Manuel GNU Emacs

Richard Stallman traduit par Laurent Garnier

16 mai 2015

Ceci est la septième édition de *Manuel GNU Emacs*, mis à jour pour la version 24.3.

Table des matières

Préface

Ce manuel décrit l'utilisation et la personnalisation simple de l'éditeur Emacs. Les personnalisations simples d'Emacs ne nécessitent pas que vous soyez un programmeur, mais si vous n'êtes pas intéressé par la personnalisation, vous pouvez ignorer les conseils de personnalisation.

Il s'agit simplement d'un manuel de référence, mais qui peut également être utilisé comme une amorce. Si vous êtes nouveau avec Emacs, nous vous recommandons de commencer avec le tutoriel intégré, apprendre par la pratique, avant de lire le manuel. Pour lancer le tutoriel, démarrer Emacs et tapez `C-h-t`. Le tutoriel décrit les commandes, vous indique quand les essayer, et explique les résultats. Le tutoriel est disponible en plusieurs langues.

En première lecture, parcourez juste les chapitres 1 et 2, qui décrivent les différentes conventions de notation du manuel et l'aspect général de l'écran d'affichage Emacs. Notez que les questions ont leurs réponses dans ces chapitres, de sorte que vous pouvez vous y référer plus tard. Après avoir lu le chapitre 4, vous devriez pratiquer les commandes qui y sont indiquées. Les prochains chapitres décrivent les techniques et les concepts fondamentaux qui sont utilisés en

permanence. Vous avez besoin de les comprendre en profondeur, afin de les expérimenter jusqu'à ce que vous soyez à l'aise.

Les chapitres 14 à 19 décrivent les fonctionnalités de niveau intermédiaire qui sont utiles pour de nombreux types d'édition. Le chapitre 20 et les suivants décrivent des fonctions facultatives mais utiles ; lisez ces chapitres lorsque vous en aurez besoin.

Lisez le chapitre Problèmes Courants (chapitre 34) si Emacs ne semble pas fonctionner correctement. Il explique comment faire face à plusieurs problèmes communs (voir Section 34.2 [S'arranger avec les problèmes Emacs], page 108), ainsi que quand et comment signaler les bogues Emacs (voir Section 34.3 [Bugs], page 108).

Pour trouver la documentation d'une commande particulière, regardez dans l'index. Touches (commandes de caractères) et noms de commandes ont des index distincts. Il y a aussi un glossaire, avec un renvoi pour chaque terme.

Ce manuel est disponible sous forme de livre imprimé et aussi en fichier Info. Le fichier Info est pour la lecture dans Emacs lui-même, ou avec le programme Info. Info est le format principal de la documentation dans le système GNU. Le fichier Info et le livre imprimé contiennent sensiblement le même texte et sont générés à partir des mêmes fichiers sources, qui sont également distribués avec GNU Emacs.

GNU Emacs est un membre de la famille de l'éditeur Emacs. Il existe de nombreux éditeurs Emacs, ils partagent tous des principes communs d'organisation. Pour plus d'informations sur la philosophie sous-jacente à Emacs et les leçons tirées de son développement, voir Emacs, l'éditeur extensible, personnalisable et auto-documenté, disponible sur `ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-519A.pdf`.

Cette version du manuel est principalement destinée à être utilisée avec GNU Emacs installé sur les systèmes GNU et Unix. GNU Emacs peut également être utilisé sur MS-DOS, Microsoft Windows et les systèmes Macintosh. La version du fichier Info de ce manuel contient un peu plus d'informations sur l'utilisation d'Emacs sur ces systèmes. Ces systèmes utilisent différentes syntaxe de nom de fichier ; en plus MS-DOS ne prend pas en charge toutes les fonctionnalités de GNU Emacs. Voir l'Annexe **??** [Microsoft Windows], page 496, pour plus d'informations sur l'utilisation d'Emacs sous Windows. Voir l'Annexe **??** [Mac OS / GNUstep], page **??**, pour plus d'informations sur l'utilisation d'Emacs sur Macintosh (et GNUstep).

Distribution

GNU Emacs est un logiciel libre ; cela signifie que chacun est libre de l'utiliser et libre de le redistribuer sous certaines conditions. GNU Emacs n'est pas dans le domaine public ; il est protégé et il y a des restrictions sur sa distribution, mais ces restrictions sont conçues pour permettre tout ce qu'un bon citoyen coopérant voudrait faire. Ce qui n'est pas autorisé est d'essayer d'empêcher les autres de partager plus d'une version de GNU Emacs qu'ils pourraient obtenir de vous. Les conditions précises se trouvent dans la GNU General Public License qui vient avec Emacs et apparaît également dans ce manuel. Voir l'Annexe **??**[Copie], page **??**.

Une façonon d'obtenir une copie de GNU Emacs est de quelqu'un d'autre qui l'a. Vous n'avez pas besoin de demander notre permission de le faire, ou le dire à personne d'autre ; il suffit de copier. Si vous avez accès à internet, vous pouvez obtenir la dernière version de la distribution de GNU Emas par FTP anonyme ; voir `://www.gnu.org/software/emacs` sur notre site Web pour plus d'informations.

Vous pouvez également recevoir GNU Emacs lorsque vous achetez un ordinateur. Les fabriquants d'ordinateurs sont libres de distribuer des copies dans les mêmes conditions qui s'appliquent à tout le monde. Ces conditions les obligent à vous donner les sources, y compris les

modifications qu'ils auraient peut-être fait, et vous permettre de redistribuer GNU Emacs reçu d'eux dans les conditions de la General Public License. En d'autres termes, le programme doit être libre pour vous lorsque vous l'obtenez, pas seulement libre pour le fabriquant.

Si vous trouvez GNU Emacs utile, s'il vous plaît envoyez un don à la Free Software Foundation pour soutenir notre travail. Les dons à la Free Software Foundation sont déductibles d'impôts aux États-Unis. Si vous utilisez GNU Emacs sur votre lieu de travail, s'il vous plaît suggérez que l'entreprise fasse un don. Pour plus d'informations sur comment vous pouvez aider, voir `http://www.gnu.org/help/help.html`.

Nous vendons aussi des versions papier de ce manuel et Introduction à la programmation en Emacs Lisp, par Robert J. Chassell. Vous pouvez visiter notre boutique en ligne `http://shop.fsf.org/`. Le revenu de la vente va soutenir l'objectif de la fondation : le développement de nouveaux logiciels libres, et l'amélioration de nos programmes existants, y compris GNU Emacs.

Si vous avez besoin de contacter la Free Software Foundation, voir `http://www.fsf.org/about/`, ou écrire à

Free Software Foundation
51 Franklin Street, Fifth Floor Boston, MA 02110-1301
USA

Introduction

Vous lisez sur GNU Emacs, l'incarnation de l'éditeur GNU, auto-documenté, avancé, personnalisable et extensible Emacs. (Le «G» de «GNU» n'est pas muet.)

Nous appelons Emacs avancé, car il peut faire beaucoup plus que de la simple insertion et suppression de texte. Il peut contrôler des sous-processus, indenter automatiquement des programmes, afficher plusieurs fichiers à la fois, et plus encore. Les commandes d'édition Emacs fonctionnent en termes de caractères, mots, lignes, phrases, paragraphes, et pages, ainsi que les expressions et commentaires dans différents langages de programmation.

Auto-documenté signifie qu'à tout moment vous pouvez utiliser des commandes spéciales, appelées commandes d'aide, pour savoir quelles sont vos options, ou pour savoir ce que n'importe quelle commande fait, ou de trouver toutes les commandes qui se rapportent à un sujet donné. Voir le chapitre 7 [Aide], page 39.

Personnalisable signifie que vous pouvez aisément modifier le comportement des commandes Emacs de façon simple. Par exemple, si vous utilisez un langage de programmation dans lequel les commentaires commencent par `'<**'` et finissent par `'**>'`, vous pouvez dire à Emacs de commenter les commandes de manipulation afin d'utiliser ces chaînes (voir Section 23.5 [Comments], page 86). Pour prendre un autre exemple, vous pouvez relier les commandes de base de déplacement du curseur (haut, bas, gauche et droite) pour toutes les touches du clavier que vous trouvez confortable. Voir le chapitre 33 [Personnalisation], page 106.

Extensible signifie que vous pouvez aller au-delà de la simple personnalisation et créer de nouvelles commandes. De nouvelles commandes sont tout simplement des programmes écrits dans le langage Lisp, qui sont gérées par un interprète Lisp d'Emacs. Les commandes existantes peuvent même être redéfinies au milieu d'une session d'édtition, sans avoir à redémarrer Emacs. La plupart des commandes d'édition dans Emacs sont écrites en Lisp ; les quelques exceptions auraient pu être écrites en Lisp mais utiliser C à la place est plus efficace. La rédaction d'une extension est de la programmation, mais les non-programmeurs peuvent les utiliser par la suite. Voir la section «Préface» dans Introduction à la programmation en Emacs Lisp, si vous voulez apprendre la programmation Emacs Lisp.

Chapitre 1

Organisation de l'écran

En mode graphique, comme sur GNU/Linux avec le système de fenêtrage X Window, Emacs occupe une «fenêtre graphique». Sur un terminal en mode texte, Emacs occupe l'écran du terminal en entier. Nous allons utiliser le terme de cadre pour désigner une fenêtre graphique ou l'écran du terminal occupée par Emacs. Emacs se comporte de manière similaire sur les deux types de cadres. Il commence normalement avec un seul cadre, mais vous pouvez créer des images supplémentaires si vous le souhaitez (voir le chapitre 18 [Cadres], page 68).

Chaque image est composée de plusieurs régions distinctes. Au sommet du cadre il y a une barre des menus, ce qui vous permet d'accéder à des commandes par l'intermédiaire d'une série de menus. En mode graphique, directement en dessous de la barre des menus il y a une barre d'outils, une rangée d'icônes qui exécutent des commandes d'édition si vous cliquez sur elles. Au bas du cadre vous trouverez une zone d'écho, où des messages d'informations sont affichés et où vous pouvez saisir des informations quand Emacs le demande.

La zone principale du cadre, en dessous de la barre d'outils (le cas échéant) et au-dessus de la zone d'écho, est appelée la fenêtre. Désormais dans ce manuel, nous allons utiliser le mot «fenêtre» dans ce sens (différent du sens usuel). Les systèmes d'affichage graphique utilisent couramment le mot «fenêtre» dans un sens différent ; mais, comme indiqué ci-dessus, nous nous référons à ces «fenêtres graphiques» comme des «cadres».

Une fenêtre Emacs est là où le tampon —le texte que vous éditez — est affiché. En mode graphique, la fenêtre possède une barre de défilement sur un côté qui peut être utilisée pour faire défiler le tampon. La dernière ligne de la fenêtre est une ligne de mode. Cette dernière affiche diverses informations sur ce qui se passe dans le tampon, par exemple, s'il y a des modifications non enregistrées, les modes d'édition qui sont en cours d'utilisation, le numéro de la ligne actuelle, et ainsi de suite.

Lorsque vous démarrez Emacs, il y a normalement une seule fenêtre dans le cadre. Cependant, vous pouvez subdiviser cette fenêtre horizontalement ou verticalement pour créer plusieurs fenêtres, dont chacune peut indépendamment afficher un tampon (voir le chapitre 17 [Fenêtres], page 65).

À tout moment, une fenêtre est la fenêtre sélectionnée. En mode graphique, la fenêtre sélectionnée affiche un curseur plus important (généralement plein et glignotant) ; les autres fenêtres affichent un curseur moins important (généralement une boîte creuse). Sur un terminal de texte, il n'existe qu'un seul curseur qui est affiché dans la fenêtre sélectionnée. Le tampon affiché dans la fenêtre sélectionnée est appelé le tampon courant, et c'est là que se produit l'édition. La plupart des commandes Emacs peuvent s'appliquer implicitement au tampon courant ; le texte affiché dans les fenêtres non sélectionnées est surtout visible pour référence. Si vous utilisez plusieurs

cadres en mode graphique, la sélection d'un cadre particulier sélectionne une fenêtre dans ce cadre.

1.1 Point

Le curseur de la fenêtre sélectionnée indique l'emplacement où la plupart des commandes d'édition prennent effet, qui est appelé point[1]. Beaucoup de commandes Emacs déplacent le point à différents endroits dans le tampon ; par exemple, vous pouvez placer en cliquant sur le bouton 1 de la souris (usuellement le gauche) à l'endroit désiré. Par défaut, le curseur de la fenêtre sélectionnée est dessiné comme un bloc plein et paraît être un caractère, mais vous devrez penser le point comme entre deux caractères ; il est situé avant le caractère sous le curseur. Par exemple ; si votre texte ressemble à «frob» avec le curseur sur «b», alors le point est entre le «o» et le «b». Si vous insérez le caractère « ! » à cette position, le résultat sera «fro !b», avec le point entre le « ! » et le «b». Ainsi, le curseur reste au-dessus du «b», comme précédemment.

Si vous éditez plusieurs fichiers dans Emacs, chacun dans son propre tampon, chaque tampon a sa propre valeur du point. Un tampon qui n'est pas actuellement affiché se souvient encore de sa valeur du point si vous l'afficher plus tard. En outre, si un tampon est affiché dans plusieurs fenêtres, chacune de ces fenêtres a sa propre valeur du point.

Voir la section 11.20 [Affichage Curseur], page 54, pour les options qui contrôlent la façon dont Emacs affiche le curseur.

1.2 La zone d'écho

La ligne au bas du cadre est la zone d'écho. Elle est utilisée pour afficher de petites quantités de texte à des fins diverses.

La zone d'écho est ainsi nommée parce que l'une de ses caractéristiques est faire l'écho, ce qui signifie l'affichage des caractères d'une commande multi-caractères. Les commandes à caractère unique ne sont pas affichées. Les commandes multi-caractères (voir la section 2.2 [Clavier], page 8) sont reprises si vous vous arrêtez pendant une seconde au milieu d'une commande. Emacs fait alors l'écho de tous les caractères de la commande à ce moment-là, pour vous inviter à saisir le reste. Une fois l'écho commencé, le reste de la commande se fait l'écho à mesure que vous tapez. Ce comportement est conçu pour donner aux utilisateurs confiants une réponse rapide, tout en donnant aux utilisateurs hésitants un maximum de retours.

La zone d'écho est également utilisée pour afficher un message d'erreur quand une commande ne peut pas faire son travail. Les messages d'erreurs peuvent être accompagnés par un bip ou par un clignotement de l'écran.

Certaines commandes affichent des messages d'information dans la zone écho pour vous dire ce que la commande a fait, ou pour vous fournir des informations spécifiques. Ces messages d'information, contrairement aux messages d'erreurs, ne sont pas accompagnés d'un signal sonore ni d'un flash. Par exemple, `C-x =` (Maintenez la touche ctrl enfoncée et tapez x, puis relâchez la touche ctrl puis tapez =) affiche un message décrivant le caractère au point, sa position dans le tampon, et sa colonne en cours dans la fenêtre. Les commandes qui prennent beaucoup de temps affichent souvent des messages se terminant par «...» pendant qu'elles travaillent (parfois aussi indiquent combien de progrès ont été réalisés, en pourcentage), et ajouter «fait» quand elles ont fini.

[1] Le terme «point» vient du caractère «.», qui était la commande TECO (le langage dans lequel l'Emacs original a été écrit) pour accéder à la position d'édition.

Les messages d'informations de la zone d'écho sont sauvegardés dans un tampon spécial nommé `*Messages*`. (Nous n'avons pas encore expliqué les tampons ; voir le chapitre 16 [tampons], page 63, pour plus d'informations à leur sujet.). Si vous ratez un message qui est apparu brièvement à l'écran, vous pouvez changer de tampon et visiter le tampon `*Messages*` afin de revoir les messages. Le tampon `*Messages*` est limité à un certain nombre de lignes, scpécifié par la variable `message-log-max`. (Nous n'avons pas encore expliqué les variables soit, voir la section 33.2 [Variables], page 106, pour plus d'informations sur ce sujet.). Au-delà de cette limite, une ligne est supprimée depuis le début à chaque fois qu'une nouvelle ligne de message est ajoutée à la fin.

Voir la section 11.23 [affichage perso], page 54, pour les options qui contrôlent la façon dont Emacs utilise la zone d'écho.

La zone d'écho est également utilisée pour afficher le mini-tampon, une fenêtre spéciale où vous pouvez saisir des arguments de commandes, telles que le nom d'un fichier à éditer. Lorsque le mini-tampon est en cours d'utilisation, le texte affiché dans la zone d'écho commence par une chaîne d'invite, et le curseur actif apparaît dans le mini-tampon, qui est considérée comme temporairement la fenêtre sélectionnée. Vous pouvez toujours sortir du mini-tampon en tapant `C-g`. Voir le chapitre 5 [mini-tampon], page 25.

1.3 Le mode ligne

Au bas de chaque fenêtre il y a une ligne de mode, qui décrit ce qui se passe dans le tampon en cours. Quand il y a une seule fenêtre, la ligne de mode apparaît juste au-dessus de la zone d'écho ; il s'agit de l'avant-dernière ligne dans le cadre. En mode graphique, la ligne de mode est dessiné avec une apparence de boîte 3D. Emacs aussi dessine habituellement la ligne de mode de la fenêtre sélectionnée avec une couleur différente de celle des fenêtres non sélectionnées, afin de la faire ressortir.

Le texte affiché dans la ligne de mode est de la forme :

```
cs :ch-fr  buf    pos line  (major minor)
```

Sur un terminal de texte, le texte est suivi par une série de traits qui s'étendent vers le bord droit de la fenêtre. Ces traits sont omis en mode graphique.

La chaîne *cs* et le caractère deux-points qui la suit décrivent le jeu de caractères de la nouvelle ligne et les conventions utilisées pour le tampon courant. Normalement, Emacs gère automatiquement ces paramètres pour vous, mais il est parfois utile d'avoir cette information.

cs décrit le jeu de caractères du texte dans le tampon (voir section 19.6 [systèmes d'encodage], page 72). S'il s'agit d'un tiret («-»), qui inidique l'absence de caractère spécial (avec de possibles exceptions de conventions de fin de ligne (césure), décrites dans le paragraphe suivant). «=» signifie aucune conversion que ce soit, et est habituellement utilisé pour les fichiers contenant des données non-textuelles. D'autres caractères représentent différents systèmes d'encodages —par exemple, «1» représente ISO Latin-1.

Sur un terminal de texte, *cs* est précédée par deux caractères supplémentaires qui décrivent les systèmes d'encodage pour la saisie au clavier et la sortie affichée. En outre, si vous utilisez une méthode de saisie, *cs* est précédée par une chaîne qui identifie la méthode de saisie (voir section 19.4 [méthodes de saisies], page 72).

Le caractère après *cs* est généralement « :». Si une chaîne de caractère est affichée, cela indique une convention de fin de ligne non négligeable pour encoder un fichier. Habituellement, les lignes de texte sont séparées par des caractères de nouvelle ligne dans un fichier, mais deux autres conventions sont parfois utilisées. La convention MS-DOS utilise un caractère «retour chariot» suivie d'un caractère «saut de ligne» ; lors de l'édition de ces fichiers, le caractère « :»

est remplacé par («\») ou «(DOS)», selon l'OS. Une autre convention, employée par les anciens Mac OS, utilise le caractère «retour chariot» au lieu d'un retour à la ligne ; lors de l'édition de ces fichiers, le caractère « :» est remplacé par («/») ou «(Mac)» . Sur certains systèmes, Emacs affiche «(Unix)» au lieu de « :» pour les fichiers qui utilisent le saut de ligne comme séparateur de ligne.

L'élément suivant la ligne de mode est indiqué par la chaîne ch. Cela montre deux tirets (« -- ») si le tampon est affiché dans la fenêtre a le même contenu que le fichier correspondant sur le disque ; à savoir si le tampon est «non modifié». Si le tampon est modifié, il montre deux étoiles («**»). Pour un tampon en lecture seule, il montre «%*» si le tampon est modifié, et «%%» autrement.

Le caractère après le *ch* est normalement un tiret («-»). Cependant, si le répertoire par défaut pour le tampon courant est sur une machine distante, «@» est affiché à la place (voir la section 15.1 [noms de fichiers], page 62).

fr donne le nom du cadre sélectionné (voir le chapitre 18[cadres], page 68). Il n'apparaît que sur les terminaux de texte. Le nom initial du cadre est «F1».

buf est le nom du tampon (buffer) affiché dans la fenêtre. Habituellement, c'est le même nom que le nom d'un fichier que vous éditez. Voir le chapitre 16 [tampons], page 63.

pos vous indique s'il y a un texte complémentaire au-dessus du haut de la fenêtre, ou en dessous. Si votre tampon est petit et tout est visible dans la fenêtre, pos est «Tout» (all). Sinon, il est «Top» si vous êtes à la recherche au début du tampon, «Bot» si vous cherchez à la fin du tampon, ou «nn%», où nn est le pourcentage du tampon au-dessus du haut de la fenêtre. Avec le mode d'indication de taille, vous pouvez afficher la taille du tampon. Voir section 11.18 [options du mode ligne], page 54.

line est le caractère «L» suivi du numéro de la ligne au point. (Vous pouvez afficher le numéro de colonne courante aussi, en activant le mode numéro de colonne. Voir la section 11.18 [options du mode ligne], page 54.)

major est le nom du mode majeur utilisé dans le tampon. Un mode majeur est un mode d'édition principale pour le tampon, telles que le mode texte, le mode Lisp, le mode C, et ainsi de suite. Voir la section 20.1 [Les principaux modes], page 73. Certains modes majeurs affichent des informations supplémentaires après le nom du mode majeur. Par exemple, des tampons de compilation et des tampons de Shell affichent l'état du sous-processus.

minor est une liste de quelques-uns des modes mineurs permis, qui sont les modes d'édition en option qui offrent des fonctionnalités supplémentaires au-dessus du mode majeur. Voir la section 20.2 [modes mineurs], page 8.

