoter du texte...

Au printemps 1998, le code source de Netscape Navigator, renommé pour l'occasion Mozilla, qui était son nom de code interne, est publié sous une licence libre. L'association Mozilla est créée afin de reprendre son développement.

Fin 1998, Microsoft vient à bout de Netscape Communications Corporation, qui sera racheté l'année suivante par America Online (AOL). De nouvelles versions de Netscape Navigator basées sur le travail effectué sur Mozilla, seront distribuées par AOL jusqu'au début 2008.

La guerre des butineurs a piétiné les principes fondateurs de la toile et laisse derrière elle de nombreux dialectes de HTML.

Le W3C a bien tenté de fédérer ce qu'il pouvait avec HTML 2.0, en 1995, puis HTML 3.2 (la version 3.0 n'ayant pas abouti), en 1997, mais qui s'en soucie ? HTML 3.2 est de toute façon une hérésie, qui se fourvoie dans un mélange d'éléments décrivant l'aspect des documents plutôt que leur structure. Pourtant la spécification des feuilles de style, qui permet d'appliquer de nombreuses modifications d'aspect à un document HTML bien structuré, a été publiée fin 1996.

Puis, toujours en 1997, le W3C publie la spécification de HTML 4.0, qui existe en trois versions. Deux d'entre elles reprennent le côté obscur de HTML 3.2 mais

la troisième renoue de manière spectaculaire avec la philosophie d'origine du Web. Avec une légère retouche en 1999, HTML 4.01 est toujours la version actuelle utilisée sur la toile aujourd'hui.

HTML cristallise évidemment d'énormes enjeux, en particuliers commerciaux, et la standardisation de HTML 5 est toujours en cours.

En 1998, le W3C publie la première spécification de XML, puis en 2000, celle de XHTML 1.0, qui est tout simplement la réécriture de HTML 4.01 à la mode XML. Suivra XHTML 1.1 en 2001. C'est la version de XHTML qui est toujours en usage aujourd'hui.

Pendant ce temps, Mozilla évolue. La version 1.0 sort en 2002. Un nouveau projet est lancé, sur la base du moteur d'affichage de Mozilla, qui aboutira en 2004 au logiciel Firefox. Mozilla sera abandonné par la suite.

Internet Explorer, dont le développement s'est beaucoup ralenti après sa victoire sur Netscape Navigator, commence à se faire rattraper par Mozilla puis Firefox. C'est le début de la seconde guerre des butineurs, qui continue aujourd'hui.

Outre Internet Explorer et Firefox, citons également les butineurs Chrome (édité par Google), Opera (édité par Opera Software) et Safari (édité par Apple), dont les parts de marché sont actuellement plus faibles mais non négligeables.

Internet Explorer perd régulièrement des utilisateurs au profit de Firefox, considéré comme plus dynamique et plus novateur. Pour preuve, la version 3.5 de Firefox, sortie le 30 juin 2009, intégrait déjà une ébauche substantielle de gestion de HTML 5. Même si certains aspects de HTML 5 ont été intégrés dans Internet Explorer 8, il aura fallu attendre Internet Explorer 9 (sorti le 14 mars 2011) pour que Microsoft comble son retard.

> http://fr.wikipedia.org/wiki/World_Wide_Web
> http://fr.wikipedia.org/wiki/Hypertexte
> http://fr.wikipedia.org/wiki/HyperCard
> http://fr.wikipedia.org/wiki/NCSA_Mosaic
> http://www.w3.org/
> http://fr.wikipedia.org/wiki/World_Wide_Web_Consortium
> http://fr.wikipedia.org/wiki/Netscape_Navigator
> http://fr.wikipedia.org/wiki/Internet_Explorer
> http://fr.wikipedia.org/wiki/Évolution_de_l'usage_des_navigateurs_Internet
> http://www.mozilla.org/
> http://fr.wikipedia.org/wiki/Mozilla_Foundation
> http://fr.wikipedia.org/wiki/Mozilla

http://www.mozilla.com/
http://fr.wikipedia.org/wiki/Mozilla_Firefox
http://www.google.com/chrome/
http://www.opera.com/
http://www.apple.com/fr/safari/
http://fr.wikipedia.org/wiki/Part_de_marché_des_navigateurs_Web
http://fr.wikipedia.org/wiki/HTML5
http://diveintohtml5.org/
http://html5gallery.com/
http://html5doctor.com/
http://www.webhostingsecretrevealed.com/featured-articles/learn-html5-10-must-read-lessons/

14

HTML, XHTML

La réalisation d'un site Web peut se faire de deux façons. La plus répandue consiste à utiliser un logiciel semblable à un traitement de texte. Généralement, l'utilisateur n'a que peu de contrôle sur le code HTML généré par ce logiciel, code qui est souvent de mauvaise qualité et rarement valide, même si le laxisme des butineurs permet de l'afficher correctement. L'autre façon de faire consiste à réaliser les pages « à la main », en concevant directement le code HTML au moyen d'un éditeur de texte.

C'est évidemment cette seconde approche que nous allons utiliser, puisqu'elle permet de maîtriser totalement le code HTML et sa conformité. C'est également un prétexte à l'utilisation d'un éditeur de texte ainsi qu'à la mise en pratique des notions étudiées jusqu'ici.

Les pages seront réalisées conformément au standard XHTML 1.1 (qui correspond à HTML 4.01 débarrassé de ses éléments superflus et écrit à la mode XML) et leur qualité sera systématiquement validée grâce au vérificateur du W3C :

```
http://validator.w3.org/
```

L'étude de l'ensemble des éléments existant en HTML dépasse le cadre de ce cours aussi nous nous limiterons aux plus utiles. Dans la description des éléments, il sera autant que possible fait référence à la partie correspondante dans la spécification de HTML 4.01. Celle-ci étant antérieure à XHTML 1.1, les éléments y sont indiqués en majuscules alors qu'ils figurent en minuscules dans ce document.

Le lecteur curieux pourra se reporter aux ouvrages suivants de la bibliographie pour approfondir ses connaissances :

- *HTML for Babies* [52]
- *Designing with Web Standards* [53]
- *HTML 4* [49]
- *Mémento XHTML* [35]
- *HTML & XHTML : The Definitive Guide* [43]
- *HTML & XHTML Pocket Reference* [46]

- *HTML5 : Up and Running* [45]
- *HTML5 Cookbook* [48]
- *Réussir son site web avec XHTML et CSS* [44]
- *Premiers pas en CSS et XHTML* [32]
- *XHTML et CSS* [50]
- *HTML5 et CSS3* [39]
- *HTML5 et CSS3* [51]
- *Sams Teach Yourself HTML and CSS in 24 Hours* [40]
- *HTML & CSS : The Good Parts* [37]
- *HTML5 & CSS3* [38]

```
http://www.w3.org/html/
http://fr.wikipedia.org/wiki/Hypertext_Markup_Language
http://fr.wikipedia.org/wiki/Extensible_HyperText_Markup_Language
http://fr.wikibooks.org/wiki/Le_langage_HTML
http://htmlhelp.com/
http://w3schools.com/
```

14.1 Structure générale d'une page XHTML

Une page XHTML minimale doit respecter la structure suivante :

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <!DOCTYPE html
4          PUBLIC "-//W3C//DTD XHTML 1.1//EN"
5          "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
6
7  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
8
9  <head>
10
11 <!-- ici se trouve l'en-tête du document -->
12
13 </head>
14
15 <body>
16
17 <!-- ici se trouve le corps du document -->
18
19 </body>
20
21 </html>
```

De manière détaillée :

- la ligne 1 est la déclaration XML du document, qui indique la version de XML utilisée ainsi que le codage du document ;
- les lignes 3, 4 et 5 (il s'agit d'une seule balise répartie sur plusieurs lignes pour en améliorer la lisibilité) indiquent la version de XHTML utilisée ainsi que l'emplacement où l'on peut télécharger la DTD correspondante ;
- la ligne 7 débute le document proprement dit par la balise `<html>` ;
- l'en-tête du document (voir le paragraphe 14.1.1 page suivante) se trouve entre les balises `<head>` et `</head>` (lignes 9 à 13) ;
- le corps du document (voir le paragraphe 14.1.2 page 295) se trouve entre les balises `<body>` et `</body>` (lignes 15 à 19) ;
- enfin, le document se termine à la ligne 21 par la balise `</html>`, qui ferme celle ouverte à la ligne 7.

À l'occasion, vous aurez remarqué que les commentaires en HTML sont délimités par `<!--` et `-->`. Ils peuvent s'étendre sur plusieurs lignes.

Vous aurez également remarqué que la balise ouvrante de l'élément `html` est accompagnée de deux attributs. Un *attribut* modifie ou complète la signification d'un élément. Les attributs sont placés dans la balise ouvrante entre le nom de l'élément et le chevron > qui marque la fin de la balise et sont de la forme `nom="valeur"`. Les guillemets sont obligatoires. Le premier attribut est séparé de l'élément par une espace et les attributs sont également séparés entre eux par des espaces.

Ici, l'élément `html` a deux attributs. L'attribut `xmlns` indique l'espace de nommage (notion qui ne sera pas abordée) utilisé. L'attribut `xml:lang` indique le code ISO 639 de la langue utilisée dans le document (`fr` pour le français, `en` pour l'anglais, etc.). L'attribut `xml:lang` sera étudié plus en détail au paragraphe 14.9 page 310.

> http://www.w3.org/TR/html401/struct/global.html
> http://www.w3.org/TR/html401/intro/sgmltut.html#h-3.2.4

14.1.1 L'en-tête

L'en-tête est délimité par les balises `<head>` et `</head>`.

Il doit obligatoirement contenir le titre du document, entre les balises `<title>` et `</title>`, qui apparaîtra dans la barre de titre de la fenêtre du butineur :

```
<title>Ceci est le titre de la page</title>
```

Il peut également contenir un ou plusieurs éléments `meta`. Celui-ci est utilisé pour donner des informations supplémentaires sur le document (auteur, liste de mots clés, etc.).

On l'utilise souvent pour indiquer le codage du document. Ceci fait double emploi avec la déclaration XML mais est nécessaire pour certains butineurs :

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
```

Notez que l'élément `meta` n'a ni balise d'ouverture ni balise de fermeture. Il ne s'agit en effet pas d'un élément entourant du texte destiné à apparaître quelque part. Un tel élément est dit *vide*. Les balises des éléments vides s'écrivent avec une barre oblique `/` juste avant le chevron > de fermeture de la balise. Les éléments vides sont

utilisés soit pour indiquer des structures très simples (nous en verrons par la suite) soit, lorsqu'ils en ont, uniquement pour leurs attributs.

L'ordre d'apparition des éléments dans l'en-tête est sans importance.

Un en-tête minimal contient donc :

```
<head>

<title>Ceci est le titre de la page</title>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>

</head>
```

http://www.w3.org/TR/html401/struct/global.html#h-7.4

14.1.2 Le corps

Le corps est délimité par les balises `<body>` et `</body>`.

Il peut contenir l'ensemble des éléments que nous allons étudier par la suite.

http://www.w3.org/TR/html401/struct/global.html#edef-BODY

14.2 Nommage et emplacement des fichiers

Chaque page XHTML doit être stockée dans un fichier ayant l'extension `.html`.

Par convention, afin de pouvoir être accessibles grâce à des URL de la forme :

`http://www.ensta-paristech.fr/~identifiant/fichier.html`

ces fichiers doivent être placés dans un répertoire `public_html` situé dans le répertoire personnel de l'utilisateur. Ainsi, l'URL ci-dessus permettra d'accéder au fichier `~/public_html/fichier.html` de l'utilisateur `identifiant`.

De même, celui de ces fichiers qui fait office de page d'accueil doit s'appeler `index.html` afin d'être accessible simplement au moyen de l'URL :

`http://www.ensta-paristech.fr/~identifiant/`

c'est-à-dire sans indiquer de nom de fichier.

14.3 Au sujet des exemples qui suivent

Chacun des exemples qui vont suivre sera supposé figurer dans le corps (ne contenant que l'exemple en question) d'une page XHTML conforme à la structure indiquée précédemment.

Chaque exemple sera immédiatement suivi du rendu graphique que la page complète donne dans Firefox 11.0.

14.4 Les éléments usuels

14.4.1 Les éléments de titres `h1`, `h2`, `h3`, `h4`, `h5` et `h6`

Les éléments `h1`, `h2`, `h3`, `h4`, `h5` et `h6` (pour *heading*) permettent de définir des titres. On utilise `h1` pour les titres de premier niveau, `h2` pour les titres de deuxième niveau et ainsi de suite :

```
<h1>Titre de premier niveau</h1>

<!-- ici on peut mettre un texte d'introduction -->

<h2>Titre de deuxième niveau</h2>

<!-- texte -->

<h2>Titre de deuxième niveau</h2>

<!-- texte -->
```

Titre de premier niveau

Titre de deuxième niveau

Titre de deuxième niveau

Un titre de niveau inférieur ne peut suivre qu'un titre de niveau immédiatement supérieur. En d'autres termes, on ne peut pas avoir un titre en `h3` après un titre en `h1`. De même, le premier titre d'un document doit être un `h1`.

Les titres sont habituellement affichés en gras et avec des caractères de taille décroissante de `h1` à `h6`.

> http://www.w3.org/TR/html401/struct/global.html#h-7.5.5

14.4.2 L'élément de paragraphe `p`

L'élément `p` permet de délimiter un paragraphe :

```
<p>
Ceci est un paragraphe. Les paragraphes sont la base
de quasiment tous les documents HTML.
</p>

<p>
Ceci est un autre paragraphe. Un document HTML peut
en contenir autant que nécessaire.
</p>
```

Ceci est un paragraphe. Les paragraphes sont la base de quasiment tous les documents HTML.

Ceci est un autre paragraphe. Un document HTML peut en contenir autant que nécessaire.

Un paragraphe ne peut pas contenir un autre paragraphe. Il ne peut pas contenir un titre et un titre ne peut pas le contenir. En revanche, on peut avoir :

```
<h1>Titre</h1>

<p>
Ceci est un paragraphe.
</p>

<h2>Sous-titre</h2>

<p>
Ceci est un paragraphe.
</p>
```

Titre

Ceci est un paragraphe.

Sous-titre

Ceci est un paragraphe.

Un document HTML ne peut pas contenir de texte isolé. Tout texte doit être contenu dans quelque chose, titre, paragraphe ou d'autres éléments non encore étudiés.

Les espaces et retours à la ligne sont ignorés. À ce sujet, remarquez comment sont disposées les balises dans l'exemple précédent. Celles des titres les encadrent sur la même ligne, parce qu'un titre est généralement court. Celles des paragraphes sont chacune sur une ligne avant et après le paragraphe, parce qu'un paragraphe est généralement plus long. Rien n'oblige bien entendu à adopter ce genre de présentation mais elle facilite la lecture du document, de même que les lignes blanches séparant les différents blocs.

> http://www.w3.org/TR/html401/struct/text.html#edef-P

14.4.3 L'élément de filet `hr`

L'élément `hr` (pour *horizontal rule*) insère une ligne — qu'on appelle en typographie un *filet* — horizontale. Il s'agit d'un élément vide dans la mesure où il n'encadre rien du tout mais figure à un endroit précis :

```
<p>
Ceci est un paragraphe.
</p>

<hr/>

<p>
Ceci est un paragraphe.
</p>
```

Ceci est un paragraphe.

Ceci est un paragraphe.

http://www.w3.org/TR/html401/present/graphics.html#edef-HR

14.4.4 L'élément de retour à la ligne `br`

L'élément vide `br` (pour *break*) génère un retour à la ligne :

```
<p>
Ô rage ! ô désespoir ! ô vieillesse ennemie ! <br/>
N'ai-je donc tant vécu que pour cette infamie ?
</p>
```

Ô rage ! ô désespoir ! ô vieillesse ennemie !
N'ai-je donc tant vécu que pour cette infamie ?

http://www.w3.org/TR/html401/struct/text.html#edef-BR

14.4.5 Les éléments textuels `abbr`, `acronym`, `cite`, `code`, `dfn`, `em`, `kbd`, `samp`, `strong` et `var`

Ces éléments sont utilisés pour différencier des parties du texte :
- `abbr` (pour *abbreviation*) sert à indiquer une abréviation ;
- `acronym` sert à indiquer un acronyme ;
- `cite` sert à indiquer une citation ;
- `code` sert à indiquer un fragment de code informatique ;
- `dfn` sert à indiquer une définition ;
- `em` (pour *emphasis*) sert à mettre en valeur ;
- `kbd` (pour *keyboard*) sert à indiquer ce qui doit être saisi au clavier ;
- `samp` (pour *sample*) sert à indiquer un exemple de résultat ;
- `strong` sert à mettre en valeur de manière plus forte que `em` ;
- `var` sert à indiquer le nom d'une variable ou d'un argument dans un exemple informatique.

Certains de ces éléments modifient l'aspect du texte (ce qui peut varier en fonction du butineur utilisé), d'autres n'ont qu'une valeur structurelle :

```
<p>
<abbr>abbr</abbr> normal <acronym>acronym</acronym>
normal <cite>cite</cite> normal <code>code</code>
normal <dfn>dfn</dfn> normal <em>em</em> normal
<kbd>kbd</kbd> normal <samp>samp</samp> normal
<strong>strong</strong> normal <var>var</var> normal
</p>
```

abbr normal acronym normal *cite* normal `code`
normal *dfn* normal *em* normal `kbd` normal `samp`
normal **strong** normal *var* normal

http://www.w3.org/TR/html401/struct/text.html#h-9.2.1

14.4.6 Les éléments visuels `b`, `i` et `tt`

Ces éléments sont à utiliser avec parcimonie parce qu'ils ne décrivent pas la structure mais l'aspect du document :

- le texte contenu dans l'élément `b` est mis en gras ;
- le texte contenu dans l'élément `i` est mis en italique ;
- le texte contenu dans l'élément `tt` est mis dans une police à chasse fixe.

```
<p>
normal <b>gras</b> <i>italique</i> <tt>chasse fixe</tt>
</p>
```

normal **gras** *italique* `chasse fixe`

http://www.w3.org/TR/html401/present/graphics.html#h-15.2.1

14.4.7 Les éléments inférieur sub et supérieur sup

Les éléments sub et sup servent respectivement à mettre leur contenu en caractères inférieurs et supérieurs (qui sont les équivalents typographiques des indices et des exposants) :

```
<p>
H<sub>2</sub>0
</p>

<p>
En français, on écrit 1<sup>er</sup>, 1<sup>re</sup>,
2<sup>e</sup>, etc.
</p>
```

H_2O

En français, on écrit 1er, 1re, 2e, etc.

http://www.w3.org/TR/html401/struct/text.html#h-9.2.3

14.4.8 Les entités

Les balises étant délimitées par des chevrons < et >, on ne peut pas utiliser ces caractères directement dans un document HTML. Il faut représenter chacun d'eux par une *entité*. Une entité est composée d'une esperluette & suivie d'un code représentant le caractère à afficher suivi d'un point-virgule ;. L'utilisation du caractère & pour introduire les entités rend également nécessaire l'utilisation d'une entité pour le représenter. En revanche, le point-virgule peut être utilisé librement.

En pratique, on utilise :

- < (*lower than*) pour représenter <;

- > (*greater than*) pour représenter >;
- & (*ampersand*) pour représenter &.

```
<p>
En HTML, les balises sont délimitées par <code>&lt;</code>
et <code>&gt;</code>.
</p>

<p>
Pour représenter ces caractères, on utilise les entités
<code>&amp;lt;</code> et <code>&amp;gt;</code>.
</p>
```

En HTML, les balises sont délimitées par < et >.

Pour représenter ces caractères, on utilise les entités < et >.

http://www.w3.org/TR/html401/charset.html#h-5.3.2

14.5 L'élément d'hyperlien a

La possibilité de relier des documents au moyen d'hyperliens est la raison d'être de la toile.

Pour créer un hyperlien vers un autre document, on utilise l'élément a (pour *anchor*, ancre), entourant le texte de l'hyperlien. Il faut obligatoirement indiquer la destination de celui-ci dans l'attribut href. Celle-ci peut être un fichier du même site, auquel cas on indique simplement son chemin d'accès, ou une page d'un site distant, auquel cas on indique son URL :

```
<p>
Pour plus d'informations, vous pouvez consulter la
<a href="liste-des-cours.html">liste des cours</a> ou visiter le
<a href="http://www.ensta-paristech.fr/">site de l'ENSTA ParisTech</a>.
</p>
```

Pour plus d'informations, vous pouvez consulter la liste des cours ou visiter le site de l'ENSTA ParisTech.

http://www.w3.org/TR/html401/struct/links.html#edef-A

14.6 L'élément d'insertion d'image `img`

Afin d'égayer les documents, on peut y insérer des images au moyen de l'élément `img`. Il s'agit d'un élément vide dans la mesure où il n'encadre rien du tout mais figure à un endroit précis. Il doit obligatoirement être utilisé avec les attributs `src` pour indiquer le chemin d'accès ou l'URL du fichier contenant l'image à insérer et `alt` pour indiquer un texte de remplacement. Celui-ci est destiné à être utilisé à la place de l'image si celle-ci n'est pas appropriée. Par exemple, les aveugles disposent de logiciels capable de lire le texte des pages et le contenu de l'attribut `alt` des éléments `img` au moyen d'un synthétiseur vocal. Par convention, afin de le différencier du texte normal des pages, on a l'habitude de placer le texte de l'attribut `alt` entre crochets.

