

Un court tutoriel pour Emacs

Contents

1	Historique	1
2	Lancement d'Emacs	1
3	Prise en main	1
3.1	Interface	2
3.1.1	<i>Buffers</i>	2
3.1.2	<i>Windows</i>	2
3.1.3	Cadre (<i>frames</i>)	2
3.1.4	<i>Modeline</i>	2
3.1.5	<i>Minibuffer</i>	2
3.2	Fonctions	2
4	Quelques raccourcis	5
4.1	Commandes générales	6
4.2	<i>minibuffer</i>	6
4.3	Obtenir de l'aide	6
4.4	Naviguer dans un <i>buffer</i>	6
4.5	Marquer une zone d'un <i>buffer</i>	7
4.6	Éditer le contenu d'un <i>buffer</i>	7
4.7	Gestion des <i>buffers</i> , fenêtres et cadres	8
4.7.1	<i>buffers</i>	8
4.7.2	Fenêtres	8
4.7.3	Cadres (<i>frames</i>)	8
4.8	Fichiers	8
4.9	Recherche	9
4.10	Remplacement de texte	11
4.11	Répétitions et macros	11
4.12	Complétion	11
5	Modes utiles	13
5.1	Modes utilitaires	13
5.1.1	Flyspell	13
5.1.2	FlyCheck	13
5.2	Interfaces avec git	13
5.2.1	GitGutter	14
5.2.2	Magit	14
5.3	Rédaction de documents L^AT_EX	14
5.3.1	AUCTeX	15
5.3.2	RefTeX	16
5.4	Programmation	17
5.4.1	Python	17

5.4.2	C++	17
5.4.3	Tags	17
5.4.4	YASnippet	18
6	Pour aller plus loin	19
6.1	Org-mode	19
6.2	Pliage et dépliage avec <code>narrow-or-widen-dwim</code>	21
6.3	Autres modes installés	21
6.4	Personnalisation	21
6.4.1	<code>custom.el</code>	21
6.4.2	Emacs lisp	22

1 Historique

Emacs (initialement pour ***Editing MACroS** running on TECO*) fut créé par Richard M. Stallman (dit RMS) et Guy L. Steele en 1976. Il s'agissait alors d'une collection de macro-commandes pour simplifier l'utilisation de TECO (1964) qui était un langage de programmation pour l'édition de fichier orienté ligne.

En 1983 RMS lance le projet GNU et l'une de ses premières tâches est de le doter d'un éditeur digne de ce nom. Profitant de son expérience et en s'inspirant de Gosling Emacs il crée GNU Emacs. Pour cela, il l'écrit en C et permet l'écriture d'extensions en Emacs Lisp. C'est probablement cette extensibilité qui fait que malgré son âge avancé, Emacs est l'un des outils les plus utilisés pour le développement.

2 Lancement d'Emacs

Pour lancer Emacs, il suffit de taper la commande

```
emacs
```

dans un terminal. Si le terminal ne permet pas l'ouverture de fenêtres graphiques (pas de `DISPLAY`), Emacs démarrera alors en mode texte. Si on utilise l'option `-nw`, pour *no window*, on force le lancement d'Emacs en mode texte.

```
emacs -nw
```

Attention dans ce mode, pour quitter Emacs, il faut utiliser le raccourci clavier `C-x C-c` (contrôle-x puis contrôle-c). Une dernière option pratique est l'option `-q`. Son effet est de lancer Emacs sans utiliser la configuration locale.

```
emacs -q
```

3 Prise en main

La prise en main d'Emacs n'est pas difficile pour une utilisation basique. En mode graphique Emacs dispose de menus qui permettent d'accéder à toutes les opérations classiques. Néanmoins, pour l'utiliser plus efficacement, il convient de comprendre un peu son fonctionnement.

Comme on l'a dit, Emacs a vu le jour comme un ensemble de **macros**, c'est-à-dire des enchaînements de commandes simples ou d'autres macros. Ces macros (ou plus précisément ces **fonctions**) sont toujours un des piliers du fonctionnement d'Emacs. Le nombre de fonctions proposées par Emacs est très important et pour y accéder rapidement Emacs permet l'usage de raccourcis clavier. C'est en apprenant et en utilisant ces raccourcis qu'on gagne en productivité. L'utilisation de la

souris avec Emacs est donc souvent contre productive. Enfin on peut facilement créer et enregistrer des enchaînements de commandes pour éviter les tâches répétitives.

Une première façon d'apprendre un peu Emacs est d'effectuer le **tutoriel intégré** qu'il propose. Pour démarrer le tutoriel il suffit de saisir le raccourcis **C-h t**. Pour le novice, cela signifie qu'on doit taper sur la touche contrôle (**C**) et sur la touche **h** simultanément, puis sur la touche **t** seulement.

3.1 Interface

Globalement, l'interface d'Emacs est assez intuitive, mais elle a conservé tous les ingrédients qui en faisaient sa richesse alors que ce n'était pas une application graphique (la première version graphique de GNU Emacs est la 19, sortie en 1994). Certains de ses concepts ne sont donc pas si communs de nos jours.

3.1.1 *Buffers*

Un *buffer* est une zone tampon et la plus petite unité d'édition. Il correspond en général à la vue d'un fichier édité.

3.1.2 *Windows*

Il est possible de visualiser plusieurs *buffers* simultanément dans Emacs. Chaque vue est appelée **fenêtre**. Attention, dans le jargon Emacs, une fenêtre n'est pas à confondre avec les fenêtres gérées par le *window-manager* de l'interface graphique (voir figure 1, où un cadre Emacs est découpé en trois fenêtres). On peut afficher un **même** *buffer* dans différentes fenêtres et il peut être modifié dans n'importe laquelle de ses vues.

3.1.3 *Cadre (frames)*

Lorsqu'il est exécuté en mode graphique, une même instance d'Emacs peut proposer plusieurs cadres (*frames*) d'affichage. Il s'agit en fait de plusieurs fenêtres du gestionnaire graphique.

Dans l'exemple de la figure 2, il s'agit bien de la même instance d'Emacs : on a lancé un seul exécutable, mais on a ouvert un cadre (*frame*) supplémentaire. Dans chacun des cadres, on affiche une vue d'un **même** *buffer* d'édition, c'est-à-dire que le **même** *buffer* peut être modifié par l'intermédiaire de l'un ou l'autre des cadres.

3.1.4 *Modeline*

Dans Emacs, chaque *buffer* est délimité **en bas** par la *modeline*. Il s'agit d'une zone dans laquelle sont affichées divers informations quant au mode de fonctionnement d'Emacs dans son *buffer*. On la voit en action sur la figure 3, le mode majeur est **CMake**, le fichier édité s'appelle **CMakeList.txt**, enfin, on travaille sur la branche **develop**.

3.1.5 *Minibuffer*

Le *minibuffer* est un *buffer* spécial d'Emacs. C'est la zone d'interaction avec l'utilisateur. C'est là qu'on doit répondre aux éventuelles questions d'Emacs et c'est là qu'on tape les commandes. Il faut donc le garder constamment à l'oeil. Voir la figure 4 (ici une exécution de **grep** dans Emacs en préparation). Pour exécuter une commande dans le *minibuffer* il faut taper **M-x**.

3.2 Fonctions

Dans Emacs, tout est fonction. Et toute fonction peut-être redéfinie localement (dans un *buffer*), globalement (dans tout Emacs) ou en fonction du **mode** (dans une famille de *buffers*).