Certaines fonctions sont répertoriées avec les modes mineurs quand ils sont allumés, même si elles ne sont pas vraiment des modes mineurs. «Narrow» signifie que le tampon est affiché à l'édition limitée à seulement une partie de son texte (voir la section 11.5 [rétrécissement], page 54). «Def» signifie qu'une macro clavier est en cours de définition (voir le chapitre 14 [macros clavier], page 59).

En outre, si Emacs est à l'intérier d'un niveau d'édition récursive, les crochets («[…]») apparaissent autour des paranthèses qui entourent les modes. Si Emacs est à un niveau d'édition récursive à l'intérieur d'un autre une paire de crochets doubles apparaît, et ainsi de suite. Puisque les niveaux d'édition récursifs affectent globalement Emacs, ces crochets apparaissent dans la ligne de mode de chaque fenêtre. Voir la section 31.9 [édition récursive], page 102.

Vous pouvez modifier l'apparence de la ligne de mode ainsi que le format de son contenu. Voir section 11.18 [options de mode ligne], page 54. En outre, la ligne de mode est sensible à la souris ; en cliquant sur les différentes parties de la ligne de mode on effectue diverses commandes. Voir la section 18.5 [mode ligne avec souris], page 68.

1.4 La barre de menu

Chaque cadre d'Emacs a normalement une barre de menu en haut que vous pouvez utiliser pour effectuer des opérations communes. Il n'est pas nécessaire de les énumérer ici, comme vous pouvez le constater par vous-même.

En mode graphique, vous pouvez utiliser la souris pour choisir une commande dans la barre de menu. Une flèche sur le bord droit d'un élément de menu signifie qu'il conduit à un menu de filiale ou sous-menu. La présence de «…» à la fin d'un élément de menu indique que la commande nécessite une (ou plusieurs) saisie(s) supplémentaires avant d'opérer.

Certaines de ces commandes dans la barre de menu ont des raccourcis clavier ordinaires ainsi ; si c'est le cas, une clé de liaison est indiquée entre parenthèses après l'item lui-même. Pour afficher le nom de la commande et la documentation de l'item, tapez `C-h k`, puis sélectionnez la barre de menu avec la souris de la façon habituelle (voir la section 7.1 [Aide pour les touches], page 42).

Au lieu d'utiliser la souris, vous pouvez également appeler le premier élément de la barre de menu en appuyant sur F10 (pour exécuter la commande `menu-bar-open`). Vous pouvez ensuite naviguer dans les menus avec les touches fléchées. Pour activer un élément de menu sélectionné, appuyez sur RET ; navigation pour annuler le menu, appuyez sur ESC.

Sur un terminal de texte, vous pouvez utiliser la barre de menu en tapant `M-'` ou F10 (ces touches lancent la commande `tmm-menubar`). Cela vous permet de sélectionner un élément de menu avec le clavier. Un choix provisoire apparaît dans la zone d'écho. Vous pouvez utiliser les touches flèches vers le haut ou vers le bas pour vous déplacer dans le menu, et alors vous pouvez taper RET pour sélectionner l'élément. Chaque élément de menu est également désigné par une lettre ou un chiffre (généralement la première d'un mot dans le nom de l'objet). Cette lettre ou ce chiffre est séparé du nom de l'élément par «`==>`». Vous pouvez taper la lettre ou le chiffre de l'élément afin de le sélectionner.

Chapitre 2

Caractères, touches et commandes

Ce chapitre explique les jeux de caractères utilisés par Emacs pour les commandes de saisie, et les concepts fondamentaux de touches et commandes, avec lesquels Emacs interprète vos saisies claviers et clics de souris.

2.1 Type de saisies

GNU Emacs est principalement conçu pour être utilisé avec le clavier. Bien qu'il soit possible d'utiliser la souris pour lancer des commandes d'édition grâce à la barre de menu et celle d'outils, ce n'est pas aussi efficace qu'à l'aide du clavier. Par conséquent, ce manuel documente principalement comment éditer avec le clavier.

Les saisies au clavier dans Emacs sont basées sur une version fortement augmentée du code ASCII. Des caractères simples, comme «a», «B», «3», «=», et le caractère espace (noté SPC), sont saisies en tapant sur les touches correspondante. Les caractères de contrôle, comme RET (entrée), TAB (tabulation), DEL, ESC (échap), F1, Accueil, et à gauche, sont également saisis de cette façon, comme le sont certains caractères figurant sur les claviers non-anglais (voir le chapitre 19 [international], page 69).

Emacs reconnaît également les caractères de contrôle qui sont saisis à l'aide des touches modificatrices. Deux touches modificatrices couramment utilisées sont la touche contrôle (généralement étiquetée Ctrl), et la touche Meta (généralement étiquetée Alt)[1]. Par exemple, Ctrl + A est saisi en maintenant la touche Ctrl enfoncée tout en appuyant sur la touche A ; nous allons nous référer à ça comme `C-a` pour faire court. De même `Meta-a`, ou `M-a` pour faire court, est entré en maintenant enfoncée la touche Alt puis en pressant la touche A. Les touches modificatrices peuvent églament être appliquées à des caractères non alphanumériques, par exemple, `C-F1` ou `M-gauche`.

Vous pouvez aussi taper des caractères Meta en utilisant des séquences de deux caractères commençant par ESC. Ainsi vous pouvez saisir `M-a` en tapant ESC a. Contrairement à Meta, ESC est saisie comme un caractère distinct. Vous ne maintenez pas la touche ESC tout en tapant le caractère suivant ; à la place, vous pressez ESC et la relâchez, puis saisissez le caractère suivant. Cette fonction est utile sur certains terminaux de texte où la touche Meta ne fonction pas de manière fiable.

[1] Pour des raisons historiques nous nous référons à Alt comme à Meta

En mode graphique, le gestionnaire de fenêtre peut bloquer certaines saisies au clavier, y compris `M-TAB`, `M-SPC`, `C-M-d` et `C-M-l`. Si vous avez ce problème, vous pouvez personnaliser votre gestionnaire de fenêtre pour ne pas bloquer les touches, ou «relier» les commandes Emacs touchées (voir chapitre 33 [Personnalisation], page 106).

Des caractères simples et des caractères de contrôle, ainsi que certaines saisies non-clavier tels que les clics de souris, sont collectivement appelés événements de saisies. Pour plus de détails sur la façon dont Emacs gère en internes les événements de saisies, voir la section «Événements de saisies» dans le manuel de référence Emacs Lisp.

2.2 Touches (du clavier)

Certaines commandes Emacs sont invoquées par un seul événement de saisie ; par exemple `C-f` avance d'un caractère dans le tampon. D'autres commandes ont deux ou plusieurs événements de saisies à invoquer, comme `C-x C-f` et `C-x 4 C-f`.

Une séquence de touches, ou touche pour faire court, est une séquence d'un ou plusieurs événements de saisies qui est significatif comme une unité. Si une séquence de touches appelle une commande, nous appelons cela une touche complète ; par exemple, `C-f`, `C-x C-f` et `C-x 4 C-f` sont des touches complètes. Si une séquence de touches n'est pas assez longue pour invoquer une commande, nous appelons cela une touche préfixée ; de l'exemple précédent nous voyons que `C-x` est `C-x 4` sont des touches préfixées. Chaque séquence de touches est soit une touche complète soit une touche préfixée.

Une touche préfixée se combine avec l'événement de saisie suivant pour faire une séquence de touches plus longue. Par exemple, `C-x` est une touche préfixée, donc `C-x` seule n'invoque pas une commande ; à la place, Emacs attend une autre entrée (si vous vous arrêtez prendant plus d'une seconde, il fait écho à la touche `C-x` pour vous inviter à poursuivre la saisie ; voir la section 1.2 [zone d'écho], page 2). `C-x` se combine avec le prochain événement de saisie pour faire une touche à deux événements ce qui pourrait être une touche préfixée (comme `C-x 4`), ou une touche complète (comme `C-x C-f`). Il n'y a pas de limite à la longueur des séquences de touches, mais en pratique, ils sont rarement plus de trois ou quatre événements de saisies.

Vous ne pouvez pas ajouter des événements de saisie sur une touche complète. Par exemple, parce que `C-f` est une touche complète, la séquence à deux événements `C-f C-k` est une séquence à deux touches, pas à une seule.

Par défaut, les touches préfixées dans Emacs sont `C-c`, `C-h`, `C-x`, `C-x RET`, `C-x @`, `C-x a`, `C-x`, `n`, `C-x r`, `C-x v`, `C-x 4`, `C-x 5`, `C-x 6`, `ESC`, `M-g` et `M-o`. (F1 et F2 sont des alias pour `C-h` et `C-x 6`.) Cette liste n'est pas coulée dans le béton ; si vous personnalisez Emacs, vous pouvez faire de nouvelles touches préfixées. Vous pouvez même en éliminer, mais ce n'est pas recommandé pour la plupart des utilisateurs ; par exemple, si vous supprimez la définition du préfixe `C-x 4`, alors `C-x 4 C-f` deviendrait une séquence de touche non valide. Voir la section 33.3 [combinaison de touches], page 106.

Taper le caractère d'aide (`C-h` ou F1) après une touche préfixée affiche une liste de commandes commençant par ce préfixe. La seule exception à cette règle est `ESC` : `ESC C-h` est équivalent à `C-M-h`, qui fait tout autre chose. Vous pouvez, cependant, utiliser F1 pour afficher une liste de commandes commençant par `ESC`.

2.3 Touches et commandes

Ce manuel est plein de passages qui vous indiquent ce que font les touches particulières. Mais Emacs n'assigne pas de sens directement aux touches. Au lieu de cela, Emacs attribue des

significations aux commandes nommées, puis donne aux touches leurs significations en les liant aux commandes.

Chaque commande a un nom choisi par un programmeur. Le nom est généralement fait de quelques mots d'anglais séparés par des tirets ; par exemple, la prochaine ligne ou avant-mot. En interne, chaque commande est un type spécial de fonction Lisp, et les actions associées à la commande sont réalisées en exécutant la fonction. Voir la section «Qu'est-ce qu'une fonction» dans le manuel de référence Emacs Lisp.

Les liens entre les touches et les commandes sont enregistrés dans des tableaux appelés claviers. Voir la section 33.3.1 [claviers], page 106.

Quand nous disons que «`C-n` déplace verticalement une ligne vers le bas» nous faisons l'impasse sur une distinction subtile qui n'est pas pertinente dans l'usage ordinaire, mais vitale pour la personnalisation d'Emacs. La commande ligne suivante fait un déplacement vertical vers le bas. `C-n` a cet effet, car cette touche est liée à la commande `prochaine-ligne`. Si vous changez le lien en liant `C-n` à la commande `prochain-mot`, alors `C-n` va se déplacer d'un mot au lieu d'une ligne.

Dans ce manuel, nous allons souvent parler de touches comme `C-n` comme d'une commande, même si à proprement parler cette touche est liée à une commande. Habituellement nous indiquons le nom de la commande qui fait vraiment le travail entre parenthèses après la mention de la touche qui l'exécute. Par exemple, nous dirons que «la commande `C-n` (`next-line`) déplace le point verticalement vers le bas», ce qui signifie que la commande `next-line` se déplace verticalement vers le bas, et la touche `C-n` est normalement liée à elle.

Puisque nous parlons de personnalisation, nous devons vous en dire plus sur les variables. Souvent la description d'une commande dira : «Pour changer cela, réglez la varibale `mumble-foo`.» Une variable est un nom utilisé pour stocker une valeur. La plupart des variables documentées dans ce manuel sont destinées à la personnalisation : certaines commandes ou d'autres parties d'Emacs examinent la variable et se comportent différemment selon la valeur que vous définissez. Vous pouvez ignorer les informations sur les variables jusqu'à ce que vous soyez intéressé par la personnalisation. Ensuite, lisez les informations de base sur les variables (voir Section 33.2 [variables], page 106) et les informations sur les variables spécifiques prendront tout leur sens.

Chapitre 3

Entrer et sortir de Emacs

Ce chapitre explique comment entrer dans Emacs, et comment en sortir.

3.1 Entrer dans Emacs

La façon habituelle d'invoquer Emacs est avec la commande emacs shell. Dans une fenêtre de terminal fonctionnant sous le système X Window, vous pouvez exécuter Emacs en arrière-plan avec emacs & ; de cette façon, Emacs ne sera pas attaché à la fenêtre du terminal, de sorte que vous pouvez l'utiliser pour exécuter d'autres commandes shell.

Quand Emacs démarre, le cadre initial affiche un tampon spécial appelé «*GNU Emacs*». Cet écran de démarrage contient des informations sur Emacs et des liens vers les tâches courantes qui sont utiles pour les utilisateurs débutants. Par exemple, en activant le lien «Tutoriel Emacs» Emacs ouvre le tutoriel ; ce qui fait la même chose que la commande `C-h t` (`help-with-tutorial`). Pour activer un lien, soit déplacer le point sur celui-ci et taper `RET`, ou cliquer dessus avec `mouse-1` (le bouton gauche de la souris).

En utilisant un argument de ligne de commande, vous pouvez demander à Emacs de visiter un ou plusieurs fichiers dès qu'il démarre. Par exemple, `emacs foo.txt` démarre Emacs avec un tampon affichant le contenu du fichier «`toto.txt`». Cette fonctionnalité existe principalement pour la compatibilité avec d'autres éditeurs, qui sont conçus pour être lancés à partir de l'enveloppe pour les sessions d'éditions courtes. Si vous appelez Emacs de cette façon, le cadre initial est divisé en deux fenêtres —une montrant le fichier spécifié, et l'autre montrant l'écran de démarrage. Voir le chapitre 17 [fenêtres], page 65.

En général, il est inutile et coûteux de commencer un nouveau Emacs chaque fois que vous souhaitez éditer un fichier. La méthode recommandée pour utiliser Emacs est de commencer juste une fois, juste après que vous vous soyez connecté faites toute votre édition dans la même session Emacs. Voir le chapitre 15 [fichiers], page 62, pour plus d'informations sur la visite de plus d'un fichier. Si vous utilisez Emacs de cette façon, la session Emacs accumule des données précieuses, comme l'anneau de kill, les registres, l'historique des actions (à défaire), l'anneau des données de marques, qui forment un ensemble d'édition plus pratique. Ces caractéristiques sont décrites plus loin dans le manuel.

Pour éditer un fichier d'un autre programme tout en gardant Emacs en cours, vous pouvez utiliser le programme `emacsclient` d'aide pour ouvrir un fichier dans la session Emacs existante. Voir la section 31.4 [serveur Emacs], page 102.

Emacs accepte d'autres arguments en ligne de commande qui lui indiquent de charger certains fichiers Lisp, où mettre la trame initiale, et ainsi de suite. Voir l'annexe C [invocation Emacs],

page 114.

Si la variable `inhibit-startup-screen` est non-`nil`, Emacs n'affiche pas l'écran de démarrage. Dans ce cas, si un ou plusieurs fichiers ont été spécifiés sur la ligne de commande, Emacs affiche simplement ces fichiers, sinon, il affiche un tampon appelé `*scratch*`, qui peut être utilisé pour évaluer de façon interactive les expressions Emacs Lisp. Voir la section 24.11 [intéraction Lisp], page 88. Vous pouvez régler la variable `inhibit-startup-screen` en utilisant le menu de personnalisation facile (voir section 33.1 [personnalisation facile], page 106), ou en éditant votre fichier d'initialisation (voir section 33.4 [fichier d'initialisation], page 106)[1].

Vous pouvez également forcer Emacs pour afficher un fichier ou un répertoire au démarrage en définissant la variable `initial-buffer-choice` à une valeur non-`nil`. (Dans ce cas, même si vous spécifiez un ou plusieurs fichiers en ligne de commande, Emacs s'ouvre mais ne les affiche pas). La valeur de `initial-buffer-choice` devrait être le nom du fichier ou du répertoire souhaité.

3.2 Sortir de Emacs

`C-x C-c` kill (quitter) Emacs (`save-buffers-kill-terminal`).

`C-z` dans un terminal texte, suspend Emacs ; en mode graphique, iconise le cadre sélectionné (`suspend-emacs`).

Kill Emacs signifie que le programme Emacs va se fermer. Pour faire cela, tapez la séquence `C-x C-c` (`save-buffers-kill-terminal`). Une séquence de touches à deux caractères est utilisée pour rendre difficile de la taper par accident. S'il y a des tampons de fichiers visités modifiés quand vous tapez `C-x C-c`, Emacs vous propose d'abord de sauvegarder les tampons. Si vous ne les sauvegardez pas tous, il vous demande encore la confirmation, alors les fichiers non sauvés seront perdus. Emacs demande également confirmation si des sous-processus sont encore en cours, car quitter Emacs va aussi clôre les sous-processus (voir section 31.3 [Shell], page 102).

`C-x C-c` se comporte spécialement si Emacs est utilisé comme serveur. Si vous la tapez à partir d'un «cadre client», il ferme la connexion du client. Voir section 31.4 [serveur Emacs], page 102.

Emacs peut, le cas échéant, enregistrer certaines informations de session quand vous le quittez, tels que les fichiez que vous visitiez à ce moment là. Cette information est alors disponible lors de la prochaine session Emacs. Voir section 31.8 [sauvegarde de sessions Emacs], page 102.

Si la valeur de la variable `confirm-kill-emacs` est non-`nil`, `C-x C-c` suppose que cette valeur est une fonction de prédicat, et appelle cette fonction. Si le résultat de la fonction appelée est non-`nil`, la session est terminée, sinon Emacs continue à tourner. Une fonction pratique à utiliser comme valeur de `confirm-kill-emacs` est la fonction `yes-or-no-p`. La valeur par défaut de `confirm-kill-emacs` est `nil`.

Pour quitter Emacs sans être invité à sauvegarder, tapez `M-x kill-emacs`.

`C-z` lance la commande `suspend-frame`. En mode graphique, cette commande iconise le cadre Emacs sélectionné, le cachant de façon à vous laisser le ramener plus tard (la façon exacte avec laquelle il se cache dépend du système de fenêtrage). Sur un terminal de texte, la commande `C-z` suspend Emacs, arrêtant temporairement le programme et retournant vers le processus parent (habituellement un shell) ; dans beaucoup de shells, vous pouvez reprendre Emacs après sa suspension avec la commande shell `%emacs`.

[1] Le réglage de la variable `inhibit-startup-screen` dans le programme `site-start.el` ne fonctionne pas, parce que l'écran de démarrage est mis en place avant la lecture du programme `site-start.el`. Voir section 33.4 [fichier d'initialisation], page 106, pour plus d'information à propos de `site-start.el`

Les terminaux textuels ont l'habitude de recevoir des caractères spéciaux pour quitter ou suspendre un programme en cours. Cette fonction du terminal est éteinte pendant que vous êtes dans Emacs. Les significations de `C-z` et de `C-x C-c` comme touches dans Emacs ont été inspirées par l'utilisation de `C-z` et `C-c` sur plusieurs systèmes d'exploitation comme caratères pour interrompre ou clôre un programme, mais c'est leur seul lien avec le système d'exploitation. Vous pouvez personnaliser ces touches pour exécuter les commandes de votre choix (voir section 33.3.1 [claviers], page 106).

Chapitre 4

Édition de commandes de base

Nous expliquons ici les bases de la saisie de texte, faire des corrections, et enregistrer le texte dans un fichier. Si ce matériel est nouveau pour vous, nous vous suggérons d'abord d'exécuter le tutoriel Emacs apprendre par la pratique, en tapant `C-h t` (`help-with-tutorial`).

4.1 Insertion de texte

Vous pouvez insérer un caractère ordinaire graphique (par exemple, «a», «B», «3», et «=») en tapant sur la touche associée. Cela ajoute le caractère au point dans le tampon. Le point d'insertion se déplace vers l'avant, de sorte que le point reste juste après le texte inséré. Voir la section 1.1 [point], page 2.

Pour terminer une ligne et en commencer une nouvelle, tapez `RET` (`newline`). (La touche `RET` peut être étiquetée `Return` ou `Enter` sur votre clavier, mais nous nous référerons à elle comme `RET` dans ce manuel). Cette commande insère un caractère de nouvelle ligne dans le tampon. Si le point est en fin de ligne, l'effet est de créer une ligne blanche après lui ; si le point est au milieu de la ligne, la ligne est partagée à cette position.

Comme nous l'expliquons plus loin dans ce manuel, vous pouvez changer la façon dont Emacs gère l'insertion de texte en basculant sur les modes mineurs. Par exemple, le mode *mineur* appelé Auto Fill partage automatiquement les lignes lorsqu'elles sont trop longues (voir section 22.5 [remplissage], page 79). Le mode mineur appelé Overwrite mode provoque l'effacement des caractères existant par insertion («écriture par dessus») de nouveaux, au lieu de pousser vers la droite. Voir section 20.2 [modes mineurs], page 73.

Seuls les caractères graphiques peuvent être insérés en tapant sur la touche associée ; les autres touches fonctionnent comme les commandes d'édition et ne s'insèrent pas. Par exemple, `DEL` exécute la commande `delete-backward-char` par défaut (certains modes se lient à une commande différente) ; il n'insère pas de caractère littéral «`DEL`» (de code caractère ASCII 127).