L'élément `img` peut figurer à tout endroit où l'on peut mettre du texte. Il ne doit pas être utilisé seul. Si l'on souhaite une image isolée, on peut par exemple l'inclure dans un paragraphe :

```
<p>
La cathédrale de Beauvais possède le chœur gothique
le plus haut du monde.
</p>

<p>
<img src="cathedrale-de-beauvais.jpg"
     alt="[Photo de la cathédrale de Beauvais]"/>
</p>
```

La cathédrale de Beauvais possède le chœur gothique le plus haut du monde.

Attention à ne pas utiliser des images à tort et à travers. Certains sites utilisent en effet de nombreuses images ne contenant en fait que du texte uniquement afin de pouvoir contrôler la mise en page. Le problème de cette approche est que le contenu des pages est alors de largeur fixe au lieu de s'adapter à la largeur de la fenêtre, ce qui

peut se révéler gênant sur de petits écrans comme ceux des téléphones mobiles. Ceci empêche également une indexation correcte par les moteurs de recherche.

http://www.w3.org/TR/html401/struct/objects.html#edef-IMG

14.7 Les listes

On peut utiliser trois types de listes dans les documents. Celles-ci doivent figurer à l'extérieur des paragraphes.

Les deux premiers types de listes sont très semblables. Les listes encadrées par l'élément `ul` (pour *unordered list*) présentent chaque composant à la suite d'un gros point. Les listes encadrées par l'élément `ol` (pour *ordered list*) présentent chaque composant à la suite d'un numéro (la numérotation est automatique). Dans ces deux types de listes, les composants sont encadrés par l'élément `li` (pour *list item*). Les composants contiennent du texte, voire plusieurs paragraphes (il faut alors utiliser des éléments `p`) pour les composants très longs.

```
<ul>
   <li> premier élément </li>
   <li> deuxième élément </li>
   <li> troisième élément </li>
</ul>

<ol>
   <li> premier élément </li>
   <li> deuxième élément </li>
   <li> troisième élément </li>
</ol>
```

- premier élément
- deuxième élément
- troisième élément

1. premier élément
2. deuxième élément
3. troisième élément

Enfin, les listes encadrées par l'élément `dl` (pour *definition list*) permettent d'introduire chaque composant par du texte. Le texte d'introduction est encadré par l'élément `dt` (pour *definition term*) et le texte du composant par l'élément `dd` (pour *definition description*), ces deux éléments allant évidemment par paire, dans cet ordre :

```
<p>
La relation d'égalité est :
</p>

<dl>
   <dt>réflexive</dt>
   <dd>quel que soit a, a = a</dd>

   <dt>symétrique</dt>
   <dd>quels que soient a et b,
       si a = b alors b = a</dd>

   <dt>transitive</dt>
   <dd>quels que soient a, b et c,
       si a = b et b = c alors a = c</dd>
</dl>
```

La relation d'égalité est :

réflexive
quel que soit a, a = a
symétrique
quels que soient a et b, si a = b alors b = a
transitive
quels que soient a, b et c, si a = b et b = c alors a = c

http://www.w3.org/TR/html401/struct/lists.html

14.8 Les tableaux

La réalisation de tableaux dans un document HTML est très simple. On décrit le tableau ligne par ligne puis colonne par colonne. Le butineur s'occupe de calculer largeur et hauteur en fonction du contenu du tableau.

Notez que les tableaux ne doivent pas figurer à l'intérieur de paragraphes ou de listes.

Voyons un exemple simple :

```
<table border="1">
<tr> <th>Nom</th>          <th>Numéro</th> </tr>
<tr> <td>Export</td>       <td>33</td>     </tr>
<tr> <td>Pastis</td>       <td>51</td>     </tr>
<tr> <td>Kronenbourg</td> <td>1664</td>   </tr>
</table>
```

Nom	Numéro
Export	33
Pastis	51
Kronenbourg	1664

Le tableau est encadré par l'élément `table`. L'attribut `border` indique l'épaisseur du cadre et des séparations entre lignes et colonnes. S'il est absent, il n'y a ni cadre ni séparations. Pendant la mise au point d'un tableau, il est utile de matérialiser cadre et séparations grâce à cet attribut, qui pourra être supprimé dans la version finale, afin de bien visualiser la structure du tableau.

Chaque ligne est encadrée par l'élément `tr` (pour *table row*).

Dans chaque ligne, les colonnes sont encadrées par l'élément `th` (pour *table header*) s'il s'agit d'un titre ou par l'élément `td` (pour *table data*) s'il s'agit d'une donnée (c'est le cas général). La majorité des butineurs affiche le texte contenu dans l'élément `th` en gras et centré, afin de le différencier des colonnes de données, qui ne subissent aucun traitement particulier.

Les lignes d'un tableau contiennent généralement le même nombre de colonnes.

Une cellule du tableau peut s'étendre sur plusieurs colonnes en utilisant l'attribut `colspan` ou sur plusieurs lignes en utilisant l'attribut `rowspan` :

```
<table border="1">
<tr> <th>Un</th> <th>Deux</th> <th>Trois</th> </tr>
<tr> <td colspan="2">sur deux colonnes</td>
     <td>bla</td> </tr>
<tr> <td>bla</td> <td>bla</td>
     <td rowspan="2">sur deux lignes</td> </tr>
<tr> <td>bla</td> <td>bla</td> </tr>
</table>
```

Un	Deux	Trois
sur deux colonnes		bla
bla	bla	sur deux lignes
bla	bla	

Enfin, il est possible d'affecter une légende à un tableau au moyen de l'élément `caption`, qui doit obligatoirement suivre la balise `<table>` :

```
<table border="1">
<caption>Tarifs du bar</caption>
<tr> <td>café</td>     <td>0,40 €</td> </tr>
<tr> <td>Guinness</td> <td>2,00 €</td> </tr>
</table>
```

Tarifs du bar

café	0,40 €
Guinness	2,00 €

http://www.w3.org/TR/html401/struct/tables.html

14.9 L'attribut de langue `xml:lang`

L'attribut `xml:lang` peut être appliqué à n'importe quel élément et permet d'indiquer la langue de son contenu au moyen de son code ISO 639 utilisé comme valeur de l'attribut :

```
<p xml:lang="en">
We are not amused.
</p>
```

Nous verrons au paragraphe 15.3.3.8.3 page 327 comment cet attribut peut être utilisé en conjonction avec la feuille de style.

> http://www.w3.org/TR/REC-xml/#sec-lang-tag
> http://fr.wikipedia.org/wiki/ISO_639
> http://fr.wikipedia.org/wiki/Liste_des_codes_ISO_639-1

14.10 Enrichir XHTML

XHTML peut être enrichi grâce à d'autres DTD, pour peu que le butineur utilisé soit capable d'afficher les documents résultants.

MathML, par exemple, permet de décrire des formules mathématiques. Son affichage est correctement géré par Firefox à partir de la version 3.0, Opera à partir de la version 9.5 et Internet Explorer à partir de la version 6.0, à condition d'installer l'extension MathPlayer (qui est téléchargeable gratuitement).

Pour intégrer des notations mathématiques au moyen de MathML dans une page XHTML, il faut en modifier la déclaration de DTD comme ceci :

```
<!DOCTYPE html
        PUBLIC "-//W3C//DTD XHTML 1.1 plus MathML 2.0//EN"
        "http://www.w3.org/Math/DTD/mathml2/xhtml-math11-f.dtd">
```

Il est alors possible d'indiquer dans le corps de la page :

```
<math xmlns="http://www.w3.org/1998/Math/MathML" display="block">
<mrow>

<munderover>
   <mo>&sum;</mo>
   <mrow> <mi>n</mi> <mo>=</mo> <mn>1</mn> </mrow>
   <mrow> <mo>+</mo> <mn>&infin;</mn> </mrow>
</munderover>

<mfrac> <mn>1</mn> <msup> <mi>n</mi> <mn>2</mn> </msup> </mfrac>

<mo>=</mo>

<mfrac> <msup> <mn>&pi;</mn> <mn>2</mn> </msup> <mn>6</mn> </mfrac>

</mrow>
</math>
```

$$\sum_{n=1}^{+\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

L'étude détaillée de MathML dépasse le cadre de ce cours. Le lecteur curieux pourra se reporter aux liens indiqués ci-après.

> http://www.w3.org/Math/
> http://fr.wikipedia.org/wiki/MathML
> https://developer.mozilla.org/en/Mozilla_MathML_Project
> http://www.dessci.com/en/products/mathplayer/

15

Feuilles de style CSS

CHACUN a l'habitude de consulter des sites Web chamarrés. Or rien dans les éléments qui ont été étudiés ne permet d'attribuer une couleur particulière à son contenu. De manière plus générale, rien ne permet de contrôler l'aspect visuel des pages une fois affichées dans un butineur.

En effet, HTML a pour but de décrire la structure des pages, pas leur apparence. HTML 3.2 a bien introduit des éléments et des attributs, dont certains ont été repris dans HTML 4, permettant de modifier l'aspect visuel du document mais leur utilisation est maintenant à proscrire.

Supposons néanmoins que les indications concernant leur aspect visuel soient intégrées aux pages HTML. Afin d'avoir une présentation cohérente de toutes les pages d'un même site, ces indications doivent donc figurer à l'identique dans chaque page. L'ajout d'une nouvelle page nécessite alors la recopie de ces indications sans en omettre. Et si l'on souhaite modifier globalement l'aspect visuel des pages existantes, il faut toutes les modifier, ce qui est difficilement gérable s'il y en a beaucoup.

Manifestement, l'approche consistant à indiquer l'aspect visuel des pages dans celles-ci n'est pas viable. Les directives de présentation sont donc regroupées dans un fichier séparé appelé *feuille de style* (*style sheet* en anglais). Les pages HTML d'un même site font alors référence à cette feuille de style pour les indications concernant leur aspect visuel. La centralisation de ces indications dans la feuille de style assure une présentation cohérente de l'ensemble des pages et en facilite les changements.

Une feuille de style est un fichier texte au format CSS (*Cascading Style Sheets*, feuilles de style en cascade, nous verrons ce qu'est la cascade au paragraphe 15.4.2 page 329). La version actuelle de ce format est la version 2.1, datant de 2004. C'est cette version que nous allons étudier par la suite. Pour information, la première version du format CSS, CSS 1, est apparue en 1996, CSS 2 en 1998 et la version 3 est actuellement en cours de développement.

Les possibilités offertes par les feuilles de style étant vastes et complexes, nous n'allons par la suite que survoler le sujet. Le lecteur curieux pourra se reporter aux ouvrages suivants de la bibliographie pour approfondir ses connaissances :

- *CSS* [31]
- *CSS 2* [33]
- *CSS 2.1* [30]
- *CSS avancées* [36]
- *Mémento CSS* [34]
- *CSS : The Definitive Guide* [41]
- *CSS Cookbook* [47]
- *CSS Pocket Reference* [42]
- *Réussir son site web avec XHTML et CSS* [44]
- *Premiers pas en CSS et XHTML* [32]
- *XHTML et CSS* [50]
- *HTML5 et CSS3* [39]
- *HTML5 et CSS3* [51]
- *Sams Teach Yourself HTML and CSS in 24 Hours* [40]
- *HTML & CSS : The Good Parts* [37]
- *HTML5 & CSS3* [38]

> http://www.w3.org/Style/CSS/
> http://jigsaw.w3.org/css-validator/
> http://fr.wikipedia.org/wiki/Feuilles_de_style_en_cascade
> http://fr.wikibooks.org/wiki/Le_langage_CSS
> http://w3schools.com/
> http://www.csszengarden.com/

15.1 Faire référence à une feuille de style dans une page HTML

Pour faire référence à une feuille de style dans une page HTML, il faut ajouter un élément vide `link` dans son en-tête :

```
<link rel="stylesheet" type="text/css" href="style.css"/>
```

L'élément `link` sert à indiquer que la page HTML doit être reliée à un autre document, en l'occurrence à une feuille de style. Il s'utilise avec trois attributs :

1. l'attribut `rel` indique à quelle catégorie de document la page doit être reliée, sa valeur doit être `stylesheet` pour une feuille de style ;

2. l'attribut `type` indique le type de ce document, sa valeur doit être `text/css` pour une feuille de style CSS (puisque le format CSS est un type particulier de fichier texte) ;
3. l'attribut `href` indique le nom du fichier qui contient la feuille de style, ici `style.css` (les feuilles de style CSS ont traditionnellement une extension `.css`).

Pour faire en sorte que toutes les pages d'un site Web soient affichées conformément à une même feuille de style, chacune de ces pages doit avoir dans son en-tête un élément `link` y faisant référence.

> http://www.w3.org/TR/html401/struct/links.html#edef-LINK

15.2 Au sujet des exemples qui suivent

Les exemples de feuilles de style qui suivent seront accompagnés si nécessaire d'un extrait de page HTML les utilisant (en supposant qu'il est bien fait référence à la feuille de style dans l'en-tête au moyen de l'élément `link`). Ils seront tous deux illustrés si besoin est par leur rendu graphique dans Firefox 11.0.

15.3 Syntaxe

Commençons par planter le décor avec une feuille de style très simple :

```
body
{
   background-color : lime ;
}
```

Il suffit de faire référence à cette feuille de style au moyen de l'élément `link` dans l'en-tête d'une page HTML pour que celle-ci se retrouve affichée sur un fond vert criard.

Le principe des feuilles de style est en effet d'appliquer des propriétés aux éléments contenus dans une page HTML, afin de modifier leur apparence. Ici, l'élément concerné est `body`, dont on modifie la propriété `background-color` (qui représente sa couleur de fond) pour lui donner la valeur `lime` (qui est une variante de vert, les couleurs seront abordées au paragraphe 15.5 page 332).

15.3.1 Les commentaires

Les commentaires sont délimités par `/*` et `*/` et peuvent s'étendre sur plusieurs lignes :

```
/* la règle suivante modifie la couleur de fond
   de la page */

body
{
   background-color : lime ;   /* vert clair */
}
```

http://www.w3.org/TR/CSS21/syndata.html#comments

15.3.2 Les règles

De manière générale, une feuille de style est constituée d'une suite de règles.
Une *règle* se compose :

1. d'un *sélecteur*, qui indique le ou les éléments dont l'apparence doit être modifiée ;
2. d'un *bloc* contenant, entre accolades, une ou plusieurs déclarations à appliquer au sélecteur.

Une *déclaration* se compose :

1. d'une *propriété*, qui est le paramètre à modifier ;
2. d'une *valeur* à donner à ce paramètre.

Dans une déclaration, propriété et valeur sont séparées par un deux-points. La déclaration est terminée par un point-virgule.

Une règle est donc de la forme suivante :

```
sélecteur
{
   propriété : valeur ;
   propriété : valeur ;
   ...
}
```

Les retours à la ligne, l'alignement vertical des accolades, l'indentation des déclarations, les espaces autour des deux-points ainsi que ceux précédant les points-virgules ne sont pas nécessaires mais permettent d'aérer la feuille de style, de mieux en visualiser la structure et de repérer plus facilement les erreurs.

On aurait donc parfaitement pu écrire :

```
sélecteur{propriété:valeur;propriété:valeur;}
```

mais il faut reconnaître que cela est bien moins lisible.

http://www.w3.org/TR/CSS21/syndata.html#rule-sets

15.3.3 Les sélecteurs

Les exemples de cette section font appel à des propriétés et des valeurs qui seront étudiées plus loin. Elles ne sont là que pour illustrer les possibilités d'utilisation des sélecteurs au moyen d'exemples réalistes et ne seront donc pas détaillées plus que nécessaire à la compréhension des différentes syntaxes de sélecteurs.

Dans une règle, le sélecteur indique le ou les éléments auxquels appliquer les déclarations figurant dans le bloc qui suit.

http://www.w3.org/TR/CSS21/selector.html

15.3.3.1 Les sélecteurs de type

Dans la plupart des cas, le sélecteur est simplement un nom d'élément, c'est ce qu'on appelle un *sélecteur de type* :

```
body
{
   background-color : lime ;
}
```

http://www.w3.org/TR/CSS21/selector.html#type-selectors

15.3.3.2 Le regroupement

Dans le cas où l'on souhaite appliquer les mêmes déclarations à plusieurs éléments, il est inutile de répéter les blocs à l'identique, on peut regrouper les éléments dans le sélecteur en les séparant par des virgules et n'avoir qu'un seul bloc, partagé par tous ces éléments :

```
h1 , h2 , h3
{
   text-align : center ;
}
```

Ici, on applique exactement le même bloc aux éléments `h1`, `h2` et `h3` en une seule règle, c'est ce qu'on appelle un *regroupement*.

Lorsque les blocs appliqués à différents éléments partagent des déclarations identiques, il est possible de les factoriser. Considérons l'exemple suivant :

```
h1
{
   text-align : center ;
   color : red ;
}

h2
{
   text-align : center ;
   color : blue ;
}
```

La première déclaration de chaque bloc est identique donc il est possible de la placer dans un bloc commun aux deux éléments :

```
h1 , h2
{
   text-align : center ;
}

h1
{
   color : red ;
}

h2
{
   color : blue ;
}
```

Lorsque plusieurs blocs s'appliquent au même élément, leurs déclarations se cumulent. Les deux exemples précédents sont donc équivalents.

En pratique, ce n'est que dans le cas où plusieurs éléments partagent de nombreuses déclarations identiques qu'il est intéressant de les factoriser, afin de simplifier la feuille de style. Les deux exemples précédents n'ont pas vraiment la taille critique pour que l'intérêt de la factorisation soit flagrant.

http://www.w3.org/TR/CSS21/selector.html#grouping

15.3.3.3 Les sélecteurs descendants

Si l'on souhaite appliquer des déclarations à un élément uniquement si celui-ci est imbriqué dans un autre, on peut utiliser cette syntaxe :

```
p code
{
   text-decoration : underline ;
}
```

Ici, le contenu de l'élément `code` sera souligné s'il est contenu dans un paragraphe, y compris s'il est niché au sein d'autres éléments :

```
<p>
L'élément <code>code</code> est souligné, même s'il
est contenu dans un autre élément, par exemple
<samp><code>samp</code></samp>.
</p>
```

L'élément code est souligné, même s'il est contenu dans un autre élément, par exemple samp.

En revanche, il ne le sera pas s'il est contenu dans une liste ou un tableau.

C'est ce qu'on appelle un *sélecteur descendant* dans la mesure où l'élément `code` est un descendant de l'élément `p`.

http://www.w3.org/TR/CSS21/selector.html#descendant-selectors

15.3.3.4 Les sélecteurs d'enfant

On peut être plus restrictif :

```
p > code
{
   text-decoration : underline ;
}
```

Le combinateur `>` signifie que l'element `code` doit être directement contenu dans l'élément `p`, sans intermédiaire :

```
<p>
L'élément <code>code</code> est souligné, mais pas s'il
est contenu dans un autre élément, par exemple
<samp><code>samp</code></samp>.
</p>
```

L'élément code est souligné, mais pas s'il est contenu dans un autre élément, par exemple samp.

Pour que la règle soit appliquée, il faut que l'élément `code` soit un descendant direct de l'élément `p`, c'est-à-dire, pour continuer l'analogie généalogique, un de ses enfants. C'est pourquoi ce type de sélecteur est appelé un *sélecteur d'enfant*.

http://www.w3.org/TR/CSS21/selector.html#child-selectors

15.3.3.5 Les sélecteurs d'attribut

Les sélecteurs précédents permettent d'appliquer un bloc de déclarations à un ou plusieurs éléments, soit inconditionnellement, soit uniquement sur des conditions de descendance par rapport à d'autres éléments.

Un *sélecteur d'attribut* permet d'appliquer un bloc de déclarations à un élément si celui-ci est muni d'un certain attribut, quelle que soit sa valeur, ou seulement s'il a une valeur particulière.

Il est constitué de l'élément, suivi de l'attribut dont il doit être muni, entre crochets :

```
élément[attribut]
{
   propriété : valeur ;
   ...
}
```

Ainsi, dans l'exemple suivant, les éléments `td` ayant un attribut `colspan`, quelle que soit sa valeur, seront affichés sur un fond de couleur jaune :

```
td[colspan]
{
   background-color : yellow ;
}
```

On peut également limiter l'application du sélecteur aux seuls éléments ayant un attribut d'une certaine valeur (contrairement à son utilisation dans une page HTML, la valeur ne doit pas figurer entre guillemets) :

```
élément[attribut=valeur]
{
   propriété : valeur ;
   ...
}
```

Ainsi, dans l'exemple suivant, les éléments `td` ayant un attribut `colspan` de valeur 2 seront affichés sur un fond de couleur jaune :

```
td[colspan=2]
{
   background-color : yellow ;
}
```

Il existe encore deux autres syntaxes de sélecteurs d'attribut, que nous n'étudierons pas ici.