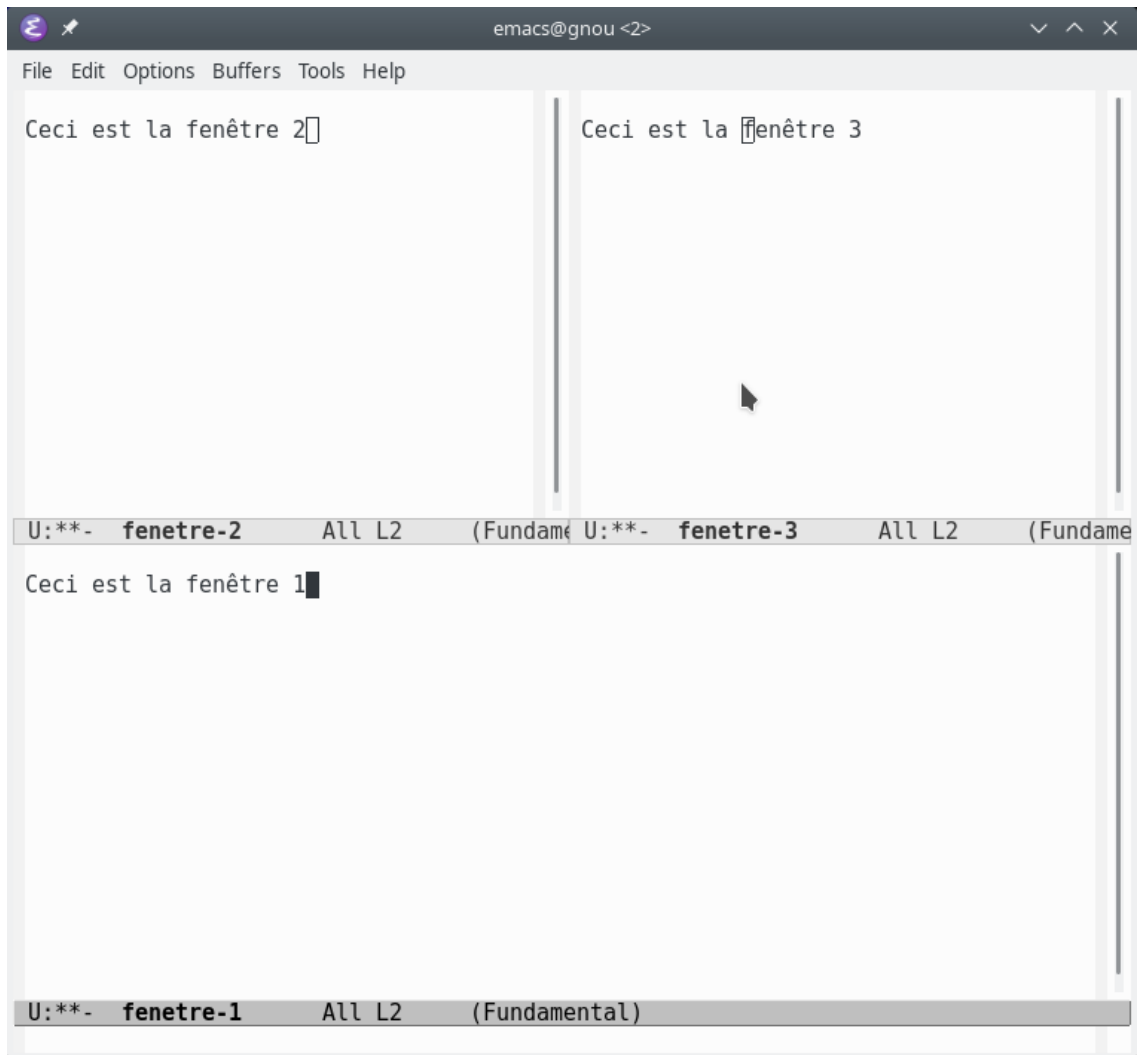


Figure 1: Trois fenêtres affichées dans Emacs

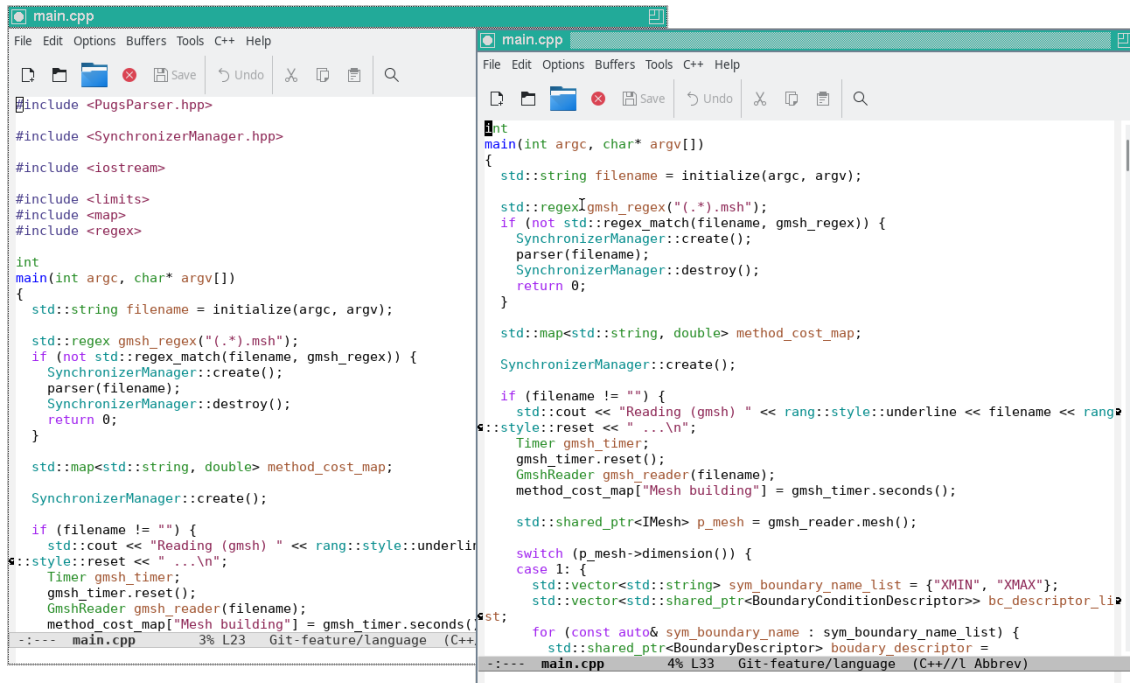


Figure 2: Deux cadres (*frames*) Emacs affichant le même *buffer*

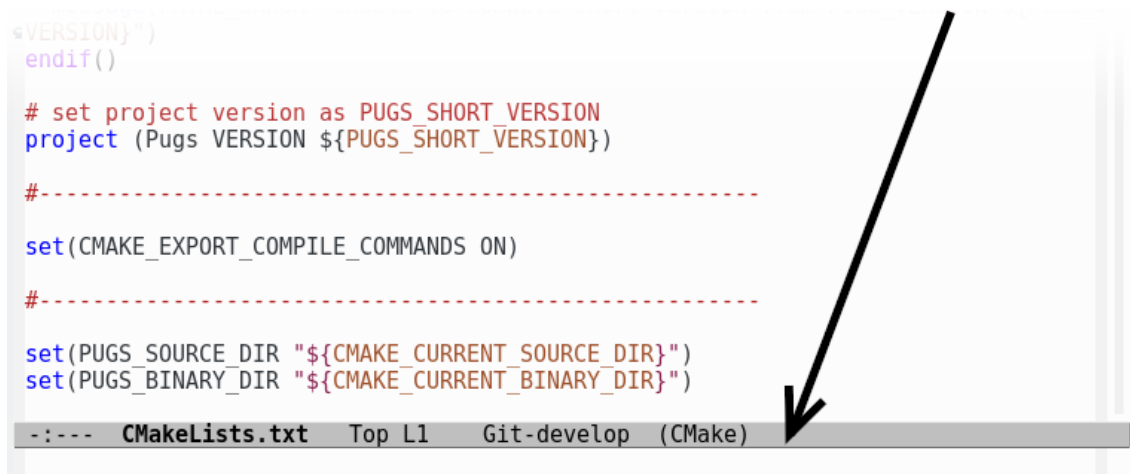
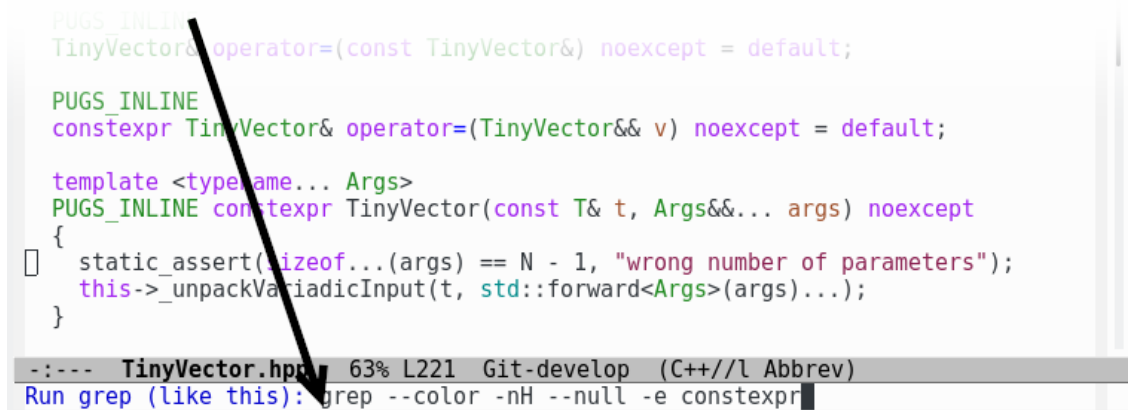


Figure 3: La *modeline*. Ici on voit qu'on est en mode **CMake**, que le fichier édité s'appelle **CMakeList.txt** et qu'on travaille sur la branche **develop**.



```
PUGS_INLINE
TinyVector& operator=(const TinyVector&) noexcept = default;

PUGS_INLINE
constexpr TinyVector& operator=(TinyVector&& v) noexcept = default;

template <typename... Args>
PUGS_INLINE constexpr TinyVector(const T& t, Args&&... args) noexcept
{
    static_assert(sizeof...(args) == N - 1, "wrong number of parameters");
    this->_unpackVariadicInput(t, std::forward<Args>(args)...);
}

-:--- TinyVector.hpp 63% L221 Git-develop (C++/l Abbrev)
Run grep (like this): grep --color -nH --null -e constexpr
```

Figure 4: Le *minibuffer* en action. Préparation de l'exécution d'une commande `grep` dans Emacs.