Pour insérer un caractère non-graphique, ou un caractère que votre clavier ne prend pas en charge, d'abord citez-le en tapant `C-q` (`quoted-insert`). Il y a deux façons d'utiliser `C-q` :

- `C-q` suivi d'un caractère non graphique (même `C-q`) insère ce caractère. Par exemple, `C-q DEL` insère un caractère littéral «`DEL`».

- `C-q` suivi d'une séquence de chiffres octaux insère le caractère correspondant à ce code octal. Vous pouvez utiliser n'importe quel nombre de chiffres en octal ; tout non-chiffre termine la séquence. Si le caractère de terminaison est `RET`, `RET` ne sert qu'à mettre fin

à la séquence. Tout autre non-chiffre termine la séquence et agit alors comme une saisie normale donc, `C-q 1 0 1` insère «AB».

L'utilisation de séquences octale est désactivée en mode Overwrite non-binaire ordinaire, pour vous donner un moyen pratique d'insérer un chiffre au lieu d'effacer avec elle (la séquence).

Pour utiliser le décimal ou l'hexadécimal au lieu de l'octal, définissez la variable `read-quoted-char-radix` à 10 ou 16. Si la base est 16, les lettres `a` et `f` servent dans le cadre d'un code de caractère, tout comme les chiffres. La casse est ignorée.

Alternativement, vous pouvez utiliser la commande `C-x 8 RET` (`insert-char`). C'est l'invite pour le nom Unicode code-point d'un caractère, en utilisant le mini-tampon. Si vous entrez un nom, la commande fournit la fin (voir section 5.4 [complétion], page 27). Si vous entrez un code-point, il doit être aussi un nombre hexadécimal (la convention pour Unicode), ou un nombre avec une base spécifique, par exemple, `#o23072` (octal) ; voir section «bases entières» dans le manuel de référence Emacs Lisp. La commande insère ensuite le caractère correspondant dans le tampon. Par exemple, les deux éléments suivants insèrent le signe de l'infini (code-point Unicode `U +221 E`) :

```
C-x 8 RET infinity RET

C-x 8 RET 221e RET
```

Un argument numérique à `C-q` ou `C-x 8 RET` spécifie le nombre de copies du caractère à insérer (voir section 4.10 [Arguments], page 23).

4.2 Changement d'emplacement du curseur

Pour faire plus de caractères d'insertion, vous devez savoir comment déplacer le point (voir la section 1.1 [point], page 2). Les commandes `C-f, C-b, C-n`, et `C-p` déplacent le point vers la droite, la gauche, le bas et le haut, respectivement. Vous pouvez aussi utiliser les flèches présentes sur la plupart des claviers : droite, gauche, bas et haut ; cependant, beaucoup d'utilisateurs Emacs trouvent qu'il est plus lent d'utiliser les touches flèches que les touches de commandes, car vous avez besoin de déplacer votre main vers la zone du clavier où se trouvent les flèches.

Vous pouvez aussi faire un clic gauche pour déplacer le point à l'endroit souhaité. Emacs fournit aussi une variété de commandes clavier additionnelles pour déplacer le point de façons plus sophistiquées.

`C-f` déplacement d'un caractère vers l'avant (`forward-char`).

`right` cette commande (`right-char`) qui a le même effet que `C-f`, à l'exception près : quand on édite de droite à gauche comme par exemple pour l'Arabe, cela fait alors un déplacement *arrière* si le paragraphe courant est de droite à gauche. Voir section **??** [édition bidirectionnelle], page **??**.

`C-b` déplacement arrière (`backward-char`).

`left` cette commande (`left-char`) se comporte comme `C-b`, excepté que le déplacement est vers l'*avant* si le paragraphe en cours est de droite à gauche. Voir section **??** [édition bidirectionnelle], page **??**.

`C-n`

down descend d'une ligne vers le bas (`next-line`). Cette commande laisse la position horizontale inchangé, donc si vous commencez au milieu d'une ligne, vous vous déplacerez au milieu de la ligne suivante.

C-p

up monte d'une ligne vers le haut (`previous-line`). Cette commande préserve la position dans la ligne, comme `C-n`.

C-a

Home va au début de la ligne (`move-beginning-of-line`).

C-e

End va à la fin de la ligne (`move-beginning-of-line`).

M-f avance d'un mot (`forward-word`).

C-right

M-right cette commande (`right-word`) se comporte comme `M-f`, excepté qu'il se déplace en *arrière* mot par mot si le paragraphe en cours est de droite à gauche. Voir section **??** [édition bidirectionnelle], page **??**.

M-b recule d'un mot (`backward-word`).

C-left

M-left cette commande (`left-word`) se comporte comme `M-f`, excepté qu'il se déplace en *avant* mot par mot si le paragraphe en cours est de droite à gauche. Voir section **??** [édition bidirectionnelle], page **??**.

M-r sans déplacer le texte de l'écran, reposition du point sur la marge de gauche de la ligne de texte la plus centrée de la fenêtre ; en conséquence des invocations successives, déplacent le point vers la marge de gauche de la ligne la plus haute, la plus basse, et ainsi de suite, de façon cyclique (`move-to-window-line-top-bottom`).

Un argument numérique dit à quelle ligne de l'écran placer le point, compter en descendant depuis la ligne du haut de la fenêtre (zéro signifie la ligne la plus haute). Un argument négatif compte les lignes depuis le bas (-1 signifie la ligne la plus basse). Voir section 4.10 [arguments], page 4.10, pour plus d'informations sur les arguments numériques.

M-< accède au haut du tampon (`beginning-of-buffer`). Avec un argument numérique n, déplace de $n/10$ depuis le haut.

M-> accède au bas du tampon (`end-of-buffer`).

C-v

PageDown

next défile l'affichage de l'écran vers l'avant, et déplace le point sur l'écran si nécessaire (`scroll-up-command`). Voir section 11.1 [défilement], page 11.1.

M-v

`PageUp`

`prior` défile l'affichage de l'écran vers l'arrière, et déplace le point sur l'écran si nécessaire (`scroll-down-command`). Voir section 11.1 [défilement], page 11.1.

`M-g c` lit un nombre n et déplace le point à la position n du tampon. Position 1 est le début du tampon.

`M-g M-g`

`M-g g` lit un nombre n et déplace le point au début de la ligne n (`goto-line`). La ligne 1 est le début du tampon. Si le point est dessus ou juste après un nombre dans le tampon, c'est par défaut n. Juste tapez `RET` dans le mini tampon pour l'utiliser. Vous pouvez aussi spécifier n en donnant à `M-g M-g` un argument numérique préfixé. Voir section 16.1 [sélection du tampon], page 16.1, pour le comportement de `M-g M-g` quand vous lui donnez un argument préfixé.

`M-g TAB` lit un nombre n et va à la colonne n de la ligne courante. La colone 0 est celle la plus à gauche. Si elle est appelée avec un argument préfixé, elle va à la colonne du numéro spécifié par la valeur de l'argument numérique.

`C-x C-n` utilise la colonne courante du point comme *colonne but semi-permanente* pour `C-n` et `C-p` (`set-goal-column`). Quand une colonne but semi-permanente est activée, ces commandes essaient toujours d'aller vers cette colonne, ou le plus près possible, après déplacements verticaux. La colonne but reste activée jusqu'à ce qu'elle soit désactivée.

`C-u C-x C-n` annule la colonne but. Désormais, `C-n` et `C-p` essaient de préserver la position horizontale, comme d'habitude.

Quand une ligne de texte du tampon est plus longue que la largeur de la fenêtre, Emacs l'affiche sur plusieurs lignes. Par commodité, `C-n` et `C-p` déplace le point par ligne, comme font les touches équivalentes `down` et `up`. Vous pouvez forcer ces commandes à se déplacer en accord avec des *lignes logiques* (i.e, en accord avec les lignes du texte dans le tampon) en configurant la variable `line-move-visual` à `nil` ; si une ligne logique occupe plusieurs lignes d'écran, le curseur saute alors les lignes d'écran supplémentaires. Pour les détails, voir section 4.8 [lignes continues], page 21. Voir section 33.2 [variables], page 106, pour comment configurer les variables comme `line-move-visual`.

Contrairement à `C-n` et `C-p`, la plupart des commandes Emacs qui fonctionnent sur les lignes fonctionnent sur les lignes *logiques*. Par exemple, `C-a` (`move-beginning-of-line`) et `C-e` (`move-end-of-line`) respectivement déplacent au début et à la fin d'une ligne logique. Chaque fois que nous rencontrons des commandes qui travaillent sur les lignes d'écrans, comme `C-n` et `C-p`, nous les signalerons.

Quand `line-move-visual` vaut `nil`, vous pouvez aussi configurer la variable `track-eol` avec une valeur non-`nil`. Alors `C-n` et `C-p`, quand le démarrage est à la fin d'une ligne logique, va à la fin de la prochaine ligne logique. Normalement, `track-eol` vaut `nil`.

`C-n` s'arrête normalement à la fin du tampon quand vous l'utilisez sur la dernière ligne du tampon. Toutefois, si vous configurez la variable `next-line-add-newlines` avec une valeur non-`nil`, `C-n` sur la dernière ligne du tampon crée une ligne supplémentaire à la fin et se déplace à la fin de cette dernière.

4.3 Effacement du texte

DEL

Backspace efface le caractère avant le point, ou la région si elle a été sélectionnée (**delete-backward-char**).

Delete efface le caractère après le point, ou la région si elle a été sélectionnée (**delete-forward-char**).

C-d efface le caractère après le point (**delete-char**).

C-k coupe la ligne (**kill-line**).

M-d coupe en avant jusqu'au prochain mot (**kill-word**).

M-DEL coupe en arrière jusqu'au mot précédent (**backward-kill-word**).

La commane **DEL** (**delete-backward-char**) efface le caractère avant le point, déplace le curseur et les caractères qui le suivent avant.

Si, toutefois, la région est sélectionnée, **DEL** efface le texte de la région. Voir chapitre 8 [marquage], page 47, pour une description de la région.

Sur la plupart des claviers, **DEL** est appelée **Backspace**, mais nous ferons référence à **DEL** dans ce manuel. (Ne pas confondre **DEL** avec la touche **Delete** (suppr.) ; nous discuterons de **Delete** momentannément.) Sur certains terminaux, Emacs ne reconnaît pas correctement la touche **DEL**. Voir section 34.2.1 [DEL ne supprime pas], page 108, si vous rencontrez ce problème.

La commande **delete** (**delete-forward-char**) supprime dans le «sens opposé» : elle supprime après le point, i.e le caractère sous le curseur. Si le point était en fin de ligne, cela joint la ligne suivante à celle en cours. Comme **DEL**, cela supprime le texte si la région est sélectionnée (voir chapitre 8 [marquage], page 47).

C-d (**delete-char**) supprime le caractère après le point, comme **delete**, mais sans tenir compte de la région sélectionnée.

Voir section 9.1.1 [suppression], page 49, pour plus d'information détaillée sur les commandes de suppression.

C-k (**kill-line**) coupe ligne par ligne. Si vous tapez **C-k** au début ou au milieu d'une ligne, cela coupe le texte jusqu'à la fin de la ligne. Si vous tapez **C-k** à la fin de la ligne, cela joint la ligne à la suivante.

Voir chapitre 9 [coupures], page 49, pour plus d'information à propos de **C-k** et les commandes reliées.

4.4 Retour en arrière des exécutions (undoing changes)

C-/ annule la dernière commande effectuée

C-x u

C-_ la même

Emacs enregistre une liste des changements réalisés dans le tampon, donc vous pouvez annuler les changements récents. Cet usage de la commande **undo**, est lié avec **C-/** (comme **C-x u** et **C-_**). Normalement, cette commande annule la dernière modification, déplace le point là où il était avant la modification. La commande d'annulation ne s'applique uniquement aux changements du tampon ; vous ne pouvez pas l'utiliser pour annuler les déplacement du curseur.

Bien que chaque commande d'édition fait habituellement une entrée séparée dans l'enregistrement des annulations, les commandes très simples peuvent être regroupées ensemble. Quelques fois, une entrée couvre juste une part d'une commande complexe.

Si vous répétez `C-/` (ou ses alias), chaque répétition annule un autre, changement précédent, jusqu'à la limite de la mémoire des modifications disponible. Si toutes les modifications enregistrées ont déjà été annulées, la commande d'annulation affiche un message d'erreur et ne fait plus rien.

Pour en apprendre plus sur la commande d'annulation, voir section 13.1 [annulation de modification], page 13.1.

4.5 Fichiers

Le texte que vous insérez dans le tampon Emacs reste seulement durant la session Emacs. Pour garder un texte en permanence vous devez le mettre dans un *fichier*.

Supposez qu'il y a un fichier nommé `test.emacs` dans votre répertoire principal (home). Pour commencer à l'éditer dans Emacs, tapez `C-x C-f test.emacs RET`.

Ici le nom du fichier est donné comme un argument de la commande `C-x C-f` (`find-file`). Cette commande utilise le *mini tampon* pour lire l'argument, et vous tapez `RET` pour terminer l'argument (voir chapitre 5 [mini tampon], page 25).

Emacs obéit à cette commande en *visitant* le fichier : il crée un tampon, copie le contenu du fichier dedans, et affiche le tampon pour l'édition. Si vous modifier le texte, vous pouvez *sauvegarder* le nouveau texte en tapant `C-x C-s` (`save-buffer`). Cela copie les contenus du tampon modifié dans le fichier `test.emacs`, ça les rend permanent. Jusqu'à ce que vous sauvegardiez, les changements n'existent que dans Emacs, et le fichier `test.emacs` est inchangé.

Pour créer un fichier, visitez-le avec `C-x C-f` comme si il existait déjà. Cela crée un tampon vide, dans lequel vous pouvez insérer le texte que vous souhaitez mettre dans le fichier. Emacs crée en fait le fichier la première fois que vous sauvegardez le tampon avec `C-x C-s`.

Pour en apprendre plus sur l'utilisation des fichiers dans Emacs, voir chapitre 15 [Fichiers], page 62.

4.6 Aide

Si vous oubliez ce que fait une touche, vous pouvez le trouver en tapant `C-h k` (`describe-key`), suivi par la touche qui vous intéresse ; par exemple, `C-h k C-n` vous dit ce que `C-n` fait.

La touche préfixée `C-h` sert pour «aide» (help). La touche `F1` sert d'alias pour `C-h`. Outre `C-h k`, il y a beaucoup d'autres commandes fournissant différentes sortes d'aide.

Voir chapitre 7 [Help], page 39, pour les détails.

4.7 Lignes blanches

Voici les commandes et techniques spéciales pour insérer et supprimer des lignes blanches.

`C-o` insère une ligne blanche après le curseur (`open-line`).

`C-x C-o` ne supprime pas tout mais une parmi plusieurs lignes blanches consécutives (`delete-blank-lines`).

Nous avons vu comment `RET` (`newline`) commence une nouvelle ligne du texte. Toutefois, il peut être plus facile de voir ce que vous faites si vous créez votre première ligne blanche et

ensuite insérez le texte désiré dedans. C'est facile de le faire en utilisant `C-o` (`open-line`), qui insère une nouvelle ligne après le point mais laisse le point en face de la ligne. Après `C-o`, tapez le texte de la nouvelle ligne.

Vous pouvez faire de nombreuses lignes blanches en tapant `C-o` plusieurs fois, ou en donnant un argument numérique spécifiant combien de lignes blanches vous créez. Voir section 4.10 [Arguments], page 4.10 pour savoir comment. Si vous avez un préfixe de remplissage, la commande `C-o` insère le préfixe de remplissage sur la nouvelle ligne, si tapé au début d'une ligne. Voir section 22.5.3 [Préfixe de remplissage], page 22.5.3.

La façon simple de se débarasser des lignes vides supplémentaires est avec la commande `C-x C-o` (`delete-blank-lines`). Si le point se trouve dans une série de plusieurs lignes vides, `C-x C-o` ne supprime pas tout mais l'une d'elles. Si le point est sur une seule ligne vierge, `C-x C-o` la supprime. Si le point est sur une ligne non vide, `C-x C-o` supprime toutes les lignes vides suivantes, s'il en existe.

4.8 Continuité des lignes

Parfois, une ligne de texte dans le tampon — une *ligne logique* — est trop longue pour tenir dans la fenêtre, et Emacs l'affiche sur deux ou plus *lignes d'écran*. On appelle ça *l'enroulement de ligne* ou *continuation*, et la longue ligne logique est appelée une *ligne continue*. En mode graphique, Emacs indique l'enroulement de ligne avec des petites flèches incurvées sur les bords gauches et droits de la fenêtre. Sur un terminal de texte, Emacs indique l'enroulement de ligne par un caractère «\» sur la marge de droite.

La plupart des commandes qui agissent sur les lignes agissent sur les lignes logiques, pas sur les lignes d'écran. Par exemple, `C-k` coupe une ligne logique. Comme décrit précédemment, `C-n` (`next-line`) et `C-p` (`previous-line`) sont des exceptions spéciales : elles déplacent le point vers le bas et le haut, respectivement, par une ligne d'écran (voir section 4.2 [déplacement du point], page 16).

Emacs peut optionnellement *tronquer* de longues lignes logiques au lieu de les continuer. Cela signifie que chaque ligne logique occupe une seule ligne d'écran ; si c'est plus grand que la largeur de la fenêtre, le reste de la ligne n'est pas affiché. En mode graphique, une ligne tronquée est indiquée par une petite flèche droite dans le coin ; sur un terminal de texte, c'est indiqué par le caractère «$» dans la marge de droite. Voir section 11.21 [troncature de ligne], page 54.

Par défaut, les lignes continues sont enroulées sur le bord droit de la fenêtre. Lorsque l'enroulement peut se réaliser au milieu d'un mot, les lignes continues peuvent être difficile à lire. La solution usuelle est de couper la ligne avant qu'elle ne soit trop longue, en insérant une nouvelle ligne. Si vous préférez, vous pouvez demander à Emacs d'insérer automatiquement une nouvelle ligne quand une ligne devient trop grande, en utilisant Auto Fill mode. Voir section 22.5 [remplissage], page 22.5.

Parfois, vous pouvez avoir besoin d'éditer des fichiers contenant beaucoup de lignes logiques, et ça peut ne pas être pratique de les couper toutes en ajoutant des nouvelles lignes. Dans ce cas, vous pouvez utiliser Visual Line mode, qui rend possible *l'enroulement de mot* : au lieu de l'enroulement de longues lignes exactement sur le bord droit de la fenêtre, Emacs les enroule au bord des mots (i.e, espace ou tabulation) les plus proches du coin droit de la fenêtre. Visual Line mode redéfinit aussi les commandes d'édition comme `C-a, C-n`, et `C-k` pour opérer sur les lignes d'écran au lieu des lignes logiques. Voir section 11.22 [Visual Line Mode], page 11.22.

4.9 Information sur la position du curseur

Voilà des commandes pour obtenir des informations concernant la taille et la position de parties du tampon, et pour compter les mots et les lignes.

M-x what-line affiche le numéro de la ligne du point.

M-x line-number-mode bascule l'affichage automatique de la ligne courante. Voir section 11.18 [Option du mode ligne], page 54.

M-x column-number-mode bascule l'affichage automatique de la colonne courante. Voir section 11.18 [Option du mode ligne], page 54.

M-= affiche le nombre de lignes, mots, et caractères présents dans la région (`count-words-region`). Voir chapitre 8 [Marques], page 47, pour les informations au sujet de la région.

M-x count-words affiche le nombre de lignes, mots, et caractères présents dans le tampon. Si la région est active (voir chapitre 8 [Marques], page 47), affiche le nombre pour la région à la place.

C-= affiche le code du caractère après le point, la position du caractère du point, et la colonne du point (`what-cursor-position`).

M-x hl-line-mode active ou désactive le surlignage de la ligne courante. Voir section 11.20 [Affichage du curseur], page 54.

M-x size-indication-mode bascule l'affichage automatique de la taille du tampon. Voir section 11.18 [Affichage du curseur], page 54.

`M-x what-line` affiche le numéro de la ligne courante dans la zone d'écho. Cette commande est habituellement redondante, parce que le numéro de ligne courante est montré dans la ligne de mode (voir section 1.3 [ligne de mode], page 3). Toutefois, si vous réduisez le tampon, la ligne de mode montre le numéro de ligne relatif à la portion accessible (voir section 11.5 [Réduction], page 54). Par contraste, `what-line` affiche à la fois le numéro de ligne relatif à la région réduite et le numéro de ligne relatif au tampon entier.

`M-=`(`count-words-region`) affiche un message rapportant le nombre de lignes, mots, et caractères dans la région (Voir chapitre 8 [Marques], page 47, pour une explication de la région). Avec un argument préfixé, `C-u M-=`, la commande affiche le compte pour le tampon entier.

La commande `M-x count-words` fait le même travail, mais avec des conventions d'appels différentes. Ça affiche le compte pour la région si la région est active, et sinon pour le tampon entier.

La commande `C-x =` (`what-cursor-position`) montre des informations à propos de la position courante du curseur et le contenu du tampon à cette position. Ça affiche une ligne dans la zone d'écho qui ressemble à ça :

```
Char : c (99, #o143, #x63) point=28062 of 36168 (78%) column=53
```

Après «`Char :`», ça montre le caractère dans le tampon au niveau du point. Le texte entre les parenthèses montre la correspondance décimale, octale et hexadécimale des codes de caractères ; pour plus d'informations au sujet de comment `C-x =` affiche l'information de caractère, voir section 19.1 [Caractères internationaux], page 70. Après «`point=`» est la position du point comme caractère compté (le premier caractère dans le tampon est en position 1, le second en position 2, et ainsi de suite). Le nombre après ça est le nombre total de caractères dans le tampon, et le

nombre entre parenthèses exprime la position comme un pourcentage du total. Après «`column=`» est la position horizontale du point, on compte les colonnes depuis le bord gauche de la fenêtre.

