Emacs pour l'impatient

Février 2020

Emacs en quelques mots

Emacs est un éditeur de texte, voire un sys d'exploitation part en diront certains. Des débats enflammés ont lieu régulièrement entre les partisans d'Emacs et ceux de Vi(m), un autre éditeur Unix versatile, tout comme on retrouve des discussions sans fin sur l'usage des taquets de tabulation ou des espaces pour l'indentation des blocs en Python. Et bien sûr, Richard Stallman, adossé à la FSF, n'est jamais bien loin lorsque l'on évoque le logiciel Emacs, même si pour l'essentiel la maintenance et le développement d'Emacs est réalisé par un groupe de volontaires bénévoles.

> I spend most of my time editing in Emacs. I read and send mail with Emacs using M-x rmail and C-x m. I have no experience with any other email client programs. In principle I would be glad to know about other free email clients, but learning about them is not a priority for me and I don't have time. — Richard Stallman, How I do my computing

Ceci étant, l'objet de ce document n'est pas d'arguer pour ou contre l'usage de Emacs pour réaliser des tâches plus ou moins complexes. Il n'est pas non plus question de fournir un manuel détaillé concernant l'utilisation d'Emacs et les milles et quelques raccourcis clavier dont dispose Emacs dans sa configuration de base. Pour cela, il est possible de consulter la documentation officielle, les ouvrages publiés chez O'Reilly, par exemple Learning GNU Emacs, ou les nombreux liens que l'on trouve sur internet : Mastering Emacs ou Practical Emacs Tutorial ; Org Mode - Organize Your Life In Plain Text! et Emacs org-mode examples and cookbook pour le mode org plus spécifiquement.

L'objet de ce document est somme toute assez modeste et on s'intéressera essentiellement à quelques "modes" spécifiques, en particulier ceux qui se révèlent le plus utile pour éditer efficacement des documents texte et interagir interactivement avec des shells ou des environnements de type {REPL | Read Eval Print Loop}. L'auteur utilise personnellement Emacs depuis une vingtaine d'années, en connaît le strict minimum pour éditer une grande variété de documents, et se contente la plupart du temps d'utiliser Emacs comme un excellent éditeur de texte. En guise d'illustration, la figure 1 permet de voir à quoi ressemble l'édition de ce document sous Emacs avec le mode org. En réalité, il ne s'agit pas tout à fait de la version de base d'Emacs, encore appelé Emacs vanilla, mais de Doom Emacs, une configuration particulière d'Emacs incluant un thème et des options bien

spécifiques. On se concentrera d'ailleurs sur cette version d'Emacs dans les prochaines sections.

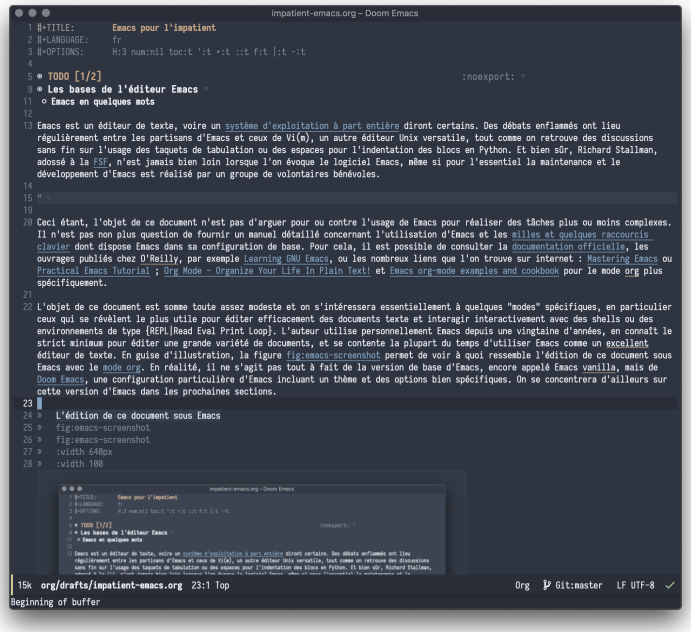

Figure 1: L'édition de ce document sous Emacs

Les bases de l'éditeur Emacs

Installation de Doom Emacs

L'installation de Doom Emacs ne pose pas de problème particulier, et tout est clairement détaillé sur Github. Il est recommendé de bien lire la FAQ pour les problèmes les plus courants, et de consulter régulièrement les tickets, notamment en cas de soucis après une mise à jour. A l'image de Spacemacs, Doom Emacs repose sur le mode Evil, même s'il est possible de le désactiver. Ceci étant, en mode insertion, la plupart des commandes Emacs classiques sont disponibles, et donc la différence se voit essentiellement lorsque l'on se retrouve en mode normal ou visuel. Attention toutefois, certaines combinaisons de touches (p.ex. dans le mode calendar ou org) ont été réaffectées pour respecter le mode Evil (voir la section Navigation et mouvements de base).