> http://www.w3.org/TR/CSS21/selector.html#attribute-selectors

15.3.3.6 Les sélecteurs de classe

Les sélecteurs de classe sont des cas particuliers de sélecteurs d'attribut. Étant très utilisés, ils bénéficient d'une syntaxe propre.

Une *classe* est une chaîne de caractères pouvant contenir :

- des lettres (minuscules ou majuscules, attention, on fait la différence entre les deux), éventuellement accentuées ;
- des chiffres (mais une classe ne peut pas commencer par un chiffre) ;
- des tirets ;
- des tirets bas.

Un *sélecteur de classe* permet d'appliquer un bloc de déclarations à un élément. Il est constitué de l'élément, suivi d'un point, puis de la classe, sans espaces autour du point :

```
élément.classe
{
   propriété : valeur ;
   ...
}
```

Dans la page HTML, il faut appliquer à l'élément l'attribut `class` avec la classe comme valeur ainsi :

```
<élément class="classe">contenu</élément>
```

Par exemple, si certains paragraphes importants doivent être affichés en rouge, on peut utiliser :

```
p.important
{
   color : red ;
}
```

```
<p class="important">
Ceci est un paragraphe important.
</p>
```

Il est également possible de ne pas appliquer la classe à un élément spécifique :

```
.classe
{
   propriété : valeur ;
   ...
}
```

Ceci permet d'utiliser cette classe avec n'importe quel élément sans avoir à définir autant de sélecteurs. Par exemple :

```
.important
{
   color : red ;
}
```

```
<p class="important">
Ceci est un paragraphe important.
</p>

<p>
Ceci est un paragraphe normal avec une
<dfn class="important">définition importante</dfn>.
</p>
```

À cette occasion, il importe de bien nommer ses classes en fonction de leur nature et non de leur aspect. Dans l'exemple précédent, on a choisi de différencier les choses importantes en leur attribuant la couleur rouge. On aurait donc pu être tenté de nommer notre classe `rouge` puisqu'elle a pour résultat de mettre le contenu des éléments auxquels elle est appliquée en rouge. Cependant, si l'on change d'avis et qu'on décide de mettre les choses importantes en bleu, il faut modifier non seulement la feuille de style mais aussi toutes les pages HTML où il est fait référence à la classe `rouge`. Encore une fois, il est préférable de raisonner en terme de structure et de nommer les classes en fonction de leur signification structurelle plutôt que de leur apparence. En appelant notre classe `important`, on indique sa nature et non son aspect. Une chose importante reste importante, qu'elle soit affichée en rouge ou en bleu. Ainsi une modfication de l'aspect des choses importantes ne nécessitera que de modifier la feuille de style, pas les pages HTML qui y font référence.

> http://www.w3.org/TR/CSS21/selector.html#class-html

15.3.3.7 Les sélecteurs d'identifiant

Les sélecteurs d'*identifiant* fonctionnent sur le même principe que les sélecteurs de classe :

```
élément#identifiant
{
   propriété : valeur ;
   ...
}
```

Le caractère utilisé dans la feuille de style pour séparer élément et identifiant n'est pas le point comme pour la classe mais le croisillon `#`.

Dans la page HTML, on utilise l'attribut `id` appliqué au même élément ainsi :

```
<élément id="identifiant">contenu</élément>
```

Il est également possible de ne pas appliquer l'identifiant à un élément spécifique :

```
#identifiant
{
   propriété : valeur ;
   ...
}
```

À la différence d'une classe, qui peut être utilisée plusieurs fois dans la même page HTML, un identifiant ne peut l'être qu'une seule fois. C'est la différence fondamentale entre classe et identifiant. Ceci permet de s'assurer qu'un même identifiant n'est utilisé qu'une et une seule fois dans une page HTML.

Les identifiants sont en pratique principalement utilisés avec l'élément `div` (qui sera étudié au paragraphe 15.11 page 353). Celui-ci permet d'indiquer des divisions logiques dans une page HTML (en-tête, barre de navigation, partie principale, pied de page, etc.). Chaque division est repérée par un identifiant et leur unicité permet de garantir qu'il n'y ait qu'une division de chaque type par page.

> http://www.w3.org/TR/CSS21/selector.html#id-selectors

15.3.3.8 Les pseudo-classes

Les sélecteurs étudiés précédemment permettent l'application des règles de la feuille de style en fonction du seul contenu de la page HTML (éléments, relations entre éléments, présence d'attributs `class` ou `id`).

Dans les cas où ces critères ne sont pas suffisants, on utilise des *pseudo-classes*. Le sélecteur d'une pseudo-classe se présente comme ceci :

```
élément:pseudo-classe
{
   propriété : valeur ;
   ...
}
```

http://www.w3.org/TR/CSS21/selector.html#pseudo-class-selectors

15.3.3.8.1 Les pseudo-classes de lien

Par défaut, les hyperliens sont soulignés et affichés en bleu s'ils n'ont pas encore été visités et en violet s'ils l'ont déjà été. Si l'on a modifié la couleur de fond, il est possible que ces deux couleurs ne conviennent pas. Les pseudo-classes de lien permettent de les modifier. Elles ne peuvent s'appliquer qu'à l'élément `a`.

La pseudo-classe `:link` permet de modifier la couleur des liens qui n'ont pas encore été visités :

```
a:link
{
   color : lime ;
}
```

La pseudo-classe `:visited` permet de modifier la couleur des liens qui ont déjà été visités :

```
a:visited
{
   color : green ;
}
```

http://www.w3.org/TR/CSS21/selector.html#link-pseudo-classes

15.3.3.8.2 Les pseudo-classes dynamiques

Les pseudo-classes dynamiques permettent d'appliquer des déclarations en fonction des actions de l'utilisateur. Contrairement aux pseudo-classes de lien, qui ne s'appliquent qu'à l'élément `a`, les pseudo-classes dynamiques peuvent s'appliquer à n'importe quel élément.

La pseudo-classe `:hover` s'applique lorsque le pointeur de la souris survole l'élément correspondant.

On peut l'utiliser, par exemple, pour donner à l'utilisateur un retour visuel plus important lorsque sa souris survole un hyperlien en modifiant la couleur de fond de celui-ci :

```
a:hover
{
   background-color : yellow ;
}
```

Il existe également des pseudo-classes `:active` et `:focus`, que nous n'étudierons pas.

http://www.w3.org/TR/CSS21/selector.html#dynamic-pseudo-classes

15.3.3.8.3 La pseudo-classe de langue

L'attribut `xml:lang`, étudié au paragraphe 14.9 page 310, permet d'indiquer la langue du contenu de l'élément auquel il est appliqué.

La pseudo-classe `:lang` permet de se baser sur la valeur de l'attribut `xml:lang` pour appliquer des déclarations en fonction de celle-ci (`XX` représente le code ISO 639 de la langue) :

```
élément:lang(XX)
{
   propriété : valeur ;
   ...
}
```

Ainsi, dans l'exemple suivant, les paragraphes ayant un attribut `xml:lang` de valeur `en` seront composés en italique :

```
p:lang(en)
{
   font-style : italic ;
}
```

L'élément peut être omis dans le sélecteur afin de pouvoir appliquer les déclarations à tous les éléments ayant la valeur correspondante de l'attribut `xml:lang` :

```
:lang(en)
{
   font-style : italic ;
}
```

http://www.w3.org/TR/CSS21/selector.html#lang

15.3.4 Les déclarations et les propriétés

Contrairement aux sélecteurs, qui offrent une grande variété de syntaxes, celle des déclarations est des plus simples :

```
{
   propriété : valeur ;
   ...
}
```

Leur richesse se situe dans les modifications qu'elles permettent d'apporter aux éléments des pages HTML.

Après une courte explication de ce que sont l'héritage et la cascade, nous allons étudier les propriétés les plus utiles selon un classement thématique.

http://www.w3.org/TR/CSS21/syndata.html#declaration

15.4 L'héritage et la cascade

15.4.1 L'héritage

L'*héritage* consiste, pour un élément contenu dans un autre, à se voir appliquées les règles concernant ce dernier. Considérons le fragment de page HTML suivant :

```
<p>
L'<em>héritage</em> consiste, pour un élément contenu dans un
autre, à se voir appliquées les règles concernant ce dernier.
</p>
```

Dans ce cas, toutes les règles qui s'appliquent à l'élément `p` s'appliqueront également à l'élément `em`. Cela permet une mise en page cohérente sans avoir à multiplier des déclarations identiques.

Si certaines propriétés s'appliquant spécifiquement à l'élément `em` sont en contradiction avec les mêmes propriétés héritées de l'élément `p` (c'est-à-dire si elles ont des valeurs différentes), ce sont bien évidemment les plus spécifiques, à savoir les premières, qui s'appliquent.

http://www.w3.org/TR/CSS21/cascade.html#inheritance

15.4.2 La cascade

Vous avez remarqué que, même sans faire référence à une feuille de style au moyen de l'élément `link`, les butineurs savent afficher certains éléments de manière adéquate, par exemple les titres, qui sont affichés en gras et dans un corps [1] plus important que celui du texte. En effet, les butineurs disposent d'une feuille de style interne, qui contient les règles correspondantes et qui permettent d'afficher les pages sans feuille de style avec une mise en page minimale mais correcte.

Il est également possible à l'utilisateur d'indiquer un certain nombre de préférences pour l'affichage des pages, comme les polices de caractères à utiliser, leur taille, les couleurs du texte, du fond et des liens, etc.

Enfin, lorsqu'on en utilise une, la feuille de style permet à l'auteur des pages d'indiquer ses propres règles d'affichage.

1. En typographie, le *corps* désigne la taille des caractères.

Puisque le butineur (au moyen de sa feuille de style interne), l'utilisateur (au moyen de ses préférences) et la feuille de style peuvent indiquer des règles contradictoires, il est nécessaire de les départager.

La *cascade* définit l'ordre de priorité dans lequel les règles sont prises en compte, de la moins prioritaire à la plus prioritaire :

1. la feuille de style interne du butineur ;
2. les préférences de l'utilisateur ;
3. la feuille de style.

Il est important de remarquer que la feuille de style est plus prioritaire que les préférences de l'utilisateur. Ceci peut poser problème si, par exemple, la feuille de style indique un corps de petite taille alors que l'utilisateur a indiqué un corps plus important dans ses préférences en raison de problèmes de vue. C'est pour cela qu'il est maladroit d'indiquer le corps dans la feuille de style de manière absolue. Il est préférable de le faire de manière relative (voir le paragraphe 15.7.4 page 338) afin de pouvoir prendre en compte les préférences de l'utilisateurs tout en ayant une mise en page conforme aux souhaits de l'auteur de la feuille de style.

En ce qui concerne les déclarations figurant dans la feuille de style, la cascade permet que toutes celles qui peuvent s'appliquer soient prises en compte, dans la mesure où elles ne sont pas contradictoires. Considérons cette feuille de style :

```
p
{
   text-align : justify ;
}

p.important
{
   color : red ;
}
```

Dans une page HTML utilisant cette feuille de style, les paragraphes appartenant à la classe `important` seront composés en rouge mais également justifiés. En effet, les déclarations correspondant au sélecteur de classe `p.important` s'appliquent à ces paragraphes mais également celles correspondant au sélecteur de type `p` car un paragraphe, même s'il appartient à une classe particulière, reste un paragraphe.

En cas de déclarations contradictoires, la cascade impose que ce soient les plus spécifiques qui s'appliquent. Considérons cette feuille de style :

```
p
{
   color : green ;
   text-align : justify ;
}

p.important
{
   color : red ;
}
```

Dans une page HTML utilisant cette feuille de style, les paragraphes appartenant à la classe `important` seront composés en rouge et néanmoins justifiés. La déclaration concernant la propriété `color` correspondant au sélecteur de type `p` est contradictoire avec celle concernant la même propriété correspondant au sélecteur de classe `p.important`. Cette dernière étant plus spécifique, c'est elle qui s'applique. En revanche, la déclaration concernant la propriété `text-align` correspondant au sélecteur de type `p` n'a pas de déclaration équivalente pour le sélecteur de classe `p.important` et peut donc s'appliquer.

Enfin, en cas de déclarations contradictoires sans qu'il y ait un avantage de spécificité, la cascade impose que ce soit la dernière déclaration qui s'applique. En effet, la feuille de style est lue de haut en bas et les règles sont prises en compte dans l'ordre de leur lecture.

```
p
{
   color : green ;
}

p
{
   color : red ;
}
```

Les pages HTML utilisant la feuille de style précédente auront donc leurs paragraphes composés en rouge.

http://www.w3.org/TR/CSS21/cascade.html#cascade

15.5 Les couleurs

L'une des principales utilisations des feuilles de style est de mettre de la couleur dans des pages auparavant ternes.

http://www.w3.org/TR/CSS21/colors.html

15.5.1 Nommage et représentation des couleurs

La façon la plus simple d'indiquer une couleur est de la désigner par son nom. Le tableau 15.1 présente les dix-sept noms utilisables dans une feuille de style.

aqua	fuchsia	lime	olive	red	white
black	gray	maroon	orange	silver	yellow
blue	green	navy	purple	teal	

TABLE 15.1 – *Noms de couleurs utilisables dans une feuille de style.*

La plupart des butineurs acceptent également d'autres noms qui ne font pas partie de la spécification des feuilles de style. Pour cette raison, il est préférable de ne pas les utiliser et de s'en tenir aux dix-sept noms du tableau 15.1.

Néanmoins, ceux-ci correspondent à des couleurs assez criardes. Il est donc également possible de spécifier des couleurs de manière plus fine en indiquant leurs niveaux de rouge, de vert et de bleu en synthèse additive.

La façon la plus simple de spécifier une couleur est d'indiquer les pourcentages respectifs de rouge, de vert et de bleu :

```
color : rgb( 80% , 100% , 80% ) ;
```

Il est important qu'il n'y ait pas d'espace entre `rgb` et la parenthèse ouvrante, de même qu'entre les nombres et leur signe pour cent.

Les niveaux peuvent également être indiqués sous forme de valeurs entières codées sur un octet, c'est-à-dire par des nombres entre 0 et 255, 0 correspondant à 0 % et 255 à 100 % :

```
color : rgb( 204 , 255 , 204 ) ;
```

Le système hexadécimal (voir le paragraphe 2.1.3.3 page 26) étant mieux adapté à la réprésentation d'octets, il est également possible d'exprimer les valeurs de rouge, de vert et de bleu par des nombres hexadécimaux allant de 00 à FF. Ainsi, la couleur précédente peut aussi s'écrire :

```
color : #CCFFCC ;
```

Le croisillon indique qu'on utilise une représentation en hexadécimal avec deux chiffres pour chaque couleur primaire. Les chiffres de A à F peuvent également être indiqués en minuscules.

Les couleurs utilisant deux fois le même chiffre hexadécimal pour chaque couleur primaire peuvent être simplifiées ainsi :

```
color : #CFC ;
```

> http://www.w3.org/TR/CSS21/syndata.html#color-units
> http://fr.wikipedia.org/wiki/Rouge_vert_bleu
> http://fr.wikipedia.org/wiki/Couleurs_du_Web
> http://www.colorpicker.com/
> http://www.febooti.com/products/iezoom/online-help/online-color-chart-picker.html

15.5.2 La propriété de couleur `color`

La propriété `color` permet de définir la couleur des éléments auxquels elle est appliquée :

```
h1
{
   color : red ;
}
```

Ici, tous les éléments `h1` seront affichés en rouge.

Les couleurs peuvent être spécifiées de toutes les manières indiquées au paragraphe 15.5.1 page 332.

> http://www.w3.org/TR/CSS21/colors.html#propdef-color

15.5.3 La propriété de couleur de fond `background-color`

La propriété `background-color` permet de définir la couleur de fond des éléments auxquels elle est appliquée :

```
body
{
   background-color : lime ;
}
```

Ici, la page HTML entière sera affichée sur un fond vert.

Les couleurs peuvent être spécifiées de toutes les manières indiquées au paragraphe 15.5.1 page 332.

> http://www.w3.org/TR/CSS21/colors.html#propdef-background-color

15.6 Les unités de mesure

Avant de poursuivre, il faut présenter les unités de mesure utilisables dans une feuille de style car plusieurs propriétés qui vont être étudiées par la suite les utilisent comme valeurs. Ces unités figurent dans le tableau 15.2 page ci-contre.

Les deux premières unités appellent quelques explications.

En typographie, un *cadratin* est une espace dont la longueur est égale à la force du corps utilisé (c'est-à-dire à la taille des caractères). Le cadratin est souvent défini comme la largeur du caractère « M » (mais ce n'est souvent qu'approximatif).

Toujours en typographie, la *hauteur d'œil* représente la hauteur d'un caractère. Ici, `1ex` représente donc la hauteur du caractère « x » (ou, si l'on préfère, de n'importe quelle minuscule sans hampe ni jambage).

Ces deux unités sont intéressantes car elles sont relatives à la taille des caractères, ce qui n'est pas le cas des autres (ces dernières sont donc rarement utilisées).

`em`	cadratin	
`ex`	hauteur d'œil d'un « x »	
`px`	pixel	
`in`	pouce (*inch* en anglais)	`1in` = `2.54cm`
`cm`	centimètre	
`mm`	millimètre	
`pt`	point	`1pt` = `1in/72`
`pc`	pica	`1pc` = `12pt`

TABLE 15.2 – *Unités de mesure utilisables dans une feuille de style.*

Lorsqu'une longueur doit être exprimée dans une feuille de style, elle l'est sous la forme d'un nombre immédiatement suivi d'une unité, par exemple : `2em`, `5mm`. Le nombre peut être décimal lorsque cela a un sens, par exemple : `0.25em` (quart de cadratin). En revanche, `0.25px` n'a pas de sens, le pixel étant le plus petit élément affichable par un écran.

Remarquez que la notation utilisée dans le tableau 15.2 pour expliquer la valeur du point, `1in/72` n'est syntaxiquement pas correcte dans une feuille de style. Cette notation n'a qu'une valeur explicative.

Enfin, il est possible d'omettre l'unité lorsque la longueur à exprimer est nulle : `0` seul est suffisant.

> http://www.w3.org/TR/CSS21/syndata.html#length-units
> http://fr.wikipedia.org/wiki/Cadratin
> http://www.orthotypographie.fr/volume-I/cabochon-citation.html#Cadratin
> http://fr.wikipedia.org/wiki/Œil_(typographie)
> http://www.orthotypographie.fr/volume-II/ocean-ouest.html#OEil

15.7 Le texte

Deux ensembles de propriétés permettent d'agir sur le texte. Celles commençant par `font` concernent les caractères, celles commençant par `text` concernent la présentation globale du texte

> http://www.w3.org/TR/CSS21/fonts.html
> http://www.w3.org/TR/CSS21/text.html

15.7.1 La propriété de police `font-family`

La propriété `font-family` permet de définir la police de caractères utilisée :

```
td
{
   font-family : sans-serif ;
}
```

Celle-ci peut être indiquée de deux manières :
- en indiquant un nom de police spécifique (comme, par exemple, `Helvetica` ou `Courier`) ;
- en indiquant un nom de police générique.

Les noms de polices génériques sont indiqués dans le tableau 15.3.

`serif`	police à empattements
`sans-serif`	police bâton
`cursive`	police cursive
`fantasy`	police décorative
`monospace`	police à chasse fixe

TABLE 15.3 – *Noms de polices génériques utilisables avec la propriété `font-family`.*

En cas d'utilisation d'un nom de police générique, la police effectivement utilisée dépend des préférences de l'utilisateur ou de la configuration par défaut du butineur.

Comme nous l'avons vu au paragraphe 15.4.2 page 329, compte tenu de la cascade, il est préférable de n'utiliser que des noms de polices génériques, afin d'en laisser le choix à l'utilisateur, sauf cas très particulier. Par ailleurs, une police spécifique peut ne pas être disponible dans toutes les configurations.

> http://www.w3.org/TR/CSS21/fonts.html#propdef-font-family
> http://fr.wikipedia.org/wiki/Empattement_(typographie)
> http://www.orthotypographie.fr/volume-I/echelle-extrait.html#Empattement

> http://fr.wikipedia.org/wiki/Chasse_(typographie)
> http://www.orthotypographie.fr/volume-I/cabochon-citation.html#Chasse-chasser

15.7.2 La propriété d'inclinaison `font-style`

La propriété `font-style` permet de définir l'inclinaison des caractères :

```
p.citation
{
   font-style : italic ;
}
```

La propriété `font-style` peut prendre l'une des valeurs indiquées dans le tableau 15.4.

`normal`	composition en *romain* (c'est-à-dire en caractères droits)
`italic`	composition en *italique*
`oblique`	composition en *oblique* (c'est-à-dire en romain incliné)

TABLE 15.4 – *Valeurs utilisables avec la propriété* `font-style`.