Par exemple, presser une touche (quel que soit le contexte) déclenche l'appel d'une fonction et cette fonction dépend du contexte dans lequel la touche est pressée. C'est ce concepts qui fait d'Emacs un outil aussi puissant et versatile. Les fonctions peuvent être de deux types. Ce sont

- soit des fonctions internes (écrites en C),
- soit des fonctions utilisateur (écrites en `ELisp`).

Par le passé, ce deuxième type de fonctions pouvait dégrader les performances d'Emacs (si bien que certains lui donnaient l'acronyme récuratif *Emacs Makes All Computer Slow*). Mais les gains de performances des ordinateurs et l'utilisation d'`ELisp` *byte-compilé* plutôt que simplement interprété ont corrigé ce défaut.

4 Quelques raccourcis

On présente dans cette partie certains raccourcis utiles. Lorsque les raccourcis de cette configuration diffèrent des raccourcis standard d'Emacs, on l'indique.

Mais rappelons tout d'abord les notations et principes.

- `C-a` signifie qu'on appuie simultanément sur les touches `Ctrl` (contrôle) et `a`.
- `C-M-a` signifie qu'on appuie simultanément sur les touches `Ctrl`, `Meta` (`Alt` sur les claviers modernes) et `a`.
- `C-S-a` signifie qu'on appuie simultanément sur les touches `Ctrl`, `Maj` et `a`.
- `C-a C-b c` signifie qu'on appuie simultanément sur les touches `Ctrl` et `a`, puis simultanément sur les touches `Ctrl` et `b`, puis sur la touche `c`.

On ne liste ici que quelques unes des commandes (et raccourcis), seulement une liste de celles qu'on juge importantes.

Noter que cette configuration utilise le mode `which-key`, dont l'une des fonctionnalités est de fournir une aide dans le *minibuffer* lors de la saisie de commandes. Par exemple, si on tape `C-x`, au bout d'un instant `which-key` affiche dans le *minibuffer* la liste des commandes qui commencent par `C-x`. C'est donc un excellent moyen de retrouver les raccourcis et d'en apprendre de nouveaux. Cette aide est évidemment contextuelle.

4.1 Commandes générales

Raccourci	Description
C-x C-c	Sort d'Emacs
C-g	Interrompt la commande en cours
C-S-g	Force l'interruption de la commande
M-x	Lance une commande (dans le <i>minibuffer</i>)
C-c C-n	Relance l'évaluation du <i>buffer</i> (normal-mode)

4.2 *minibuffer*

Lors d'une interaction avec le *minibuffer*, on a accès à certains raccourcis qui facilitent son utilisation.

Raccourci	Description
M-x	Lance une interaction dans le <i>minibuffer</i>
C-x ESC ESC	Édite la dernière commande dans le <i>minibuffer</i>
TAB	Complète l'entrée autant que possible

Cette configuration utilise **swiper**, donc toutes les interactions effectuées dans le *minibuffer* proposent la liste des commandes disponibles qui correspondent à la chaîne entrée (SPC faisant office de n'importe quelle liste de caractères). Il est alors possible de naviguer dans cette liste (en utilisant les flèches du clavier par exemple)

4.3 Obtenir de l'aide

On peut lancer l'aide en tapant C-h et en suivant les indications du *minibuffer*.

Raccourci	Description
C-h t	Lance le tutoriel intégré
C-h m	Informations sur le mode courant
C-h i	Accède à la documentation (info)
C-h a	à propos (interaction dans le <i>minibuffer</i>)

4.4 Naviguer dans un *buffer*

Au delà des flèches et autres touches usuelles du clavier, Emacs propose d'autres moyens de se déplacer dans un *buffer*.

Raccourci	Description
C-b	Reculer d'un caractère
C-f	Avancer d'un caractère
M-b	Reculer d'un mot
M-f	Avancer d'un mot
C-p	Ligne précédente
C-n	Ligne suivante
C-a	Début de ligne
C-e	Fin de ligne
M-a	Début de la phrase
M-e	Fin de la phrase
M-m	Début de l'indentation
M-g g <i>numéro</i>	Positionne le curseur à la ligne <i>numéro</i>

Noter qu'on peut généralement utiliser des combinaisons de **Ctrl**, **Meta**,... avec les touches de déplacement du clavier (comme les flèches).

4.5 Marquer une zone d'un *buffer*

Une zone est délimitée par une **marque** (position de départ) et la position courante du curseur. Créer une zone consiste donc à poser une **marque**, puis à déplacer le curseur pour la délimiter. Tous les caractères compris entre ces deux positions font partie de la zone.

Raccourci	Description
C-SPC ou C-@	Pose une marque
C-g	Supprime la marque
C-x C-x	Inverse la marque et le curseur
C-=	Sélectionne un mot ou étend la sélection

Noter qu'un enchaînement de C-= permet d'étendre rapidement la zone aux entités faisant sens. En effet, C-= s'arrête aux guillemets encadrant une chaîne de caractères, aux paramètres ou au corps d'une fonction par exemple, ce qui permet de facilement les sélectionner.

4.6 Éditer le contenu d'un *buffer*

Les commandes citées ici concernent essentiellement les **modes** d'édition classiques. Il est possible que pour certains modes les raccourcis ne fonctionnent pas ou n'aient simplement pas de sens. Dans ces modes d'édition classiques (modes de saisie par exemple), le clavier fonctionne normalement. On précise simplement ici quelques commandes usuelles.

L'action des commandes d'édition décrites ci-dessous est effectuée relativement à la position courante du curseur.

Raccourci	Description
DEL	Efface un caractère vers la gauche
C-d ou SUPR	Efface le caractère sous le curseur
DEL ou SUPR	Efface une zone sélectionnée
C-DEL, M-DEL ou C-w	Coupe le mot vers la gauche
M-d	Coupe le mot vers la droite
M-O C-k	Coupe jusqu'au début de la ligne
C-k	Coupe jusqu'à la fin de la ligne
C-w	Coupe une zone sélectionnée
M-w	Copie une zone sélectionnée
M-z <i>caractère</i>	Coupe jusqu'au prochain caractère <i>caractère</i>
C-_ ou C-/ ou C-x u	Annule la dernière saisie/commande

Noter qu'avec cette configuration d'Emacs, si le caractère à effacer est un **blanc**, tous les blancs suivants dans la direction d'effacement sont supprimés.

Puisqu'on vient d'aborder comment **copier** ou **couper** une zone du texte, nous allons décrire le **coller**. Là encore, le vocable Emacs n'est pas standard. L'action **couper** dans Emacs se nomme **kill** et **coller**, **yank**. On comprend ainsi mieux les choix de lettres **k** et **y** dans ce contexte.

Raccourci	Description
C-y	Colle la dernière sélection copiée
M-y	Remplace le texte collé par la copie précédente

En fait, Emacs conserve toutes les copies en mémoire (*kill-ring*) et on peut donc les utiliser à tout moment. Dans sa version de base, parcourir le *kill-ring* nécessite d'abord un **coller**. C'est-à-dire qu'il faut effectuer C-y and de taper M-y pour le parcourir.

Cette configuration d'Emacs utilise **swiper** et le parcours du *kill-ring* est amélioré. En effet, **swiper** propose l'utilisation directe de M-y et plutôt que de remplacer le texte *en ligne* comme le fait la version de base, on y accède dans le *minibuffer*. On peut alors le parcourir, soit en continuant à presser M-y, soit en utilisant les flèches du clavier, soit même en entrant les une partie du texte comme pour les recherches.

4.7 Gestion des *buffers*, fenêtres et cadres

4.7.1 *buffers*

Dans cette configuration, on utilise le paquet `IBuffer` à la place du mode `Buffer` classique. Il fonctionne avec les mêmes raccourcis de base.

Raccourci	Description
<code>C-x b</code>	Choisit un autre <i>buffer</i>
<code>C-x C-b</code>	Ouvre le mode <code>IBuffer</code>
<code>C-x k</code>	Choisit un <i>buffer</i> à fermer
<code>C-x k RET</code>	Ferme le <i>buffer</i> courant
<code>C-x left</code>	Retourne au <i>buffer</i> précédent
<code>C-x right</code>	Retourne au <i>buffer</i> suivant

Noter que les raccourcis `C-x left` et `C-x right` (où `left` et `right` désignent les flèches du clavier) sont extrêmement utiles et il est important de les mémoriser. On parle bien ici de retrouver le *buffer* précédent ou suivant dans la même fenêtre.