Si le tampon a été réduit, cela fait qu'une partie du texte au début et à la fin sont temporairement inaccessible, `C-x =` affiche du texte supplémentaire décrivant les rangs accessible. Par exemple, ça pourrait afficher ça :

```
Char : C (67, #o103, #x43) point=252 of 889 (28%) <231-599> column=0
```

où les deux nombres supplémentaires donnent la position la plus petite et la plus grande des caractères autorisée. Les caractères entre ces deux positions sont accessibles. Voir section 11.5 [Réduction], page 54.

4.10 Arguments numériques

Dans la terminologie mathématique et informatique, *argument* signifie «donnée fournie à une fonction ou opération». Vous pouvez donner à n'importe quelle commande Emacs un *argument numérique* (aussi appelé un *argument préfixé*). Certaines commandes interprètent l'argument comme une répétition de compte. Par exemple, donner à `C-f` un argument de dix provoque le déplacement du point de dix caractères vers l'avant au lieu d'un seul. Avec ces commandes, l'absence d'argument est équivalente à un argument de un, et les arguments négatifs provoquent le déplacement ou l'action dans le sens opposé.

La façon la plus simple de spécifier un argument numérique est de taper un chiffre et/ou un signe moins tout en maintenant la touche `META` enfoncée. Par exemple,

```
M-5 C-n
```

se déplace vers le bas de 5 lignes. Les touches `M-1, M-2`, et ainsi de suite, tout comme `M--`, sont bornées aux commandes (`digit-argument` et `negative-argument`) qui configurent un argument numérique de la prochaine commande. `Meta--` sans chiffre signifie -1.

Si vous saisissez plus qu'un chiffre vous devez relâcher la touche `META` pour le second chiffre et les suivants. Donc, pour se déplacer de 50 lignes vers le bas, tapez

```
M-5 0 C-n
```

Remarquez que cela *ne fait pas* l'insertion de cinq copies de «0» et le déplacement d'une ligne vers le bas, comme vous l'aurez deviné — le «0» est traité comme une partie de l'argument préfixé.

(Que faire si vous voulez insérer cinq copies de «0» ? Tapez `M-5 C-u 0`. Ici, `C-u` «finit» l'exécution. Remarquez que cela signifie que `C-u` s'applique uniquement dans ce cas. Pour le rôle habituel de `C-u`, voir ci-dessous.)

Au lieu de taper `M-1, M-2`, et ainsi de suite, une autre façon de spécifier un argument numérique est de taper `C-u` (`universal-argument`) suivi par quelques chiffres, ou (pour un argument négatif) un signe moins suivi par des chiffres. Un signe moins sans chiffre signifie normalement -1.

`C-u` seul a le sens spécial de «quatre fois» : ça multiplie l'argument de la prochaine commande par quatre.`C-u C-u` multiplie par seize. Donc, `C-u C-u C-f` se déplace de seize caractères vers l'avant. D'autres combinaisons utiles sont `C-u C-n, C-u C-u C-n` (se déplace vers le bas d'une bonne fraction de l'écran), `C-u C-u C-o` (crée «beaucoup» de lignes blanches), et `C-u C-k` (supprime quatre lignes).

Vous pouvez utiliser un argument numérique avant une auto-insertion de caractère afin d'insérer des copies multiples de celui-ci. C'est très direct quand le caractère n'est pas un chiffre ; par

exemple, `C-u 64 a` insère 64 copies du caractère «a». Mais ça ne marche pas pour l'insertion de nombre ; `C-u 6 4 1` spécifie un argument de 641. Vous pouvez séparer les arguments des chiffres pour insérer d'autres `C-u` ; par exemple, `C-u 64 C-u 1` fait insérer 64 copies du caractères «1».

Certaines commandes font attention si il y a un argument, mais ignore sa valeur. Par exemple la commande `M-q` (`fill-paragraph`) remplit le texte ; avec un argument, ça justifie le texte aussi. (Voir section 22.5 [Remplissage], page 79, pour plus d'information sur `M-q`.) Pour ces commandes, c'est suffisant de spécifier l'argument avec un simple `C-u`.

Certaines commandes utilisent la valeur de l'argument comme une répétition de compte, mais font quelque chose de spécial quand il n'y a pas d'argument. Par exemple, la commande `C-k` (`kill-line`) avec un argument n supprime n lignes, incluant les nouvelles lignes de fin. Mais `C-k` sans argument spécial : ça supprime le texte au dessus de la nouvelle ligne, ou, si le point est à la fin de la ligne, ça supprime la ligne elle-même. Donc, deux `C-k` sans argument peuvent supprimer une ligne non blanche, comme `C-k` avec un argument de un. (Voir chapitre 9 [Suppression], page 49, pour plus d'information sur `C-k`.)

Quelques commandes traitent `C-u` différemment qu'un argument ordinaire. Quelques autres peuvent traiter un argument signe moins différemment d'un argument -1. Ces cas non usuels sont décrits quand ils arrivent ; ils existent pour créer une commande individuelle plus commode, et ils sont documentés dans la chaîne de documentation de commande.

Nous utilisons le terme *argument préfixé* pour insister sur le fait que vous tapez ces arguments avant la commande, et pour les distinguer des arguments du mini-tampon (voir chapter 5 [Mini-tampon], page 25), qui sont saisis après l'invocation de la commande.

4.11 Répétition de commande

De nombreuses commandes simples, comme celles invoquées avec une touche simple ou avec `M-x` *`command-name`* `RET`, peuvent être répétées par leur invocation avec un argument numérique qui sert de compte de répétition (voir section 4.10 [Arguments] page 23). Toutefois, si la commande que vous souhaitez répéter vous invite à la saisie, ou utilise un argument numérique d'une autre façon, la méthode ne marchera pas.

La commande `C-x z` (`repeat`) fournit ne autre façon de répéter des commandes Emacs plusieurs fois. Cette commande répète la commande Emacs précédente, qu'elle quelle soit. La répétition d'une commande utilise le même argument qui était utilisé avant ; ça ne lit pas de nouveaux arguments à chaque fois.

Pour répéter la commande plus d'une fois, tapez des `z` supplémentaires : chaque `z` répète la commande une fois de plus. La répétition finit quand vous tapez un autre caractère que `z`, ou cliquez sur un bouton de la souris.

Par exemple, supposez que vous tapez `C-u 20 C-d` pour supprimer 20 caractères. Vous pouvez répéter cette commande (incluant ses arguments) trois fois supplémentaires, pour effacer un total de 80 caractères, en tapant `C-x z z z`. Le premier `C-x z` répète la commande une fois, et chaque `z` suivant la répète une fois encore.

Chapitre 5

Le minibuffer ou mini-tampon

Le *mini-tampon* est l'endroit où les commandes Emacs lisent les arguments compliqués, comme des noms de fichiers, noms de tampons, noms de commandes Emacs, ou expressions Lisp. On l'appelle le «mini-tampon» parce que c'est un tampon à usage spécial avec une petite taille d'espace d'écran. Vous pouvez utiliser les commandes d'édition Emacs usuelles dans le mini-tampon pour éditer le texte de l'argument.

5.1 Utilisation du mini-tampon

Quand le mini-tampon est utilisé, il apparaît dans la zone d'écho, avec un curseur. Le mini-tampon commence avec une *invite* (prompt), terminant habituellement avec deux points. L'invite indique le type de saisie attendue, et comment elle sera utilisée. L'invite est surlignée et utilise la face `minibuffer-prompt` (voir section 11.8 [Faces], page 54).

La façon la plus simple de saisir un argument dans le mini-tampon est de taper le texte, puis `RET` pour soumettre l'argument et sortir du mini-tampon. Alternativement, vous pouvez taper `C-g` pour sortir du mini-tampon en annulant la commande demandant l'argument (voir section 34.1 [Quitter], page 108).

Parfois, l'invite montre un *argument par défaut*, à l'intérieur de parenthèses avant les deux points. Cet argument par défaut sera utilisé comme l'argument si vous tapez juste `RET`. Par exemple, des commandes qui lisent les noms de tampons montrent usuellement un nom de tampon par défaut ; vous pouvez taper `RET` pour travailler dans ce tampon par défaut.

Si vous activez Minibuffer Electric Default mode, un mode mineur complet, Emacs cache l'argument par défaut dès que vous modifiez les contenus du mini-tampon (alors taper `RET` ne soumettra plus l'argument par défaut). Si jamais vous ramenez le texte original du mini-tampon, l'invite montrera à nouveau l'argument par défaut. En outre, si vous changez la variable `minibuffer-eldef-shorten-default` avec une valeur non-`nil`, l'argument par défaut est affiché comme «`[default]`» au lieu de «`(default` *`default`*`)`», sauvegardant un peu d'espace d'écran. Pour activer ce mode mineur, tapez `M-x minibuffer-electric-default-mode`.

Dès que le mini-tampon apparaît dans la zone d'écho, il peut y avoir conflit avec d'autres usages de la zone d'écho. Si un message d'erreur ou un message d'information est émis pendant que le buffer est actif, le message cache le mini-tampon pour quelques secondes, ou jusqu'à ce que vous tapiez quelque chose ; alors le mini-tampon revient. Pendant que le mini-tampon est en usage, les raccourcis ne font pas d'écho.

5.2 Minibuffer pour les noms de fichiers

Des commandes comme `C-x C-f` (`find-file`) utilisent le mini-tampon pour lire un argument de nom de fichier (voir section 4.5 [Fichiers de base], page 20). Quand le mini-tampon est utilisé pour lire un nom de fichier, ça commence typiquement avec du texte initial se terminant par une barre oblique. C'est le *répertoire par défaut*. Par exemple, ça peut commençait par quelque chose comme ça :

```
Find file : /u2/emacs/src/
```

Ici, «`Find file :`» est l'invite et «`/u2/emacs/src/`» est le répertoire par défaut. Si vous tapez maintenant `buffer.c` comme saisie, cela spécifie le fichier `/u2/emacs/src/buffer.c`. Voir section 15.1 [Noms de fichiers], page 62, pour information à propos du répertoire par défaut.

Vous pouvez spécifier le répertoire parent avec `..` : `/a/b/../foo.el` qui est équivalent à `/a/foo/el`. Alternativement, vous pouvez utiliser `M-DEL` pour supprimer le nom du répertoire (voir section 22.1 [Mots], page 77).

Pour spécifier un fichier dans un répertoire complètement différent, vous pouvez supprimer entièrement le répertoir par défaut avec `C-a C-k` (voir section 5.3 [Édition du mini-tampon], page 26). Alternativement, vous pouvez ignorer le repertoire par défaut, et saisir un chemin absolu en commençant par une barre oblique ou un tilde après le répertoire par défaut. Par exemple, si vous spécifiez `/etc/termcap` comme suit :

```
Find file : /u2/emacs/src//etc/termcap
```

Emacs interprète une double barre oblique comme «ignore tout ce qu'il y a avant la seconde barre oblique dans la paire». Dans l'exemple ci-dessus, `/u2/emacs/src/` est ignoré, donc l'argument que vous soumettez est `/etc/termcap`. La partie ignorée du chemin ignoré est grisée si le terminal le permet. (Pour désactiver ce grisement, désactiver le mode File Name Shadow avec la commande `M-x file-name-shadow-mode`.)

Emacs interprète `~/` comme votre répertoir personnel. Donc, `~/foo/bar.txt` spécifie un nom de fichier `bar.txt`, dans un répertoire nommé `foo`, qui est localisé dans votre répertoire personnel. De plus, `~user-id/` signifie : le répertoire personnel d'un utilisateur dont l'identifiant est *user-id*. N'importe quel nom de répertoire avant le `~` est ignoré : donc, `/u2/emacs/~/foo/bar.txt` est équivalent à `~/foo/bar.txt`.

Sur des systèmes MS-Windows et MS-DOS, où l'utilisateur n'a pas toujours de repertoire personnel, Emacs utilise plusieurs alternatives. Pour MS-Windows, voir section **??** [Windows HOME], page **??** ; pour MS-DOS, voir section «MS-DOS File Names» dans *la version numérique du manuel Emacs*. Sur ces systèmes, le `~user-id/` construit est supporté uniquement pour l'utilisateur en court i.e, seulement si *user-id* est l'identifiant de l'utilisateur en court.

Pour prévenir Emacs de l'insertion du répertoire par défaut lors de la lecture de noms de fichiers, modifier la variable `insert-default-directory` à `nil`. Dans ce cas, le mini-tampon démarre vide. Néanmoins, les arguments de noms de fichiers sont encore interprétés basés sur le même répertoire par défaut.

Vous pouvez aussi saisir des noms de fichiers distants dans le mini-tampon. Voir section 15.13 [Fichiers distants], page 62.

5.3 Édition dans le minibuffer

Le mini-tampon est un tampon Emacs, quoique particulier, et les commandes Emacs usuelles sont disponibles pour l'édition du texte de l'argument. (L'invite, toutefois, est *en lecture seule*, et ne peut pas être modifiée.)

Dès que `RET` dans le mini-tampon soumet l'argument, vous ne pouvez pas l'utiliser pour insérer une nouvelle ligne. Vous pouvez faire ça avec `C-q C-j`, qui insère un caractère de contrôle `C-j`, qui est formellement équivalent à un caractère de nouvelle ligne (voir section 4.1 [Insertion de texte], page 4.1). Alternativement, vous pouvez utiliser la commande `C-o (open-line)` (voir section 4.7 [Lignes vierges], page 4.7).

Dans le mini-tampon, les touches `TAB`,`SPC` et `?` sont souvent liées à *des commmandes de complétion*, qui vous permettent de facilement remplir le texte désiré sans le taper en entier. Voir section 5.4 [Complétion], page 5.4. Comme avec `RET`, vous pouvez utiliser `C-q` pour insérer un caractère `TAB`,`SPC` ou `?`.

Par commodité, `C-a (moving-beginning-of-line)` dans le mini-tampon déplace le point au début de l'argument de texte, mais pas au début de l'invite. Par exemple, cela vous permet d'effacer l'argument entier avec `C-a C-k`.

Quand le mini-tampon est actif, la zone d'écho est traitée plus comme une fenêtre Emacs ordinaire. Par exemple, vous pouvez changer pour une autre fenêtre (avec `C-x o`), éditer du texte là, ensuite revenir à la fenêtre du mini-tampon pour finir l'argument. Vous pouvez même supprimer du texte dans une autre fenêtre, revenir dans la fenêtre du mini-tampon, et coller le texte dans l'argument. Il y a quelques restrictions dans la fenêtre du mini-tampon, toutefois : par exemple, vous ne pouvez pas la découper. Voir chapitre 17 [Fenêtres], page 65.

Normalement, la fenêtre de mini-tapon occupe une seule ligne d'écran. Toutefois, si vous ajouter deux ou plusieurs lignes de texte dans le mini-tampon, sa taille augmente automatiquement de façon à s'adapter au texte. La variable `resize-mini-windows` contrôle la redimension du mini-tampon. La valeur par défaut est `grow-only`, qui signifie le comportement que nous venons juste de décrire. Si la valeur est `t`, la fenêtre du mini-tampon sera aussi automatiquement rétractable si vous effacez des lignes de texte du mini-tampon, jusqu'au minimum d'une ligne d'écran. Si la valeur est `nil`, la fenêtre du mini-tampon ne changera jamais de taille automatiquement, mais vous pouvez utiliser la commande usuelle de redimensionnement de la fenêtre (voir chapitre 17 [Fenêtres], page 65).

La variable `max-mini-window-height` contrôle la hauteur maximale de redimensionnement de la fenêtre du mini-tampon. Un nombre à virgule spécifie une fraction de la hauteur du cadre ; un entier spécifie le nombre maximal de lignes ; `nil` signifie : ne pas redimensionner la fenêtre du mini-tampon automatiquement. La valeur par défaut est 0.25.

La commande `C-M-v` dans le mini-tampon fait défiler le texte d'aide de commandes qui affiche l'aide de texte de toute sorte dans une autre fenêtre. Vous pouvez aussi faire défiler l'aide de texte avec `M-prior` et `M-next` (ou, de façon équivalente, `M-PageUp` et `M-PageDown`). C'est particulièrement utile avec des listes longues de complétions possible. Voir section 17.3 [Autre fenêtre], page 17.3.

Emacs normalement ne permet pas la plupart des commandes qui utilisent le mini-tampon pendant que le mini-tampon est actif. Pour autoriser de telles commandes dans le mini-tampon, initialiser la variable `enable-recursive-minibuffers` à `t`.

Quand il n'est pas actif, le mini-tampon est en `minibuffer-inactive-mode`, et en cliquant `Mouse-1` cela montre le tampon `*Messages*`. Si vous utilisez un cadre dédié pour le mini-tampon, Emacs reconnaît aussi certaines touches là, par exemple `n` pour créer un nouveau cadre.

5.4 Complétion (ou complémentation)

Vous pouvez souvent utiliser la fonction appelée *complétion* qui aide à saisir les arguments. Cela signifie qu'après que vous tapiez une partie de l'argument, Emacs peut remplir le reste, ou une partie, basée sur ce qui a déjà été rempli.

Quand la complétion est disponible, certaines touches (usuellement `TAB`, `RET`, et `SPC`) sont réservées dans le mini-tampon pour des commandes de complétions spéciales (voir section 5.4.2 [Commandes de complétion], page 5.4.2). Ces commandes attendent de compléter le texte dans le mini-tampon, basées sur un *ensemble de complétion alternatives* fournies par la commande qui requiert un argument. Vous pouvez généralement taper `?` pour voir une liste de complétions alternatives.

Bien que la complétion est généralement faite dans le mini-tampon, la fonction est parfois disponible dans les tampons ordinaires aussi. Voir section 23.8 [Symbole de complétion], page 86.

5.4.1 Exemple de complétion

Un exemple simple peut aider ici. `M-x` utilise le minitampon pour lire le nom d'une commande, donc la complétion fonctionne par correspondance du texte du minitampon avec les noms des commandes Emacs existantes. Supposons que vous souhaitiez lancer la commande `auto-fill-mode`. Vous pouvez le faire en tapant `M-x auto-fill-mode RET`, mais il est plus facile d'utiliser la complétion.

Si vous tapez `M-x a u TAB`, la `TAB` cherche les complétions alternatives (dans ce cas, les noms de commandes) qui démarrent par «`au`». Il y en a plusieurs, incluant `auto-fill-mode` et `autoconf-mode`, mais elles commencent toutes avec `auto`, donc le «`au`» dans le minitampon se complète en «`auto`». (Plus de commandes peuvent être définies dans votre session Emacs. Par exemple, si une commande appelée `authorize-me` était définie, Emacs pourrait seulement compléter jusqu'à «`aut`».)

Si vous tapez `TAB` encore immédiatement, il ne peut pas déterminer le prochain caractère ; cela pourrait être «`-`», «`a`», ou «`c`». Donc il n'ajoute aucun caractère ; au lieu, `TAB` affiche une liste de toutes les complétions possibles dans une autre fenêtre.

Ensuite, tapez `-f`. Le minitampon contient désormais «`auto-f`», et le seul nom de commande qui démarre avec ça est «`auto-fill-mode`». Si vous tapez maintenant `TAB`, la complétion rempli le reste de l'argument «`auto-fill-mode`» dans le minitampon.

D'où, taper juste `a u TAB -f TAB` vous permet d'entrer «`auto-fill-mode`».

5.4.2 Commandes de complétion

Voici une liste des commandes de complétions définies dans le minitampon la complétion est permise.

`TAB` Complète le texte dans le minitampon autant que possible ; si pas possible de compléter, affiche une liste des complétions possibles (`minibuffer-complete`).

`SPC` Complète jusqu'à un mot du texte du minitampon avant le point (`minibuffer-complete-word`). Cette commande n'est pas disponible pour des arguments qui incluent souvent des espaces, comme des noms de fichiers.

`RET` Présente le texte dans le minitampon comme argument, complétant éventuellement le premier (`minibuffer-complete-and-exit`). Voir section 5.4.3 [Sortie de complétion], page 29.

`?` Affiche une liste des complétions (`minibuffer-completion-help`).

`TAB` (`minibuffer-complete`) est la commande de complétion la plus fondamentale. Elle cherche toutes les complétions possibles qui correspondent au texte existant dans le minitampon, et tente de le compléter autant que possible. Voir section 5.4.4[Styles de complétion], page 30, pour savoir comment les complétions alternatives sont choisies.

`SPC` (`minibuffer-complete-word`) complète comme `TAB`, mais seulement jusqu'au prochain tiret ou espace. Si vous avez «`auto-f`» dans le minitampon et tapez `SPC`, cela trouvera la com-

plétion «auto-fill-mode», mais cela insère seulement «ill-», donnant «auto-fill». Un autre SPC à cet endroit complète le reste jusqu'à «auto-fill-mode».

Si TAB ou SPC n'arrive pas à compléter, cela affiche une liste des complétions alternatives correspondantes (s'il y en a) dans une autre fenêtre. Vous pouvez afficher la même liste avec ? (minibuffer-completion-help). Les commandes suivantes peuvent être utilisées avec la liste de complétion :

Mouse-1
Mouse-2 Cliquer sur le bouton 1 ou 2 sur une complétion alternative la choisis (mouse-choose-completion).