Pour installer Emacs sous macOS avec Homebrew, il suffit de taper l'instruction suivante :

```
brew install emacs-plus --without-spacemacs-icon --with-no-titlebar \
    --with-mailutils --with-jansson --with-emacs-27-branch --HEAD
```

Depuis 2020, les formules Homebrew ont été mises à jour de sorte qu'il suffit de taper `brew install emacs-plus@27` ou `emacs-plus@28` pour installer le version courante (27.1 depuis août 2020) ou la version de développement (`HEAD`, compilée depuis Github). La version 27 d'Emacs est conseillé pour bénéficier des meilleures performances avec le mode `lsp`, notamment en raison de l'option de traitement des fichiers JSON.

Emacs est parfaitement fonctionnel dans un terminal, et il est possible d'appliquer le même thème de couleur qu'en mode GUI sous réserve d'utiliser un terminal en mode 24 bits. La documentation de Doom Emacs fournit les instructions aire. [1]

[1] Sans utiliser de configuration spécifique, sous réserve de disposer du bon fichier `terminfo`, on peut toujours lancer Emacs en mode terminal de la manière suivante : `env TERM=xterm-24bit emacs -nw`.

Particularités de Doom Emacs

leader key = `SPC` local leader key = `SPC m`

TL;DR

Encore une fois, il n'est pas question de transformer Emacs en une application tout en un — pour cela, il suffit de consulter l'excellente page citée plus haut, Org Mode - Organize Your Life In Plain Text!, mais à l'exploiter comme une boîte à outils centrale dans son travail quotidien. Il est tout à fait possible de n'utiliser Emacs que pour éditer des fichiers texte, voire même en faire l'outil de base de son travail d'écriture. L'auteur de *Mastering Emacs*[2] fournit d'ailleurs quelques conseils bien utiles pour rédiger un ouvrage sous Emacs.

[2] Petersen 2015, l'auteur fournit par ailleurs de nombreux exemples de configuration pour Emacs dans la plupart des modes sur son site personnel.

Voici trois des aspects d'Emacs qui le distingue des autres éditeurs largement utilisés pour éditer du texte ou du code source. Premièrement, un des immenses avantages d'Emacs est la possibilité de lier n'importe quel shell ou process à un buffer dans lequel se trouve un document en cours d'édition (document Org ou code source). Il est alors possible d'évaluer directement des expressions ou des instructions d'un code donné à partir du document source et d'insérer les résultats dans ce même document. A l'extrême, le texte remplace les commentaires d'un code source et on a véritablement un document avec un corps de texte, des éléments balisés contenant du code source et les sorties produites par ce même code source. On parlera de programmation lettrée (en anglais, *literate programming*) pour désigner cette approche :

> Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do. — Donald E. Knuth

Emacs peut donc d'ores et déjà servir d'éditeur de code et de doc-

umentation, quelle que soit la manière dont on envisage la relation entre ces deux constituant d'un programme informatique. Le mode Org, par exemple, offre une version modernisée de Noweb.

En plus de documents interactifs, Emacs permet de gérer un agenda ainsi que son courriel, organiser des listes de tâches à faire, naviguer dans un système de fichier local ou distant, gérer des projets informatiques à plus grande échelle tout en facilitant l'interaction avec un gestionnaire de version tel que Git.

Enfin, si Emacs dispose de paquets dédiés à l'édition de documents en format structuré tels que Markdown ou reStructured text, le mode Org fournit un ensemble incroyable de ressources, tant pour la programmation lettrée que pour l'organisation d'une session de travail. En ce sens, il réunit en un seul paquet les deux points évoqués plus haut, à savoir la possibilité de combiner du texte et du code auto-évalué dans un même document, tout en gérant un projet informatique bien au-delà de ses aspects purement techniques.