Lorsqu'on ferme un *buffer*, si les données n'ont pas été sauvegardées Emacs invite l'utilisateur à le faire.

4.7.2 Fenêtres

Raccourci	Description
<code>C-x 1</code>	Agrandit la fenêtre pour qu'elle occupe tout le cadre
<code>C-x 2</code>	Découpe la fenêtre courante horizontalement
<code>C-x 3</code>	Découpe la fenêtre courante verticalement
<code>C-x 0</code>	Ferme la fenêtre courante
<code>C-x o</code>	Positionne le curseur dans une autre fenêtre

Pour la gestion des fenêtres, on utilise le mode `ace-window` qui ajoute la fonctionnalité suivante. Si plus de deux fenêtres sont actives dans le cadre courant, `C-x o` affiche alors des numéros dans chacune des fenêtres. On entre ce numéro pour basculer dans la fenêtre choisie.

4.7.3 Cadres (*frames*)

Raccourci	Description
<code>C-x 5 0</code>	Ferme le cadre actif
<code>C-x 5 1</code>	Ferme tous les autres cadres
<code>C-x 5 2</code>	Crée un nouveau cadre

4.8 Fichiers

Raccourci	Description
<code>C-x C-f</code>	Recherche incrémentale d'un fichier ou le crée
<code>C-x C-s</code>	Sauvegarde le <i>buffer</i> courant
<code>C-x s</code>	Sauvegarde tous les <i>buffers</i>
<code>C-x C-w</code>	Sauvegarde le <i>buffer</i> courant dans un nouveau fichier
<code>C-x i</code>	Insère le contenu d'un fichier dans le <i>buffer</i>

IMPORTANT

Si on souhaite créer un nouveau fichier avec `C-x C-f` dont le nom est une sous chaîne d'un fichier existant, il ne faut pas utiliser `RET` pour valider, mais `C-M-j`. Utiliser `RET` valide la recherche.

Une autre façon de créer le fichier `foo`, si un fichier correspondant à l'expression `foo` existe

(par exemple **foobar**), consiste à interrompre la recherche incrémentale en sélectionnant la **zone de saisie** avec les flèches puis à taper **RET**.

Ouverture de fichiers/répertoires distants. À noter que cette configuration d'Emacs utilise le mode **TRAMP** pour (*Transparent Remote Access, Multiple Protocols*) qui permet l'ouverture de fichiers à distance depuis Emacs.

Pour que l'édition distante fonctionne, il suffit que la connexion par **ssh** soit autorisée.

L'intérêt de ce mode est évident puisqu'on n'a pas besoin de réinstaller l'environnement Emacs sur cette nouvelle machine, pour travailler dans son environnement usuel. Il suffit de pouvoir s'y connecter par **ssh**.

Pour ouvrir un fichier distant, il suffit d'utiliser le raccourci **C-x C-f**, puis d'écrire dans le **minibuffer**

```
/sshx:username@hostname:
```

À partir de là, si on presse **RET**, Emacs ouvrira un **direc buffer** (c'est à dire un buffer permettant de manipuler les fichiers du répertoire **HOME** de l'utilisateur sur la machine distante. On peut aussi utiliser **TAB** pour voir alors la liste des fichiers accessibles dans le **minibuffer**, la complétion fonctionnant comme en local.

Une fois un fichier ouvert, il est manipulé classiquement dans le **buffer** et seule la sauvegarde du fichier nécessite la connexion.

4.9 Recherche

Raccourci	Description
C-s	Recherche une chaîne dans le <i>buffer</i> vers le bas
C-r	Recherche une chaîne dans le <i>buffer</i> vers le haut
C-s C-w	Complète la chaîne à chercher avec le mot courant
C-M-s	Recherche une expression régulière vers le bas
C-M-r	Recherche une expression régulière vers le haut

Une fois une occurrence trouvée, on peut poursuivre la recherche vers le bas ou vers le haut, en faisant respectivement **C-s** ou **C-r**. Puisqu'on utilise pas le mode standard de recherche d'Emacs, mais le mode **swiper**, il est possible de naviguer dans le **minibuffer** avec les flèches pour se déplacer d'une occurrence à l'autre.

Noter qu'en mode recherche (on a déjà pressé **C-s**) le raccourci **C-w** ajoute le mot suivant le curseur à la recherche. On recherche ainsi rapidement une suite de mots (les mots cherchés doivent être sur une même ligne).

Quand la recherche est terminée,

- soit on utilise **C-g** pour revenir où on était quand on a commencé la recherche,
- soit **RET** pour continuer l'édition du *buffer* à la position de l'occurrence.

Une autre particularité intéressante de **swiper** est son utilisation de **SPC** qui sert de joker pendant les saisies. Voir par exemple, la figure 5, où on a lancé une recherche puis tapé **y t x(** dans le **minibuffer**, les lignes correspondantes y sont affichées. On peut parcourir cette liste à l'aide des flèches du clavier, le *buffer* de recherche suit l'occurrence sélectionnée.

Attention si on utilise **C-S-s C-w**, on n'utilise pas **swiper** pour la recherche, mais l'outil recherche incrémentale classique d'Emacs. Il est possible d'ajouter des mots à la recherche en faisant à nouveau **C-w**.

The image shows an Emacs editor window titled 'emacs@gnou <3>'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'C++', and 'Help'. The main text area contains C++ code for a 'TinyMatrix' class. The code includes a static assertion for arithmetic types, a constructor, and several inline operators. A 'swiper' search overlay is visible at the bottom, showing a search for 'y t x(' and listing several matches from 'TinyMatrix.hpp'.

```

{
    static_assert(std::is_arithmetic<T>(), "Cannot assign 'identity' value for non-
arithmetic types");
    for (size_t i = 0; i < N; ++i) {
        for (size_t j = 0; j < N; ++j) {
            m_values[_index(i, j)] = (i == j) ? 1 : 0;
        }
    }
    return *this;
}

PUGS_INLINE
constexpr TinyMatrix& operator=(const TinyMatrix& A) noexcept = default;

PUGS_INLINE
constexpr TinyMatrix& operator=(TinyMatrix&& A) noexcept = default;

template <typename... Args>
PUGS_INLINE constexpr TinyMatrix(const T& t, Args&&... args) noexcept
{
    static_assert(sizeof...(args) == N * N - 1, "wrong number of parameters");
    this->_unpackVariadicInput(t, std::forward<Args>(args)...);
}

```

--- TinyMatrix.hpp 43% L262 Git-feature/language [(C++//l Abbrev)]

1/7 Swiper: y t x(

- PUGS_INLINE constexpr TinyMatrix(const T& t, Args&&... args) noexcept
- constexpr TinyMatrix() noexcept {}
- constexpr TinyMatrix(const ZeroType&) noexcept
- constexpr TinyMatrix(const IdentityType&) noexcept
- constexpr TinyMatrix(const TinyMatrix&) noexcept = default;
- TinyMatrix(TinyMatrix&& A) noexcept = default;
- ~TinyMatrix() = default;

Figure 5: Exemple de recherche avec ‘swiper’

4.10 Remplacement de texte

Raccourci	Description
M-%	Remplacer les occurrences d'une chaîne par une autre

La recherche des chaînes à remplacer se déroule depuis la position du curseur vers la fin du *buffer*.

Il s'agit d'un mode **interactif**. Pour chaque occurrence de la chaîne à remplacer, Emacs attend une validation.

- Pour remplacer l'occurrence trouvée, taper **y**.
- Pour ne pas la remplacer, presser **n**.
- Pour interrompre le processus de remplacement, taper **q** ou **C-g**.
- Pour tout remplacer, jusqu'à la fin du *buffer*, taper **!**.

Noter que le mode de remplacement est *intelligent* au sens où il respecte la casse si possible.

Si on demande le remplacement de **foo** par **bar** on obtient les résultats suivants.

Chaîne initiale	Résultat
foobar	barbar
barfooFoobar	barbarBarbar
Foobar	Barbar
FOO	BAR

4.11 Répétitions et macros

Il existe de nombreuses façons d'automatiser des actions dans Emacs. Les deux plus simples consistent à répéter une action (**macro** ou autre) ou à créer une macro *interactivement*.

Raccourci	Description
C-u <i>nombre commande</i>	Exécute la commande <i>commande</i> , <i>nombre</i> de fois

Par exemple, si on tape **C-u 10 C-n**, on se déplace de 10 lignes vers le bas dans le fichier.

L'autre manière basique est la création et l'utilisation de macros *clavier*.

Raccourci	Description
C-(Début la création d'une macro
f3	Début la création d'une macro ou insère le compteur
C-)	Termine la définition d'une macro
C-x e ou f4	Exécute la dernière macro créée

Il est donc très facile de créer et d'exécuter une macro clavier avec Emacs. Mais on peut aller facilement un peu plus loin en utilisant les commandes suivantes. On rappelle que pour exécuter la commande *cmd*, il suffit de taper **M-x cmd**.