M-v
PageUp
prior Taper M-v, dans le minitampon, sélectionne la fenêtre montrant la liste de complétion (switch-to-completions). Cela ouvre la voie à l'aide des commandes ci-dessous. PageUp ou prior fait la même. Vous pouvez aussi sélectionner la fenêtre d'autres manières (voir 17[Fenêtres], page 65).

RET Dans le tampon de la liste de complétion, cela choisit la complétion au point (choose-completion).

Right Dans le tampon de la liste de complétion, cela déplace le point à la complétion alternative suivante (next-completion).

Left Dans le tampon de la liste de complétion, cela déplace le point à la complétion alternative précédente (previous-completion).

5.4.3 Sortie de complétion

Quand une commande lit un argument utilisant le minitampon avec complétion, cela contrôle aussi ce qui se passe lorsque vous tapez RET (minibuffer-complete-and-exit) pour présenter l'argument. Il y a quatre types de comportements :

- *Complétion stricte* accepte seulement les complétions correspondantes exactes. Taper RET sort du minitampon seulement si le texte du minitampon est une correspondance exacte, ou la complète en une. Sinon, Emacs refuse de quitter le minitampon ; au lieu de cela il essaie de compléter, et si aucune complétion ne peut être faite il affiche momentanément «[No match]» après le texte du minitampon. (Vous pouvez toujours quitter le minitampon en tapant C-g pour annuler la commande.)

  Un exemple d'une commande qui utilise ce comportement est M-x, car ça n'a pas de sens pour elle d'accepter un de commande inexistante.

- *La complétion prudente* est comme la complétion stricte, sauf que RET sort seulement si le texte est déjà une correspondance exacte. Si le texte complète en une correspondance exacte, RET accompli cette complétion mais ne sort pas encore ; vous devez taper RET une seconde fois pour sortir.

  La complétion prudente est utilisée pour lire les noms de fichiers de fichiers qui existent déjà, par exemple.

- *La complétion permissive* autorise n'importe qu'elle saisie ; les candidats à la complétion sont justes des suggestions. Taper RET ne complète pas, ça présente juste l'argument comme vous l'avez entré.

- *La complétion permissive avec confirmation* est comme la complétion permissive, à une exception : si vous tapez TAB et que ça complète le texte jusqu'à un état intermédiaire

(i.e, un qui n'est pas une complétion correspondante exacte), taper `RET` juste après ne présente pas l'argument. Au lieu de ça, Emacs demande une confirmation en affichant momentanément «[`Confirm`]» après le texte ; taper `RET` encore pour confirmer et présenter le texte. Cela empêche une erreur classique, dans laquelle une frappe `RET` avant de realiser que `TAB` n'avait pas complété aussi loin que souhaité.

Vous pouvez modifier le comportement de la confirmation par la personnalisation de la variable `confirm-nonexistent-file-or-buffer`. La valeur par défaut, `after-completion`, donne le comportement que nous venons de décrire. Si vous la changez à `nil`, Emacs ne demandera pas de confirmation, retombant en complétion permissive. Si vous la changez en n'importe qu'elle autre valeur non-`nil`, Emacs demandera une confirmation que la précédente comme soit `TAB` ou pas.

Ce comportement est utilisé par la plupart des commandes qui lisent des noms de fichiers, comme `C-x C-f`, et les commandes qui lisent les noms de tampons, comme `C-x b`.

5.4.4 Comment sont choisies les alternatives de complétion

Les commandes de complétion fonctionnent par rétrécissement d'une grande liste de complétions alternatives possibles vers un sous ensemble qui «correspond» avec ce que vous avez tapé dans le mini tampon. Dans la section 5.4.1 [Exemple de complétion], page 28, nous avons donné un exemple simple d'une telle correspondance. Emacs tente d'offrir les complétions plausibles dans la plupart des cas.

Emacs réalise les complétions en utilisant un ou plusieurs *styles de complétion*—ensembles de critères de correspondance pour le texte du mini tampon pour les complétions alternatives. Durant la complétion, Emacs essaie chaque style de complétion tour à tour. Si un style donne une ou plusieurs correspondances, qui est utilisée comme liste de complétions alternatives. Si un style ne produit pas de correspondance, Emacs retombe sur le style suivant.

Cette variable liste `completion-styles` spécifie les styles de complétions à utiliser. Chaque élément de la liste est le nom d'un style de complétion (un symbole Lisp). Les styles de complétion par défaut sont (par ordre) :

`basic` une complétion alternative doit avoir le même début que le texte dans le mini-tampon avant le point. En outre, s'il y a un texte dans le mini-tampon après le point, le reste de la complétion alternative doit contenir ce texte comme sous chaîne.

`partial-completion` Ce style de complétion agressif divise le texte du mini-tampon en mots séparés par des tirets ou des espaces, et termine chaque mot séparément. (Par exemple, quand il complète les noms de commandes, «`em-l-m`» complète en «`emacs-lisp-mode`».)

En outre, une «*» dans le texte du mini-tampon est traité comme un `joker`—ça correspond à n'importe quel caractère à la position associée dans la complétion alternative.

`emacs22` Ce style de complétion est similaire du `basic`, sauf qu'il ignore le texte dans le mini-tampon après le point. Il est appelé ainsi parce qu'il correspond au comportement de complétion dans Emacs 22.

Les styles de complétion supplémentaires suivants sont aussi définis, et vous pouvez les ajouter au `completion-styles` si vous le souhaitez (voir Chapitre 33 [Personnalisation], page 106) :

`substring` une correspondance de complétion alternative doit contenir le texte dans le mini-tampon avant le point, et le texte dans le mini-tampon après le point, comme chaînes (dans le même ordre).

Alors, si le texte dans le mini-tampon est «`foobar`», avec le point entre «`foo`» et «`bar`», qui correspond à «`afoo`*b*`barc`», où *a*,*b*, et *c* peuvent être n'importe quelle chaîne incluant la chaîne vide.

`initials` Ce style de complétion très agressif tente de compléter les acronymes et sigles. Par exemple, quand il complète les noms de commandes, il fait correspondre «`lch`» avec «`list-command-history`».

Il y a aussi un style de complétion très simple appelé `emacs21`. Dans ce style, si le texte dans le mini-tampon est «`foobar`», seules les correspondances commençant avec «`foobar`» sont considérées.

Vous pouvez utiliser des styles de complétion différents dans différentes situations, en réglant la variable `completion-category-overrides`. Par exemple, le réglage par défaut dit d'utiliser seulement la complétion `basic` et `substring` pour les noms de tampons.

5.4.5 Options de complétion

Un cas est important au moment de remplir les arguments de la casse, comme par exemple les noms de commandes, '`AU`' ne se complète pas en '`auto-fill-mode`'. Différents cas sont ignorés au moment de remplir des arguments dans ce cas ce n'est pas grave.

Lorsque vous remplissez les noms de fichiers, les différences de casse sont ignorées si la variable `read-file-name-completion-ignore-case` est non-`nil`. La valeur par défaut est `nil` sur les systèmes qui ont des noms de fichiers sensibles à la casse, comme GNU/Linux ; il est non-`nil` sur les systèmes qui sont insensibles à la casse, comme Microsoft Windows. Lorsque vous remplissez les noms de tampons, les différences de casse sont ignorées si la variable `read-buffer-completion-ignore-case` est non-`nil` ; par défaut elle vaut `nil`.

Lorsque vous remplissez les noms de fichiers, Emacs omet généralement certaines alternatives qui sont considérées peu probable d'être choisies, telle que déterminée par la variable liste `completion-ignored-extensions`. Chaque élément de la liste doit être une chaîne ; n'importe quel nom de fichier se terminant par une telle chaîne est ignoré comme alternative d'achèvement. Tout élément se terminant par une barre oblique (/) représente un nom de sous répertoire. La valeur standard de `completion-ignored-extensions` comporte plusieurs éléments dont «`.o`», «`.elc`», et «`~`». Par exemple, si un répertoire contient '`foo.c`' et '`foo.elc`', '`foo`' se complète en '`foo.c`'. Toutefois, si toutes les complétions possibles finissent dans les chaînes «ignorées», elles ne sont pas ignorées : dans l'exemple précédent, '`foo.e`' se complète en '`foo.elc`'. Emacs ignore `completion-ignored-extensions` en montrant des alternatives de complétions dans la liste des terminaisons possibles.

Si `completion-auto-help` est réglé à `nil`, les commandes de complétion n'affiche jamais le tampon de la liste d'achèvement ; vous devez taper `?` pour afficher la liste. Si la valeur est `lazy`, Emacs affiche uniquement le tampon de la liste d'achèvement pour la deuxième tentative de complétion. En d'autres termes, s'il n'y a rien à compléter, le premier écho de `TAB` sera '`Next char not unique`' ; le second `TAB` affichera le tampon de la liste de complétion.

Si `completion-cycle-threshold` est non-`nil`, les commandes de complétion peuvent «boucler» à travers les complétions alternatives. Normalement, s'il y a plus d'une alternative d'achèvement pour le texte dans le mini-tampon, une commande de complétion complète jusqu'à la plus longue sous-chaîne commune. Si vous changez `completion-cycle-threshold` pour `t`, la commande de complétion au lieu de compléter la première de ces complétions alternatives ; chaque appel ultérieur de la commande de complétion remplace celle de la prochaine complétion alternative, de manière cyclique. Si vous donnez à `completion-cycle-threshold` une valeur numérique

n, les commandes de complétion passeront au comportement cyclique seulement quand il aura n ou moins alternatives.

Le mode incomplet présente un affichage constamment mis à jour qui vous dit quelles complétions sont disponibles pour le texte que vous avez saisi jusque là. La commande pour activer ou désactiver ce mode mineur est `M-x incomplete-mode`.

5.5 Historique du minibuffer

Tous les arguments que vous entrez avec le mini-tampon sont enregistrés dans une liste d'historique du mini-tampon de sorte que vous pouvez facilement les utiliser plus tard. Vous pouvez utiliser les arguments suivants pour aller chercher rapidement un argument plus tôt dans le mini-tampon :

`M-p`
`Up` Déplacez à l'élément précédent dans l'historique du mini-tampon, un argument plus tôt (`previous-history-element`).

`M-n`
`Down` Déplacez à l'élément suivant dans l'historique du mini-tampon (`next-history-element`).

`M-r` *regexp* `RET` Déplacer vers un point plus tôt dans l'histoire du mini-tampon qui correspond à l'expression régulière (`previous-matching-history-element`)

`M-s` *regexp* `RET` Déplacer vers un point plus tard dans l'histoire du mini-tampon qui correspond à l'expression régulière (`next-matching-history-element`)

Alors que dans le mini-tampon, `M-p` ou `Up` (`previous-history-element`) se déplace à travers la liste d'historique du mini-tampon, un élément à la fois. Chaque `M-p` récupère un élément plus tôt dans la liste de l'historique du mini-tampon, remplaçant son contenu existant. Taper `M-n` ou `Down` (`next-history-element`) se déplace à travers la liste d'historique du mini-tampon dans la direction opposée, aller chercher les entrées plus tard dans le mini-tampon.

Si vous tapez `M-n` dans le mini-tampon lorsqu'il n'y a pas d'entrée plus tard dans l'historique du mini-tampon (par exemple, si vous n'avez pas déjà tapé `M-p`), Emacs essaie d'aller chercher dans une liste d'arguments par défaut : les valeurs que vous êtes susceptibles d'entrer. Vous pouvez penser à cela comme un déplacement à travers «le futur historique».

Si vous modifiez le texte inséré par les commandes d'historique du mini-tampon `M-p` ou `M-n`, cela ne change pas son entrée dans la liste de l'historique. Toutefois, l'argument édité va à la fin de l'historique lorsque vous le soumettez.

Vous pouvez utiliser `M-r` (`previous-matching-history-element`) pour chercher à travers les anciens éléments dans la liste de l'historique, et `M-s` (`next-matching-history-element`) pour chercher à travers les entrées plus récentes. Chacune de ces commandes demande une expression régulière comme un argument, et récupère la première entrée correspondante dans le mini-tampon.

Voir section 12.6 [Regexps], page 56, pour une explication des expressions régulières. Un argument numérique préfixé par n signifie chercher la n-ième entrée correspondante. Ces commandes sont inhabituelles, en ce qu'elles utilisent le mini-tampon pour lire l'argument de l'expression régulière, même si elles sont invoquées à partir du mini-tampon. Un lettre majuscule dans l'expression régulière rend la recherche sensible à la casse (voir section 12.9 [Recherche sensible à la casse], page 56.

Vous pouvez également rechercher à travers l'historique en utilisant une recherche incrémentale. Voir section 12.1.7 [Recherche incrémentale dans le mini-tampon], page 56.

Emacs conserve des listes d'historique séparées pour plusieurs types d'arguments. Par exemple, il y a une liste des noms de fichiers, utilisés par toutes les commandes qui lisent des noms de fichiers. Les autres listes d'historiques comprennent les noms de tampons, noms de commandes (utilisés par `M-x`), et les arguments de commande (utilisés par des commandes comme `query-replace`).

La variable `history-length` spécifie la longueur maximale d'une liste d'historique du mini-tampon ; l'ajout d'un nouvel élément supprime l'élément le plus ancien si la liste est trop longue. Si la valeur est `t`, il n'y a pas de longueur maximale.

La variable `history-delete-duplicates` indique s'il faut supprimer les doublons dans l'historique. S'il est non-`nil`, l'ajout d'un nouvel élément dans la liste supprime tous les autres éléments qui lui sont égaux. La valeur par défaut est `nil`.

5.6 Répétition des commandes du minibuffer

Chaque commande qui utilise le mini-tampon une fois est enregistrée sur une liste d'historique particulière, l'historique des commandes, ainsi que les valeurs de ses arguments, de sorte que vous pouvez répéter la commande entière. En particulier, chaque utilisation de `M-x` est enregistrée là, puisque `M-x` utilise le mini-tampon pour lire le nom de la commande.

`C-x ESC ESC`	Ré-exécute une commande de mini-tampon récente dans l'historique de commande (`repeat-complex-command`)
`M-x list-command-history`	Afficher tout l'historique de commande, montrant toutes les commandes `C-x ESC ESC` peuvent se répéter, les plus récentes apparaissant en premier.

`C-x ESC ESC` ré-exécute une commande récente qui a utilisé le mini-tampon. Sans argument, elle répète telle quelle la dernière commande. Un argument numérique spécifie quelle commande répéter ; 1 signifie le dernier, 2 le précédent, et ainsi de suite.

`C-x ESC ESC` fonctionne en tournant la commande précédente en une expression Lisp puis en entrant un mini-tampon initialisé avec le texte de cette expression. Même si vous ne connaissez pas Lisp, il sera probablement évident de voir quelle commande s'affiche pour la répétition. Si vous tapez juste `RET`, qui répète la commande inchangée. Vous pouvez également modifier la commande en modifiant l'expression Lisp avant de l'exécuter. La commande exécutée est ajoutée à l'avant de l'historique des commandes que si elle est identique à l'élément le plus récent.

Une fois à l'intérieur du mini-tampon pour `C-x ESC ESC`, vous pouvez utiliser les commandes habituelles d'historique du mini-tampon (voir section 5.5 [Historique du mini-tampon], page 32) pour vous déplacer dans la liste de l'histoire. Après avoir trouvé la commande précédente désirée, vous pouvez modifier son expression comme d'habitude, puis l'exécuter en tapant `RET`.

La recherche incrémentale, à proprement parler, n'utilise pas le mini-tampon. Par conséquent, même si elle se comporte comme une commande complexe, elle n'apparaît normalement pas dans la liste de l'historique pour `C-x ESC ESC`. Vous pouvez faire apparaître des commandes de recherche supplémentaires dans l'historique en configurant `isearch-resume-in-command-history` à une valeur non-`nil`. Voir section 12.1 [Recherche incrémentale], page 56.

La liste des commandes précédentes utilisant le mini-tampon est stockée comme une liste Lisp dans la variable `command-history`. Chaque élément est une expression Lisp qui décrit une commande et ses arguments. Les programmes Lisp peuvent ré-exécuter une commande en appelant `eval` avec l'élément `command-history`.

5.7 Création de mot de passe

Parfois, vous devrez peut-être un mot de passe dans Emacs. Par exemple, quand vous dites à Emacs de visiter un fichier sur une autre machine via un protocole FTP, vous avez souvent besoin de fournir un mot de passe pour accéder à la machine (voir section 15.13 [Fichiers distants], page 62).

La saisie d'un mot de passe est similaire à l'aide d'un mini-tampon. Emacs affiche une invite dans la zone écho (comme '`Password`') ; apès avoir tapé le mot de passe requis, appuyez sur `RET` pour le soumettre. Pour empêcher les autres de voir votre mot de passe, chaque caractère que vous tapez est affiché comme un point ('`.`') au lieu de sa forme habituelle.

La plupart des fonctions et commandes associées avec le mini-tampon ne peut être utilisé lors de la saisie d'un mot de passe. Il n'y a pas d'historique de la complétion, et vous ne pouvez pas changer de fenêtre ou effectuer toute autre action avec Emacs jusqu'à ce que vous ayez soumis le mot de passe.

Pendant que vous tapez le mot de passe, vous pouvez appuyer sur `DEL` pour effacer en arrière, enlever le dernier caractère saisi. `C-u` supprime tout ce que vous avez saisi jusqu'ici. `C-g` quitte l'invite de mot de passe (voir section 34.1 [Quitter], page 108). `C-y` insère la suppression courante dans le mot de passe (voir section 9 [Suppression], page 49). Vous pouvez taper soit `RET` soit `ESC` pour soumettre le mot de passe. Toute autre touche de caractère auto-insertion insère le caractère associé dans le mot de passe, et toute autre entrée est ignorée.

5.8 Oui ou Non du prompt (invite de commande)

Une commande Emacs peut vous demander de répondre à une question «oui ou non» au cours de son exécution. Ces requêtes sont disponibles en deux variétés principales.

Pour le premier type de requête «oui ou non», l'invite se termine par '`(y or n)`'. Une telle requête n'utilise pas le mini-tampon ; l'invite apparaît dans la zone écho, et que vous répondez en tapant '`y`' ou '`n`', qui fournit immédiatement la réponse. Par exemple, si vous tapez `C-x C-w` (`(write-file`) pour enregistrer un tampon, et entrez le nom d'un fichier existant, Emacs émet une invite comme ceci :

```
File 'foo.el' exists ; overwrite ? (y or n)
```

Parce que cette requête n'utilise pas le mini-tampon, les commandes d'édition usuelles du mini-tampon ne peuvent être utilisées. Toutefois, vous pouvez effectuer certaines opérations de défilement de fenêtre pendant que la requête est active : `C-l` recentre la fenêtre sélectionnée ; `M-v` (ou `PageDown` ou `next`) fait défiler vers l'avant ; `C-v` (ou `PageUp` ou `prior`) défile en arrière ; `C-M-v` défile en avant dans la prochaine fenêtre ; et `C-M-S-v` défile en arrière dans la fenêtre suivante. Taper `C-g` annule la requête, et quitte la commande qui l'a délivré (voir section 34.1[Quitter], page 108).

Le deuxième type de requête «oui ou non» est généralement utilisé si donner la mauvaise réponse aurait de graves conséquences ; il utilise le mini-tampon, et dispose d'une fin rapide avec '`(yes or no)`'. Par exemple, si vous invoquez `C-x k` (`kill-buffer`) sur un tampon de fichier visiter avec des modifications non enregistrées, Emacs active le mini-tampon avec une invite comme ceci :

```
Buffer foo.el modified ; kill anyway ? (yes or no)
```

Pour répondre, vous devez taper '`yes`' ou '`no`' dans le mini-tampon, suivi par `RET`. Le mini-tampon se comporte comme décrit dans les sections précédentes ; vous pouvez passer à une autre

fenêtre avec `C-x o`, utiliser l'historique des commandes `M-p` et `M-f`, etc. Tapez `C-g` quitte le mini-tampon et la commande d'interrogation.

Chapitre 6

Lancer des commandes par leur nom

Chaque commande Emacs a un nom que vous pouvez utiliser pour l'exécuter. Pour plus de commodité, de nombreuses commandes ont également des raccourcis clavier. Vous pouvez exécuter ces commandes en tapant ces touches, ou les exécuter par leurs noms. La plupart des commandes Emacs n'ont pas de raccourcis clavier, la seule façon de les lancer est par leurs noms. (Voir section 33.3 [Raccourcis clavier], page 106, pour savoir comment mettre en place les raccourcis clavier.).

Par convention, un nom de commande se compose d'un ou plusieurs mots, séparés par des tirets ; par exemple, `auto-fill-mode` ou `manual-entry`. Les noms de commandes utilisent le plus souvent des mots complets en anglais pour les rendre plus facile à retenir.

Pour exécuter une commande par son nom, commencez par `M-x`, tapez le nom de la commande, puis terminer par `RET`. `M-x` utilise le mini-tampon pour lire le nom de la commande. La chaîne '`M-x`' apparaît au début du mini-tampon comme invite pour vous rappeler d'entrer un nom de commande à exécuter. `RET` sort le mini-tampon et exécute la commande. Voir le chapitre 5 [Mini-tampon], page 25, pour plus d'informations sur le mini-tampon.

Vous pouvez utiliser la complétion pour entrer le nom de la commande. Par exemple, pour appeler la commande `forward-char`, vous pouvez taper `M-x forward-char RET` ou `M-x forw TAB c RET`.

Notez que `forward-char` est la même commande que vous invoquez avec la touche `C-f`. L'existence d'un raccourci ne vous empêche pas d'exécuter la commande par son nom.