Navigation et mouvement de base

Organisation des fichiers et des buffers

Ce que l'on appelle un buffer, comme sous Vim, correspond en réalité à une vue d'un fichier du système de fichiers lui-même. Ainsi, le même fichier physique peut être affiché dans différents buffers, voire dans différentes fenêtres d'Emacs. Ceci étant, avec Doom Emacs les fenêtres sont regroupées sous la forme de "workspaces".

La gestion des buffers se fait à partir du menu `SPC-b`. mais on dispose également des raccourcis Emacs standards, `C-x f` (ouvrir) et `C-x k` (fermer), ainsi que de quelques autres raccourcis bien pratiques : `SPC SPC` et `SPC ,`.

Mouvements de base

Il existe un excellent tutoriel de Guy Lapalme, GNU-EMACS - Presentation simplifiee, qui couvre l'essentiel des commandes permettant de se déplacer dans un fichier et des raccourcis associés. Spacemacs offre un mode hybride qui permet de conserver les raccourcis Emacs en mode insertion et les raccourcis Vim en mode normal. Avec Doom Emacs, le mode Evil qui permet d'émuler le comportement Vim sous Emacs peut être activé de manière globale et donc on dispose également de cette distinction de raccourcis selon le mode d'édition courant (insertion versus normal ou visuel, principalement). Notons toutefois que tous les raccourcis Emacs ne sont pas respectés en mode insertion : par exemple, `C-n` décrit ci-après ne fonctionnera pas sous Doom.

Dans ce qui suit, on utilisera le mode hybride qui permet d'utiliser tous les raccourcis Emacs en mode *insertion*, et les raccourcis Vi en mode *normal*. Voici un moyen mnémotechnique pour retenir les commandes de déplacement sur une ligne de texte : la direction du déplacement se contrôle avec `b` pour "backward" (reculer) et `f` pour "forward" (avancer), l'unité de déplacement est soit le caractère (`C` ou `^`) soit le mot (`M` ou `ESC`). Les commandes `C-a` et `C-e`, également très utiles dans un terminal Unix, permettent d'aller en début et en fin de ligne (physique) ; avec `M-a` et `M-e`, le déplacement porte sur la phrase entière. Quant à `M-<` et `M->`, cela permet d'aller au tout début ou à la toute fin du tampon ou "buffer". Voici une illustration sur un texte arbitraire dans lequel la position du curseur est représentée par le symbole `|` :

```
Vivre, c'est passer d'un e|space à un autre en essayant le plus possible de ne pas se cogner.
^                          ^     ^                                                          ^
C-a                      C-f   M-f                                                       C-e

Vivre, c'e|st passer d'un espace à un autre en essayant le plus possible de ne pas se cogner.
^        ^
M-b     C-b
```

On passe d'une ligne à la suivante ou à la précédente à l'aide de `C-n` et `C-p`, et d'un paragraphe à l'autre à l'aide de `M-}` et `M-{`. En pratique, comme on peut utiliser les flèches du clavier pour se déplacer d'un caractère à l'autre, ou d'une ligne à l'autre, seules les commandes permettant de naviguer entre les mots ou les blocs de mots (phrases ou paragraphes) se révèlent vraiment intéressantes.[3]

[3] Notons que avec Doom Emacs, les raccourcis `C-n` et `C-p` sont déjà réservés.

On trouvera dans le document Vim 101 une description exhaustive du mode Evil et des raccourcis associés, principalement pour la sélection et les déplacements dans un document. Pour l'essentiel, l'édition du texte se déroule en mode insertion (`i` pour y accéder depuis le mode normal), et les mouvements ou transformations de texte sont réalisées en mode normal (`ESC` pour y accéder depuis le mode insertion). On dispose également d'un mode visuel (`v` pour y accéder depuis le mode normal) ou de remplacement (`R` pour y accéder depuis le mode normal). Les principaux raccourcis pour se déplacer sont indiqués dans le tableau 1 (reproduit de Vim 101).