> http://www.w3.org/TR/CSS21/fonts.html#propdef-font-style
> http://fr.wikipedia.org/wiki/Police_romaine
> http://fr.wikipedia.org/wiki/Italique_(typographie)
> http://www.orthotypographie.fr/volume-II/incipit-italique.html#Italique

15.7.3 La propriété de graisse `font-weight`

La propriété `font-weight` permet de définir la graisse des caractères :

```
td
{
   font-weight : bold ;
}
```

La propriété `font-weight` peut prendre trois types de valeurs :
- des valeurs absolues, `normal` pour des caractères normaux, `bold` pour des caractères gras ;
- des valeurs relatives par rapport à ce que serait la graisse sans cette propriété, `bolder` et `lighter` ;
- des valeurs numériques, `100`, `200`, `300`, `400`, `500`, `600`, `700`, `800` et `900` (`normal` correspondant à `400` et `bold` à `700`).

La plupart des polices ne proposant qu'un seul type de gras, il est préférable de s'en tenir à `normal` et `bold`.

> http://www.w3.org/TR/CSS21/fonts.html#propdef-font-weight
> http://fr.wikipedia.org/wiki/Graisse_(typographie)
> http://www.orthotypographie.fr/volume-II/gallerie-guillemet.html#Graisse

15.7.4 La propriété de corps `font-size`

La propriété `font-size` permet de définir le corps des caractères :

```
li
{
   font-size : 75% ;
}
```

Celui-ci peut être indiqué de deux manières :
- de manière absolue, en indiquant une taille avec une unité de mesure ;
- de manière relative, en fonction du corps défini par les préférences de l'utilisateur.

Nous avons vu au paragraphe 15.4.2 page 329 que, compte tenu de la cascade, il était maladroit de définir le corps des caractères de manière absolue. Aussi cette façon de faire ne sera pas détaillée ici.

Concernant la manière relative, il existe trois syntaxes différentes. La plus souple et la plus facilement compréhensible (nous n'étudierons pas les autres) consiste à indiquer un pourcentage de ce que serait le corps sans la propriété `font-size`, comme il est fait dans l'exemple précédent.

> http://www.w3.org/TR/CSS21/fonts.html#propdef-font-size
> http://fr.wikipedia.org/wiki/Corps_typographique
> http://www.orthotypographie.fr/volume-I/classe-corps.html#Corps

15.7.5 La propriété d'alinéa `text-indent`

La propriété `text-indent` permet de définir la taille de l'alinéa débutant chaque paragraphe (qui est par défaut nulle avec la plupart des butineurs) :

```
p
{
   text-indent : 1em ;
}
```

Traditionnellement, l'alinéa ne doit pas être inférieur à un cadratin. Il peut aller jusqu'à quatre cadratins (`4em`) d'après le *Lexique des règles typographiques en usage à l'Imprimerie nationale*, page 153. Le cadratin (`em`) est donc l'unité de mesure la mieux adaptée à la propriété `text-indent`.

> http://www.w3.org/TR/CSS21/text.html#propdef-text-indent
> http://fr.wikipedia.org/wiki/Alinéa
> http://www.orthotypographie.fr/volume-I/accolade-allemand.html#Alinea

15.7.6 La propriété d'alignement `text-align`

La propriété `text-align` permet de définir la façon d'aligner des blocs de texte :

```
p
{
   text-align : justify ;
}
```

La propriété `text-align` peut prendre l'une des valeurs indiquées dans le tableau 15.5.

`left`	le texte est aligné à gauche (présentation dite *en drapeau, au fer* à gauche)
`right`	le texte est aligné à droite (présentation dite *en drapeau, au fer* à droite)
`center`	le texte est centré
`justify`	le texte est justifié

TABLE 15.5 – *Valeurs utilisables avec la propriété* `text-align`.

Par défaut, le texte est aligné à gauche.

La propriété `text-align` peut s'appliquer principalement aux éléments `p`, `li` (pour les valeurs `left` et `justify`) ainsi qu'aux cellules des tableaux (éléments `td` et `th`).

En pratique, elle influence tout ce qui se trouve à l'intérieur de l'élément auquel on l'applique, que ce soit du texte ou autre chose. En particulier, on peut l'utiliser pour centrer une image :

```
p.image
{
   text-align : center ;
}
```

```
<p class="image">
<img src="limousin.jpg" alt="[danse traditionnelle]"/>
</p>
```

Le fait qu'une image soit centrée est en effet une propriété de l'élément qui la contient, pas de l'élément `img` en lui-même.

En ce qui concerne les tableaux, qui ne peuvent pas être contenus dans des paragraphes, la façon de les centrer est différente et sera étudiée au paragraphe 15.9.3 page 352.

http://www.w3.org/TR/CSS21/text.html#propdef-text-align

15.7.7 La propriété de décoration `text-decoration`

La propriété `text-decoration` permet d'ajouter une décoration au texte contenu dans un élément :

```
abbr
{
   text-decoration : underline ;
}
```

La propriété `text-decoration` peut prendre l'une des valeurs indiquées dans le tableau 15.6.

`none`	aucune décoration n'est ajoutée
`underline`	souligne le texte
`overline`	surligne le texte
`line-through`	raye le texte d'une ligne en son milieu
`blink`	fait clignoter le texte

TABLE 15.6 – *Valeurs utilisables avec la propriété `text-decoration`.*

http://www.w3.org/TR/CSS21/text.html#propdef-text-decoration

15.8 Marge, espacement et bordure

Dans une page HTML, tous les éléments sont inscrits dans un rectangle. Il est possible d'agir sur trois types de paramètres, la marge, l'espacement et la bordure, afin respectivement d'éloigner ce rectangle des objets l'entourant, d'augmenter ses dimensions et d'en matérialiser les limites.

Il sera plus facile d'en discuter sur la base d'un exemple. Ainsi, vous trouverez :

- dans la figure 15.1 page suivante un exemple de feuille de style utilisant les propriétés permettant de modifier les paramètres de marge, d'espacement et de bordure ;

- dans la figure 15.2 page ci-contre un exemple de page HTML mettant en œuvre cette feuille de style ;
- dans la figure 15.3 page 344 le rendu de cette page HTML dans un butineur.

```
p
{
   background-color : #CCC ;
   text-align : justify ;
}

p.test
{
   margin : 2em ;

   padding : 1em ;

   border-width : 1em ;
   border-color : black ;
   border-style : solid ;
}
```

FIGURE 15.1 – *Marge, espacement et bordure : exemple de feuille de style.*

Dans la feuille de style (figure 15.1), le rectangle contenant les paragraphes a été matérialisé en l'affichant sur un fond gris au moyen de la propriété `background-color`. Le texte a été justifié grâce à la propriété `text-align` afin qu'il occupe toute la largeur du paragraphe (ce qui permet de mieux en visualier les limites). Ces propriétés s'appliquent à tous les paragraphes. Il a ensuite été créé une classe particulière de paragraphes, dans le but de mettre en œuvre les propriétés permettant de modifier les paramètres de marge, d'espacement et de bordure, que nous allons étudier par la suite.

La page HTML (figure 15.2 page suivante) contient simplement quatre paragraphes au contenu identique, deux paragraphes n'ayant rien de particulier puis deux utilisant la classe `test` afin de montrer l'effet des propriétés permettant de modifier les paramètres de marge, d'espacement et de bordure.

Le rendu de cette page dans un butineur (figure 15.3 page 344) permet de constater que les deux derniers paragraphes sont différents des deux premiers en trois

```
<p>
L'ordinateur fait ce que vous lui dites de faire,
pas ce que vous voulez qu'il fasse.
</p>

<p>
L'ordinateur fait ce que vous lui dites de faire,
pas ce que vous voulez qu'il fasse.
</p>

<p class="test">
L'ordinateur fait ce que vous lui dites de faire,
pas ce que vous voulez qu'il fasse.
</p>

<p class="test">
L'ordinateur fait ce que vous lui dites de faire,
pas ce que vous voulez qu'il fasse.
</p>
```

FIGURE 15.2 – *Marge, espacement et bordure : exemple de page HTML.*

points :

- ils sont davantage éloignés des bords à gauche et à droite ainsi que des objets qui les précèdent et qui les suivent, c'est la *marge* ;
- le fond gris est plus détaché du texte, c'est l'*espacement* ;
- le fond gris est entouré d'un cadre noir, c'est la *bordure*.

La figure 15.4 page 345 schématise les positions respectives de la marge, de l'espacement et de la bordure par rapport au contenu.

On peut voir tous ces éléments comme des poupées russes emboîtées les unes dans les autres. Le contenu ultime de la balise est entouré par l'espacement, lui-même entouré par la bordure, elle-même entourée par la marge.

Par défaut, marge, espacement et bordure sont inexistants et il faut utiliser les propriétés que nous allons maintenant étudier pour leur donner vie.

FIGURE 15.3 – *Marge, espacement et bordure : rendu dans un butineur.*

FIGURE 15.4 – *Marge, espacement et bordure : schéma de principe.*

http://www.w3.org/TR/CSS21/box.html

15.8.1 La marge

Parler de *marge* (*margin* en anglais) au singulier est une simplification. Il existe en effet quatre marges : en haut, à droite, en bas et à gauche. Ces marges peuvent prendre des valeurs différentes qui sont définies respectivement par les propriétés `margin-top`, `margin-right`, `margin-bottom` et `margin-left` :

```
p
{
   margin-top    : 1em ;
   margin-right  : 2em ;
   margin-bottom : 3em ;
   margin-left   : 4em ;
}
```

Si l'on souhaite affecter la même valeur aux quatre marges, on peut utiliser la propriété `margin` :

```
p
{
   margin : 2em ;
}
```

Celle-ci accepte également deux valeurs (qui sont affectées respectivement aux marges haute et basse pour la première, droite et gauche pour la seconde), trois valeurs (qui sont affectées respectivement aux marges haute pour la première, droite et gauche pour la deuxième et basse pour la troisième) ou quatre valeurs (qui sont affectées respectivement aux marges haute, droite, basse et gauche). Néanmoins, cette façon de procéder est difficilement compréhensible et source d'erreur. Si l'on souhaite qu'une des marges ait une valeur différente des autres, il est donc préférable d'utiliser la propriété idoine.

Il est bien entendu possible de ne définir que certaines marges, les autres restant à leur valeur par défaut (généralement nulle) :

```
p
{
   margin-right : 2em ;
   margin-left  : 2em ;
}
```

Vous pouvez remarquer que les marges des deux derniers paragraphes de la figure 15.3 page 344 ne sont pas affichées sur fond gris, contrairement aux paragraphes et à leurs espacements. En effet, les marges sont toujours transparentes.

Par ailleurs, dans la même figure, remarquez que la marge séparant les deux derniers paragraphes semble être de la même taille que les marges gauche et droite alors qu'elle devrait être deux fois plus importante, puisqu'elle cumule la marge du bas du troisième paragraphe et la marge du haut du quatrième paragraphe. Ce n'est pas le cas parce que les marges verticales séparant deux éléments fusionnent, la marge résultante étant la plus grande des deux. Ceci ne se produit pas avec les marges horizontales.

Enfin, il est également possible d'utiliser la valeur `auto` pour laisser le butineur calculer automatiquement certaines marges. C'est en particulier le cas pour centrer les tableaux (voir le paragraphe 15.9.3 page 352).

> http://www.w3.org/TR/CSS21/box.html#margin-properties

15.8.2 L'espacement

Là aussi, l'*espacement* (*padding* en anglais) se subdivise en quatre : en haut, à droite, en bas et à gauche. Ces espacements peuvent prendre des valeurs différentes

qui sont définies respectivement par les propriétés `padding-top`, `padding-right`, `padding-bottom` et `padding-left` :

```
p
{
   padding-top    : 1em ;
   padding-right  : 2em ;
   padding-bottom : 3em ;
   padding-left   : 4em ;
}
```

Si l'on souhaite affecter la même valeur aux quatre espacements, on peut utiliser la propriété `padding` :

```
p
{
   padding : 2em ;
}
```

Celle-ci accepte également deux, trois ou quatre valeurs sur le même modèle que la propriété `margin`.

Il est bien entendu possible de ne définir que certains espacements, les autres restant à leur valeur par défaut (généralement nulle) :

```
p
{
   padding-right : 2em ;
   padding-left  : 2em ;
}
```

Remarquez que les espacements des deux derniers paragraphes de la figure 15.3 page 344 ont hérité de la couleur de fond attribuée aux paragraphes au moyen de la propriété `background-color`. Il n'est pas possible d'attribuer à un espacement une couleur de fond différente de celle de son élément.

http://www.w3.org/TR/CSS21/box.html#padding-properties

15.8.3 La bordure

La *bordure* (*border* en anglais) matérialise la frontière entre espacement et marge :

```
p
{
   border-width : 1em ;
   border-color : black ;
   border-style : solid ;
}
```

Trois propriétés permettent d'en définir l'aspect :
- la propriété `border-width` indique son épaisseur ;
- la propriété `border-color` indique sa couleur ;
- la propriété `border-style` indique son style.

La propriété `border-style` peut prendre l'une des valeurs indiquées dans le tableau 15.7.

`none`	n'affiche pas la bordure
`hidden`	identique à `none` sauf dans des cas particuliers impliquant des cellules de tableaux (que nous ne détaillerons pas)
`dotted`	affiche la bordure en pointillés
`dashed`	affiche la bordure sous la forme d'un trait interrompu
`solid`	affiche la bordure sous la forme d'un train plein
`double`	affiche la bordure sous la forme d'un double trait
`groove`	affiche la bordure en pseudo-relief sous la forme d'une ligne rentrante
`ridge`	affiche la bordure en pseudo-relief sous la forme d'une ligne saillante
`inset`	affiche la bordure en pseudo-relief en donnant l'impression que l'élément et son espacement sont incrustés dans la page
`outset`	affiche la bordure en pseudo-relief en donnant l'impression que l'élément et son espacement sortent de la page

TABLE 15.7 – *Valeurs utilisables avec la propriété* `border-style`.

Les propriétés `border-width`, `border-color` et `border-style` s'utilisent en fait comme les propriétés `margin` et `padding`, avec une, deux, trois ou quatre valeurs, contrôlant les quatre côtés de la bordure.

Signalons également l'existence de propriétés qui permettent d'indiquer individuellement l'épaisseur, la couleur et le style d'un des quatre côtés de la bordure. Ces propriétés sont présentées dans le tableau 15.8.

Épaisseur	Couleur	Style
`border-top-width`	`border-top-color`	`border-top-style`
`border-right-width`	`border-right-color`	`border-right-style`
`border-bottom-width`	`border-bottom-color`	`border-bottom-style`
`border-left-width`	`border-left-color`	`border-left-style`

TABLE 15.8 – *Propriétés de configuration de l'épaisseur, de la couleur et du style des quatre côtés de la bordure.*

Néanmoins, il est particulièrement inesthétique que les quatre côtés de la bordure ne soient pas homogènes. Aussi, l'utilisation de ces propriétés est vivement déconseillée.

http://www.w3.org/TR/CSS21/box.html#border-properties

15.9 Les tableaux

Plusieurs propriétés permettent de modifier la présentation des tableaux. Nous allons laisser de côté celles qui sont complexes et qui ont peu d'intérêt pour nous concentrer sur le positionnement de la légende, la fusion des bordures et le centrage d'un tableau.

http://www.w3.org/TR/CSS21/tables.html

15.9.1 La propriété de positionnement de légende `caption-side`

La propriété `caption-side` permet de définir le positionnement de l'élément `caption` :

```
caption
{
   caption-side : bottom ;
}
```

La propriété `caption-side` peut prendre l'une des valeurs indiquées dans le tableau 15.9.

`top`	la légende est placée au-dessus du tableau (c'est le cas par défaut)
`bottom`	la légende est placée au-dessous du tableau

TABLE 15.9 – *Valeurs utilisables avec la propriété `caption-side`.*

Par ailleurs, la légende est par défaut centrée par rapport au tableau, son positionnement horizontal peut être modifié au moyen de la propriété `text-align` (voir le paragraphe 15.7.6 page 339).

http://www.w3.org/TR/CSS21/tables.html#propdef-caption-side

15.9.2 La propriété de fusion des bordures `border-collapse`

Vous l'aurez remarqué, l'aspect par défaut des séparations horizontales et verticales des tableaux n'est pas vraiment avenant. En particulier, le fait d'utiliser des lignes doubles en alourdit la présentation.

Il est possible de changer cela au moyen de la propriété `border-collapse` appliquée à l'élément `table` et en matérialisant la bordure des cellules grâce aux propriétés étudiées au paragraphe 15.8.3 page 348 :

```
table
{
   border-collapse : collapse ;
}

td , th
{
   border-width : 2px ;
   border-style : solid ;
}
```

```
<table>
<tr> <th>Nom</th>          <th>Numéro</th> </tr>
<tr> <td>Export</td>       <td>33</td>     </tr>
<tr> <td>Pastis</td>       <td>51</td>     </tr>
<tr> <td>Kronenbourg</td> <td>1664</td>   </tr>
</table>
```

Nom	Numéro
Export	33
Pastis	51
Kronenbourg	1664

La propriété `border-collapse` peut prendre l'une des valeurs indiquées dans le tableau 15.10.

`collapse`	provoque la fusion des bordures des cellules
`separate`	indique que les bordures des cellules doivent rester distinctes (c'est la valeur par défaut)

TABLE 15.10 – *Valeurs utilisables avec la propriété* `border-collapse`.

Notez qu'il est inutile d'utiliser l'attribut `border` de l'élément `table` lorsqu'on matérialise la bordure des cellules.

http://www.w3.org/TR/CSS21/tables.html#propdef-border-collapse

15.9.3 Centrage d'un tableau

Étant donné qu'un tableau ne peut pas être contenu dans un paragraphe, le centrage d'un tableau ne peut pas être effectué au moyen de la propriété `text-align`, étudiée au paragraphe 15.7.6 page 339.

Pour centrer un tableau, il faut laisser le butineur calculer les marges gauche et droite en leur donnant la valeur `auto` :

```
table
{
   margin-left  : auto ;
   margin-right : auto ;
}
```

15.10 L'élément span

Les propriétés étudiées jusqu'ici permettent de modifier l'aspect de divers éléments. Mais que faire si l'on souhaite modifier l'aspect d'une partie du texte qui n'est encadrée par aucun élément ? L'élément `span` permet de résoudre ce problème.

Par exemple, il est d'usage de composer en italique les mots issus d'une langue étrangère. Nous avons vu au paragraphe 15.3.3.8.3 page 327 comment utiliser la pseudo-classe `:lang` pour appliquer des déclarations en fonction de la valeur de l'attribut `xml:lang`. On peut alors écrire, dans la page HTML :

```
<p>
Y a des gens qui n'ont pas réussi parce qu'ils ne sont
pas <span xml:lang="en">aware</span>, ils ne sont pas
au courant.
</p>
```

Et le mot *aware* sera mis en italique grâce à la règle suivante dans la feuille de style :

```
:lang(en)
{
   font-style : italic ;
}
```

Le sélecteur de langue ne spécifie pas d'élément afin de pouvoir être appliqué à d'autres éléments que span au besoin.

Au lieu d'utiliser l'élément span, l'attribut xml:lang et un sélecteur de langue, il aurait été bien plus simple d'utiliser l'élément i pour mettre directement le mot en italique. Néanmoins, cette solution n'est pas élégante car elle ne donne aucune information structurelle (elle ne permet pas, en particulier, d'identifier la langue des mots étrangers).

L'élément span peut être considéré comme un élément à tout faire dès qu'il s'agit de modifier l'aspect d'une partie du texte n'étant pas délimitée par un élément (du moins avant de rajouter l'élément span). Il est principalement utilisé avec des sélecteurs de classe :

```
.important
{
   color : red ;
}
```

```
<p>
Par ailleurs, il est absolument essentiel de
<span class="important">ne pas nourrir le mogwaï
après minuit</span> car les conséquences peuvent
se révéler désastreuses.
</p>
```

http://www.w3.org/TR/html401/struct/global.html#edef-SPAN

15.11 L'élément div et les divisions

L'élément div permet de définir des *divisions* dans une page HTML. Dans un site, les différentes pages sont généralement construites selon le même gabarit. Elles sont

formées d'un ensemble de blocs rectangulaires agencés de la même façon dans toutes les pages afin d'avoir une certaine cohérence visuelle. Par exemple, un bloc pour l'en-tête, un bloc pour la barre latérale de navigation, un bloc pour la partie principale de la page et un bloc pour le pied de page. Ces blocs constituent les divisions. Chaque bloc est contenu dans un élément `div` dont la valeur de l'attribut `id` détermine la fonction. Grâce à des sélecteurs d'identifiant (voir le paragraphe 15.3.3.7 page 324), la feuille de style permet d'agencer les divisions les unes par rapport aux autres.

Nous allons par la suite étudier comment utiliser les divisions pour réaliser quelques mises en pages classiques.

> http://www.w3.org/TR/html401/struct/global.html#edef-DIV

15.11.1 Les deux méthodes d'agencement des divisions

Il existe deux méthodes d'agencement des divisions, chacune ayant ses avantages et ses inconvénients.

> http://www.w3.org/TR/CSS21/visuren.html
> http://www.w3.org/TR/CSS21/visudet.html

15.11.1.1 Agencement des divisions avec la propriété `position`

La première méthode d'agencement des divisions repose sur l'utilisation de la propriété `position` et des quatre propriétés `top`, `right`, `bottom` et `left` permettant de positionner avec précision le rectangle contenant la division par rapport à un ou plusieurs côtés de la page.