Pour annuler le `M-x` et ne pas lancer une commande, tapez `C-g` lieu d'entrer le nom de la commande. Cela vous ramène au niveau de la commande.

Pour passer un argument numérique pour la commande que vous invoquez avec `M-x`, spécifiez l'argument numérique avant `M-x`. La valeur de l'argument apparaît dans l'invite alors que le nom de la commande est en cours de lecture, et enfin `M-x` passe l'argument à cette commande.

Lorsque la commande que vous exécutez avec `M-x` dispose d'un raccourci, Emacs le mentionne dans la zone écho après l'exécution de la commande. Par exemple, si vous tapez `M-x forward-word`, le message indique que vous pouvez exécuter la même commande en tapant `M-f`. Vous pouvez désactiver ces messages en définissant la variable `suggest-key-bindings` à `nil`.

Dans ce manuel, lorsque nous parlons de l'exécution d'une commande par son nom, nous omettons souvent le `RET` qui termine le nom. Ainsi, nous pourrions dire `M-x auto-fill-mode RET`. Nous mentionnons le `RET` seulement pour l'accent, comme lorsque la commande est suivie par des arguments.

`M-x` fonctionne en exécutant la commande `execute-extended-command`, qui est chargée de lire le nom d'une autre commande et l'invoquer.

Chapitre 7

Aide

Emacs fournit une grande variété de commandes d'aide, toutes accessibles par la touche de préfixe `C-h` (ou, de façon équivalente, la touche `F1`). Ces commandes d'aide sont décrites dans les sections suivantes. Vous pouvez également taper `C-h C-h` pour afficher une liste des commandes d'aide (`help-for-help`). Vous pouvez faire défiler la liste avec la `SPC` et de `DEL`, puis tapez la commande d'aide que vous voulez. Pour annuler taper `C-g`.

Beaucoup de commandes d'aide affichent leurs informations dans un tampon d'aide spécial. Dans ce tampon, vous pouvez taper `SPC` et `DEL` pour faire défiler et tapez `RET` pour suivre des liens hypertexte. Voir la section 7.4 [Mode aide], page 44.

Si vous cherchez une certaine fonction, mais ne savez pas comment elle s'appelle ou où chercher, nous vous recommandons trois méthodes. Premièrement, essayez une commande à propos, puis essayez de chercher dans l'index du Manuel, puis regardez dans les `FAQ` et les mots clés.

`C-h a topics RET`
: Cette fonction recherche des commandes dont les noms correspondent avec les arguments donnés. L'argument peut être un mot-clé, une liste de mots-clés, ou une expression régulière (voir Section 12.6 [Regexps], page 56. Voir la Section 7.3 [A propos], page 43.

`C-h i d m emacs RET i topic RET`
: Cette fonction recherche les sujets dans les indices du manuel Emacs Info, l'affichage de la première correspondance trouvée. Appuyez, pour voir les correspondances suivantes. Vous pouvez utiliser une expression régulière comme sujet.

`C-h i d m emacs RET s topic RET`
: Similaire, mais cherche le texte du manuel plutôt que les indices.

`C-h C-f`
: Affiche les FAQ Emacs, utilisant Info.

`C-h p`
: Affiche les paquetages Emacs disponibles basés sur le clavier. Voir Section 7.5 [Paquetages du clavier], page 45.

`C-h` ou `F1` signifie «aide» dans divers autres contextes. Par exemple, vous pouvez les saisir après une touche préfixée pour afficher une liste des touches qui peuvent suivre la touche préfixée. (Quelques touches préfixées ne supportent pas `C-h` de cette manière, car ils définissent d'autres significations pour ça, mais ils soutiennent tous `F1` pour l'aide.)

Voici un résumé des commandes d'aide pour accéder à la documentation intégrée. La plupart d'entre elles sont décrites en détail dans les sections suivantes.

`C-h a topics RET`
: Affiche une liste de commandes dont les noms correspondent aux sujets (`apropos-command`) avec les arguments donnés. L'argument peut être un mot-clé, une liste de mots-clés, ou une expression régulière (voir Section 12.6 [Regexps], page 56. Voir la Section 7.3 [A propos], page 43.

`C-h b`
: Affiche tous les raccourcis clavier actifs ; ceux des modes mineurs en premier, puis ceux des modes majeurs (`describe-bindings`).

`C-h c key`
: Affiche le nom de la commande associée au raccourci clavier (`describe-key-briefly`). Ici c est synonyme de «caractère». Pour des informations plus complètes sur la touche, utiliser `C-h k`.

`C-h d topics RET`
: Affiche les commandes et les variables dont la documentation correspond aux sujets (`apropos-documentation`).

`C-h e`
: Affiche le tampon `*Messages*` (`view-echo-area-messages`).

`C-h f function RET`
: La documentation de l'affichage de la fonction Lisp nommé (`describe-function`). Les commandes étant des fonctions Lisp, cela fonctionne pour les commandes aussi.

`C-h h`
: Affiche le fichier `HELLO`, qui montre des exemples de différents jeux de caractères.

`C-h i`
: Lance Info, le navigateur de documentation GNU (info). Le manuel Emacs est disponible dans Info.

`C-h k key`
: Affiche le nom et la documentation de la commande que ce raccourci lance (`describe-key`).

`C-h`
: Affiche le message d'aide pour une zone de texte spéciale, si le point est dans l'une d'elle (`display-local-help`). (Cela inclut, par exemple, les liens dans les tampons `*Help*`.)

`C-h w command RET`
: Montre quelles touches exécutent les commandes nommées *commande* (`where-is`).

`C-h C coding RET`
: Décrit le système d'encodage *coding* (`describe-coding-system`).

`C-h C RET`
: Décrit les systèmes d'encodage couramment utilisés.

`C-h l`
: Affiche une description des 300 dernières touches (`view-lossage`).

`C-h m`
: Affiche la documentation du mode majeur courant (`describe-mode`).

`C-h n`
: Affiche les changements récents dans Emacs (`view-emacs-news`).

`C-h p`
: Trouve les paquetages par sujet de raccourci (`finder-by-keyword`). Cela liste les paquetages utilisant un tampon de menu de paquetage. Voir Chapitre 32[Paquetages], page 103.

`C-h P package RET`
: Affiche la documentation à propos du paquetage nommé paquetage (`describe-package`).

`C-h r`
: Affiche le manuel Emacs dans Info (`info-emacs-manual`).

`C-h s`
: Affiche le contenu de la table de syntaxe courante (`describe-syntax`). La table de syntaxe dit quels caractères sont les délimiteurs ouvrant, quels sont ceux qui sont des parties de mots, et ainsi de suite. Voir Section "Tables de syntax" dans le *Emacs Lisp Reference Manual* pour plus détails.

`C-h t`
: Entre dans le tutoriel interactif de Emacs (`help-with-tutorial`).

`C-h v var RET`
: Affiche la documentation de la variable Lisp var (`describe-variable`).

`C-h F command RET`
: Lance Info et va au noeud correspondant à la commande Emacs *commande* (`Info-goto-emacs-command-node`).

`C-h I method RET`
: Décrit la méthode de saisie *method* (`describe-input-method`).

`C-h K key`
: Lance Info et va au noeud qui documente la séquence de touches *key* (`Info-goto-emacs-key-command-node`).

`C-h L language-env RET`
: Affiche des informations sur les ensembles de caractères, systèmes d'encodage, et méthode de saisie utilisées dans l'environnement de langage *language-env* (`describe-language-environment`).

`C-h S symbol RET`
: Affiche la documentation Info sur le symbole *symbol* en accord avec le langage de programmation que vous éditez (`info-lookup-symbol`).

7.1 Documentation pour une clé (touche)

Les commandes d'aide pour obtenir des informations sur une séquence de touches sont `C-h c` (`describe-key-briefly`) et `C-h k` (`describe-key`).

`C-h c` ***key*** affiche dans la zone de répercussion du nom de la commande qui est liée à la clé. Par exemple, `C-h c C-f` affiche '`forward-char`'.

`C-h k` ***key*** est similaire mais donne plus d'information : il affiche un tampon d'aide contenant la *chaîne de documentation* de la commande, qui décrit exactement ce que fait la commande.

`C-h K` ***key*** affiche la section du manuel Emacs qui décrit la commande correspondant à la clé.

`C-h c`, `C-h k` et `C-h K` travaillent pour toute sorte de séquences clés, y compris les touches de fonction, les menus et le événements de souris. Par exemple, après `C-h k` vous pouvez sélectionner un élément de menu dans la barre de menu, pour afficher la chaîne de documentation de la commande qui s'exécute.

`C-h w` ***command*** `RET` répertorie les touches qui sont liées à *command*. Il affiche la liste dans la zone d'écho. Si il dit que la commande n'est sur aucune touche, ça signifie que vous devez utiliser `M-x` pour la lancer. `C-h w` lance la commande `where-is`.

7.2 Aide par commande ou par nom de variable

`C-h f` ***function*** `RET (describe-function)` affiche la documentation de la fonction Lisp *function*, dans une fenêtre. Puisque les commandes sont des fonctions Lisp, vous pouvez utiliser cette méthode pour voir la documentation de n'importe qu'elle commande dont vous connaissez le nom. Par exemple,

`C-h f auto-fill-mode RET`

affiche la documentation de `auto-fill-mode`. C'est le seul moyen d'obtenir la documentation d'une commande qui n'est liée à aucune touche (une que vous lanceriez normalement en utilisant `M-x`).

`C-h f` est aussi utile pour les fonctions Lisp que vous utilisez dans un programme Lisp. Par exemple, si vous venez d'écrire l'expression `(make-vector len)` et souhaitez vérifier que vous utilisez `make-vector` proprement, tapez `C-h f make-vector RET`. Puisque `C-h f` autorise tous les noms de fonctions, pas seulement les noms de commandes, vous pouvez trouver certaines de vos complétions d'abréviations favorites qui fonctionnent avec `M-x` mais pas avec `C-h f`. Une abbréviation qui est unique parmi les noms de commandes peut ne pas être unique parmi tous les noms de fonctions.

Si vous tapez `C-h RET`, cela décrit la fonction appelée l'expression Lisp la plus intérieure dans le tampon autour du point, *sous réserve* que le nom de fonction soit valide, et la fonction Lisp définie. (Ce nom apparaît par défaut lorsque vous entrez l'argument.) Par exemple, si le point est placé après le texte «`(make-vector (car x)`», la liste la plus intérieure contenant le point est celle qui commence avec «`(make-vector`», donc `C-h f RET` décrit la fonction `make-vector`.

`C-h f` est aussi utile pour vérifier que vous avez écrit correctement un nom de fonction. Si l'invite du min-tampon pour `C-h f` montre le nom de fonction depuis le tampon par défaut, cela signifie que ce nom est définit comme une fonction Lisp. Tapez `C-g` pour annuler la commande `C-h f` si vous ne voulez pas vraiment voir la documentation.

`C-h v (describe-variable)` est comme `C-h f` mais décrit des variables Lisp au lieu de fonctions Lisp. Par défaut c'est le symbole Lisp autour ou avant le point, si c'est le nom d'une variable Lisp définie. Voir Section 33.2 [Variables], page 106.

Les tampons d'aide qui décrivent les variables et les functions Emacs ont normalement des hyperliens vers le code source correspondant, si vous avez installés les fichiers sources (voir Section

31.11 [Hyperliens], page 102).

Pour trouver une documentation de commande dans un manuel, utilisez `C-h F (Info-goto-emacs-command-node)`. Cela renseigne sur des manuels variés, pas seulement le manuel Emacs, cela trouve celui approprié.

7.3 Apropos

Les commandes *apropos* répondent aux questions comme, «Quelles sont les commandes pour travailler avec les fichiers ?» Plus précisément, vous spécifiez un *modèle apropos* dans le mini-tampon, ce qui signifie soit un mot, une liste de mots, ou une expression régulière.

Chacune des commandes apropos suit un modèle apropos dans le mini-tampon, recherche les éléments qui correspondent au modèle, et affiche les résultats dans une fenêtre différente.

`C-h a`
: Recherche de commandes `(apropos-command)`. Avec un argument préfixé, la recherche de fonctions non interactives aussi.

`M-x apropos`
: Recherche pour les fonctions et variables. Autant les fonction interactives (commandes) que pour les fonctions non interactives peuvent être trouvées avec ça.

`M-x apropos-user-option`
: Recherche pour les variables personnalisables par l'utilisateur. Avec un argument préfixé, recherche des variables non personnalisables aussi.

`M-x apropos-variable`
: Recherche pour les variables. Avec un argument préfixé, la recherche pour les variables personnalisables seulement.

`M-x apropos-value`
: Recherche pour les variables dont les valeurs correspondent au modèle spécifié. Avec un argument préfixé, consulte également les fonctions avec les définitions correspondant au modèle, et les symboles Lisp avec des propriétés correspondant au modèle.

`C-h d`
: Recherche pour les fonctions et les variables dont les chaînes de documentation correspondent au modèle spécifié `(apropos-documentation)`.

Le type le plus simple de modèle est un mot. N'importe quoi contenant ce mot correspond au modèle. Ainsi, pour trouver les commandes qui travaillent sur des fichiers, tapez `C-h a file RET`. Cela affiche une liste de tous les noms de commandes qui contiennent `'file'` y compris `copy-file, find-file`, et ainsi de suite. Chaque nom de commande vient avec une brève description et une liste de touche (ou combinaison de touches) que vous pouvez invoquer. Dans notre exemple, ce serait dire que vous pouvez invoquer `find-file` en tapant `C-x C-f`.

Pour plus d'information à propos d'une définition de fonction, d'une propriété d'une variable ou d'un symbole listé dans un tampon apropos, vous pouvez cliquer sur `Mouse-1` ou `Mouse-2`, ou vous déplacer là-bas et taper `RET`.

Quand vous spécifiez plus d'un mot dans le chemin apropos, un nom doit contenir au moins deux des mots afin de correspondre. Ainsi, si vous cherchez pour supprimer du texte avant le point, vous pouvez essayer `C-h a kill back backward behind before RET`. Le vrai nom de la commande `kill-backward` correspondra avec ; s'il y avait une `kill-text-before`, cela aurait aussi correspondu, puisqu'il contient deux des mots spécifiés.

Pour encore plus de flexibilité, vous pouvez spécifier une expression régulière (voir Section 12.6 [Regexps], page 56). Un chemin apropos est interprété comme une expression régulière si il contient l'un (au moins) des caractères spéciaux, `'$*+?.\['`.

Suivants les conventions de dénomination des commandes Emacs, voici quelques mots que vous pourrez trouver utiles dans les chemins apropos. En les utilisant dans `C-h a`, vous obtiendrez aussi une sensation pour les conventions de nommage.

char, line, word, sentence, paragraph, region, page, sexp, list, defun, rect, buffer, frame, window, face, file, dir, register, mode, beginning, end, forward, backward, next, previous, up, down, search, goto, kill, delete, mark, insert, yank, fill, indent, case, change, set, what, list, find, view, describe, default.

Si la variable `apropos-do-all` est non-`nil`, les commandes apropos se comportent toujours comme si elles étaient données en argument préfixé.

Par défaut, toutes les commandes apropos sauf `apropos-documentation` liste leur résultats par ordre alphabétique. Si la variable `apropos-sort-by-scores` est non-`nil`, alors les commandes essaient de deviner la pertinence de chaque résultat, et affichent les résultats les plus pertinents en premier. La commande `apropos-documentation` liste les résultats par ordre de pertinence par défaut ; pour les lister par ordre alphabétique, changer la variable `apropos-documentation-sort-by-scores` à `nil`.

7.4 Commandes du mode aide

Les tampons d'aide fournissent les mêmes commandes que le View mode (voir Section 11.6 [View Mode], page 54) ; par exemple, `SPC` défile vers l'avant, et `DEL` ou `S-SPC` défile vers l'arrière. Quelques commandes spéciales sont également fournies :

`RET`	Suivre une référence croisée au point `(help-follow)`.
`TAB`	Déplace le point vers l'avant jusqu'au prochain lien `(forward-button)`.
`S-TAB`	Déplace le point vers l'arrière jusqu'au lien précédent `(backward-button)`.
`Mouse-1`	
`Mouse-2`	Suis un lien sur lequel vous avez cliqué.
`C-c C-c`	Montre toute la documentation à propos du symbole au point `(help-follow-symbol)`.
`C-c C-b`	Retourne au sujet précédent `(help-go-back)`.

Lorsqu'un nom de fonction, un nom de variable, ou un nom de face (voir Section 11.8 [Faces], page 54) apparaît dans la documentation dans le tampon d'aide, il est normalement un lien souligné. Pour voir la documentation associée, déplacez le point à cet endroit et tapez `RET (help-follow)`, ou cliquez sur le lien avec `Mouse-1` ou `Mouse-2`. Procédant ainsi remplace le contenu du tampon d'aide ; pour revenir sur vos pas, tapez `C-c C-b (help-go-back)`.

Un tampon d'aide peut aussi contenir des liens des manuels Info, des codes sources de définitions, et des URLs (pages web). Le premier des deux est ouvert dans Emacs, et le troisième

utilise un navigateur web via la commande `browse-url` (voir Section 31.11.1 [Browse-URL], page 102).

Dans un tampon d'aide, `TAB (forward-button)` déplace le point vers l'avant jusqu'au prochain lien tandis que `S-TAB (backward-button)` pointe vers le lien précédent. Ces commandes sont cycliques ; par exemple, taper `TAB` au dernier lien renvoie au premier lien.

Pour voir toute la documentation au sujet d'un symbole dans le texte, déplacez le point à cet endroit et tapez `C-c C-c (help-follow-symbol)`. Cela montre toute la documentation disponible au sujet du symbole—comme une variable, fonction et/ou face.

7.5 Mot-clé pour la recherche d'extension

7.6 Aide pour les langues internationales

7.7 Autres commandes d'aide

7.8 Fichiers d'aide

7.9 Aide dans du texte actif et infobulles

Chapitre 8

Sélection et région

Chapitre 9

Couper et déplacer du texte

9.1 Suppression et coupure

9.1.1 Suppression

9.1.2 Coupure par ligne

9.1.3 Autres commandes de coupure

9.1.4 Options de coupure

9.2 Collage

9.2.1 L'anneau de coupure

9.2.2 Coller les coupures précédentes

9.2.3 Ajout de coupures

9.3 «Couper et coller» opération sur l'affichage graphique

9.3.1 Usage du clippboard (menu avec boutons)

9.3.2 Couper et coller avec d'autres applications fenêtre

9.3.3 Sélection secondaire

9.4 Accumulation de texte

9.5 Rectangles

9.6 CUA raccourcis

Chapitre 10

Enregistrements

Chapitre 11

Contrôle de l'affichage

Chapitre 12

Recherche et remplacement

Chapitre 13

Commandes pour corriger les coquilles

13.1 Undo (annulation de la dernière action)

13.2 Transposition de texte

13.3 Conversion de casse

13.4 Détecter et corriger l'orthographe

Chapitre 14

Macros clavier ou raccourci

Chapitre 15

Traitement de fichier

15.1 Noms de fichier

15.2 Visionnage de fichiers

15.3 Sauvegarde de fichiers

15.3.1 Commandes de sauvergardes des fichiers

15.3.2 Fichiers de sauvegardes

Sauvegardes uniques ou numérotées

Destruction automatique des sauvegardes

Copie vs. renommage

15.3.3 Personnalisation de la sauvegarde de fichier

15.3.4 Protection contre l'édition simultanée

15.3.5 Fichiers cachés

15.3.6 Mise à jour automatique de l'heure

15.4 Retour vers un buffer

15.5 Sauvegarde automatique : protection contre les désastres

15.5.1 Sauvegarde automatique des fichiers

15.5.2 Contrôle de la sauvegarde automatique

15.5.3 Récupérer des données à partir des sauvegardes automatiques

15.6 Alias

15.7 Répertoires

15.8 Comparaison de fichiers

15.9 Mode diff

15.10 Opérations diverses sur les fichiers

15.11 Accéder à des fichiers compressés

Chapitre 16

Utilisation de multiple buffers (ou tampons)

Chapitre 17

Fenêtrage multiple

Chapitre 18

Diapositives et affichages graphique

Chapitre 19

Caractères internationaux et paramétrages

Emacs supporte une grande variété de jeux de caractères internationaux, y compris les variantes européennes et vietnamiennes de l'alphabet latin, ainsi que le cyrillique, devanagari (pour l'hindi et le marathi), éthiopien, grec, Han (pour le chinois et le japonais), Hangul (pour le coréen, l'hébreu, l'IPA, le kannada, le Laos, le malayalam, tamoul, thaï, tibétain, et vietnamiens. Emacs prend également en charge divers encodages de ces caractères qui sont utilisés par un autre logiciel internationalisé, tels que traitement de texte et assistant de courriels.

Emacs permet l'édition de texte avec des caractères internationaux en soutenant toutes les activités liées :

- Vous pouvez visiter les fichiers avec des caractères non-ASCII, enregistrer du texte non-ASCII, et passer du texte non-ASCII entre Emacs et des programmes qui l'invoquent (tels que des compilateurs, des correcteurs orthographiques, et des assistants de courriels). La configuration de votre environnement de langue (vois Section 19.3 [Environnements de langue], page 72) prend soin de mettre en place les systèmes d'encodage et d'autres options pour une langue ou une culture. Alternativement, vous pouvez spécifier comment Emacs devrait encoder ou décoder un texte pour chaque commande ; voir Section 19.10 [Encodage de texte], page 72.