Fonctions avancées

Recherche simple

Voici les principales manières de rechercher du texte ou un symbole dans un document : `/`, `SPC-/`, `*` et `SPC-*`. Les deux premières reposent sont des outils assez rapides pour marquer les occurrences

Clavier	Description
b	Déplacer le curseur au mot précédent
w	Déplacer le curseur au mot suivant
0	Aller en début de ligne
$	Aller en fin de ligne
gg	Aller au début du buffer
G	Aller à la fin du buffer
:X	Aller à la ligne numéro x
f<char>	Aller à la prochaine occurrence du caractère
F<char>	Aller à la précédente occurrence du caractère
C-u	Aller à l'écran précédent
C-d	Aller à l'écran suivant
o	Alterner entre le début et la fin de la sélection visuelle
%	Alterner entre le début et la fin des délimiteurs appariés
(	Aller au début du paragraphe
)	Aller à la fin du paragraphe
{	Aller à la prochaine ligne vide
}	Aller à la précédente ligne vide

Table 1: Raccourcis de base pour les déplacements en mode visuel

d'un motif en utilisant le raccourci Vim ou `swiper` du package Ivy (voir section suivante). Dans le premier cas, toutes les occurrences sont surlignées de manière incrémentale à mesure que le motif de recherche est trouvé ; une fois qu'on a obtenu le résultat désiré, il suffit de valider en appuyant sur entrée, et à partir de là il est possible de visiter chaque occurrence en appuyant sur `n`. Le raccourci `SPC-/` (ou `SPC-s b`) permet d'appeler swiper (ou swiper-all) permet de réaliser la même chose mais en affichant chaque ligne contenant le motif recherché dans un minibuffer. Les deux dernières instructions, `*` et `SPC-*`, permettent de surligner le symbole sous le curseur dans tout le document, ou dans tout le projet, respectivement.

Ivy

On a vu dans les paragraphes précédents deux des principaux outils de recherche textuelle proposé par le package Ivy.

Utiliser un terminal

Emacs fournit un véritable shell écrit en Lisp, `eshell`, et des émulateurs de terminal (`term`, `ansi-term`, `multi-term`). Eshell présente ses intérêts, que l'on discutera plus loin, mais si l'on souhaite travailler avec un véritable terminal et non un émulateur sous Emacs, il est conseillé d'installer le package vterm. Attention, il s'agit d'un véritable programme externe, qui doit être compilé avant de pouvoir l'utiliser. Ceci est également valable après chaque mise à jour du package ou lorsque l'on change de version d'Emacs, à l'image de `pdf-tools`.

L'utilisation d'un shell sous Emacs peut paraître inutile dans la mesure où il est possible d'attacher un process à un buffer actif, par exemple un shell avec R ou Python pendant que l'on édite un document R Markdown ou un script Python, et puisque le mode `dired` permet d'opérer sur le système de fichiers de manière relativement efficace. Toutefois, cela évite dans bien des cas de lancer un terminal à côté, et le transfert de données (copier-coller de régions par exemple) est beaucoup plus simple lorsque le shell est embarqué dans Emacs directement.

Les sections qui suivent indiquent comment configurer et utiliser les deux principaux shell interactifs sous Emacs.

Utilisation de eshell

Contrairement à term et ses variantes, Eshell contitue un véritable shell, et pas seulement un émulateur. Toutes les commandes habituellement disponibles (`cd`, `ls`, `mkdir`, etc.) ont été réécrites en Lisp. L'intérêt d'Eshell par rapport aux autres émulateurs ou pseudo-émulateurs de terminal est qu'il autorise l'usage de commandes Emacs-Lisp. par exemple, il est possible d'appeler la fonction `magit-status` du package Magit directement dans le shell. Plus généralement, il est possible de définir des "alias" pour Eshell. Voici par exemple ce que l'on peut mettre dans son fichier de configuration (`doom.d/config.el`) :

```
(after! eshell
  (set-eshell-alias!
   "f"   "(other-window 1) && find-file $1"
   "l"   "ls -lh"
   ".."  "cd ../"
   "d"   "dired $1"
   "gl"  "(call-interactively 'magit-log-current)"
   "gs"  "magit-status"
   "gc"  "magit-commit"))
```

Configuration de vterm

Le package vterm est assez récent et il repose sur la librairie libvterm développée par les concepteurs de Neovim, qui propose un véritable émulateur de terminal contrairement aux anciennces version de Vim. Par défaut, vterm utilisera le shell défini dans la variable d'environnement $SHELL, si elle est renseignée. Pour redéfinir le shell utilisé par vterm, il suffit de mettre à jour son fichier de configuration (`doom.d/config.el`) :

```
(setq vterm-shell "/bin/zsh")
```

A partir de là, il n'y a vraiment plus aucune différence entre utiliser Zsh depuis Emacs ou avec une application de terminal. On prendra garde au fait que le shell Fish nécessite quelques étapes de configuration supplémentaires.