Les propriétés `top`, `right`, `bottom` et `left` acceptent chacune une valeur pouvant être une longueur (voir le paragraphe 15.6 page 334) ou un pourcentage (les pourcentages sont relatifs à la taille de la fenêtre du butineur). Cette valeur indique la distance qui sépare le côté correspondant de la division par rapport au bord de la fenêtre du butineur. On utilise généralement une ou deux de ces propriétés afin de positionner la division verticalement et horizontalement. Il est rare d'utiliser trois ou quatre de ces propriétés.

La propriété `position` permet de définir le comportement de la division quand on fait défiler le contenu de la fenêtre du butineur (à supposer qu'elle ne permette pas d'afficher la totalité du contenu de la page). Elle peut prendre l'une des valeurs indiquées dans le tableau 15.11 page suivante.

`absolute`	la division défile avec le reste de la page
`fixed`	la division reste toujours au même endroit, même après défilement

TABLE 15.11 – *Valeurs utilisables avec la propriété* `position`.

Le choix entre `absolute` et `fixed` est avant tout une affaire de préférence personnelle. Une division (par exemple une barre latérale de navigation) qui ne défile pas peut être intéressante parce qu'elle est toujours accessible. En revanche, si elle est suffisamment haute pour dépasser de la fenêtre du butineur vers le bas (ce qui peut arriver sur des écrans de petite taille comme ceux des téléphones mobiles), cela rend le bas de la barre latérale de navigation inaccessible.

Enfin, pour être absolument exact, la propriété `position` peut également prendre les valeurs `static` et `relative`, qui ne présentent pas d'intérêt pour positionner une division et qui ne seront donc pas détaillées.

> http://www.w3.org/TR/CSS21/visuren.html#propdef-position
> http://www.w3.org/TR/CSS21/visuren.html#propdef-top
> http://www.w3.org/TR/CSS21/visuren.html#propdef-right
> http://www.w3.org/TR/CSS21/visuren.html#propdef-bottom
> http://www.w3.org/TR/CSS21/visuren.html#propdef-left

15.11.1.2 Agencement des divisions avec la propriété `float`

La seconde méthode d'agencement des divisions repose sur l'utilisation de la propriété `float` et permet de positionner les divisions de manière plus approximative mais plus souple.

La propriété `float` peut prendre deux valeurs, `left` et `right` indiquant si la division doit être placée respectivement à gauche ou à droite. À gauche ou à droite de quoi ? De ce qu'il y a déjà du même côté. L'ordre d'apparition des divisions dans la page HTML en fonction de l'endroit où l'on souhaite les placer les unes par rapport aux autres a donc son importance.

Outre l'agencement des divisions, la propriété `float` permet également d'envelopper une image contenue dans un paragraphe avec le texte de celui-ci (et des suivants si l'image est suffisamment haute et le paragraphe suffisamment court) après l'avoir placée à gauche ou à droite de la page :

```
p img
{
   float : left ;
   margin : 0.5em ;
}
```

```
<p>
Former des ingénieurs capables d’assurer la conception,
la réalisation et la direction de systèmes complexes,
sous des contraintes économiques fortes et dans un
<img src="ENSTA-ParisTech.jpg" alt="[logo ENSTA]"/>
environnement international.
</p>

<p>
Former des ingénieurs capables d’assurer la conception,
la réalisation et la direction de systèmes complexes,
sous des contraintes économiques fortes et dans un
environnement international.
</p>
```

Former des ingénieurs capables d'assurer la conception, la réalisation et la direction de systèmes complexes, sous des contraintes économiques fortes et dans un environnement international.

ENSTA ParisTech

Former des ingénieurs capables d'assurer la conception, la réalisation et la direction de systèmes complexes, sous des contraintes économiques fortes et dans un environnement international.

> http://www.w3.org/TR/CSS21/visuren.html#propdef-float

15.11.1.3 Quelle méthode utiliser ?

Dans les cas simples comportant deux ou trois divisions, ces deux méthodes permettent d'aboutir au même résultat. C'est dans les cas plus complexes que l'une est généralement préférable à l'autre. Les exemples qui suivent vont illustrer l'utilisation de ces propriétés.

15.11.2 Dimensionnement des divisions

Les dimensions du rectangle contenant une division s'adaptent automatiquement à son contenu, comme c'est le cas pour tous les éléments étudiés. Néanmoins il est possible que le contenu d'une division varie d'une page à une autre, entraînant des changements des dimensions de cette division qui peuvent être inesthétiques.

Il est donc courant de spécifier la largeur ou la hauteur d'une division (en pratique, on peut le faire pour n'importe quel élément mais cette possibilité est rarement utilisée en dehors des divisions) au moyen respectivement des propriétés `width` et `height`. Celles-ci acceptent comme valeur soit une longueur (voir le paragraphe 15.6 page 334) soit un pourcentage (les pourcentages sont relatifs à la taille de la fenêtre du butineur).

> http://www.w3.org/TR/CSS21/visudet.html#propdef-width
> http://www.w3.org/TR/CSS21/visudet.html#propdef-height

15.11.3 Une barre latérale de navigation

L'utilisation la plus répandue des divisions consiste à inclure une barre latérale de navigation permettant d'accéder aux différentes pages ou rubriques d'un site sans avoir à le parcourir d'une manière linéaire.

Dans la page HTML (figure 15.5 page suivante), ceci se fait simplement en rajoutant deux éléments `div`, l'un encadrant une liste d'hyperliens pour créer la barre latérale de navigation, l'autre encadrant la partie principale de la page. Chaque division est munie de son propre identifiant. Si plusieurs pages d'un même site doivent être munies d'une barre latérale de navigation, la division correspondante doit figurer dans chaque page.

Il est important que la division correspondant à la barre latérale de navigation soit indiquée avant celle correspondant à la partie principale de la page pour l'agencement

```
<div id="barre">

<ul>
   <li> <a href="page1.html">page1</a> </li>
   <li> <a href="page2.html">page2</a> </li>
   <li> <a href="page3.html">page3</a> </li>
   <li> <a href="page4.html">page4</a> </li>
   <li> <a href="page5.html">page5</a> </li>
</ul>

</div>

<div id="principal">

<p>
Former des ingénieurs capables d'assurer la conception,
la réalisation et la direction de systèmes complexes,
sous des contraintes économiques fortes et dans un
environnement international.
</p>

</div>
```

FIGURE 15.5 – *Barre latérale de navigation : exemple de page HTML.*

avec la propriété `float`. En effet, avec l'ordre inverse, plutôt que d'avoir les deux divisions côte à côte, le haut de la barre latérale de navigation serait placé au niveau du bas de la partie principale. En revanche, l'ordre des divisions n'a pas d'importance pour l'agencement avec la propriété `position`.

Bien entendu, il n'est indiqué nulle part dans la page HTML où doivent être placées les deux divisions. Nous allons étudier comment les agencer au moyen de la propriété `position` (paragraphe 15.11.3.1 page ci-contre) puis au moyen de la propriété `float` (paragraphe 15.11.3.2 page 360). La barre latérale de navigation sera placée à gauche et la partie principale à droite. La disposition inverse est laissée en exercice au lecteur.

15.11.3.1 Agencement avec la propriété position

La feuille de style présentée dans la figure 15.6 permet l'agencement de la barre latérale de navigation et de la partie principale de la page HTML de la figure 15.5 page précédente au moyen de la propriété position.

```
div#barre
{
   background-color : #cccccc ;
   position : absolute ;
   top : 0 ;
   left : 0 ;
   width : 20% ;
}

div#principal
{
   position : absolute ;
   top : 0 ;
   left : 0 ;
   margin-left : 20% ;
}
```

FIGURE 15.6 – *Barre latérale de navigation (propriété position) : exemple de feuille de style.*

Tout d'abord, étudions les propriétés de la division barre, correspondant à la barre latérale de navigation :

- la propriété background-color permet de matérialiser le fond de la barre latérale de navigation afin de bien en discerner les limites et de la différencier visuellement de la partie principale ;
- la propriété position indique, par sa valeur absolute, que la barre latérale de navigation pourra défiler avec le reste de la page ;
- la propriété top indique que la barre latérale de navigation sera placée à la distance 0 du haut de la page ;
- la propriété left indique que la barre latérale de navigation sera placée à la distance 0 du bord gauche de la page ;

– la propriété `width` indique que la barre latérale de navigation aura une largeur égale à 20 % de celle de la page.

La combinaison des propriétés `top` et `left` fait que la barre latérale de navigation sera positionnée en haut et à gauche de la page.

Les propriétés de la division `principal`, correspondant à la partie principale de la page, n'appellent que peu de commentaires. Cette division est positionnée de la même façon que la barre latérale de navigation. Néanmoins, une marge gauche (propriété `margin-left`) de 20 % permet d'en décaler le contenu afin de laisser suffisamment d'espace à la barre latérale de navigation. Il est important que cette marge ait une largeur supérieure ou égale à celle de la barre latérale de navigation.

FIGURE 15.7 – *Barre latérale de navigation (propriété `position`) : rendu dans un butineur.*

Enfin, le rendu de la page HTML dans un butineur (figure 15.7) permet d'apprécier le résultat. On se rend compte à cette occasion que le texte de la division principale est collé au fond coloré de la barre de navigation latérale. Attribuer une petite valeur (par exemple `1em`) à l'espacement gauche de la division principale permettrait de l'aérer un peu. De même, il n'est pas nécessaire que la liste de liens soit si éloignée du bord gauche de la barre latérale de navigation. Les listes ont par défaut un espacement gauche important. Lui attribuer une valeur inférieure (par exemple `1.5em`) pourrait être intéressant.

Lorsqu'on agence des divisions, il arrive fréquemment de se rendre compte qu'il faut agir sur une marge ou un espacement ici ou là.

15.11.3.2 Agencement avec la propriété `float`

La feuille de style présentée dans la figure 15.8 page suivante permet l'agencement de la barre latérale de navigation et de la partie principale de la page HTML de la figure 15.5 page 358 au moyen de la propriété `float`.

```
div#barre
{
   background-color : #cccccc ;
   float : left ;
   width : 20% ;
}

div#principal
{
   margin-left : 20% ;
}
```

FIGURE 15.8 – *Barre latérale de navigation (propriété `float`) : exemple de feuille de style.*

Dans la division `barre`, on retrouve les propriétés `background-color` et `width` alors que les propriétés `position`, `top` et `left` ont été remplacées par la seule propriété `float`, dont la valeur est `left`. Ceci indique que la division doit être placée à gauche (par rapport à ce que serait sa position autrement). Dans la mesure où elle est déjà en haut de la page, il n'est nul besoin d'indiquer sa position verticale.

De la même façon, dans la division `principal`, il n'est nécessaire d'indiquer que la largeur de la marge gauche.

FIGURE 15.9 – *Barre latérale de navigation (propriété `float`) : rendu dans un butineur.*

Enfin, le rendu de la page HTML dans un butineur (figure 15.9) montre que la barre latérale de navigation est ici plus éloignée des bords haut et gauche de la page. Elle est en effet soumise aux marges par défaut imposé à l'élément `body` par

le butineur, ce qui n'est pas le cas avec la propriété `position`. On se rend compte qu'il faudrait également agir sur les largeurs de l'espacement gauche de la division principale et de la liste de liens, comme dans l'exemple précédent.

15.11.4 Deux barres latérales

Une autre disposition courante comporte deux barres latérales, l'une à gauche et l'autre à droite.

Dans la page HTML (figure 15.10 page ci-contre), ceci se fait simplement avec trois éléments `div`, un pour chaque barre latérale et un pour la partie principale de la page.

Comme dans l'exemple avec une seule barre, il est important que la division correspondant à la partie principale de la page soit la dernière pour que l'agencement avec la propriété `float` soit correct.

15.11.4.1 Agencement avec la propriété `position`

La feuille de style présentée dans la figure 15.11 page 364 permet l'agencement des barres latérales et de la partie principale de la page HTML de la figure 15.10 page ci-contre au moyen de la propriété `position`.

Le principe est dans la lignée de celui de l'exemple avec une seule barre et n'appelle pas de commentaires particuliers.

Enfin, le rendu de la page HTML dans un butineur (figure 15.12 page 365) permet d'apprécier le résultat.

15.11.4.2 Agencement avec la propriété `float`

La feuille de style présentée dans la figure 15.13 page 365 permet l'agencement des barres latérales et de la partie principale de la page HTML de la figure 15.10 page suivante au moyen de la propriété `float`.

Le principe est dans la lignée de celui de l'exemple avec une seule barre et n'appelle pas de commentaires particuliers.

Enfin, le rendu de la page HTML dans un butineur (figure 15.14 page 366) permet d'apprécier le résultat.

```
<div id="barre-1">

<ul>
   <li> <a href="page1.html">page1</a> </li>
   <li> <a href="page2.html">page2</a> </li>
   <li> <a href="page3.html">page3</a> </li>
   <li> <a href="page4.html">page4</a> </li>
   <li> <a href="page5.html">page5</a> </li>
</ul>

</div>

<div id="barre-2">

<p>
Ici, on peut indiquer les dernières nouvelles.
</p>

<hr/>

<p>
Ici, on peut indiquer les dernières nouvelles.
</p>

</div>

<div id="principal">

<p>
Former des ingénieurs capables d'assurer la conception,
la réalisation et la direction de systèmes complexes,
sous des contraintes économiques fortes et dans un
environnement international.
</p>

</div>
```

FIGURE 15.10 – *Deux barres latérales : exemple de page HTML.*

```
div#barre-1
{
   background-color : #cccccc ;
   position : absolute ;
   top : 0 ;
   left : 0 ;
   width : 30% ;
}

div#barre-2
{
   background-color : #cccccc ;
   position : absolute ;
   top : 0 ;
   right : 0 ;
   width : 30% ;
}

div#principal
{
   position : absolute ;
   top : 0 ;
   left : 0 ;
   margin-left : 30% ;
   margin-right : 30% ;
}
```

FIGURE 15.11 – *Deux barres latérales (propriété `position`) : exemple de feuille de style.*

FIGURE 15.12 – *Deux barres latérale (propriété `position`) : rendu dans un butineur.*

```
div#barre-1
{
   background-color : #cccccc ;
   float : left ;
   width : 30% ;
}

div#barre-2
{
   background-color : #cccccc ;
   float : right ;
   width : 30% ;
}

div#principal
{
   margin-left : 30% ;
   margin-right : 30% ;
}
```

FIGURE 15.13 – *Deux barres latérales (propriété `float`) : exemple de feuille de style.*

FIGURE 15.14 – *Deux barre latérales (propriété `float`) : rendu dans un butineur.*

Annexes

A

Récapitulatif des commandes des éditeurs de texte

Déplacement	
aller en début de ligne	`C-a`
aller en fin de ligne	`C-e`
se déplacer de mot en mot	`C-` ←
	`C-` →
se déplacer de paragraphe en paragraphe	`C-` ↑
	`C-` ↓
aller sur la ligne numéro *n*	`M-g M-g` ou `M-g g` puis `saisir` *n* `dans le minibuffer`
Suppression	
supprimer le caractère sous le curseur	`C-d`
supprimer la fin du mot à partir du curseur	`M-d`
supprimer la fin de la ligne	`C-k`
Exécution multiple	
exécuter *n* fois une commande	`M-`*n* `<commande>`
répéter la dernière commande	`C-x z`
Copier-coller, couper-coller	
copier-coller	à la souris
couper-coller	souris, `C-w` et souris
Annulation	
annuler la modification précédente	`C-/` ou `C-_`
annuler la commande en cours	`C-g`

TABLE A.1 – *Récapitulatif des commandes d'Emacs étudiées.*

Fichiers	
sauvegarder le fichier	`C-x C-s`
sauvegarder le fichier en le nommant	`C-x C-w`
lire un fichier	`C-x C-f`
Recherche, remplacement	
recherche interactive (vers le bas)	`C-s`
recherche interactive (vers le haut)	`C-r`
recherche et remplacement	`M-%`
Divers	
reformater le paragraphe	`M-q`
appeler une commande par son nom complet	`M-x`
couper la fenêtre en deux	`C-x 2`
Quitter	
quitter	`C-x C-c`

TABLE A.1 – *Récapitulatif des commandes d'Emacs étudiées (suite et fin).*

Passage en mode insertion, en mode commande	
passage en mode insertion au caractère situé sous le curseur	`i`
passage en mode insertion au caractère situé après le curseur	`a`
passage en mode insertion en début de ligne	`I`
passage en mode insertion en fin de ligne	`A`
passage en mode insertion au-dessous de la ligne actuelle	`o`
passage en mode insertion au-dessus de la ligne actuelle	`O`
passage en mode commande	Esc
Déplacement	
gauche, bas, haut, droite	`h j k l`
aller en début de ligne	`0`
aller en fin de ligne	`$`
se déplacer de mot en mot	`W B`

TABLE A.2 – *Récapitulatif des commandes de `vi` et de Vim étudiées.*

se déplacer de paragraphe en paragraphe	`{ }`
aller sur la dernière ligne	`G`
aller sur la ligne numéro *n*	*n*`G`

TABLE A.2 – *Récapitulatif des commandes de* `vi` *et de Vim étudiées.*

Suppression	
supprimer le caractère sous le curseur	`x`
supprimer la fin du mot à partir du curseur	`dw`
supprimer la fin de la ligne	`D`
supprimer la ligne	`dd`
Exécution multiple	
exécuter *n* fois une commande	*n*`<commande>`
répéter la dernière commande	`.`
Copier-coller, couper-coller	
copier-coller	`[`*n*`]yy p`
couper-coller	`[`*n*`]dd p`
Annulation	
annuler la modification précédente	`u`
annuler la commande en cours	Esc
Fichiers	
sauvegarder le fichier	`:w`
lire un fichier	`:e`
Recherche, remplacement	
recherche (vers le bas)	`/`
recherche (vers le haut)	`?`
recherche et remplacement	`:s`
Divers	
joindre la ligne suivante à la ligne courante	`J`
Les commandes suivantes sont spécifiques à Vim.	
limiter la saisie des lignes à *n* caractères	`:set textwidth=`*n*
reformater le paragraphe	`gqap`
Quitter	
quitter	`:q`
sauvegarder le fichier et quitter	`:wq :x ZZ`

TABLE A.2 – *Récapitulatif des commandes de* `vi` *et de Vim étudiées.*

quitter sans sauvegarder le fichier	`:q!`

TABLE A.2 – *Récapitulatif des commandes de* `vi` *et de Vim étudiées (suite et fin).*

B

Récapitulatif des commandes

Commande	Description	Où
`date`	affiche la date et l'heure	§ 3.5 page 36
`passwd`	change le mot de passe de l'utilisateur	§ 3.6 page 37

TABLE B.1 – *Récapitulatif des commandes étudiées dans la première partie, « Introduction ».*

Commande	Description	Où
`iconv`	convertit le codage d'un fichier texte	§ 4.5.1 page 53
`emacs`	éditeur de texte Emacs	§ 5.1 page 57
`vi`	éditeur de texte `vi`	§ 6.1 page 73
`vim`	éditeur de texte Vim	§ 6.1 page 73

TABLE B.2 – *Récapitulatif des commandes étudiées dans la deuxième partie, « Du texte et des éditeurs de texte ».*

Commande	Description	Où
`sh`	Bourne shell	§ 7.1 page 88
`csh`	C shell	§ 7.1 page 88
`tcsh`	TENEX C shell	§ 7.1 page 88
`ksh`	Korn shell	§ 7.1 page 88
`bash`	Bourne-again shell	§ 7.1 page 88
`zsh`	Z shell	§ 7.1 page 88

TABLE B.3 – *Récapitulatif des commandes étudiées dans le chapitre 7, « L'interpréteur de commandes ».*

Commande	Description	Où
`cal`	affiche le calendrier	§ 7.2.3 page 93
`ls`	affiche la liste des fichiers	§ 7.3 page 97
`cat`	affiche le contenu d'un fichier texte	§ 7.4.1 page 101
`more`	affiche le contenu d'un fichier texte	§ 7.4.2 page 102
	avec contrôle	§ 9.5.1 page 204
`less`	affiche le contenu d'un fichier texte	§ 7.4.3 page 103
	avec contrôle	§ 9.5.1 page 204
`man`	affiche le manuel d'une commande	§ 7.5 page 103
`clear`	efface l'écran de l'interpréteur de commandes	§ 7.6 page 104
`exit`	quitte l'interpréteur de commandes	§ 7.7 page 105

TABLE B.3 – *Récapitulatif des commandes étudiées dans le chapitre 7, « L'interpréteur de commandes » (suite et fin).*