- Vous pouvez afficher les caractères non-ASCII encodés par les différents scripts. Cela fonctionne en utilisant des polices appropriées sur les écrans graphiques (voir Section 19.15 [Définition des jeux de polices], page 72), et en envoyant des codes spéciaux pour les écrans de texte (voir Section 19.13 [Encodage des terminaux], page 72). Si certains caractères ne s'affichent pas correctement, reportez-vous à la Section 19.17 [Caractères non-affichables], page 72, qui décrit les problèmes possibles et explique comment les résoudre.

- Les caractères des scripts qui sont naturellement de droite à gauche sont réorganisés pour l'affichage (voir Section **??** [Edition bidirectionnelle], page **??**). Ces scripts comprennent l'arabe, l'hébreu, le syriaque, le Thaana, et quelques autres.

- Vous pouvez insérer des caractères non-ASCII ou en chercher. Pour ce faire, vous permettant de spécifier une méthode d'entrée (voir Section 19.5 [Sélectionnez la méthode d'entrée], page 72) adapté à votre langue, ou utilisez la méthode d'entrée par défaut mise en place lorsque vous avez choisi votre environnement de langue. Si votre clavier peut produire des caractères non-ASCII, vous pouvez sélectionner un système d'encodage du clavier

approprié (voir Section 19.13 [Encodage des terminaux], page 72), et Emacs acceptera ces caractères. Les caractères Latin-1 peuvent également être entré en utilisant le préfixe `C-x`, voir la Section 19.18 [Mode uni-octet], page 72.

Avec le système X Window, vos paramètres régionaux doivent être réglé sur une valeur appropriée pour s'assurer qu'Emacs interprète correctement les entrées au clavier ; voir Section 19.3 [Environnements de langue], page 72.

Le reste de ce chapitre décrit ces questions en détail.

19.1 Introduction des ensembles de caractères internationaux

Les utilisateurs des jeux de caractères internationaux ont établi beaucoup de standard de systèmes d'encodage pour stocker les fichiers. Ces systèmes d'encodage sont généralement multi-octets, ce qui signifie que les suites de deux ou plusieurs octets sont utilisées pour représenter les caractères individuels non-ASCII.

En interne, Emacs utilise son propre encodage de caractères multi-octets, qui est un sur-ensemble de la norme *Unicode*. Cet encodage interne permet des caractères de presque chaque script connu d'être mélangés dans un seul tampon ou dans une chaîne de caractères. Emacs traduit entre l'encodage multi-octets et divers autres de systèmes d'encodage lors de la lecture et de l'écriture de fichiers, et quand il échange des données avec les sous-processus.

La commande `C-h h (view-hello-file)` affiche le fichier `etc/HELLO`, qui illustre divers scripts montrant comment dire «bonjour» dans de nombreuses langues. Si certains caractères ne peuvent être affihcés sur votre terminal, ils apparaissent comme '?' ou comme caissons creux (voir Section 19.17 [Caractères non affichables], page 72).

Les claviers, même dans les pays où ces jeux de caractères sont utilisés, n'ont généralement pas les touches pour tous les caractères. Vous pouvez insérer des caractères que votre clavier ne prend pas en charge, en utilisant `C-q (quoted-insert)` ou `C-x 8 RET (insert-char)`. Voir Section 4.1 [Insertion de texte], page 15. Emacs prend également en charge diverse méthodes de saisie, généralement une pour chaque script ou langue, ce qui rend plus facile de taper des caractères dans le script. Voir la Section 19.4 [Les méthodes de saisie], page 72.

La touche préfixée `C-x RET` est utilisée pour les commandes qui se rapportent aux caractères multi-octets, systèmes d'encodage, et les méthodes d'entrée.

La commande `C-x = (what-cursor-position)` renseigne sur le caractère au point. En plus de la position du caractère, qui a été décrit dans la Section 4.9 [Infos sur la position], page 22, cette commande affiche comment le caractère est codé. Par exemple, il affiche la ligne suivante dans la zone d'écho pour le caractère «c» :

```
Char : c (99, #o143, #x63) point=28062 of 36168 (78%) column=53
```

Les quatre valeurs après «`Char :`» décrivent le caractère qui suit le point, d'abord en montrant, puis en donnant son code de caractère en décimal, octal et hexadécimal. Pour un caractère multi-octets non ASCII, ceux-ci sont suivis par «`file`» et la représentation du caractère, en hexadécimal, dans le système d'encodage du tampon, si ce système d'encodage code le caractère en toute sécurité avec un seul octet (voir Section 19.6 [Système d'encodage], à la page 72. Si l'encodage du caractère est plus d'un octet, Emacs montre «`file …`»

Comme cas particulier, si le caractère se trouve dans la plage de 128 (0200 en octal) à 159 (0237 en octal), cela signifie un octet «brut» qui ne correspond pas à n'importe quel caractère spécial affichable. Un tel «caractère» se trouve dans le jeu de caractère `eight-bit-control` et

est affiché comme un code de caractère octal échappé. Dans ce cas, `C-x =` montre «`part of display …`» au lieu de «`file`».

Avec un argument préfixé (`C-u C-x =`), cette commande affiche une description détaillée du caractère dans une fenêtre :

- Le nom du jeu de caractères et les codes qui identifient le caractère dans ce jeu de caractères ; les caractères ASCII sont identifiés comme appartenant à l'ensemble de caractères `ascii`.
- La syntaxe et les catégories du caractère.
- Les encodages du caractère, tant à l'interne dans le tampon, qu'à l'extérieur si vous deviez enregistrer le fichier.
- Quelles touches taper pour entrer le caractère de la méthode d'entrée naturelle (si elle prend en charge le caractère).
- Si vous utilisez Emacs avec un affichage graphique, le nom de police et le code de glyphe pour le caractère. Si vous lancez Emacs dans un terminal de texte, le(s) code(s) est(sont) envoyé(s) au terminal.
- Les propriétés du caractère de texte (voir la Section «Text Properties» in the Emacs Lisp Reference Manual), y compris les faces non-défaut utiisées pour afficher le caractère, et les incrustations en contenant (voir la section «overlays» dans le même manuel).

Voici un exemple montrant le caractère Latin-1 A avec accent grave, dans un tampon dont le système d'encodage est `utf-8-unix` :

```
            position :  1 of 1 (0%), column : 0
           character :  À (displayed as À) (codepoint 192, #o300, #xc0)
     prefered charset :  unicode (Unicode ISO10646))
code point in charset :  0xC0
              syntax :  w   wihich means : word
            category :  . :Base, L : Left-to-right (strong),
                        j :Japanese, l :Latin, v :Viet
         buffer code :  #xC3 #x80
           file code :  not encodable by coding system undecided-unix
             display :  by this font (glyph code)
               xft :-unknown-DejaVu Sans Mono-normal-normal-
                  normal-*-13-*-*-*-m-0-iso10646-1 (#x82)

            Character code properties : customize what to show
                 name : LATIN CAPITAL LETTER A WITH GRAVE
                 old-name : LATIN CAPITAL LETTER A GRAVE
                general-category : Lu (Letter, Uppercase)
                   decomposition : (65 768) ('A' '`')
```

Chapitre 20

Modes majeur et mineur

20.1 Modes majeur

20.2 Modes mineur

20.3 Choix des modes de fichier

Chapitre 21

Indentation

Chapitre 22

Commandes pour les langues humaines

Ce chapitre décrit les commandes Emacs qui agissent sur le texte, nous entendons par là des séquences de caractères dans un langage humain (par opposition à, disons, un langage de programmation informatique). Ces commandes agissent de façon à prendre en compte les conventions syntaxiques et stylistiques de langues humaines : conventions impliquant des mots, des phrases, des paragraphes, et les lettres majuscules. Il y a aussi des commandes pour le remplissage, ce qui signifie réorganisation des lignes d'un paragraphe à environ la même longueur. Ces commandes, tandis que destinées principalement pour l'édition de texte, sont aussi souvent utile pour les éditions de programmes.

Emacs dispose de plusieurs modes majeurs pour l'édition de texte de langue humaine. Si le fichier contient du texte ordinaire, utilisez le mode texte, qui personnalise Emacs dans de petits moyens pour les conventions syntaxiques de texte. Le mode Contour fournit des commandes spéciales pour opérer sur le texte avec une structure hiérarchique. Le mode Org étend le mode Contour et transforme Emacs en organiseur à part entière : vous pouvez gérer des listes à faire, stocker des notes et les publier dans de nombreux formats.

Voir section 22.8 [Mode Contour], page 79.

Emacs a d'autres modes majeurs pour le texte qui contient des commandes «embarquées», comme TEX ou LATEX (voir section 22.10[Mode TEX] page 79) ; HTML et SGML (voir section 22.11 [Mode HTML], page 84) ; XML (voir le nXML mode Info manuel, qui est distribué avec Emacs) ; et Groff et Nroff (voir section 22.12 [Mode nroff], page 84).

Si vous avez besoin d'éditer des photos fabriquées à partir de caractères de texte (communément appelé «l'art ASCII»), utilisez le mode d'image, un mode majeur pour l'édition spéciale de telles images. Voir la section “Picture Mode” dans *Specialized Emacs Features*.

22.1 Mots

Emacs définit plusieurs commandes pour déplacer ou opérer sur les mots :

`M-f`	avance d'un mot (`forward-word`)
`M-b`	recule d'un mot (`backward-word`)
`M-d`	supprime jusqu'à la fin d'un mot (`kill-word`)
`M-DEL`	supprime jusqu'au début du mot (`backward-kill-word`)
`M-@`	marque la fin du prochain mot (`mark-word`)
`M-t`	transpose deux mots ou fait glisser un mot dans d'autres (`transpose-words`).

Remarquez comment ces touches forment une série qui est parallèle à la base de caractère `C-f`, `C-b`, `C-d`, `DEL` et `C-t`. `M-@` est apparenté à `C-@`, qui est un alias pour `C-SPC`.

Les commandes `M-f` (`forward-word`) et `M-b` (`backward-word`) se déplacent vers l'avant et vers l'arrière sur les mots. Ces séquences de touches à base de `Meta` sont analogues aux séquences de touches `C-f` et `C-b`, qui se déplacent sur des caractères simples. L'analogie s'étend aux arguments numériques, qui servent de nombre de répétions. `M-f` avec un argument négatif déplace vers l'arrière, et `M-o` avec un argument négatif déplace vers l'avant. Le mouvement vers l'avant s'arrête juste après la dernière lettre du mot, tout mouvement vers l'arrière s'arrête juste avant la première lettre.

`M-d` (`kill-word`) efface le mot après le point. Pour être précis, il tue tout du point jusqu'à l'endroit où `M-f` se déplacerait. Ainsi, si le point est au milieu d'un mot, `M-d` tue juste la partie après le point. Si certains signe de ponctuation viennent entre le point et le mot suivant, ils sont tués avec le mot. (Si vous souhaitez seuelement tuer le mot suivant, mais pas les signes de ponctuation avant, il suffit de faire `M-f` pour obtenir la fin, et de tuer le mot à l'envers avec `M-DEL`.) `M-d` prend des arguments comme `M-f`.

`M-DEL` (`backward-kill-word`) tue le mot avant le point. Il tue tout du point de retour à l'endroit où `M-b` serait déplacé. Par exemple, si le point est après l'espace en '`FOO, BAR`', il tue '`FOO, `'. Si vous souhaitez simplement tuer '`FOO`', et non la virgule et l'espace, utilisez `M-b M-d` à la place de `M-DEL`.

`M-t` (`transpose-words`) échange le mot avant ou contenant le point avec le mot suivant. Les caractères de délimitation entre les mots ne se déplacent pas. Par exemple, '`FOO, BAR`' se transpose en '`BAR, FOO`' plutôt que '`BAR FOO, `'. Voir la section 13.2 [Transposition], page 57, pour plus de détails sur la transposition.

Pour opérer sur les mots avec une opération qui agit sur la région, utiliser la commande `M-@` (`mark-word`). Cette commande définit la marque où `M-f` serait déplacé. Voir la section 8.2 [Marquage des objets], page 47, pour plus d'informations sur cette commande.

La compréhension des commandes de mots des limites de mots est contrôlée par la table de syntaxe. Tout caractère peut, par exemple, être déclaré un mot délimiteur. Voir la section «Tables de syntaxe» dans le manuel Emacs Lisp référence.

En outre, voir la section 4.9 [Infos de position], page 22 pour les commandes `M-=` (`count-words-region`) et `M-x count-words`, qui comptent et signalent le nombre de mots dans la région ou le tampon.

22.2 Phrases

22.3 Paragraphes

22.4 Pages

22.5 Remplissement du texte

22.5.1 Mode auto-remplissage

22.5.2 Commandes explicites de remplissage

22.5.3 Préfixe de remplissage

22.5.4 Remplissage adaptable

22.6 Commandes de conversion de casse

22.7 Mode texte

22.8 Mode Contour

22.8.1 Format hors-ligne

22.8.2 Commandes de contour

22.8.3 Commandes de visibilité hors-ligne

22.8.4 Vue hors-ligne en fenêtre multiple

22.8.5 Édition de pilage

22.9 Mode org

22.9.1 Org comme organisateur

22.9.2 Org comme système d'autorisation

22.10 Mode TEX

Emacs fournit les principaux modes spéciaux pour l'édition de fichiers écrits en TEX et ses formats connexes. TEX est un formateur de texte puissant écrit par Donald Knuth ; comme GNU Emacs, il est un logiciel libre. LATEX est un format d'entrée simplifiée pour TEX, mis en oeuvre en utilisant des macros de TEX. DocTEX est un format de fichier spécial dans lequel les sources LATEX sont écrits, combinant sources avec la documentation. SliTEX est une forme spéciale obsolète de LATEX[1].

Le mode TEX a quatres variantes : Plain TEX mode, LATEX mode, DocTEX, et SliTEX mode. Ces modes majeurs distincts ne diffèrent que légèrement, et sont conçus pour l'édition des quatre

[1] Elle a été remplacé par la classe de document '`slides`', qui vient avec LATEX

formats différents. Emacs sélectionne le mode approprié en examinant le contenu de la mémoire tampon. (Cela se fait par la commande `tex-mode`, qui est normalement appelée automatiquement lorsque vous visitez un fichier TEX-like. Voir section 20.3 [Choisir un mode], page 73.) Si les contenus sont insuffisants pour déterminer, Emacs choisit le mode spécifié par la variable `tex-default-mode` ; sa valeur par défaut est le `latex-mode`. Si Emacs ne devine pas correctement, vous pouvez sélectionner la variante correcte de TEX en utilisant la commande `M-x plain-tex-mode`, `M-x latex-mode`, `M-x slitex-mode`, ou `doctex-mode`.

Les sections suivantes documentent les caractéristiques du mode TEX et de ses variantes. Il y a plusieurs paquets Emacs relatifs à TEX, qui ne sont pas documentés dans ce manuel :

- BibTEX mode est un mode majeur pour les fichiers BibTEX, qui sont couramment utilisés pour garder les références bibliographiques des documents LATEX. Pour plus d'informations, voir la chaîne de document pour `bibtex-mode`

- Le paquet RefTEX fournit un mode mineur qui peut être utilisé avec le mode LATEX pour gérer les références bibliographiques. Pour plus d'informations, consultez le manuel RefTEX Info, qui est distribué avec Emacs.

- Le paquet AUCTEX fournit plus de fonctions avancées pour éditer du TEX et ses formats relatifs, incluant la capacité de prévisualiser les équations TEX dans les tampons Emacs. Contrairement au mode BibTEX et le paquet RefTEX, AUCTEX n'est pas distribué avec Emacs par défaut. Il peut être téléchargé via le menu des paquets (voir le chapitre 32 [Paquets], page 103) ; une fois installé, consultez le manuel de AUCTEX, qui est inclus dans le paquet.

22.10.1 Édition de commandes TEX

`"`	insère, selon le contexte, soit '“' soit '”' ou '”' (`tex-insert-quote`).
`C-j`	insère une coupure de paragraphe (deux nouvelles lignes) et vérifie les équilibres d'accolades ou de signe dollar (`tex-terminate-paragraph`).
`M-x tex-validate-region`	vérifie chaque paragraphe dans la région pour des accolades asymétriques ou signes de dollar.
`C-c {`	insère '{}' et le point entre elles (`tex-insert-braces`)
`C-c }`	avance passé la prochaine accolade fermante inégalée (`up-list`).

Dans TEX, le caractère '“' n'est normalement pas utilisé ; mais à la place, les citations commencent avec '“' et finissent par '”'. Le mode TEX lie donc la touche `` ` `` à la commande `tex-insert-quote`. Cela insère “ “ après un espace ou une accolade ouverte, '“' après une barre oblique inverse, “ “ après tout caractère.

Comme une exception, si vous tapez “ lorsque le texte avant le point est soit “ “ ou “ “, Emacs remplace celle qui précède le texte avec un seul '“' caractère. Vous pouvez donc taper “ “ pour insérer '“', si jamais vous avez besoin de le faire. (Vous pouvez également utiliser `C-q` “ pour insérer ce caractère.)

En mode TEX, '$' a un code de syntaxe spécial qui tente de comprendre la façon dont le mode mathématique de TEX délimite les correspondances. Lorsque vous insérez un '$', qui est destiné à quitter le mode de calcul, la position de l'appariement '$', qui est entré en mode mathématique est affiché pendant une seconde. C'est la même fonctionnalité qui affiche l'accolade ouverte qui correspond à une accolade fermante qui est inséré. Toutefois, il est impossible de dire si un '$' passe en mode maths ou le quitte ; ainsi, lorsque vous insérez un '$' qui entre dans le mode

maths, le précédent «$» est présenté comme si c'était un appariemement, même si elles sont effectivement indépendantes.

TEX utilise des accolades comme délimiteurs qui doivent correspondre. Certains utilisateur préfèrent garder les accolades équilibrées en tout temps, plutôt que de les insérer individuellement. Utiliser `C-c {` (`tex-insert-braces`) pour insérer une paire d'accolades. Il laisse le point entre les deux accolades de sorte que vous pouvez insérer le texte qui appartient à l'intérieur. Ensuite, utilisez la commande `C-c }` (`up-list`) pour avancer au-delà du accolade fermante.

Il y a deux commandes de contrôle de la correspondance des accolades. `C-j` (`tex-terminate-paragraph`) vérifie le paragraphe avant le point, et insère deux nouvelles lignes pour commencer un nouveau paragraphe. Il affiche un message dans la zone écho le cas échéant si une discordance est trouvée. `M-x tex-validate-region` vérifie une région, paragraphe par paragraphe. Les erreurs sont répertoriées dans un tampon *Occur* ; vous pouvez utiliser les commandes habituelles dans ce tampon, comme `C-c C-c` pour visiter un décalage particulier (voir section 12.11 [Autres recherche répétition], page 56).

Notez que les commandes Emacs comptent les crochets et les parenthèses en mode TEX, pas seulement les accolades. Ce n'est pas strictement exact à des fins de vérification de syntaxe TEX. Cependant, les parenthèses et crochets sont susceptibles d'être utilisés dans le texte comme des séparateurs correspondant, et il est utile pour les différentes commandes de mouvement et affichage automatique de correspondance de travailler avec eux.

22.10.2 Édition de commandes LATEX

Le mode LATEX offre quelques fonctionnalités supplémentaires non applicables aux plain TEX :

`C-c C-o` insérez '`\begin`' et '`\end`' pour le bloc de LATEX et point de position sur une ligne entre eux (`tex-latex-bloc`)

`C-c C-e` ferme le bloc LATEX le plus encapsulé pas encre fermé (`tex-close-latex-block`).

En entrée de LATEX, les étiquettes '`\begin`' et '`\end`', laissant une ligne blanche entre les deux et le point de s'y installer.

Lors de la saisie de l'argument de type bloc `C-c C-o`, vous pouvez utiliser les commandes de complétion habituelles (voir section 5.4 [Complétion], page 27). La liste d'achèvement par défaut contient les types de blocs de LATEX standard. Si vous voulez des blocs supplémentaires pour l'achèvement, personnalisez la variable liste `latex-block-names`.

En entrée de LATEX, les étiquettes '`\begin`' et '`\end`', doivent s'équilibrer. Vous pouvez utiliser `C-c C-e` (`tex-close-latex-block`) pour insérer une balise '`\end`' qui correspond à la dernière inégalée '`\begin`'. Il décale aussi le '`\end`' pour correspondre à la balise '`\begin`', et insère un saut de ligne après la balise '`\end`' si les point est au début de la ligne. Le mode mineur `latex-electric-env-pair-mode` insère automatiquement un '`\end`' ou '`\begin`' pour vous lorsque vous tapez celui correspondant.

22.10.3 Impression de commandes TEX

Vous pouvez appeler TEX comme un sous-processus de Emacs, soit fournir tout le contenu de la mémoire tampon ou seulement une parte de celle-ci (par exemple, un chapitre d'un document important).

`C-c C-b` invoque TEX dans le tampon courant dans son entier (`tex-buffer`).
`C-c C-r` invoque TEX dans la région courante, avec l'en-tête du tampon (`tex-region`).
`C-c C-f` invoque TEX dans le fichier courant (`tex-file`).
`C-c C-v` prévisualise la sortie depuis la dernière commande `C-c C-r`, `C-c C-b`, ou `C-c C-f` (`tex-view`).
`C-c C-p` imprime la sortie depuis la dernière commande `C-c C-r`, `C-c C-b`, ou `C-c C-f` (`tex-print`).
`C-c TAB` invoque BibTEX sur le fichier courant (`tex-bibtex-file`).
`C-c C-l` recentre la fenêtre d'affichage de sortie depuis TEX de sorte que la dernière ligne puisse être vue (`tex-recenter-output-buffer`).
`C-c C-k` tue les sous-processus TEX (`tex-kill-job`).
`C-c C-c` invoque quelques autres commandes de compilation sur le tampon courant dans son ensemble (`tex-compile`).