Commande	Description
insert-kbd-macro	Écrit le code (ELisp) de la dernière macro définie
name-last-kbd-macro	Donne un nom à la dernière macro définie

4.12 Complétion

Emacs propose de nombreux moyens de compléter un mot afin d'accélérer la saisie de texte.

Raccourci	Description
M-/	Complète le mot ou propose une nouvelle complétion

Les complétions proposées sont les mots qui composent les *buffers* actifs. Cela n'est pas toujours suffisant mais permet d'éviter certaines erreurs de frappe.

En fait, cette configuration utilise **company** qui est un moteur de complétion plus souple et configurable. Le fonctionnement de **company** est très intuitif. En effet pendant la saisie de texte, si des complétion sont possibles, une fenêtre *pop-up* qui les propose apparaît (voir figure 6).

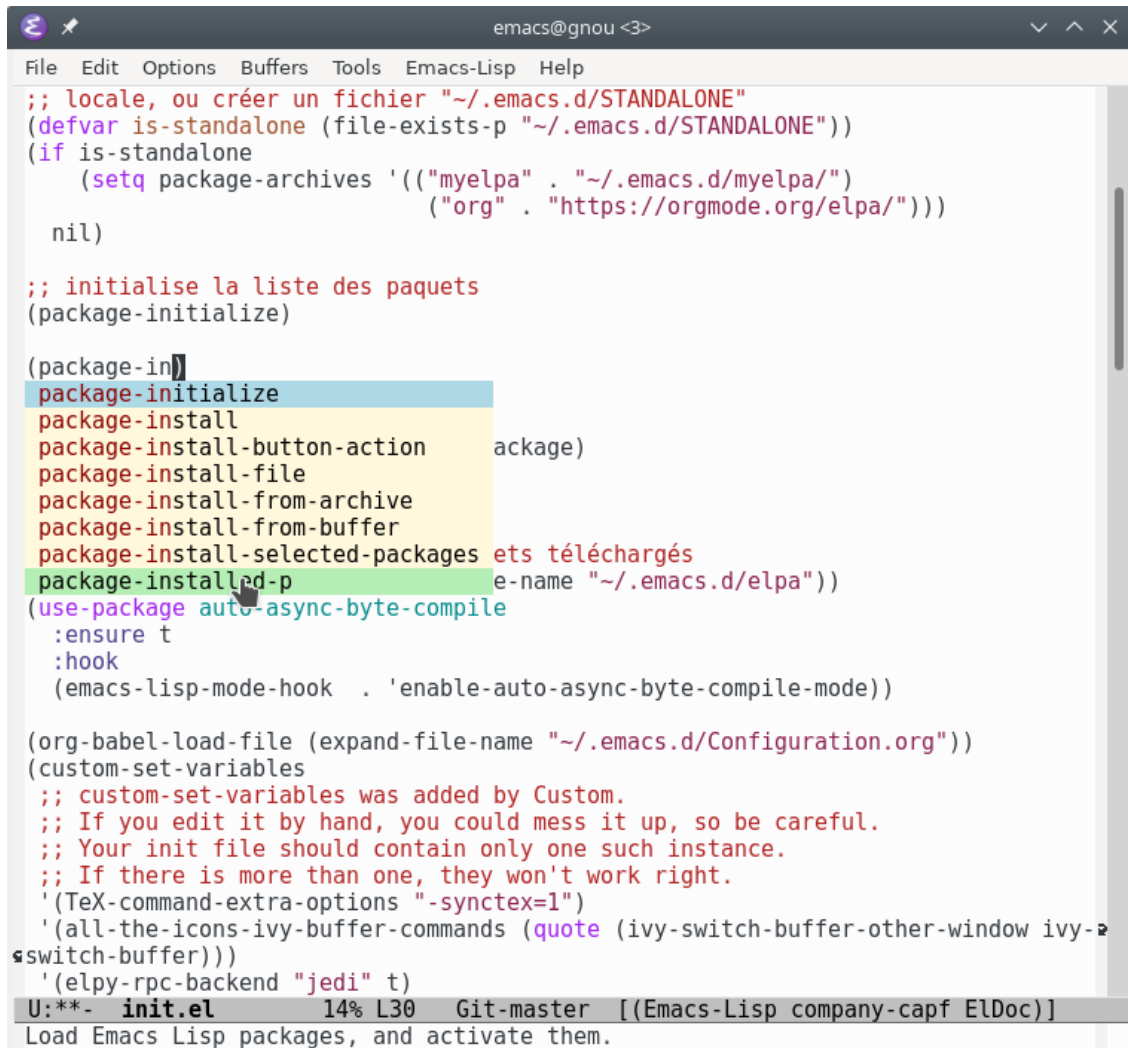


Figure 6: *Pop-up* présentant les complétions possibles

Pour compléter le texte, plusieurs possibilités sont offertes :

- l'utilisation de la touche **TAB** complète le mot au maximum (s'arrêtant après complétion de la partie commune à toutes les alternatives). On peut alors continuer la saisie en entrant le caractère suivant, et ainsi de suite.
- Une autre possibilité consiste à se déplacer dans le menu *pop-up* pour choisir une entrée.

Si l'entrée n'est pas dans la liste, le *pop-up* se ferme automatiquement lors de la saisie.

Enfin, il est possible de quitter le *pop-up* en pressant **C-g** ou en déplaçant le curseur vers la droite ou vers la gauche.

À noter qu'avec cette configuration, la complétion peut-être beaucoup plus évoluée selon le mode majeur actif. Par exemple, les modes de programmation proposent une complétion contextuelle. En mode **LaTeX**, les commandes **L^AT_EX** (qui commencent par un ****) sont proposées (même si elles ne sont pas présentes dans les *buffers* ouverts). De même, en **C++** ou en **Python** par exemple, la complétion est contextuelle (les types et variables accessibles, les fonctions ou fonctions membres sont proposées).

Dans certains cas, la complétion donne accès à un mode de remplissage interactif des arguments de fonctions. On passe d'un argument à l'autre en tapant **TAB** et **S-TAB** (suivant et précédent).

5 Modes utiles

On présente ici quelques modes particulièrement utiles. La liste complète des modes utilisés dans cette configuration (ainsi que quelques raccourcis basiques est accessible dans le fichier `Configuration.org`).

De plus, on ne décrit que brièvement les modes en question, une description complète étant hors de portée d'un tel document et deviendrait inévitablement obsolète.

5.1 Modes utilitaires

5.1.1 Flyspell

Il s'agit d'un mode mineur qui vérifie l'orthographe (uniquement lexicale) dans un document à la volée.

Il est actif dans cette configuration pour certains modes. Par défaut on a choisi le dictionnaire *american* qui est finalement souvent le plus utile.

On peut cependant facilement choisir localement le dictionnaire dans un *buffer* en utilisant la commande `M-x ispell-change-dictionary` et en choisissant le dictionnaire adéquate ou en définissant une variable locale dans le *buffer* pour que ce changement intervienne à chaque fois que le fichier sera ouvert. Par exemple en `LATEX`, on pourra mettre le bloc suivant en fin de fichier.

```
% Local Variables:
% ispell-local-dictionary: "français"
% End:
```

Pour désactiver le `flyspell` dans un fichier, on peut utiliser la configuration locale. Par exemple en `Python`, on écrit

```
# Local Variables:
# eval: (flyspell-mode -1)
# End:
```

5.1.2 FlyCheck

Ce mode mineur permet de vérifier la validité du code à la volée. Il fonctionne pour les langages `LATEX`, `C++`, `Python`, les scripts `shell`,...

Généralement, ce mode n'est pas très consommateur de CPU car la compilation est très légère. Il est cependant possible de le désactiver localement dans un *buffer*, soit en utilisant la commande `M-x flycheck-mode`, soit en définissant une variable locale. En `C++`, on peut par exemple utiliser

```
// Local Variables:
// eval: (flycheck-mode -1)
// End:
```

5.2 Interfaces avec git

Emacs possède différentes interfaces à `git`. Les deux plus abouties sont probablement `magit` et `GitGutter`. Ces deux modes sont complémentaires, `GitGutter` permet d'afficher dans le *buffer* quelles lignes ont été modifiées et de voir rapidement ce qu'on a changé. Quant à `magit`, c'est une excellente interface à `git`. En fait le mode `magit` à lui seul justifie l'utilisation d'Emacs.

5.2.1 GitGutter

Le principe de **GitGutter** est d’afficher (en marge de gauche) les modifications faites au fichier. La mise à jour (pour ne pas surcharger l’outil est effectuée à la sauvegarde du fichier).

Cette configuration interface **GitGutter** avec **Hydra** ce qui permet de simplifier l’utilisation des raccourcis clavier.