Commande	Description	Où
`cp`	copie un fichier	§ 8.1.1 page 107
	copie plusieurs fichiers	§ 8.3.3.1 page 122
`mv`	renomme un fichier	§ 8.1.2 page 109
	déplace des fichiers	§ 8.3.3.2 page 124
`rm`	supprime des fichiers	§ 8.1.3 page 110
	supprime des répertoires	§ 8.3.7.2 page 134
`ln`	crée des liens	§ 8.1.4 page 111
		§ 8.3.3.3 page 124
`mkdir`	crée des répertoires	§ 8.3.1 page 118
`cd`	change de répertoire	§ 8.3.4 page 125
`pwd`	affiche le répertoire courant	§ 8.3.4 page 125
`rmdir`	supprime des répertoires	§ 8.3.7.1 page 133
`chmod`	modifie les droits d'accès	§ 8.5.3 page 145
`umask`	modifie les droits d'accès	§ 8.5.4 page 147
`find`	retrouve des fichiers	§ 8.7.1.1 page 151
`quota`	affiche ses quotas	§ 8.7.1.2 page 154
`du`	affiche l'espace occupé par chaque répertoire	§ 8.7.1.3 page 156
`df`	affiche l'occupation des disques	§ 8.7.1.4 page 158

TABLE B.4 – *Récapitulatif des commandes étudiées dans le chapitre 8, « Gestion, organisation et traitement des fichiers ».*

Commande	Description	Où
`compress`	comprime des fichiers	§ 8.7.2.1 page 161
`uncompress`	décomprime des fichiers	§ 8.7.2.1 page 161
`gzip`	comprime des fichiers	§ 8.7.2.2 page 161
`gunzip`	décomprime des fichiers	§ 8.7.2.2 page 161
`bzip2`	comprime des fichiers	§ 8.7.2.3 page 163
`bunzip2`	décomprime des fichiers	§ 8.7.2.3 page 163
`lzma`	comprime des fichiers	§ 8.7.2.4 page 164
`unlzma`	décomprime des fichiers	§ 8.7.2.4 page 164
`xz`	comprime des fichiers	§ 8.7.2.5 page 164
`unxz`	décomprime des fichiers	§ 8.7.2.5 page 164
`tar`	archivage	§ 8.7.3.1 page 166
`zip`	archivage et compression	§ 8.7.4.1 page 169
`unzip`	désarchivage	§ 8.7.4.1 page 169
`unrar`	désarchivage	§ 8.7.4.2 page 171

TABLE B.4 – *Récapitulatif des commandes étudiées dans le chapitre 8, « Gestion, organisation et traitement des fichiers » (suite et fin).*

Commande	Description	Où
`history`	affiche l'historique des commandes	§ 9.1 page 173
`type`	affiche le chemin d'accès absolu d'une commande	§ 9.2.3 page 179
`which`	affiche le chemin d'accès absolu d'une commande	§ 9.2.3 page 179
`ps`	affiche la liste des processus	§ 9.3.3 page 185
`top`	affiche la liste des processus	§ 9.3.4 page 187
`kill`	envoie un signal à un processus	§ 9.3.5 page 188
`true`	envoie un statut de sortie vrai	§ 9.3.6.3 page 192
`false`	envoie un statut de sortie faux	§ 9.3.6.3 page 192
`grep`	recherche utilisant une expression rationnelle	§ 9.5.2 page 205
`awk`	utilise le langage de programmation AWK	§ 9.5.3 page 209
`sed`	éditeur de texte non interactif	§ 9.5.4 page 214
`sort`	trie les lignes d'un texte	§ 9.5.5 page 216

TABLE B.5 – *Récapitulatif des commandes étudiées dans le chapitre 9, « Commandes, processus et redirections ».*

Commande	Description	Où
`wc`	affiche le nombre de lignes, de mots et d'octets d'un texte	§ 9.5.6 page 219
`nohup`	lance un processus en l'émancipant	§ 9.6.1 page 221
`nice`	lance un processus en modifiant sa priorité	§ 9.6.2 page 222
`renice`	modifie la priorité d'un processus	§ 9.6.2 page 222
`screen`	multiplexeur de terminaux	§ 9.6.3 page 224
`tmux`	multiplexeur de terminaux	§ 9.6.4 page 226

TABLE B.5 – *Récapitulatif des commandes étudiées dans le chapitre 9, « Commandes, processus et redirections ».*

Commande	Description	Où
`lpq`	affiche la file d'attente d'une imprimante	§ 10.2.1 page 230
`lprm`	supprime un travail d'impression	§ 10.2.2 page 231
`lpr`	imprime un fichier PostScript	§ 10.4.1 page 235
`psselect`	sélectionne les pages à imprimer	§ 10.4.2.1 page 236
`psnup`	imprime plusieurs pages par feuille de papier	§ 10.4.2.2 page 236
`a2ps`	imprime un fichier texte	§ 10.5.1 page 237
`enscript`	imprime un fichier texte	§ 10.5.2 page 240
`iconv`	convertit le codage d'un fichier texte	§ 10.5.3 page 242
`pdftops`	convertit un fichier PDF en fichier PostScript	§ 10.6 page 242

TABLE B.6 – *Récapitulatif des commandes étudiées dans le chapitre 10, « Impression ».*

Commande	Description	Où
`ssh`	connexion sur un ordinateur distant	§ 11.2 page 249
`ssh-keygen`	affiche l'empreinte d'une clé publique	§ 11.2.1 page 249
`hostname`	affiche le nom de l'ordinateur	§ 11.2.1 page 249
`scp`	transfert de fichier avec un ordinateur distant	§ 11.3 page 253
`who`	affiche la liste des utilisateurs connectés	§ 11.4 page 254
`w`	affiche la liste des utilisateurs connectés	§ 11.4 page 254

TABLE B.7 – *Récapitulatif des commandes étudiées dans le chapitre 11, « Connexion à distance ».*

Commande	Description	Où
`test`	teste fichiers, chaînes de caractères et entiers	§ 12.7.1.1 page 265

TABLE B.8 – *Récapitulatif des commandes étudiées dans le chapitre 12, « Programmation »*.

C

Récapitulatif des éléments HTML et des propriétés CSS

Élément	Description	Où
`a`	hyperlien	§ 14.5 page 303
`abbr`	abréviation	§ 14.4.5 page 300
`acronym`	acronyme	§ 14.4.5 page 300
`b`	gras	§ 14.4.6 page 301
`body`	corps de la page	§ 14.1.2 page 295
`br`	retour à la ligne	§ 14.4.4 page 300
`caption`	légende d'un tableau	§ 14.8 page 308
`cite`	citation	§ 14.4.5 page 300
`code`	code informatique	§ 14.4.5 page 300
`dd`	description d'un élément de liste `dl`	§ 14.7 page 306
`dfn`	définition	§ 14.4.5 page 300
`div`	division	§ 15.11 page 353
`dl`	liste contenant des définitions	§ 14.7 page 306
`dt`	terme d'un élément de liste `dl`	§ 14.7 page 306
`em`	mise en valeur	§ 14.4.5 page 300
`h1`	titre de premier niveau	§ 14.4.1 page 296
`h2`	titre de deuxième niveau	§ 14.4.1 page 296
`h3`	titre de troisième niveau	§ 14.4.1 page 296
`h4`	titre de quatrième niveau	§ 14.4.1 page 296
`h5`	titre de cinquième niveau	§ 14.4.1 page 296
`h6`	titre de sixième niveau	§ 14.4.1 page 296
`head`	en-tête de la page	§ 14.1.1 page 294
`hr`	filet	§ 14.4.3 page 299
`html`	document HTML	§ 14.1 page 292

TABLE C.1 – *Récapitulatif des éléments HTML étudiés.*

Élément	Description	Où
`i`	italique	§ 14.4.6 page 301
`img`	image	§ 14.6 page 304
`kbd`	saisie au clavier	§ 14.4.5 page 300
`li`	élément de liste `ul` et `ol`	§ 14.7 page 306
`link`	fait référence à une feuille de style	§ 15.1 page 314
`meta`	informations supplémentaires sur le document	§ 14.1.1 page 294
`ol`	liste ordonnée	§ 14.7 page 306
`p`	paragraphe	§ 14.4.2 page 297
`samp`	exemple de résultat	§ 14.4.5 page 300
`span`	élément à tout faire	§ 15.10 page 352
`strong`	mise en valeur de manière plus forte que `em`	§ 14.4.5 page 300
`sub`	inférieur	§ 14.4.7 page 302
`sup`	supérieur	§ 14.4.7 page 302
`table`	tableau	§ 14.8 page 308
`td`	cellule de donnée d'un tableau	§ 14.8 page 308
`th`	cellule de titre d'un tableau	§ 14.8 page 308
`title`	titre	§ 14.1.1 page 294
`tr`	ligne d'un tableau	§ 14.8 page 308
`tt`	police à chasse fixe	§ 14.4.6 page 301
`ul`	liste non ordonnée	§ 14.7 page 306
`var`	nom d'une variable ou d'un argument	§ 14.4.5 page 300

TABLE C.1 – *Récapitulatif des éléments HTML étudiés (suite et fin).*

Propriété	Description	Où
`background-color`	couleur de fond	§ 15.5.3 page 334
`border-collapse`	fusion des bordures d'un tableau	§ 15.9.2 page 350
`border-color`	couleur de la bordure	§ 15.8.3 page 348
`border-style`	style de la bordure	§ 15.8.3 page 348
`border-width`	épaisseur de la bordure	§ 15.8.3 page 348
`bottom`	position d'une division	§ 15.11.1.1 page 354
`caption-side`	position de la légende d'un tableau	§ 15.9.1 page 349
`color`	couleur	§ 15.5.2 page 333

TABLE C.2 – *Récapitulatif des propriétés CSS étudiées.*

Propriété	Description	Où
`float`	position d'une division	§ 15.11.1.2 page 355
`font-family`	police de caractères	§ 15.7.1 page 336
`font-size`	corps des caractères	§ 15.7.4 page 338
`font-style`	inclinaison des caractères	§ 15.7.2 page 337
`font-weight`	graisse des caractères	§ 15.7.3 page 337
`height`	hauteur	§ 15.11.2 page 357
`left`	position d'une division	§ 15.11.1.1 page 354
`margin`	marge	§ 15.8.1 page 345
`margin-bottom`	marge du bas	§ 15.8.1 page 345
`margin-left`	marge de gauche	§ 15.8.1 page 345
`margin-right`	marge de droite	§ 15.8.1 page 345
`margin-top`	marge du haut	§ 15.8.1 page 345
`padding`	espacement	§ 15.8.2 page 346
`padding-bottom`	espacement du bas	§ 15.8.2 page 346
`padding-left`	espacement de gauche	§ 15.8.2 page 346
`padding-right`	espacement de droite	§ 15.8.2 page 346
`padding-top`	espacement du haut	§ 15.8.2 page 346
`position`	position d'une division	§ 15.11.1.1 page 354
`right`	position d'une division	§ 15.11.1.1 page 354
`text-align`	alignement du texte	§ 15.7.6 page 339
`text-decoration`	décoration du texte	§ 15.7.7 page 341
`text-indent`	taille de l'alinéa	§ 15.7.5 page 339
`top`	position d'une division	§ 15.11.1.1 page 354
`width`	largeur	§ 15.11.2 page 357

TABLE C.2 – *Récapitulatif des propriétés CSS étudiées (suite et fin).*

Bibliographie

De nombreux ouvrages figurant dans cette bibliographie sont consultables à la bibliothèque de l'ENSTA ParisTech.

Du texte et des éditeurs de texte

[1] Debra CAMERON.
GNU Emacs Pocket Reference.
O'Reilly Media, novembre 1998.
URL : http://shop.oreilly.com/product/9781565924963.do.

[2] Debra CAMERON, James ELLIOTT, Marc LOY, Eric RAYMOND et Bill ROSENBLATT.
Learning GNU Emacs.
3e édition.
O'Reilly Media, décembre 2004.
URL : http://shop.oreilly.com/product/9780596006488.do.

[3] Steve OUALLINE.
Vi IMproved—Vim.
Landmark.
Sams, 2001.
URL : http://www.informit.com/store/product.aspx?isbn=0735710015.

[4] Arnold ROBBINS.
vi Editor Pocket Reference.
O'Reilly Media, janvier 1999.
URL : http://shop.oreilly.com/product/9781565924970.do.

[5] Arnold ROBBINS, Elbert HANNAH et Linda LAMB.
Learning the vi and Vim Editors.
7e édition.
O'Reilly Media, juillet 2008.
URL : http://shop.oreilly.com/product/9780596529833.do.

[6] Michael A. SCHOONOVER, John S. BOWIE et William R. ARNOLD.
GNU Emacs : UNIX Text Editing and Programming.
Addison-Wesley Professional, 1992.
URL : http://www.informit.com/store/product.aspx?isbn=0201563452.

[7] Kim SCHULZ.
Hacking Vim 7.2.
Packt Publishing, avril 2010.
URL : https://www.packtpub.com/hacking-vim-7-2/book.

[8] Richard STALLMAN.
GNU Emacs Manual.
16e édition.
Free Software Foundation, 2009.
URL : http://www.gnu.org/software/emacs/manual/emacs.html.

L'interpréteur de commandes

[9] Alfred V. AHO, Brian W. KERNIGHAN et Peter J. WEINBERGER.
The AWK Programming Language.
Addison-Wesley, 1988.
URL : http://cm.bell-labs.com/cm/cs/awkbook/.

[10] Carl ALBING, JP VOSSEN et Cameron NEWHAM.
bash Cookbook.
O'Reilly Media, mai 2007.
URL : http://shop.oreilly.com/product/9780596526788.do.

[11] Gail ANDERSON et Paul ANDERSON.
The UNIX C Shell Field Guide.
Prentice Hall, avril 1986.
URL : http://www.asgteach.com/books/unix.htm.

[12] John BAMBENEK et Agnieszka KLUS.
grep Pocket Reference.
O'Reilly Media, janvier 2009.
URL : http://shop.oreilly.com/product/9780596153618.do.

[13] Daniel J. BARRETT, Richard SILVERMAN et Robert G. BYRNES.
SSH, The Secure Shell : The Definitive Guide.
2e édition.
O'Reilly Media, mai 2005.
URL : http://shop.oreilly.com/product/9780596008956.do.

[14] Christine DEFFAIX RÉMY.
Programmation shell sous Unix/Linux. sh (Bourne), ksh, bash.
2e édition.
Éditions ENI, mai 2008.
URL : http://www.editions-eni.fr/.

[15] Dale DOUGHERTY et Arnold ROBBINS.
sed & awk.
2e édition.
O'Reilly Media, mars 1997.
URL : http://shop.oreilly.com/product/9781565922259.do.

[16] Paul DUBOIS.
Using csh & tcsh.
O'Reilly Media, juillet 1995.
URL : http://shop.oreilly.com/product/9781565921320.do.

[17] Jeffrey E.F. FRIEDL.
Mastering Regular Expressions.
3e édition.
O'Reilly Media, août 2006.
URL : http://shop.oreilly.com/product/9780596528126.do.

[18] Jan GOYVAERTS et Steven LEVITHAN.
Regular Expressions Cookbook.
O'Reilly Media, mai 2009.
URL : http://shop.oreilly.com/product/9780596520694.do.

[19] Michael W. LUCAS.
SSH Mastery.
Tilted Windmill Press, janvier 2012.
URL : http://www.michaelwlucas.com/nonfiction/ssh-mastery.

[20] Cameron NEWHAM et Bill ROSENBLATT.
Learning the bash Shell.
3e édition.
O'Reilly Media, mars 2005.
URL : http://shop.oreilly.com/product/9780596009656.do.

[21] Ellie QUIGLEY.
UNIX Shells by Example.
4e édition.
Prentice Hall, 2005.
URL : http://www.informit.com/store/product.aspx?isbn=013147572X.

[22] Arnold ROBBINS.
Effective awk Programming.
3e édition.
O'Reilly Media, mai 2001.
URL : http://shop.oreilly.com/product/9780596000707.do.

[23] Arnold ROBBINS.
sed & awk Pocket Reference.
2e édition.
O'Reilly Media, juin 2002.
URL : http://shop.oreilly.com/product/9780596003524.do.

[24] Arnold ROBBINS.
bash Quick Reference.
O'Reilly Media, juin 2006.
URL : http://shop.oreilly.com/product/9780596527761.do.

[25] Arnold ROBBINS et Nelson H.F. BEEBE.
Classic Shell Scripting.
O'Reilly Media, mai 2005.
URL : http://shop.oreilly.com/product/9780596005955.do.

[26] Bill ROSENBLATT et Arnold ROBBINS.
Learning the Korn Shell.
2e édition.
O'Reilly Media, avril 2002.
URL : http://shop.oreilly.com/product/9780596001957.do.

[27] Michael STAHNKE.
Pro OpenSSH.
Apress, octobre 2005.
URL : http://www.apress.com/9781590594766.

[28] Tony STUBBLEBINE.
Regular Expression Pocket Reference.
2e édition.
O'Reilly Media, juillet 2007.
URL : http://shop.oreilly.com/product/9780596514273.do.

[29] Sriranga VEERARAGHAVAN.
Sams Teach Yourself Shell Programming in 24 Hours.
2e édition.
Sams Teach Yourself.
Sams, 2002.
URL : http://www.informit.com/store/product.aspx?isbn=0672323583.

Création d'un site Web

[30] Christophe AUBRY.
CSS 2.1. Adoptez les feuilles de style pour maîtriser les standards du web.
2e édition.
Éditions ENI, juin 2008.
URL : http://www.editions-eni.fr/.

[31] Christophe AUBRY.
CSS. Réussir la mise en page de vos sites Web.
Éditions ENI, janvier 2010.
URL : http://www.editions-eni.fr/.

[32] Francis DRAILLARD.
Premiers pas en CSS et XHTML.
3e édition.
Eyrolles, avril 2010.
URL : http://www.eyrolles.com/Informatique/Livre/premiers-pas-en-css-et-xhtml-9782212127249.

[33] Raphaël GOETTER.
CSS 2. Pratique du design web.
3e édition.
Eyrolles, janvier 2009.
URL : http://www.goetter.fr/livres/css2/.

[34] Raphaël GOETTER.
Mémento CSS.
2e édition.
Eyrolles, juillet 2009.
URL : http://www.eyrolles.com/Informatique/Livre/memento-css-9782212125429.

[35] Raphaël GOETTER.
Mémento XHTML.
2e édition.
Eyrolles, juillet 2009.
URL : http://www.eyrolles.com/Informatique/Livre/memento-xhtml-9782212125412.

[36] Raphaël GOETTER.
CSS avancées. Vers HTML5 et CSS3.
Eyrolles, mars 2011.
URL : http://www.goetter.fr/livres/css-avancees/.

[37] Ben HENICK.
HTML & CSS : The Good Parts.
O'Reilly Media, février 2010.
URL : http://shop.oreilly.com/product/9780596157616.do.

[38] Brian P. HOGAN.
HTML5 & CSS3. Develop with Tomorrow's Standards Today.
The Pragmatic Programmers, décembre 2010.
URL : http://www.pragprog.com/titles/bhh5/html5-and-css3.

[39] Michel MARTIN.
HTML5 et CSS3. L'essentiel des pratiques actuelles.
Pearson, mars 2011.
URL : http://www.pearson.fr/livre/?GCOI=27440100242490.

[40] Julie C. MELONI et Michael MORRISON.
Sams Teach Yourself HTML and CSS in 24 Hours.
8e édition.
Sams Teach Yourself.
Sams, 2010.
URL : http://www.informit.com/store/product.aspx?isbn=0672330970.

[41] Eric A. MEYER.
CSS : The Definitive Guide.
3e édition.
O'Reilly Media, novembre 2006.
URL : http://shop.oreilly.com/product/9780596527334.do.

[42] Eric A. MEYER.
CSS Pocket Reference.
3e édition.
O'Reilly Media, octobre 2007.
URL : http://shop.oreilly.com/product/9780596515058.do.

[43] Chuck MUSCIANO et Bill KENNEDY.
HTML & XHTML : The Definitive Guide.
6e édition.
O'Reilly Media, octobre 2006.
URL : http://shop.oreilly.com/product/9780596527327.do.

[44] Mathieu NEBRA.
Réussir son site web avec XHTML et CSS.
3e édition.
Eyrolles, janvier 2010.
URL : http://www.eyrolles.com/Informatique/Livre/reussir-son-site-web-avec-xhtml-et-css-9782212124859.

[45] Mark PILGRIM.
HTML5 : Up and Running.
O'Reilly Media, août 2010.
URL : http://shop.oreilly.com/product/9780596806033.do.

[46] Jennifer Niederst ROBBINS.
HTML & XHTML Pocket Reference.
4e édition.
O'Reilly Media, décembre 2009.
URL : http://shop.oreilly.com/product/9780596805876.do.

[47] Christopher SCHMITT.
CSS Cookbook.
3e édition.
O'Reilly Media, décembre 2009.
URL : http://shop.oreilly.com/product/9780596155940.do.

[48] Christopher SCHMITT et Kyle SIMPSON.
HTML5 Cookbook.
O'Reilly Media, septembre 2011.
URL : http://shop.oreilly.com/product/0636920016038.do.

[49] Luc VAN LANCKER.
HTML 4. Maîtrisez le code source.
3e édition.
Éditions ENI, octobre 2008.
URL : http://www.editions-eni.fr/.