Pour passer le tampon à travers TEX, tapez `C-c C-b` (`tex-buffer`). La sortie formatée va dans un fichier temporaire, normalement un fichier `.dvi`. Ensuite, vous pouvez taper `C-c C-v` (`tex-view`) de lancer un programme externe, comme `xdvi`, pour voir ce fichier de sortie. Vous pouvez également taper `C-c C-p` (`tex-print`) en imprimer une copie du fichier de sortie.

Par défaut, `C-c C-b` lance TEX dans le répertoire courant. La sortie de TEX va également dans ce répertoire. Pour exécuter TEX dans un répertoire différent, changer la variable `tex-directory` pour le nom du répertoire désiré. Si votre environnement de variable `TEXINPUTS` contient les noms relatifs de répertoire, ou si vos fichiers contiennent les commandes '`\input`' avec des noms de fichiers relatifs, puis `tex-directory` doit être "`.`" ou vous obtiendrez des résultats erronés. Sinon, il est sûr de spécifier un autre répertoire, tel que "`/tmp`".

La variante de TEX de la mémoire tampon détermine ce shell de commande `C-c C-b` fonctionne effectivement. En mode plain TEX, il est précsié par la variable `tex-run-command`, qui est par défaut "`tex`". En mode LATEX, il est précisé par la variable `latex-run-command`, qui est par défaut "`latex`". La commande shell `C-c C-v` qui lance le `.dvi` est déterminée par la variable `tex-dvi-view-command` quelle que soit la variante TEX. La commande shell `C-c C-p` qui fonctionne pour imprimer la sortie est déterminée par la variable `tex-dvi-print-command`.

Normalement, Emacs ajoute automatiquement le nom du fichier de sortie pour les chaînes de commande shell décrites dans le paragraphe précédent. Par exemple, si `tex-dvi-view-command` est "`xdvi`", `C-c C-v` lance "`xdvi`". Dans certains cas, cependant, le nom du fichier doit être intégré dans la commande, par exemple, si vous avez besoin de fournir le nom de fichier comme argument pour une commande dont la sortie est l'ambiance à l'autre. Vous pouvez spécifier l'endroit où placer le nom de fichier par '`*`' dans la chaîne de commandement. Par exemple, (`setq tex-dvi-print-command ``dvips -f * | lpr''`)

La borne de sortie de TEX, y compris les messages d'erreur, apparaît dans un tampon appelé `*tex-shell*`. Si TEX obtient une erreur, vous pouvez passer à ce tampon et nourrir l'entrée (cela fonctionne comme en mode Shell, voir la section **??** [Shell intéractif], page **??**). Sans passer à ce tampon, vous pouvez la faire défiler de sorte que sa dernière ligne soit visible en tapant `C-c C-l`.

Tapez `C-c C-k` (`tex-kill-job`) pour tuer le processus de TEX si vous voyez que sa sortie n'est plus utile. L'utilisation de `C-c C-b C-r` ou `C-c` tue aussi tout processus de TEX toujours en cours.

Vous pouvez également passer une région arbitraire à TEX en tapant `C-c C-r` (`tex-region`). C'est délicat, cependant, parce que la plupart des fichiers source TEX contiennent des commandes au début pour définir les paramètres et défiinir des macros, sans laquelle aucune partie ultétieure du fichier ne sera formaté correctement. Pour résoudre ce problème, `C-c C-r` vous permet de désigner une partie du fichier comme contenant des commandes essentielles ; il est inclus avant

la région déterminée dans le cadre de l'entrée de TEX. La partie désignée du fichier est appelée l'en-tête.

Pour indiquer les limites de l'en-tête en mode Plain TEX, vous insérez deux chaînes spéciales dans le fichier. Insérer '`\%**` début de tête' avant l'en-tête, et '`\%**`fin de tête' après. Chaque chaîne doit apparaître entièrement sur une seule ligne, mais il peut y avoir un autre texte sur la ligne avant ou après. Les lignes contenant les deux chaînes sont incluses dans l'en-tête. Si '`\%**`début de tête' n'apparaît pas dans les 100 premières lignes de la mémoire tampon, `C-c C-r` suppose qu'il n'y a pas de tête.

En mode LATEX, l'en-tête commence par '`\documentclass`' ou '`\documentstyle`' et se termine par '`\begin{document}`'. Ce sont des commandes que LATEX vous oblige à utiliser dans tous les cas, donc rien à faire pour identifier la tête.

Les commandes (`tex-buffer`) et (`tex-region`) font tout leur travail dans un répertoire temporaire, et ne sont pas disponible pour tous les fichiers auxiliaires nécessaires à TEX pour les références croisées ; ces commandes ne sont généralement pas adaptés pour l'exécution de la copie finale dans laquelle tous les renvois besoin d'être correct.

Lorsque vous voulez que les fichiers auxiliaires pour les références croisées, utilisez `C-c C-f` (`tex-file`) qui étend TEX dans le fichier du tampon courant, dans le répertoire de ce fichier. Avant d'exécuter TEX, il propose d'enregistrer tous les tampons modifiés. En règle générale, vous devez utiliser (`tex-file`) deux fois pour obtenir les renvois corrects.

La valeur des variables `tex-start-options` spécifie les options pour l'exécution de TEX. La valeur des variables `tex-start-commands` spécifie les commandes pour le démarrage de TEX. Les causes de la valeur par défaut de TEX pour fonctionner en mode non-stop. Pour exécuter TEX intéractivement définissez la variable sur "`.`".

Les grands documents de TEX sont souvent divisés en plusieurs fichiers —un fichier principal, ainsi que des sous-fichiers. Lancer TEX sur un sous-fichier ne fonctionne généralement pas ; vous devez exécuter le fichier principal. Afin de rendre `tex-file` utile lorsque vous modifiez un sous-fichier, vous pouvez définir la variable `tex-main-file` pour le nom du fichier principal. Puis `tex-file` lance TEX sur ce fichier.

La façon la plus pratique d'utiliser `tex-main-file` est de la spécifier dans une variable locale dans chacun des sous-fichiers. Voir la section 33.2.5 [Variables de fichiers], page 106.

Pour les fichiers LATEX, vous pouvez utiliser BibTEX pour traiter le fichier auxiliaire pour le dossier du tamon courant. BibTEX regarde les citations bibliographiques dans une base de données et prépare les références citées pour la bibliographie. La commande `C- c TAB` (`tex-bibtex-file`) lance la commande shell (`tex-bibtex-command`) pour produire un fichier `'.bbl'` pour le dossier du tampon courant.

Généralement, vous devez faire `C-c C-f` (`tex-file`) une fois pour générer le fichier `'.aux'`, puis faire pour `C-c TAB` (`tex-bibtex-file`), puis répéter `C-c C-f` (`tex-file`) deux fois de plus pour corriger les références croisées.

Pour appeler un autre programme de compilation sur le tampon de TEX en cours, tapez `C-c C-c` (`tex-compile`). Cette commande sait comment passer des arguments à de nombreux programmes communs, y compris `pdflatex, yap, xdvi` et `dvips`. Vous pouvez choisir le programme de compilation de votre choix en utilisant les touches d'achèvement standard (voir section 5.4 [Complétion], page 5.4).

22.10.4 Mode divers TEX

Saisie de toute variante en mode TEX exécute les crochets `text-mode-hook` et `tex-mode-hook`. Puis il s'exécute soit en `plain-tex-mode-hook`, `latex-mode-hook`, ou `slitex-mode-hook`, selon

le cas. Lancer le shell TEX exécute le crochet `tex-shell-hook`. Voir la section 33.2.2 [Crochets], page 106.

Les commandes `M-x iso-iso2tex`, `M-x iso-tex2iso`, `M-x iso-iso2gtex` et `M-x iso-gtex2iso` peuvent être utilisées pour convertir entre l'encodage Latin-1 et des équivalents de TEX.

22.11 Modes SGML et HTML

22.12 Mode nroff

22.13 Texte enrichi

22.13.1 Mode enrichi

22.13.2 Nouvelles lignes hard et soft

22.13.3 Information du format d'édition

22.13.4 Faces en texte enrichi

22.13.5 Indentation en texte enrichi

22.13.6 Justification en texte enrichi

22.13.7 Réglages d'autres propriétés de textes

22.14 Édition de tables basées sur du texte

22.14.1 Qu'est-ce qu'une table basée sur du texte ?

22.14.2 Création de table

22.14.3 Table de reconnaissance

22.14.4 Commandes pour les tableaux de cellules

22.14.5 Justification des cellules

22.14.6 Table lignes et colonnes

22.14.7 Conversion entre texte plein et tables

22.14.8 Diverses tables

22.15 Édition deux colonnes

Chapitre 23

Édition de programmes

Chapitre 24

Compilation et test de programmes

Chapitre 25

Maintenace de gros programmes

25.1 Contrôle de version

25.1.1 Introduction à la contrôle de version

Compréhension du problème adressé

contrôle de version du système supportée

Concepts de contrôle de version

Merge-based vs lock-based contrôle de version

Changeset-based vs File-based contrôle de version

Décentralisé vs répertoires centralisés

Types de fichiers log

25.1.2 Contrôle de version et mode ligne

25.1.3 Édition de base sous contrôle de version

Contrôle de version de base avec fusion

Contrôle de version de base avec verrouillage

Contrôle avancé avec `C-x v v`

25.1.4 Fonctions du log d'entrée de buffer

25.1.5 Enregistrement d'un fichier de contrôle de version

25.1.6 Examen et comparaison des anciennes versions

25.1.7 VC Chance Log

25.1.8 Refaire les actions de contrôle de version

25.1.9 VC mode répertoire

Le buffer VC répertoire

VC commandes de répertoire

25.1.10 Branches de contrôle de version

Changement entre les branches

Changement dans une branche

Fusion de branches

Création de nouvelles branches

25.2 Changement de logs

25.2.1 Commandes de changements de logs

Chapitre 26

Abbréviation

Chapitre 27

Dired, the Directory Editor (l'éditeur de répertoire)

Chapitre 28

Calendrier et agenda

Chapitre 29

Envoyer des mails (courriels)

Chapitre 30

Lecture de mail avec Rmail

Chapitre 31

Commandes diverses

Chapitre 32

Extensions Emacs Lisp

Chapitre 33

Personnalisation

33.1 Interface de personnalisation simple

33.1.1 Groupes de personnalisation

33.1.2 Naviguer et chercher pour des réglages

33.1.3 Changer une variable

33.1.4 Sauvegarde des personnalisations

33.1.5 Faces de personnalisation

33.1.6 Personnalisation d'items spécifiques

33.1.7 Personnaliser des thèmes

33.1.8 Création de thèmes personnalisés

33.2 Variables

33.2.1 Exmanen et réglages de variables

33.2.2 Crochers

33.2.3 Variables locales

33.2.4 Variables locales dans des fichiers

Spécification de fichiers de variables

Sécurité de fichiers de variables

33.2.5 Variables locales par répertoire

33.3 Personnalisation des raccourcis clavier

33.3.1 Carte des clés

33.3.2 Préfixes de la carte des clés

33.3.3 Carte des clés locale

33.3.4 Minibuffer de la carte des clés

33.3.5 Changement de raccourci clavier intéractivement

33.3.6 Renommer un raccourci clavier dans son fichier d'initialisation

33.3.7 Touches modificatrices

33.3.8 Nom ASCII du contrôle des caractères

33.3.9 Renommer les boutons de la souris

Chapitre 34

S'arranger avec les problèmes communs

Annexe A

gnu general public license

Annexe B

GNU Free Documentation License

Annexe C

Arguments de ligne de commandes pour invocation par Emacs

Annexe D

X options et ressources

D.1 Ressources X

D.2 Table des ressources X pour Emacs

D.3 Ressources GTK

D.3.1 Ressources de bases pour GTK

D.3.2 Noms de widget pour GTK

D.3.3 Noms de widget pour GTK dans Emacs

D.3.4 Styles GTK

Annexe E

Emacs 23

Annexe F

Emacs et Mac OS/GNUstep

Cette section décrit les particularités d'utiliser l'Emacs construit avec les bibliothèques GNUSTEP sur GNU/LINUX ou d'autres systèmes d'exploitation, ou sur Mac OS X avec système de fenêtre natif. Sur le Mac OS X, l'Emacs peut être construit sans support système de fenêtre, avec X11, ou avec l'interface de Cocoa ; cette section s'applique seulement à la construction Cocoa. Ceci ne concerne pas les versions de Mac OS X antérieures à la version 10.4.

Pour des raisons historiques et techniques diverses, l'Emacs utilise le terme «`Nextstep`» intérieurement, au lieu «de Cocoa» ou «Mac OS X» ; par exemple, la plupart des commandes et des variables décrites dans cette section commencent par «`ns-`», qui est un raccourci pour «`Nextstep`». NeXTstep était une interface d'application sortie par NeXT Inc pendant les années 1980, dont le Cocoa est un descendant direct. En dehors du Cocoa, il y a un autre système de NeXTstep-style : GNUstep, qui est le logiciel libre. À partir de cette écriture, le support Emacs GNUstep est le statut alpha (voir **??**), mais nous espérons l'améliorer dans l'avenir.

F.1 Utilisation de base Emacs sous Mac OS et GNUstep

Par défaut, les touches `alt` et `option` sont les même que `Meta`. La touche Mac `Cmd` est la même que `Super`, et Emacs fournit un jeu de raccourcis clavier utilisant cette touche modificatrice comme imitation d'autres applications Mac / GNUstep (voir **??**). Vous pouvez changer ces raccourcis clavier en utilisant la manière habituelle (voir page 420).

La variable `ns-right-alternate-modifier` contrôle le comportement de la touche droite `alt` et `options`. Cette touche se comporte comme la touche gauche si la valeur est `left` (par défaut). Une valeur de `control`, `meta`, `alt`, `super`, ou `hyper` les fait se comporter comme des touches modificatrices ; une valeur `none` dit à Emacs de les ignorer.

`S-Mouse-1` ajuste la région à la position du clic, comme `Mouse-3` (`mouse-save-then-kill`) ; cela ne produit pas un menu pop up pour changer la face par défaut, comme `S-Mouse-1` fait normalement (voir page 76). Ce changement fait Emacs se comporter plus comme les autres applications Mac / GNUstep.

Quand vous ouvrez ou sauvez des fichiers en utilisant les menus, ou en utilisant `Cmd-O` et `Cmd-S`, Emacs utilise les boîtes de dialogues pour lire les noms de fichiers.

Sur GNUstep, dans un environnement X-windows vous avez besoin d'utiliser `Cmd-C` à la place de `C-w` ou `M-w` pour transférer du texte vers la première sélection X ; d'autre part, Emacs va utiliser la sélection «clipboard». Comme, `Cmd-y` (au lieu de `C-y`) pour coller depuis une première sélection X au lieu de kill-ring ou clipboard.

F.1.1 Variables d'environnement grabbing

Beaucoup de programmes qui peuvent tourner sous Emacs, comme latex ou man, dépendent de la configuration des variables d'environnement. Si Emacs est lancé depuis le shell, il héritera automatiquement de ces variables d'environnement et ses sous-processus en hériteront aussi. Mais si Emacs est lancé depuis le Finder il n'est pas un descendant d'aucun shell, donc son environnement de variables n'a pas été configuré, ce qui provoque souvent que les sous-processus qu'il déclenche se comportent différemment que s'ils étaient lancés depuis le shell.

Pour le PATH et MANPATH des variables, un large système de méthodes de configuration PATH est recommandé sur Mac OS X 10.5 et antérieur, en utilisant les fichiers `/etc/paths` et le répertoire `/etc/paths.d`.

F.2 Personnalisation Mac / GNUstep

Emacs peut être personnalisé de plusieurs manières en plus des buffers de personnalisation standard et le menu d'Options.

F.2.1 Panels de fontes et couleurs

Les panels de fontes et couleurs standard Mac / GNUstep sont accessible via des commandes Lisp. Le panel de fontes peut être obtenu avec `M-x ns-popup-font-panel`. Ca configurera par défaut la fonte de la diapositive la plus récente ou cliquer dessus.

Vous pouvez soulever un panneau coloré avec `M-x ns-popup-color-panel` et glisser la couleur que vous voulez sur la face d'Emacs que vous voulez changer. Le glissement normal altèrera la couleur de premier plan. Le glissement shift altèrera la couleur de fond. Pour renoncer (annuler) les changements (de configuration), créer une nouvelle diapositive et fermer celle qui a subi les altérations.

Très utile dans ce contexte, la liste complète des face est obtenue avec `M-x list-faces-display`.

F.2.2 Personnalisation spécifique pour Mac OS / GNUstep

Les options de personnalisation suivantes sont spécifiques à Nextstep `ns-auto-hide-menu-bar`

Non-nil signifie que la barre de menu est cachée par défaut, mais apparaît si vous déplacez le pointeur de la souris dessus. (Nécessite Mac OS X 10.6 ou ultérieur.)

F.3 Système de fenêtrage sous Mac OS / GNUstep

Les applications Nextstep reçoivent un nombre spécial d'événements qui n'ont pas d'équivalent X. Ils sont envoyés comme clés spécialement définies, ce qui ne correspond à aucune combinaisons de touche. Sous Emacs, ces événements «clés» peuvent être ajoutés aux fonctions comme n'importe quel macros.

Voici une liste des ces événements.

`ns-open-file` cet événement se produit lorsqu'une autre application Nextstep demande à Emacs d'ouvrir un fichier. Une raison typique pour cela serait qu'un utilisateur double-clique sur un fichier dans le Finder. Par défaut, Emacs répond à cet événement en ouvrant un nouveau cadre (frame) et en visitant le fichier dans ce cadre `(ns-find-file)`. Par exception, si le tampon (buffer) choisi est le tampon `*scratch*`, Emacs visite le fichier dans le cadre sélectionné.

Vous pouvez changer la façon dont Emacs répond à un événement `ns-open-file` en changeant la variable `ns-pop-up-frames`. Sa valeur par défaut, `'fresh'`, est ce que nous venons de décrire. Une valeur de `t` signifie que l'on visite toujours le fichier dans un nouveau cadre. Une valeur de `nil` signifie que l'on visite toujours le fichier dans le cadre existant.

`ns-open-temp-file` cet événement se produit lorsqu'une autre application demande à Emacs d'ouvrir un fichier temporaire. Par défaut, c'est géré par la création d'un événement `ns-open-file`, dont les résultats sont décrits ci-dessus.

`ns-open-file-line` certaines applications, telles que ProjectBuilder et gdb, demandent non seulement un fichier particulier, mais aussi une ligne ou une suite de lignes particulières dans le fichier. Emacs gère en visitant ce fichier et en soulignant la ligne demandée `(ns-open-file-select-line)`.

`ns-drag-file` cet événement se produit lorsqu'un utilisateur fait glisser des fichiers à partir d'une autre application dans un cadre Emacs. Le comportement par défaut est d'insérer le contenu de tous les fichiers déplacés dans le tampon courant `(ns-insert-files)`. La liste des fichiers déplacés est stockée dans la variable `ns-input-file`.

ns-drag-color cet événement se produit lorsqu'un utilisateur fait glisser une couleur depuis le color well (ou d'une autre source) dans un cadre Emacs. Le comportement par défaut est de modifier la couleur de premier plan de la zone dont la couleur a été glissée sur `(ns-set-foreground-at-mouse)`. Si cet événement est délivré avec le modifieur Shift, Emacs change la couleur de fond à la place `(ns-set-background-at-mouse)`. Le nom de la couleur glissée est stockée dans la variable `ns-input-color`.

`ns-change-font` cet événement se produit lorsque l'utilisateur sélectionne une fonte (police) dans un panel Nextstep (qui peut être ouvert avec `Cmd-t`. Le comportement par défaut est d'ajuster la fonte (police) du cadre sélectionné `(ns-respond-to-changefont)`. Le nom et la taille de la fonte (police) sélectionnnée sont stockés dans les variables `ns-input-fontsize`, respectivement.

`ns-power-off` cet événement se produit lorsque l'utilisateur se déconnecte et Emacs est encore en cours d'exécution, ou lorsque `'Quit Emacs'` est choisi dans le menu de l'application. Le comportement par défaut est de sauver tous les tampons de visite de fichier.

Emacs permet également aux utilisateurs de faire usage de services Nextstep, via un ensemble de commandes dont le nom commence par `'ns-service'` et se termine par le nom du service. Tapez `M-x ns-service-TAB` pour voir la liste de ces commandes. Ces fonctions opèrent soit sur le texte marqué (le remplaçant par la suite) ou prennent un argument de chaîne et renvoient le résultat sous forme de chaîne. Vous pouvez également utiliser la fonction Lisp `ns-perform-service` pour passer des chaînes arbitraires à l'appel services arbitraires et recevoir les résultats en retour. Notez que vous devrez peut-être redémarrer Emacs pour accéder aux services nouvellement disponibles.

F.4 Support GNUstep

Emacs peut être construit et tourner sous GNUstep, mais il y a encore des problèmes à régler. Les développeurs intéressés devraient contacter `emacs-devel@gnu.org`.

Annexe G

Emacs et Microsoft Windows/MS-DOS

The GNU Manifesto

Glossaire

Index des clés (caractères)

Index des commandes et fonctions

Index des variables

Index des concepts