Raccourci	Description
M-g M-g	Ouvre le <i>pop-up</i> Hydra pour GitGutter

Une fois qu’on a tapé M-g M-g une fenêtre *pop-up* apparaît et on peut alors suivre les indications pour effectuer les commandes de révision de la version. C’est-à-dire, qu’on peut parcourir les changements, afficher les changements, ou annuler des changements.

On peut également **ajouter** des zones particulières pour préparer un commit **git** mais pour ce genre d’opérations, on préférera utiliser **Magit**.

5.2.2 Magit

Magit est une superbe interface à **git** pour Emacs. Encore une fois, il ne serait pas raisonnable de tenter de fournir une documentation complète du mode dans ce tutoriel. On se contentera de décrire simplement quelques éléments de son fonctionnement. Pour une description plus précise, on se référera au site web de **Magit**. Enfin il est important de préciser que l’aide en ligne de **Magit** est simplement excellente.

Raccourci	Description
C-x g	Lance git dans un autre <i>buffer</i>

Le *buffer* **Magit** (obtenu en tapant donc C-x g) n’est pas un *buffer* d’édition, mais un *buffer* spécial. Dans ce *buffer*, les touches du clavier sont directement surchargées. On a ainsi accès à des nouveaux raccourcis très simples.

Raccourci Magit	Description
h	Affiche l’aide en ligne
TAB	(dé)plie la zone du curseur
RET	Affiche la zone du curseur
g	Rafraîchit le <i>buffer</i>
s	<i>Stage</i> des modifications (git add)
u	<i>Unstage</i> des modifications
c	Démarre un git commit (propose plusieurs options)
F	Lance un git pull (propose plusieurs options)
p	Lance un git push (peut proposer plusieurs options)
k	Annule des modifications ou efface un fichier non suivi

On insiste sur le fait qu’on n’a listé ici que quelques raccourcis des plus utiles. L’aide en ligne, accessible donnée par h est très complète. Il est par exemple très facile de choisir quelques lignes à ajouter (*stager*) en sélectionnant une zone.

5.3 Rédaction de documents L^AT_EX

La rédaction d’un document L^AT_EX s’apparente plus à de la programmation qu’à de l’édition simple de fichiers. De nombreux outils existent maintenant pour éditer des documents L^AT_EX, mais il est également possible de bénéficier de la puissance d’Emacs et de nombreuses années d’expériences qui ont permis le développement de modes qui permettent une excellente intégration Emacs/L^AT_EX.

Avant de décrire les principaux modes, rappelons que cette configuration utilise **FlyCheck** de sorte que les erreurs de syntaxe L^AT_EX sont indiquées à la saisie, ce qui facilite grandement l’écriture. Cette configuration propose aussi le mode **Preview-LaTeX** qui ne sera pas abordé dans ce tutoriel.

5.3.1 AUCTeX

AUCTeX est un ensemble de macros Emacs qui permet d'éditer facilement (et efficacement) des fichiers \LaTeX . C'est le mode qui offre la plupart des fonctionnalités utiles à l'édition \LaTeX .

Nous rappelons ici quelques raccourcis de base. Pour aller accéder à la documentation complète d'AUCTeX, on peut parcourir l'aide intégrée d'Emacs en tapant `C-h i` et en consultant la rubrique qui lui est associée.

1. Typographie Pour les changements de styles typographiques, tous les raccourcis débutent par la séquence `C-c C-f`. Le changement de fonte dépend du contexte de saisie : texte ou mathématique. Noter que si on saisit `C-c C-f` suivi de n'importe quel caractère Emacs affiche la liste des possibilités.

Raccourcis	Style	Comm. (texte)	Style	Comm. (math)
<code>C-c C-f C-a</code>	—	—	calligraphique	<code>\mathcal</code>
<code>C-c C-f C-b</code>	gras	<code>\textbf</code>	gras	<code>\mathbf</code>
<code>C-c C-f C-c</code>	petites capitales	<code>\textsc</code>	—	—
<code>C-c C-f C-e</code>	emphase	<code>\emph</code>	emphase	<code>\emph</code>
<code>C-c C-f C-f</code>	sans serif	<code>\textsf</code>	sans serif	<code>\mathsf</code>
<code>C-c C-f TAB</code>	italique	<code>\textit</code>	italique	<code>\mathit</code>
<code>C-c C-f C-n</code>	normal	<code>\textnormal</code>	normal	<code>\mathnormal</code>
<code>C-c C-f C-r</code>	roman	<code>\textrm</code>	roman	<code>\mathrm</code>
<code>C-c C-f C-s</code>	penché	<code>\textsl</code>	symbole	<code>\mathbb</code>
<code>C-c C-f C-t</code>	machine à écrire	<code>\texttt</code>	machine à écrire	<code>\mathtt</code>
<code>C-c C-f C-d</code>	supprime le style	—	supprime le style	—
<code>C-c C-f C-k</code>	—	—	eufraque	<code>\mathfrak</code>

Noter que si une région est sélectionnée, la commande est appliquée à la région, sinon, le curseur est positionné entre les accolades.

Enfin, si on préfixe les raccourcis précédents par `C-u` alors, le style est remplacé par le style demandé. Par exemple, si on avait le texte suivant

```
\emph{Un texte}
```

si on utilise la commande `C-u C-c C-f C-b`, il sera transformé en

```
\textbf{Un texte}
```

2. Sections AUCTeX propose moyen de créer les éléments structurants du document.

Raccourci	Description
<code>C-c C-s</code>	Crée une section

Une fois ce raccourci utilisé, une interaction dans le *minibuffer* permet d'entrer son titre et éventuellement le label à lui attribuer. Comme toujours en \LaTeX , il est recommandé d'indiquer des noms explicites. On préférera utiliser le caractère - plutôt que _ pour éventuellement séparer les termes qui composent le label.

3. Environnements De la même manière, on écrit des environnements simplement en utilisant des raccourcis.

Raccourci	Description
<code>C-c C-e</code>	Ajoute un environnement

Ici encore, quand le raccourci a été utilisé, les environnements possibles sont proposés dans le *minibuffer* et éventuellement, on propose de saisir un label.

Comme dans le cas des changements typographiques, si une région était sélectionnée, les balises **begin** et **end** entourent la zone. Sinon, le curseur est placé entre les balises.

4. Compilation et visualisation

Raccourci	Description
C-c C-c ou C-c C-z	Accède à une commande \LaTeX
C-c C-k	Tue le processus \LaTeX actif
C-c C-l	Affiche la sortie de compilation

En fait, lorsqu'on invoque **C-c C-c** ou **C-c C-z**, la liste des commandes proposées comprend la compilation du document, la compilation de la bibliographie, la visualisation du document produit, ... Par défaut **AUCTeX** propose l'action qu'il juge naturelle. Par exemple, si le document n'a pas besoin d'être recompilé il proposera sa visualisation.

Cette configuration utilise **pdf-tools** pour visualiser les documents produits. Cela a plusieurs avantages, les **pdf** sont affichés directement dans Emacs (dans une nouvelle **fenêtre**, mais on peut évidemment utiliser un autre **cadre**. De plus, on utilise **synctex** qui permet de synchroniser le source **tex** et le **pdf** produit. Ainsi si on utilise **C-mouse1** ou le double-clic (**mouse1 mouse1**) sur une zone du document **pdf** produit, Emacs positionne le curseur sur la partie du source \LaTeX associé. De même, on peut visualiser la zone du document **pdf** correspondant au curseur dans les sources en utilisant le raccourci **C-c C-g**. On rappelle dans le tableau suivant quelques raccourcis utiles pour la visualisation.

Raccourci	Description
<i>Dans les sources</i>	
C-c C-a	Génère et affiche le pdf (enchaîne toutes les commandes)
C-c C-v	Affiche le pdf s'il existe
C-c C-g	Synchronise la vue du pdf avec les sources au curseur
<i>Dans le pdf</i>	
C-mouse1 (ou double clic)	Positionne le curseur dans les sources correspondantes
PageUp	Aller à la page précédente
PageDown	Aller à la page suivante
SPC	Avance dans le document
DEL	Reculé dans le document
B	Retourne en arrière (page précédemment visitée)
N	Retourne en avant (inverse de B)

5.3.2 RefTeX

AUCTeX est distribué avec **RefTeX** qui est un mode mineur qui permet une gestion simplifiée des références. Que ce soient des références à des équations, des figures, des tables ou des citations bibliographiques. **RefTeX** analyse le fichier source (ou les fichiers si nécessaire) pour déterminer les références possibles.

Raccourci	Description
C-c (Ajoute un label
C-c)	Crée une référence
C-c [Insère une référence bibliographique
C-c /	Ajoute une référence à un index ou un glossaire

Une fois la commande activée, l'interaction est classique et se déroule soit dans le *minibuffer* soit dans un *buffer* annexe.