[50] Luc VAN LANCKER.
XHTML et CSS. Les nouveaux standards du code source.
2e édition.
Éditions ENI, octobre 2008.
URL : http://www.editions-eni.fr/.

[51] Luc VAN LANCKER.
HTML5 et CSS3. Maîtrisez les standards des applications Web.
Éditions ENI, mars 2011.
URL : http://www.editions-eni.fr/.

[52] John C. VANDEN-HEUVEL Sr.
HTML for Babies.
Tome 1.
Web Design for Babies.
Code Babies Media, 2011.
URL : http://codebabies.com/.

[53] Jeffrey ZELDMAN et Ethan MARCOTTE.
Designing with Web Standards.
3e édition.
New Riders, octobre 2009.
URL : http://www.zeldman.com/dwws/.

Typographie

[54] Jean-Pierre LACROUX.
Orthotypographie, Orthographe & Typographie françaises, Dictionnaire raisonné.
2007.
URL : http://www.orthotypographie.fr/.

[55] *Lexique des règles typographiques en usage à l'Imprimerie nationale*.
3e édition.
Imprimerie nationale. Juillet 2002.

Divers

[56] Louis LÉON et Françoise MILLET.
La machine UNIX.
Éditions Cépaduès, 1992.
URL : http://www.cepadues.com/livre_details.asp?l=267.

[57] Alfred J. MENEZES, Paul C. van OORSCHOT et Scott A. VANSTONE.
Handbook of Applied Cryptography.
CRC Press, octobre 1996.
URL : http://www.cacr.math.uwaterloo.ca/hac/.

[58] Bruce SCHNEIER.
Applied Cryptography.
2e édition.
John Wiley & Sons, 1996.
URL : http://www.schneier.com/book-applied.html.

Index

A

account ..... *voir* compte
AIX 19
architecture en couches 34
AZERTY 29

B

babasse 29
bépo 30
binaire ... *voir* système de numération
bit 25
boîte à outils 34
BSD 19
bureau virtuel 34

C

chiffrement 246
composant d'interface graphique .. 33
compte 32

D

Darwin 19
décimal .. *voir* système de numération
desktop environment ..... *voir* environnement de bureau
display manager ..... *voir* gestionnaire d'affichage
Dvorak 30

E

empreinte 248
environnement de bureau 32, 35
exaoctet 27
exbioctet 27

F

fichier 46
fingerprint ..... *voir* empreinte
format 47, 281
  fermé 282
  ouvert 282
FreeBSD 19

G

gestionnaire d'affichage 31
gestionnaire de fenêtres 34
gibioctet 27
gigaoctet 27

H

hexadécimal ..... *voir* système de numération
home directory ..... *voir* répertoire personnel
HP-UX 19

I

identifiant 32
Internet 40

ISO 8601 149

J

JBM 40

K

kibioctet 27
kilooctet 27

L

Linux 19, 29
login ..... *voir* identifiant

M

Mac OS X 19
mébioctet 27
mégaoctet 27
mot de passe 32
Multics 18
multitâche 20, 181
    préemptif 182
multi-utilisateur 20, 139, 245

N

NetBSD 19

O

octal ..... *voir* système de numération
octet ..... 26, *voir aussi* exaoctet ; exbioctet ; gibioctet ; gigaoctet ; kibioctet ; kilooctet ; mébioctet ; mégaoctet ; pébioctet ; pétaoctet ; tébioctet ; téraoctet ; yobioctet ; yottaoctet ; zébioctet ; zettaoctet
OpenBSD 19, 226
operating system ..... *voir* système d'exploitation
Oracle Solaris 19
ordonnancement 181
ordonnanceur 181

P

password ..... *voir* mot de passe
pébioctet 27
pétaoctet 27

Q

quantum 181
QWERTY 29
QWERTZ 30

R

répertoire personnel 32

S

scheduler ..... *voir* ordonnanceur
scheduling ..... *voir* ordonnancement
session 32
système de numération 23
    binaire 25
    décimal 24
    hexadécimal 26
    octal 25
système d'exploitation 18

T

tébioctet 27
téraoctet 27
toolkit ..... *voir* boîte à outils

U

UNIX 18

V

virtual desktop .... *voir* bureau virtuel

W

widget .... *voir* composant d'interface graphique
window manager . *voir* gestionnaire de fenêtres

X

X ..... *voir* X Window System
X Window System 33
X11 ..... *voir* X Window System

Y

yobioctet 27
yottaoctet 27

Z

zébioctet 27
zettaoctet 27

Index – Du texte et des éditeurs de texte

Les commandes des éditeurs de texte sont récapitulées :
- dans le tableau A.1 page 369 pour Emacs ;
- dans le tableau A.2 page 372 pour `vi` et Vim.

A

ASCII 48

C

codage 47
  conversion 53
copier-coller 64

E

éditeur de texte 54
Emacs 57
`emacs` 57

F

fichier texte 45

I

`iconv` 53
ISO 8859 48

L

ligne de mode 58

M

minibuffer 58
mode line ..... *voir* ligne de mode

T

`.txt` (extension) 57

U

Unicode 50
UTF-8 51

V

`vi` 73
Vim 73
`vim` 74

Index – L'interpréteur de commandes

Un récapitulatif des commandes étudiées figure également dans l'annexe B page 375.

Symboles

! (exécution d'une commande de l'historique) 174, 175
!! (exécution de la commande précédente) 175
" . . . " (inhibition de caractères spéciaux) 149
(commentaire) 261
(variable spéciale) 273
#! (script) 260
$ (expression rationnelle) 206
$ (valeur d'une variable) 178, 263
$((. . .)) (calcul entier) 264
& (arrière-plan) 183
&& (opérateur et) 191
' . . . ' (inhibition de caractères spéciaux) 149
(. . .) (expression rationnelle) 206, 207
) (instruction case) 270
* (expression rationnelle) 206, 207
* (métacaractère) 137
+ (expression rationnelle) 206, 207
- (suppression du premier de ses travaux d'impression) 232
. (expression rationnelle) 205, 206
. (répertoire courant) 127
.. (répertoire parent) 127
/ (répertoire racine) 129
/ (séparateur) 119, 148
2> (redirection de la sortie d'erreur standard) 196
; (séquence) 192
;; (instruction case) 270
< (redirection de l'entrée standard) 197
> (redirection de la sortie standard) 195
>> (redirection de la sortie standard) 195
? (expression rationnelle) 206, 207
? (métacaractère) 135
? (variable spéciale) 190
@ (variable spéciale) 273
[. . .] (commande test) 267
[. . .] (expression rationnelle) 205, 206
[. . .] (métacaractères) 136
[^ . . .] (expression rationnelle) . . . 206
\ (continuation de ligne) 260
\ (inhibition d'un caractère spécial) 148
^ (expression rationnelle) 206
^ (touche Ctrl) 94

‘ ... ‘ (substitution de commande) 203
{ ... } (expression rationnelle) 206
| (expression rationnelle) 206, 207
| (instruction case) 270
| (tuyau) 201
|| (opérateur ou) 192
~ (répertoire personnel) 132

A

a2ps 237, 242
  option -1 238
  option -2 238
  option -4 238
  option -o 239
  option -P 237
  option -s 239
alias 99
archivage 165
archive 165
argument 100
arrière-plan 183
awk 199, 209, 264
  option -f 214

B

background ..... *voir* arrière-plan
bash 90, 257
bc 197
bg 184
Bourne shell ..... *voir* sh
Bourne-again shell ..... *voir* bash
bunzip2 163
.bz2 (extension) 163
bzip2 163

C

C shell ..... *voir* csh
cal 93
case 269
casse 94
cat 101
catalog ..... *voir* catalogue
catalogue 117
cd 125
chemin d'accès 119, 131
  absolu 131
  relatif 131
chmod 145
  option -R 146
clear 104
commande 87
  externe 177
  interne 177
commentaire 261
complètement 115, 119
completion ..... *voir* complètement
compress 161
compression 160
cp 107, 123
  option -i 108
csh 89, 257

D

date 36
décompression 160
df 158
  option -h 158
  option -k 158
directory ..... *voir* répertoire
do 271, 273
done 271, 273
dossier 117
droits d'accès 139
du 156, 219
  option -k 156
  option -s 157

E

echo 135, 178, 263
egrep 208
elif 268
else 268
émulateur de terminal 91
enscript 240, 242
  option -D 240
  option -E 240
  option -o 241
  option -P 240
  option -U 240
entrée standard 193
  redirection 197
esac 270
exit 105, 226, 228, 251, 276
expression rationnelle 205

F

false 192
fi 268
fichier
  copie 107
  renommage 109
  suppression 110
  types de fichiers
    - (fichier ordinaire) 98
    b (fichier spécial de type bloc) . 98
    c (fichier spécial de type caractère) 98
    d (répertoire) 98, 118
    l (lien symbolique) 98
    p (tuyau nommé) 98
    s (prise réseau) 98
fichier spécial 193
filtre 200, 203
find 151
  option -mtime 153
  option -name 153
folder ..... *voir* dossier
for 271
ftp 245

G

grep 198, 205
  option -E 208
  option -i 208
  option -l 208
  option -n 209
  option -v 201, 209
gunzip 162
gv 234
.gz (extension) 162
gzip 161
  option -9 162

H

historique 96, 173
history 173
home directory ..... *voir* répertoire personnel
hostname 251

I

iconv 242
identifiant de processus 183, 185
if 267
in 270, 271
indentation 268, 277
inode ..... *voir* i-nœud
interpréteur de commandes 21, 87
i-nœud 111
invite 36, 92, 126

K

kill 188
  option -9 189
  option -KILL 189

option -l 188
Korn shell ..... *voir* ksh
ksh 89, 257

L

less 103, 204
lien 111
lien symbolique 98, 114
link ..... *voir* lien
ll 99
ln 111, 124
option -i 113
option -s 114
lpq 230
option -P 230
lpr 235
option -P 235
lprm 231
option -P 231
ls 97
option -a 99
option -d 118
option -i 111
option -l 97
lzma 164
.lzma (extension) 164

M

man 104
métacaractère 135
mkdir 118
more 102, 204
mv 109, 124
option -i 109

N

nice 223
option -n 223
nohup 221

P

passwd 38
path ..... *voir* chemin d'accès
PATH (variable d'environnement) .. 178
pdftops 242
option -duplex 243
PID ..... *voir* identifiant de processus
pipe ..... *voir* tuyau
PostScript 232
priorité 222
process ..... *voir* processus
process identifier .... *voir* identifiant de processus
processus 181
identifiant 185
statut de sortie 190
prompt ..... *voir* invite
ps 185
option -a 186
option -u 185
option -w 186
option -x 185
.ps (extension) 234
psnup 236
option -2 236
option -4 236
psselect 236
option -p 236
pwd 126

Q

quota 154

R

.rar (extension) 171
redirection 193
entrée standard 197
sortie d'erreur standard 196
sortie standard 194

regular expression ..... *voir* expression rationnelle
`renice` 223
répertoire ..... 98, 117, *voir aussi* sous-répertoire
  création 118
  suppression 133
répertoire courant 126
répertoire personnel 128
répertoire racine 129
`return` 276
`rm` 110
  option `-i` 110
  option `-r` 134
`rmdir` 133
root directory ... *voir* répertoire racine

S

`scp` 253
`screen` 224
  option `-r` 225
  option `-S` 225
script 258
`sed` 215
  option `-E` 215
  option `-e` 216
  option `-f` 216
`sh` 88, 257
`.sh` (extension) 258
shell . *voir* interpréteur de commandes
signal 188
`sort` 158, 216
  option `-f` 217
  option `-k` 217, 218
  option `-n` 218
  option `-r` 219
  option `-t` 218
sortie d'erreur standard 193
  redirection 196
sortie standard 193
  redirection 194
sous-répertoire 122
`ssh` 249
  option `-l` 252
  option `-t` 253
  option `-X` 251
  option `-Y` 252
`ssh-keygen` 250
  option `-f` 250
  option `-l` 250
`status` (variable spéciale) 190
`stderr` 193
`stdin` 193
`stdout` 193
substitution de commande 202

T

tâche de fond ..... *voir* arrière-plan
`tar` 166
  option `-c` 166
  option `-f` 166
  option `-j` 169
  option `-t` 168
  option `-v` 166
  option `-x` 168
  option `-Z` 169
  option `-z` 169
`.tar` (extension) 166
`tcsh` 89, 257
`telnet` 245
TENEX C shell ..... *voir* `tcsh`
terminal 91
`test` 265
`then` 268
`tmux` 226
  option `-t` 227

`top` 187
`true` 192
tube ..... *voir* tuyau
tuyau 200
`type` 179

U

`umask` 147
`uncompress` 161
`unlzma` 164
`unrar` 171
`until` 273
`unxz` 165
`unzip` 170
  option `-l` 170

V

variable 261
  d'environnement 178, 262
  spéciale 190, 262

W

`w` 254
`wc` 219
  option `-c` 220
  option `-l` 220
  option `-m` 220
  option `-w` 220
`which` 179
`while` 272
`who` 254
working directory ..... *voir* répertoire courant

X

`xz` 164
`.xz` (extension) 165

Z

`.Z` (extension) 161
`^Z` (suspension du processus) 184
Z shell ..... *voir* `zsh`
`zip` 170
  option `-r` 170
`.zip` (extension) 169
`zsh` 90, 257

Index – Création d'un site Web

Des récapitulatifs figurent également :
- dans le tableau C.1 page 381 pour les éléments HTML ;
- dans le tableau C.2 page 382 pour les propriétés CSS.

Symboles

. (sélecteur de classe) 322
(sélecteur d'identifiant) 325
, (regroupement) 318
/* ... */ (commentaire CSS) 316
<!-- ... --> (commentaire HTML) 294
> (combinateur) 320
[...] (sélecteur d'attribut) 321

A

a (élément) 303, 326
  attribut href 303
abbr (élément) 300
acronym (élément) 300
alt (attribut)
  élément img 304
& 303
attribut 294

B

b (élément) 301
background-color (propriété) 334, 347
balise 283
bloc 316
body (élément) 293, 295
border ..... *voir* bordure
border (attribut)
  élément table 309, 351
border-collapse (propriété) 350
border-color (propriété) 348
border-style (propriété) 348
border-width (propriété) 348
bordure 343, 348
bottom (propriété) 354
br (élément) 300
browser ..... *voir* butineur
butineur 286

C

cadratin 334
caption (élément) 310, 349
caption-side (propriété) 349
cascade 329
cite (élément) 300
class (attribut) 323
classe 322
cm (unité de mesure) 335

code (élément) 300
color (propriété) 333
commentaire
  CSS 316
  HTML 294
couleurs 332
CSS 281, 313
  1 313
  2 313
  2.1 313
  3 313
.css (extension) 315

D

dd (élément) 307
déclaration 316, 328
dfn (élément) 300
div (élément) 325, 353
division 353
dl (élément) 307
dt (élément) 307
DTD 285

E

élément 283
  vide 294
em (élément) 300
em (unité de mesure) 335
entité 302
espacement 343, 346
ex (unité de mesure) 335

F

feuille de style 281, 313
float (propriété) 355
font-family (propriété) 336
font-size (propriété) 338
font-style (propriété) 337
font-weight (propriété) 337

G

> 303

H

h1 (élément) 296
h2 (élément) 296
h3 (élément) 296
h4 (élément) 296
h5 (élément) 296
h6 (élément) 296
head (élément) 293, 294
height (propriété) 357
héritage 329
:hover (pseudo-classe) 327
hr (élément) 299
href (attribut)
  élément a 303
  élément link 315
HTML 281, 286
  2.0 288
  3.2 288
  4.0 288
  4.01 289
  5 289
html (élément) 293
  attribut xmlns 294
.html (extension) 295
hyperlien 286
hypertexte 286

I

i (élément) 301
id (attribut) 325, 354
identifiant 324
img (élément) 304
  attribut alt 304
  attribut src 304
in (unité de mesure) 335
index.html 295

ISO 639 ..... 294, 310, 327

K

kbd (élément) 300

L

:lang (pseudo-classe) 327
left (propriété) 354
li (élément) 306
link (élément) 314
  attribut href 315
  attribut rel 314
  attribut type 315
:link (pseudo-classe) 326
< 302

M

marge 343, 345
margin ..... *voir* marge
margin (propriété) 345
margin-bottom (propriété) 345
margin-left (propriété) 345
margin-right (propriété) 345
margin-top (propriété) 345
MathML 311
meta (élément) 294
mm (unité de mesure) 335

N

navigateur ..... *voir* butineur

O

ol (élément) 306

P

p (élément) 297
padding ..... *voir* espacement
padding (propriété) 347
padding-bottom (propriété) 347
padding-left (propriété) 347
padding-right (propriété) 347
padding-top (propriété) 347
page HTML 281
pc (unité de mesure) 335
position (propriété) 354
propriété 316, 328
pseudo-classe 325
  de langue 327
  de lien 326
  dynamique 327
pt (unité de mesure) 335
public_html 295
px (unité de mesure) 335

R

règle 316
rel (attribut)
  élément link 314
right (propriété) 354

S

samp (élément) 300
sélecteur 316, 317
  d'attribut 321
  de classe 322
  de type 317
  d'enfant 320
  descendant 319
  d'identifiant 324
  regroupement 318
SGML 283
span (élément) 352
src (attribut)
  élément img 304
strong (élément) 300
style sheet ..... *voir* feuille de style
sub (élément) 302
sup (élément) 302

T

`table` (élément) 309
attribut `border` 309, 351
tag ..... *voir* balise
`td` (élément) 309
`text-align` (propriété) 339, 350
`text-decoration` (propriété) 341
`text-indent` (propriété) 339
`th` (élément) 309
`title` (élément) 294
`top` (propriété) 354
`tr` (élément) 309
`tt` (élément) 301
`type` (attribut)
élément `link` 315

U

`ul` (élément) 306

V

valeur 316
`var` (élément) 300
`:visited` (pseudo-classe) 326

W

`width` (propriété) 357

X

XHTML
1.0 289
1.1 ..... 289, 291
XML 285
`xml:lang` (attribut) 294, 310, 327
`xmlns` (attribut)
élément `html` 294

Liste des tableaux

2.1 Chiffres utilisés en hexadécimal 26
2.2 Unités multiples de l'octet 27

4.1 Le codage ASCII 49
4.2 Les codages ISO 8859 50
4.3 Codage de la chaîne de caractères « coucou » en ASCII 51
4.4 Codage de la chaîne de caractères « coucou » en Unicode 51
4.5 Codage de la chaîne de caractères « pâté » en ISO 8859-1 52
4.6 Codage de la chaîne de caractères « pâté » en UTF-8 52

5.1 Combinaisons permettant de saisir des caractères usuels en français avec Emacs 69

6.1 Les six façons de passer en mode insertion dans `vi` 76
6.2 Combinaisons permettant de saisir des caractères usuels en français avec Vim 81

7.1 Diminutifs, noms et créateurs des interpréteurs de commandes 89
7.2 Types de fichiers 98

8.1 Chemin d'accès relatif (exemple 2), `toto1` et `toto2` 132
8.2 Chemin d'accès relatif (exemple 2), `toto3` et `toto4` 133
8.3 Droits d'accès 141
8.4 Droits d'accès en binaire et en octal 144

9.1 Caractères utilisables dans une expression rationnelle 206

12.1 Opérateurs utilisables dans une expression encadrée par `$(( et ))` . . 264
12.2 Expressions les plus utiles pouvant suivre la commande `test` 266

15.1 Noms de couleurs utilisables dans une feuille de style 332

15.2 Unités de mesure utilisables dans une feuille de style 335
15.3 Noms de polices génériques utilisables avec la propriété `font-family` 336
15.4 Valeurs utilisables avec la propriété `font-style` 337
15.5 Valeurs utilisables avec la propriété `text-align` 340
15.6 Valeurs utilisables avec la propriété `text-decoration` 341
15.7 Valeurs utilisables avec la propriété `border-style` 348
15.8 Propriétés de configuration de l'épaisseur, de la couleur et du style des quatre côtés de la bordure 349
15.9 Valeurs utilisables avec la propriété `caption-side` 350
15.10 Valeurs utilisables avec la propriété `border-collapse` 351
15.11 Valeurs utilisables avec la propriété `position` 355

A.1 Récapitulatif des commandes d'Emacs étudiées 369
A.2 Récapitulatif des commandes de `vi` et de Vim étudiées 370
A.2 Récapitulatif des commandes de `vi` et de Vim étudiées 371
A.2 Récapitulatif des commandes de `vi` et de Vim étudiées 372

B.1 Récapitulatif des commandes étudiées dans la première partie, « Introduction » 375
B.2 Récapitulatif des commandes étudiées dans la deuxième partie, « Du texte et des éditeurs de texte » 375
B.3 Récapitulatif des commandes étudiées dans le chapitre 7, « L'interpréteur de commandes » 375
B.4 Récapitulatif des commandes étudiées dans le chapitre 8, « Gestion, organisation et traitement des fichiers » 376
B.5 Récapitulatif des commandes étudiées dans le chapitre 9, « Commandes, processus et redirections » 377
B.5 Récapitulatif des commandes étudiées dans le chapitre 9, « Commandes, processus et redirections » 378
B.6 Récapitulatif des commandes étudiées dans le chapitre 10, « Impression » 378
B.7 Récapitulatif des commandes étudiées dans le chapitre 11, « Connexion à distance » 378
B.8 Récapitulatif des commandes étudiées dans le chapitre 12, « Programmation » 379

C.1 Récapitulatif des éléments HTML étudiés 381
C.2 Récapitulatif des propriétés CSS étudiées 382

Table des figures

3.1 Gestionnaire d'affichage utilisé à l'ENSTA ParisTech 31
3.2 Environnement de bureau utilisé à l'ENSTA ParisTech 35

4.1 Exemple de fichier 46
4.2 Exemple de fichier contenant une image en couleurs 47

5.1 Emacs 58
5.2 Emacs : continuation de ligne 60
5.3 Emacs : affichage dans le minibuffer d'une commande partiellement saisie 62
5.4 Emacs : état de la ligne de mode après sauvegarde du fichier 66

6.1 Vim 75

7.1 Arbre généalogique des interpréteurs de commandes 89
7.2 Quelques commandes 94
7.3 Quelques commandes (décomposition temporelle) 95

8.1 Structure d'un fichier 112
8.2 Structure d'un fichier ayant deux noms 113
8.3 Principe du lien symbolique 114
8.4 Structure du répertoire `rep` 120
8.5 Structure du répertoire `rep` contenant `rep2` 121
8.6 Arborescence imaginaire 121
8.7 Exemple d'utilisation de `ln` 125
8.8 Structure du système de fichiers sous UNIX 129
8.9 Chemin d'accès relatif (exemple 1) 131
8.10 Chemin d'accès relatif (exemple 2) 132
8.11 Arborescence utilisée pour illustrer la commande `find` 151
8.12 Arborescence utilisée pour illustrer la commande `du` 156

9.1 Ordonnancement 182
9.2 Entrée, sortie et sortie d'erreur standards 193
9.3 Redirection de la sortie standard 194
9.4 Redirection de la sortie d'erreur standard 196
9.5 Redirection des sortie et sortie d'erreur standards 196
9.6 Redirection de l'entrée standard 197
9.7 Obtention de l'identifiant de processus avec des fichiers temporaires 200
9.8 Obtention de l'identifiant de processus avec des tuyaux 202
9.9 Substitution de commande 203

10.1 Rendu graphique du fichier PostScript du listage 10.1 234
10.2 Fichier imprimé avec la commande `a2ps` 238
10.3 Fichier imprimé avec la commande `enscript` 241

15.1 Marge, espacement et bordure : exemple de feuille de style 342
15.2 Marge, espacement et bordure : exemple de page HTML 343
15.3 Marge, espacement et bordure : rendu dans un butineur 344
15.4 Marge, espacement et bordure : schéma de principe 345
15.5 Barre latérale de navigation : exemple de page HTML 358
15.6 Barre latérale de navigation (propriété `position`) : exemple de feuille de style 359
15.7 Barre latérale de navigation (propriété `position`) : rendu dans un butineur 360
15.8 Barre latérale de navigation (propriété `float`) : exemple de feuille de style 361
15.9 Barre latérale de navigation (propriété `float`) : rendu dans un butineur 361
15.10 Deux barres latérales : exemple de page HTML 363
15.11 Deux barres latérales (propriété `position`) : exemple de feuille de style 364
15.12 Deux barres latérale (propriété `position`) : rendu dans un butineur . 365
15.13 Deux barres latérales (propriété `float`) : exemple de feuille de style . 365
15.14 Deux barre latérales (propriété `float`) : rendu dans un butineur . . . 366

Table des listages

4.1 Conversion de codage de caractères en Perl 54

9.1 Exemple de programme en langage AWK 213

10.1 Exemple de fichier PostScript 233

13.1 Exemple de fichier SGML 284
13.2 Exemple de fichier XML 287

Table des matières

Sommaire 9

Avant-propos 11

I Introduction 15

1 Le stage d'informatique pour l'ingénieur 17
1.1 Objectifs du cours 18
1.2 Qu'est-ce qu'un système d'exploitation ? 18
1.2.1 Généralités 18
1.2.2 UNIX 18
1.2.3 Pourquoi UNIX à l'ENSTA ParisTech ? 20
1.3 L'informatique efficace 21

2 Représentation et mesure de l'information 23
2.1 Représentation des nombres 23
2.1.1 Système décimal 24
2.1.2 Notation positionnelle et bases de numération 24
2.1.3 Systèmes de numération utilisés en informatique 25
2.1.3.1 Système binaire 25
2.1.3.2 Système octal 25
2.1.3.3 Système hexadécimal 26
2.2 Unités de mesure de l'information 26
2.2.1 L'octet 26
2.2.2 Représentation d'un octet en hexadécimal 26
2.2.3 Unités multiples de l'octet 27

3 Premier contact **29**
3.1 La babasse 29
3.1.1 Le clavier 29
3.1.2 Le gestionnaire d'affichage 31
3.2 Le compte 32
3.3 Ouverture de session 32
3.4 L'interface graphique 32
3.4.1 X Window System 33
3.4.2 Les boîtes à outils 33
3.4.3 Le gestionnaire de fenêtres 34
3.4.4 L'environnement de bureau 35
3.5 L'interpréteur de commandes 36
3.6 Changement du mot de passe 37
3.7 Verrouillage et fin de session 39
3.8 L'environnement informatique de l'ENSTA ParisTech 40

II Du texte et des éditeurs de texte **43**

4 Le texte et son codage **45**
4.1 Le texte, c'est quoi ? 45
4.2 Qu'est-ce qu'un fichier ? 46
4.3 Codage du texte 47
4.3.1 Le codage ASCII 48
4.3.2 Les codages ISO 8859 48
4.3.3 Unicode et le codage UTF-8 50
4.4 Codage des fichiers 53
4.5 Conversion d'un codage vers un autre 53
4.5.1 La commande `iconv` 53
4.5.2 Dans un logiciel 54
4.6 Les éditeurs de texte 54

5 L'éditeur de texte Emacs **57**
5.1 Lancement d'Emacs 57
5.2 Saisie du texte 59
5.3 Les commandes 61
5.4 Commandes de déplacement 63
5.5 Commandes de suppression 63

5.6 Exécution multiple d'une commande 64
5.7 Copier-coller 64
5.8 Couper-coller 65
5.9 Commande d'annulation des modifications 65
5.10 Commandes de gestion de fichiers 65
5.11 Commandes de recherche et de remplacement 67
5.12 Saisie de caractères ne figurant pas sur le clavier 68
5.13 Commande d'annulation de la commande en cours de saisie . . 69
5.14 Commandes diverses 69
5.15 Quitter 70
5.16 Conclusion 71

6 Les éditeurs de texte `vi` et Vim 73
6.1 Lancement de `vi` et de Vim 73
6.2 Saisie du texte 74
6.3 Les commandes 77
6.4 Commandes de déplacement 78
6.5 Commandes de suppression 78
6.6 Exécution multiple d'une commande 78
6.7 Copier-coller 79
6.8 Couper-coller 79
6.9 Commande d'annulation des modifications 79
6.10 Commandes de gestion de fichiers 80
6.11 Commandes de recherche et de remplacement 80
6.12 Saisie de caractères ne figurant pas sur le clavier 81
6.13 Commande d'annulation de la commande en cours de saisie . . 82
6.14 Commandes diverses 82
6.15 Quitter 82
6.16 Conclusion 83

III L'interpréteur de commandes 85

7 L'interpréteur de commandes 87
7.1 Historique des interpréteurs de commandes 88
7.2 Concepts fondamentaux 91
7.2.1 L'émulateur de terminal 91
7.2.2 L'invite 92

7.2.3 Passé, présent et futur 93
7.2.4 Notation des combinaisons de touches 94
7.2.5 Sensibilité à la casse 94
7.2.6 Édition de la ligne de commande et historique 95
7.2.7 Forme des exemples dans la suite du document 96
7.3 La commande `ls`, la syntaxe des commandes 97
7.3.1 Les options 97
7.3.2 Les arguments 100
7.4 Les commandes d'affichage : `cat`, `more` et `less` 101
7.4.1 La commande `cat` 101
7.4.2 La commande `more` 102
7.4.3 La commande `less` 103
7.5 La commande `man` 103
7.6 La commande `clear` 104
7.7 La commande `exit` 105

8 Gestion, organisation et traitement des fichiers 107
8.1 Gestion des fichiers 107
8.1.1 Copie de fichiers : la commande `cp` 107
8.1.2 Renommage de fichiers : la commande `mv` 109
8.1.3 Suppression de fichiers : la commande `rm` 110
8.1.4 Création de liens : la commande `ln` 111
8.2 Le complètement 115
8.3 Organisation des fichiers 117
8.3.1 Création de répertoires : la commande `mkdir` 118
8.3.2 Chemin d'accès et structure arborescente 119
8.3.3 Retour sur les commandes de gestion des fichiers 122
8.3.3.1 La commande `cp` et les répertoires 122
8.3.3.2 La commande `mv` et les répertoires 124
8.3.3.3 La commande `ln` et les répertoires 124
8.3.4 Se déplacer dans un répertoire : la commande `cd` 125
8.3.5 Promenade dans le système de fichiers 127
8.3.6 Chemin d'accès absolu et chemin d'accès relatif 131
8.3.7 Suppression de répertoires 133
8.3.7.1 La commande `rmdir` 133
8.3.7.2 L'option `-r` de la commande `rm` 134
8.3.8 Quelques conseils pour organiser vos fichiers 134
8.4 Les métacaractères de génération de noms de fichiers 134

8.4.1 Le métacaractère ? 135
8.4.2 Les métacaractères [et] 136
8.4.3 Le métacaractère * 137
8.4.4 Utilisation avancée 138
8.4.5 Gestion des erreurs 138
8.5 Les droits d'accès 139
8.5.1 Principe des droits d'accès 139
8.5.2 Représentation numérique des droits d'accès 142
8.5.3 La commande chmod 145
8.5.4 La commande umask 147
8.6 Les noms des fichiers 148
8.7 Gestion de l'espace de stockage, compression et archivage 150
8.7.1 Gestion de l'espace de stockage 151
8.7.1.1 Retrouver des fichiers : la commande find . . . 151
8.7.1.2 La commande quota 154
8.7.1.3 La commande du 156
8.7.1.4 La commande df 158
8.7.2 Compression 160
8.7.2.1 Les commandes compress et uncompress 161
8.7.2.2 Les commandes gzip et gunzip 161
8.7.2.3 Les commandes bzip2 et bunzip2 163
8.7.2.4 Les commandes lzma et unlzma 164
8.7.2.5 Les commandes xz et unxz 164
8.7.3 Archivage 165
8.7.3.1 La commande tar 166
8.7.3.1.1 Créer une archive 166
8.7.3.1.2 Extraire une archive 167
8.7.3.1.3 Afficher le contenu d'une archive 168
8.7.3.1.4 Compression et décompression d'une archive 168
8.7.4 Échanges avec d'autres systèmes d'exploitation 169
8.7.4.1 Les commandes zip et unzip 169
8.7.4.2 La commande unrar 171

9 Commandes, processus et redirections 173
9.1 L'historique des commandes 173
9.1.1 La commande history 173
9.1.2 Au sujet des exemples qui suivent 174

9.1.3 Exécuter à nouveau une commande dans son intégralité 174
9.1.4 Utiliser certains mots d'une commande précédente . . . 176
9.2 Les chemins de recherche des commandes 177
9.2.1 Commandes externes et commandes internes 177
9.2.2 La variable d'environnement `PATH` 178
9.2.3 Les commandes `type` et `which` 179
9.2.4 Mise en garde 179
9.3 Gestion des processus 181
9.3.1 L'ordonnancement 181
9.3.2 Lancement d'un processus en arrière-plan 183
9.3.3 La commande `ps` 185
9.3.4 La commande `top` 187
9.3.5 La commande `kill` 188
9.3.6 Le statut de sortie d'un processus 190
9.3.6.1 Les variables spéciales `?` et `status` 190
9.3.6.2 Succession de commandes 191
9.3.6.3 Les commandes `true` et `false` 192
9.4 Redirections 193
9.4.1 Sous UNIX, tout est fichier 193
9.4.2 Redirection de la sortie standard 194
9.4.3 Redirection de la sortie d'erreur standard 196
9.4.4 Redirection de l'entrée standard 197
9.4.5 Les tuyaux 198
9.4.6 Substitution de commande 202
9.5 Quelques filtres usuels 203
9.5.1 Les commandes `more` et `less` 204
9.5.2 La commande `grep` 205
9.5.2.1 Les expressions rationnelles 205
9.5.2.2 Utilisation d'une expression rationnelle avec la commande `grep` 207
9.5.2.3 Quelques options de la commande `grep` 208
9.5.3 La commande `awk` 209
9.5.4 La commande `sed` 214
9.5.5 La commande `sort` 216
9.5.6 La commande `wc` 219
9.6 Gestion des longs processus 221
9.6.1 La commande `nohup` 221
9.6.2 Les commandes `nice` et `renice` 222

9.6.3 La commande screen 224
9.6.4 La commande tmux 226

10 Impression **229**
10.1 Nommage des imprimantes 229
10.2 Avant d'imprimer 230
10.2.1 La commande lpq 230
10.2.2 La commande lprm 231
10.3 Le format PostScript 232
10.4 Impression de fichiers PostScript 234
10.4.1 La commande lpr 235
10.4.2 Les utilitaires PSUtils 235
10.4.2.1 La commande psselect 236
10.4.2.2 La commande psnup 236
10.5 Impression de fichiers texte 237
10.5.1 La commande a2ps 237
10.5.2 La commande enscript 240
10.5.3 Impression de fichiers texte codés en UTF-8 242
10.6 Impression de fichiers PDF 242

11 Connexion à distance **245**
11.1 Le chiffrement 246
11.1.1 Chiffrement symétrique 247
11.1.2 Chiffrement asymétrique 247
11.1.3 Chiffrement hybride 248
11.2 La commande ssh 249
11.2.1 Connexion simple 249
11.2.2 Connexion avec affichage graphique déporté 251
11.2.3 Connexion avec un identifiant différent 252
11.2.4 Connexion depuis l'extérieur de l'ENSTA ParisTech . . 252
11.3 La commande scp 253
11.4 Savoir qui est connecté sur un ordinateur : les commandes who et w 254

12 Programmation **257**
12.1 Pourquoi programmer ? 257
12.2 Comment programmer ? 258
12.2.1 Les scripts 258

12.2.2 Exécution d'un script 258
12.3 Continuation de ligne 260
12.4 Les commentaires 261
12.5 Les variables 261
12.6 Le calcul entier 264
12.7 Les instructions de contrôle 265
12.7.1 Les tests 265
12.7.1.1 La commande `test` 265
12.7.1.2 Le test `if` 267
12.7.1.3 Le test `case` 269
12.7.2 Les boucles 271
12.7.2.1 La boucle `for` 271
12.7.2.2 Les boucles `while` et `until` 272
12.8 Les arguments 273
12.9 Les fonctions 275
12.10 Statut de sortie 276
12.11 Quelques conseils généraux sur la programmation 277

IV Création d'un site Web 279

13 Formats de documents et historique de la toile 281
13.1 Des formats de documents 281
13.1.1 Formats ouverts, formats fermés 282
13.1.2 Un exemple de format ouvert : SGML 283
13.1.3 L'évolution de SGML : XML 285
13.2 Historique de la toile 286

14 HTML, XHTML 291
14.1 Structure générale d'une page XHTML 292
14.1.1 L'en-tête 294
14.1.2 Le corps 295
14.2 Nommage et emplacement des fichiers 295
14.3 Au sujet des exemples qui suivent 296
14.4 Les éléments usuels 296
14.4.1 Les éléments de titres `h1`, `h2`, `h3`, `h4`, `h5` et `h6` 296
14.4.2 L'élément de paragraphe `p` 297
14.4.3 L'élément de filet `hr` 299

14.4.4 L'élément de retour à la ligne `br` 300
14.4.5 Les éléments textuels `abbr`, `acronym`, `cite`, `code`, `dfn`, `em`, `kbd`, `samp`, `strong` et `var` 300
14.4.6 Les éléments visuels `b`, `i` et `tt` 301
14.4.7 Les éléments inférieur `sub` et supérieur `sup` 302
14.4.8 Les entités 302
14.5 L'élément d'hyperlien `a` 303
14.6 L'élément d'insertion d'image `img` 304
14.7 Les listes 306
14.8 Les tableaux 308
14.9 L'attribut de langue `xml:lang` 310
14.10 Enrichir XHTML 311

15 Feuilles de style CSS 313
15.1 Faire référence à une feuille de style dans une page HTML 314
15.2 Au sujet des exemples qui suivent 315
15.3 Syntaxe 315
15.3.1 Les commentaires 316
15.3.2 Les règles 316
15.3.3 Les sélecteurs 317
15.3.3.1 Les sélecteurs de type 317
15.3.3.2 Le regroupement 318
15.3.3.3 Les sélecteurs descendants 319
15.3.3.4 Les sélecteurs d'enfant 320
15.3.3.5 Les sélecteurs d'attribut 321
15.3.3.6 Les sélecteurs de classe 322
15.3.3.7 Les sélecteurs d'identifiant 324
15.3.3.8 Les pseudo-classes 325
15.3.3.8.1 Les pseudo-classes de lien 326
15.3.3.8.2 Les pseudo-classes dynamiques . . 327
15.3.3.8.3 La pseudo-classe de langue 327
15.3.4 Les déclarations et les propriétés 328
15.4 L'héritage et la cascade 329
15.4.1 L'héritage 329
15.4.2 La cascade 329
15.5 Les couleurs 332
15.5.1 Nommage et représentation des couleurs 332
15.5.2 La propriété de couleur `color` 333

15.5.3 La propriété de couleur de fond `background-color` 334
15.6 Les unités de mesure 334
15.7 Le texte 335
15.7.1 La propriété de police `font-family` 336
15.7.2 La propriété d'inclinaison `font-style` 337
15.7.3 La propriété de graisse `font-weight` 337
15.7.4 La propriété de corps `font-size` 338
15.7.5 La propriété d'alinéa `text-indent` 339
15.7.6 La propriété d'alignement `text-align` 339
15.7.7 La propriété de décoration `text-decoration` 341
15.8 Marge, espacement et bordure 341
15.8.1 La marge 345
15.8.2 L'espacement 346
15.8.3 La bordure 348
15.9 Les tableaux 349
15.9.1 La propriété de positionnement de légende `caption-side` 349
15.9.2 La propriété de fusion des bordures `border-collapse` . . 350
15.9.3 Centrage d'un tableau 352
15.10 L'élément `span` 352
15.11 L'élément `div` et les divisions 353
15.11.1 Les deux méthodes d'agencement des divisions 354
15.11.1.1 Agencement des divisions avec la propriété `position` 354
15.11.1.2 Agencement des divisions avec la propriété `float` 355
15.11.1.3 Quelle méthode utiliser ? 357
15.11.2 Dimensionnement des divisions 357
15.11.3 Une barre latérale de navigation 357
15.11.3.1 Agencement avec la propriété `position` 359
15.11.3.2 Agencement avec la propriété `float` 360
15.11.4 Deux barres latérales 362
15.11.4.1 Agencement avec la propriété `position` 362
15.11.4.2 Agencement avec la propriété `float` 362

Annexes **367**

A Récapitulatif des commandes des éditeurs de texte **369**

B Récapitulatif des commandes **375**

C Récapitulatif des éléments HTML et des propriétés CSS **381**

Bibliographie **385**
Du texte et des éditeurs de texte 385
L'interpréteur de commandes 386
Création d'un site Web 389
Typographie 393
Divers 393

Index **395**

Index – Du texte et des éditeurs de texte **399**

Index – L'interpréteur de commandes **401**

Index – Création d'un site Web **407**

Liste des tableaux **411**

Table des figures **413**

Table des listages **415**

Table des matières **417**

Le présent ouvrage a été mis en pages par son auteur au moyen du système de préparation de documents LaTeX — créé par Leslie Lamport et entretenu par l'équipe du projet LaTeX3 — reposant sur le système de composition TeX — créé par Donald Knuth. Sa réalisation a été assurée par le moteur pdfTeX — créé par Hàn Thế Thành —, la classe memoir — créée par Peter Wilson et entretenue par Lars Madsen — ainsi qu'une trentaine d'extensions. L'ensemble a été fourni par la distribution TeX Live.

Son texte a été composé en URW Garamond corps 10, la couverture et les titres de chapitres en URW Nimbus Sans et les exemples en Bera Mono.

Ses fichiers sources ont été saisis au moyen de l'éditeur de texte Vim et leur historique a été confié aux bons soins du logiciel de gestion de versions Subversion (cette édition correspond au numéro de révision 209 du dépôt).

Le cycle de production du fichier PDF ayant permis son impression a été entièrement réalisé sur le système d'exploitation NetBSD.

École nationale supérieure
de **techniques avancées**

32 boulevard Victor - 75739 PARIS CEDEX 15 - France

www.ensta-paristech.fr

ENSTA
ParisTech