5.4 Programmation

On l'a déjà dit, cette configuration d'Emacs propose par défaut l'utilisation de `flycheck` qui active une compilation (légère) à la volée du *buffer*, ce qui aide à rapidement trouver les erreurs de syntaxe.

La plupart des modes de programmation nécessitent l'installation d'outils annexes pour fonctionner. On se reportera donc au fichier `Installation.org` pour des détails à ce sujet.

5.4.1 Python

On utilise le paquet `jedi` qui gère la complétion dans Emacs pour `Python`. On se reportera donc à la documentation en ligne <http://tkf.github.io/emacs-jedi/latest/>, pour plus d'informations.

Raccourci	Description
<code>C-c C-c</code>	Exécute le script
<code>C-c C-z</code>	Passe du <i>buffer</i> du code à la console d'exécution ou inversement
<code>S-TAB</code> ou <code>backtab</code>	Supprime une indentation

Il existe également, une interface à `Jupyter`, mais cette dernière ne fait pas partie de cette configuration.

Enfin, si on exécute les scripts dans Emacs `C-c C-c`, le terminal occupe un *buffer* interactif (c'est-à-dire qu'on peut y entrer des commandes comme dans n'importe quel terminal), où il est possible, comme dans les *buffers* de compilation de cliquer sur les lignes d'erreur pour positionner le curseur dans la ligne correspondante des sources.

5.4.2 C++

Pour la complétion et la correction syntaxique à la volée, on utilise le paquet `irony`. Il est nécessaire pour cette fonctionnalité de configurer correctement le système et en particulier de disposer d'une version adaptée du compilateur `clang` (voir les consignes d'installation).

Raccourci	Description
<code>C-c C-c</code>	Affiche l'invite de compilation dans le <i>minibuffer</i>
<code>C-c '</code>	Positionne le curseur à la première erreur de compilation

Le *buffer* de compilation est interactif : si on clique sur un message d'erreur, les sources sont ouvertes et le curseur est positionné à l'endroit correspondant. Il est donc très pratique de compiler dans Emacs plutôt que dans un terminal.

On notera également que si on laisse le curseur positionné sur un mot du code source, le *minibuffer* affiche des informations concernant le type ou l'objet en question.

5.4.3 Tags

Les *tags* (ou étiquettes en français) sont utilisés pour parcourir facilement les sources d'un code. Cette installation est configurée pour fonctionner avec `GNU Global` (voir <http://www.gnu.org/software/global/>) qui indexe et exploite des bases de *tags*. Une telle base est une base de données qui répertorie des informations sur les sources du code : quel est le type d'une variable, où est-elle déclarée ou utilisée ? Quel est le fichier qui définit une classe, un type, etc... L'utilisation d'une telle base permet donc de facilement accéder à de nombreuses informations sans avoir à faire de recherches par ailleurs.

Comme cette configuration utilise les `ggtags` et repose donc sur le logiciel tiers `GNU Global`, il convient de l'installer. Pour cela, on se référera au fichier `Installation.org`.

Avant de rappeler quelques raccourcis usuels de ce mode, rappelons qu'à la première utilisation des `ggtags` pour un code, la base sera créée.

Enfin précisons que les *tags* ne sont qu'une base de données d'occurrences de textes et même si elle est plutôt efficace, il est parfois possible que les références ne correspondent pas au même objet.

Raccourci	Description
M-.	Cherche un <i>tag</i>
C-M-.	Cherche les <i>tags</i> qui correspondent à une expression régulière
M-,	Continue à chercher un <i>tag</i>
C-c M-%	Remplace un <i>tag</i> par une nouvelle chaîne dans tout le code
C-c M-g	Lance un grep dans la base des <i>tags</i>
C-c M-f	Ouvre un fichier référencé dans la base des <i>tags</i>

Dans ce contexte, on rappelle que pour revenir dans le *buffer* il est conseillé d'utiliser les raccourcis de navigation classiques :

Raccourci	Description
C-x left	Retourne au <i>buffer</i> précédent
C-x right	Retourne au <i>buffer</i> suivant

5.4.4 YASnippet

Il n'est pas envisageable d'écrire un tutoriel Emacs sans évoquer les **snippets**. Commençons donc par rappeler de quoi il s'agit. Les **snippets** (littéralement **fragments** en français) sont des patrons (*template*) de code qui évitent de taper toujours et encore des motifs de code qui se répètent. Emacs offre depuis très longtemps des **snippets** qui permettent souvent de n'écrire que les zones du code faisant sens. **YASnippet** s'est imposé comme la version la plus aboutie et est maintenant la plus utilisée.

Il faut bien intégrer le fait que les **snippets** ne sont pas des morceaux de code simplement insérés mais de véritables modèles. Quand on les utilise, Emacs invite l'utilisateur à renseigner certains champs (en ligne) ce qui permet d'obtenir directement le code correspondant au patron sans autre modification.

Notons également que **YASnippet** est livré avec une collection impressionnante de modèles. Ces patrons sont accessibles selon le contexte d'édition : les patrons utiles au **Python**, à la rédaction **L^AT_EX** ou au **C++**, par exemple ne sont proposés que dans leur mode. Au vu de leur nombre, il serait impensable d'en faire une quelconque liste ici, mais on peut en avoir la liste dans le menu **YASnippet** ou en tapant la commande **M-x yas-describe-tables** qui ouvre un buffer contenant tous les **snippets** disponibles.

Un autre avantage de **YASnippet** est qu'il est très facile d'ajouter de nouveaux modèles, mais nous ne décrivons pas la procédure ici (voir <http://joaotavora.github.io/yasnippet> pour les détails).

Raccourci	Description
C-RET	Développe le modèle

Dans cette configuration, on a désactivé le raccourci usuel **TAB**, qui pouvait avoir des effets inattendus et désagréables, au profit de **C RET**. En effet, l'effet principal de **TAB** est l'indentation du code, mais si le *mini-mode* de **YASnippet** interceptait un développement de modèle possible, il prenait la main et remplaçait l'indentation demandée par une expansion de modèle. Le nouveau raccourci reste simple et n'interfère plus avec les fonctions usuelles.

On termine cette courte introduction en donnant un exemple d'utilisation en **C++**. Si on écrit **cls** et si on utilise immédiatement le raccourci **C-RET**, le modèle (simple) de **classe** est utilisé. On obtient alors

```
class Name
{
public:
    Name();
```

```
virtual ~Name();  
};
```

Le curseur est alors positionné sur le premier `Name` qui est en fait éditable. On est toujours en train d'appliquer le modèle et il suffit maintenant (sans autre action) d'entrer le nom de la classe. Par exemple, si on écrit `Tableau` (une seule fois), on obtient

```
class Tableau  
{  
public:  
    Tableau();  
    virtual ~Tableau();  
};
```

On n'a modifié qu'un champ pour obtenir ce résultat. `TAB` permet de passer au champ suivant s'il y en a d'autres et `S-TAB` (ou `backtab`) permet de revenir à l'édition d'un champ précédent. Pour obtenir ce résultat, on a simplement pressé la suite de touches suivantes.

```
c l s C-R E T a b l e a u
```

Cet exemple simple illustre bien l'efficacité de l'approche.

6 Pour aller plus loin

6.1 Org-mode

Une fois encore, il est inenvisageable de faire un tour complet de ce mode dans un tel document. On va donc seulement en tracer les grandes lignes et décrire certaines de ses fonctionnalités. Pour vraiment en appréhender les possibilités, il faut consulter les multiples ressources disponibles sur le net, et en particulier des vidéos de démonstrations.

Initialement, `Org-mode` est un outil conçu pour la gestion, le suivi et l'archivage de tâches personnelles. La gestion d'agenda n'en est plus aujourd'hui qu'un aspect mineur. Son intégration à Emacs est extrêmement poussée¹ et tire profit de l'interopérabilité des différents modes d'Emacs. Il bénéficie donc d'une extensibilité assez spectaculaire. `Org-mode` définit un véritable langage de (méta-)programmation et utilise son propre format de fichier `.org`.

Même si un document `Org-mode` est **lisible** avec n'importe quel éditeur de texte, de nombreux rendus sont possibles : `Markdown`, texte (ASCII ou UTF-8), `PDF` (générés en `LATEX`), `HTML`, `OpenDocument`,... On peut publier des *blogs*, générer des présentations (`Beamer` ou autres). Et `Org-mode` permet beaucoup d'autres choses encore.

L'une des forces d'`Org-mode` est la possibilité d'insérer des blocs de code (dans de nombreux langages) directement dans le document. `Org-mode` peut alors les évaluer en les compilant ou en les exécutant par exemple. Dans un même document, on peut par exemple utiliser des programmes `C++` ou `Python`, des script `shell`, des commandes `LATEX`, des instructions `gnuplot`, tracer des graphes ou des diagrammes avec `dot`,... `Org-mode` permet en plus de faire communiquer ces différents programmes. Dans un document `Org-mode`, il est possible d'afficher des images en ligne. On peut aussi créer des tableaux dynamiques (provenant d'évaluations de blocs de code par exemple) ou statiques. On peut écrire des formules mathématiques en ligne, c'est à dire sans définir des zones de code source, en utilisant directement des formules `LATEX` (`$(a+b)^2 = a^2+2ab+b^2$` est interprété directement $(a + b)^2 = a^2 + 2ab + b^2$),...

Ce document (et ce n'est qu'un exemple **très** simple) est rédigé en `Org-mode`. C'est aussi le cas du fichier de configuration (`Configuration.org`). Cette pratique est devenue un standard de nos jours. On préfère écrire une configuration en `Org-mode` plutôt que directement en `Elisp`. Cela donne

¹`Org-mode` n'existe que dans Emacs. Il n'y a pas d'application autonome qui offre les mêmes fonctionnalités. Cela n'aurait pas vraiment de sens.

une véritable structure au document et donc, en améliore beaucoup la lisibilité. Au delà, on peut même exporter le contenu (en HTML par exemple) pour obtenir une véritable documentation de la configuration. Notons aussi que `Gitlab` et `Github` affichent raisonnablement le contenu des fichiers `.org` (sans toute fois interpréter les zones de code ou les équations $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$). C’est commode pour partager une configuration.

Dans le cas d’un fichier de configuration, ces blocs sont principalement des morceaux de codes `ELisp` dont l’évaluation modifie le comportement d’Emacs. Mais, en général, il est aussi possible de récupérer le résultat de l’exécution et de l’insérer automatiquement dans le document. De ce fait, le document est extrêmement dynamique. La rédaction de documentations ou de tutoriels par exemple est donc grandement facilitée, puisqu’Emacs vérifie par l’évaluation du document `Org-mode` que tous les morceaux de code sont valides et met à jour les résultats.

Par certains aspects, `Org-mode` est similaire à certains langages de description de contenus comme par exemple `Markdown`, puisqu’il est très simple de définir des hiérarchies de sections, des listes, des hyperliens, de créer des tableaux,...

Par exemple pour créer une section de niveau n , on débute une ligne avec n étoiles (\star) consécutives suivies d’un espace et du titre. La création d’une liste est tout aussi simple, il suffit par exemple de commencer les lignes par un tiret ($-$), l’indentation définit le niveau de l’item.

On termine cette très brève introduction par un rappel de quelques raccourcis basiques.

Les raccourcis suivants fonctionnent pour les sections et les listes. Les mêmes raccourcis ont des effets voisins dans les tableaux, voir plus bas.

Raccourci	Description
M-up	Échange l’élément avec l’élément précédent (de même niveau)
M-down	Échange l’élément avec l’élément suivant (de même niveau)
M-left	Diminue le niveau de l’élément (plus haut dans la hiérarchie)
M-right	Augmente le niveau de l’élément (plus bas dans la hiérarchie)
TAB	Plie ou déplie le contenu
S-left ou S-right	Modifie le statut de l’élément (<code>TODO</code> , <code>DONE</code> ,...)
M-RET	Crée un élément de même niveau que l’élément courant

Dans le cas des listes, **S-left** et **S-right** modifient le type d’élément de la liste (et des éléments de même niveau), on peut ainsi obtenir une numérotation par exemple.

Enfin, si on utilise **M-RET** au milieu d’un élément, il est décomposé en deux éléments, le contenu reste associé à la seconde partie de l’élément.

Insistons sur le fait qu’en cas de déplacement d’un élément, son contenu lui reste attaché (les sous items en particulier). Dans le cas des listes numérotées, la numérotation est maintenue à jour par toutes les opérations.

Dans le cas des tableaux, comme on l’a dit, ces raccourcis sont réutilisés avec des effets très voisins

Raccourci	Description
M-up	Échange la ligne avec la ligne précédente
M-down	Échange la ligne avec la ligne suivante
M-left	Échange la colonne avec la colonne précédente
M-right	Échange la colonne avec la colonne suivante
TAB	Positionne le curseur sur la cellule suivante (ajoute une ligne si nécessaire)
S-TAB ou backtab	Positionne le curseur sur la cellule précédente

En fin de ligne (après le dernier `|`), si on presse **TAB**, alors le tableau est réaligné. Il existe de nombreux autres raccourcis pour les tableaux, (suppression/ajout de lignes ou de colonnes) mais comme certains sont dangereux nous ne les rappelons pas ici. Ces raccourcis sont rappelés dans le menu `Tbl`.

Nous terminons cette description des tableaux en donnant **un** moyen de les créer. Il est extrêmement simple. Il suffit de commencer une ligne par un `|` puis de séparer les colonnes par d’autres `|`.

Pour ajouter une ligne de séparation, il suffit de commencer une ligne par la séquence `| -` puis de presser `TAB` n'importe où dans la zone du tableau.

Finalement, on donne quelques raccourcis plus généraux pour **Org-mode**. On ne décris pas ici comment générer des zones de code, ni autres comportements dynamiques. Les fonctionnalités basiques sont accessibles dans le menu, mais il est préférable pour cela de consulter des documentations dédiées (voir par exemple orgmode.org).

Raccourci	Description
<code>C-c C-c</code>	Évalue une zone de code et met à jour le résultat
<code>C-c C-v C-b</code>	Évalue tout le document et met à jour les résultats
<code>C-c C-e</code>	Lance le menu Hydra pour déclencher les exports
<code>C-c '</code>	Édite une zone de code dans le mode approprié

Ce dernier raccourci : `C-c '` est **extrêmement** important pour la modification des blocs de codes. Il faut éviter de modifier directement ces éléments avec le mode **Org**.

6.2 Pliage et dépliage avec `narrow-or-widen-dwim`

Emacs propose des fonctions de (dé)-pliage de zones. La dénomination pour ces fonctionnalités dans Emacs est *narrowing* (pliage) et *widening* (dépliage).

Il s'agit d'un outil très puissant pour l'édition avec Emacs. Son fonctionnement n'est pas de cacher des zones de texte comme le proposent de nombreux outils, mais de modifier le *buffer* pour qu'il ne contienne plus que la région sélectionnée. Le *buffer* **n'est donc plus** le reflet du fichier édité, mais simplement le reflet de la zone.

Les modifications apportées à la zone sont donc bien sauvées dans le fichier, mais on ne peut plus modifier le reste du fichier. C'est donc très pratique lorsqu'on veut appliquer une série de modifications sans avoir à se préoccuper de leur impact hors de la région. Un bon exemple est l'utilisation d'une macro clavier.

Dans cette configuration, on utilise `narrow-or-widen-dwim` qui simplifie la sélection des régions à éditer, c'est l'effet *dwim* (*Do What I Mean*). Un seul raccourci est nécessaire à l'activation du pliage ou du dépliage d'une zone.

Raccourci	Description
<code>C-x n</code>	Active ou désactive le pliage

Là où ce mode est intelligent c'est en sa façon de reconnaître quelle est la région qui sera éditée. En fait, si une région est sélectionnée c'est cette région qui sera considérée, mais sinon, c'est le paragraphe, la section ou la fonction courante (dans un mode de programmation) même qui occupera tout le *buffer*.

6.3 Autres modes installés

En dehors des modes internes fournis avec Emacs, on pourra consulter le fichier de configuration (Configuration.org) où sont décrits succinctement des modes plutôt utiles comme **WGrep**, **iedit**, **Dumb Jump**, **AutoYASnippet**, **Avy**, **projectile**, ...

Ils ne sont pas abordés dans ce tutoriel.

6.4 Personnalisation

6.4.1 `custom.el`

Même si on souhaite pouvoir suivre les développements de cette configuration et les mettre à jour avec **git**, il est possible de la personnaliser un peu.

Pour cela, les modifications doivent être apportées au fichier `~/.emacs.d/custom.el`. En fait, ce fichier particulier est ignoré par **git**, mais il est généré au lancement d'Emacs s'il n'existe pas.

Une façon de **supprimer** toutes les personnalisations est donc de l'effacer puis de relancer Emacs. Si des personnalisations sont importantes, il convient donc de les sauvegarder ailleurs.

6.4.2 Emacs lisp

Rappelons le, Emacs n'est pas un simple éditeur. C'est en fait une véritable machine interprétant le langage **ELisp** dont des fonctionnalités sont adaptées à l'édition de sources.

Donc pour une personnalisation (extrêmement) poussée, ou pour développer de nouveaux modes, il est indispensable de connaître suffisamment **ELisp**. Évidemment, on ne couvre pas cet aspect des choses dans ce document. Il faudra donc se référer aux nombreuses documentations disponibles sur internet.

Voici quelques références, classées par ordre croissant de complexité.

- Un guide rapide des bases d'ELisp
- Practical Emacs Lisp
- Manuel de référence