Gestion des répertoires avec Dired

Bien qu'il soit tout à fait possible de naviguer dans son système de fichiers en utilisant un terminal ou un explorateur de fichier externe, Emacs fournit un outil très puissant pour gérer ses fichiers : le mode `dired`.

Gestion de projets avec Projectile

Les commandes de gestion de projets sont accessibles à l'aide du raccourci `SPC-p`.

SPC-p p

Gestion de l'agenda et des notes

Gestion du courriel

Gestion de documents texte

Le mode texte simple

Emacs offre les mêmes fonctionnalités d'édition de texte simple qu'un éditeur tel que vi(m), Sublime, Atom ou VS Code.

Plutôt que de laisser courir le texte indéfiniment sur la même ligne (il s'agit du mode `longlines-mode`), il est possible de formater l'affichage du texte dans le buffer à l'aide de modes mineurs. On distingue alors principalement le mode avec arrêt automatique sur le bord de la fenêtre (`visual-line-mode`) ou à un certain nombre de caractères (`auto-fill-mode`). Dans le premier cas de figure, cela n'affecte pas le rendu final du document texte (chaque ligne reste disposée sur une seule et même ligne physique), alors que dans le second cas de figure le document final est vraiment formaté selon le nombre de colonnes spécifiées, généralement 80 charactères pour respecter les limitations de certains terminaux.

Pour le reste, on dispose des commandes de base de Emacs concernant n'importe quelle saisie de texte. Il peut exister des subtilités selon, par exemple, que le mode "électrique" est activé ou non,

En mode d'édition texte (cela est valable également dans le cas de documents Markdown ou Org), il est toujours possible d'activer

le correcteur orthographique qui généralement repose sur le programme `ispell` ou son équivalent `aspell`. Il peut être nécessaire de modifier le dictionnaire choisi par défaut à l'aide de la commande `ispell-change-dictionnary`. Lorsqu'un mot présente une erreur, il est sous-ligné et il est alors possible de le corriger en tapant `M-$` (`ispell-word`) : une sous-fenêtre propose différents choix possibles pour le remplacement et il suffit d'indiquer le numéro correspondant ou à défaut de taper sur la touche entrée.

Le dictionnaire sélectionné par défaut est défini à partir de la variable d'environnement `$LANG` du terminal, ou de la variable `flyspell-default-dictionary` si elle est définie. Dans le cas où l'on est amené à éditer fréquemment des documents dans deux langues, il est préférable de définir un raccourci clavier permettant d'alterner rapidement entre les deux dictionnaires, comme suggéré sur le site de Flyspell :

```
(defun ispell-cycle-dictionary ()
  "Cycle between fr <-> en dictionaries"
  (interactive)
  (let* ((curr ispell-current-dictionary)
         (next (if (string= curr "fr") "en" "fr")))
        (ispell-change-dictionary next)
        (message "Dictionary switched from %s to %s" curr next)))
```

Markdown

L'édition de fichier Markdown peut naturellement se faire en mode texte simple, mais il est préférable d'utiliser le mode majeur correspondant. Les raccourcis sont disponibles sous le "local leader key" (`SPC m`).

Pour faciliter le balisage de certains éléments, tels que la mise en gras ou en italique, il suffit de sélectionner une partie de texte, qui devient alors la "région active", et d'utiliser le raccourci clavier correspondant en utilisant la séquence `C-c C-s`. Il est également possible de définir ses propres raccourcis clavier. Par exemple, dans le fichier `doom.d/+bindings.el`, on peut rajouter les instructions suivantes :

```
(map!
 (:map markdown-mode-map ;; (GUI only)
  :i "s-i" #'markdown-insert-italic
  :i "s-b" #'markdown-insert-bold))
```

Le raccourci `C-c C-d` est utilisé pour réaliser des actions telles que basculer d'une référence de note de bas de page à la note de bas de page elle-même, alterner entre les références et les définitions, etc.

Concernant la visualisation du document Markdown, il est possible d'utiliser le mode grip ou un visualisateur externe. Sous macOS, l'application Marked2 convient parfaitement et il suffit de la définir comme visualisateur par défaut. Voici un exemple de configuration possible : [4]

[4] La première instruction n'est indispensable mais permet de s'assurer que poly-markdown n'est pas activé lorsqu'il s'agit d'un simple document Markdown.

```
(add-to-list 'auto-mode-alist '("\\.md" . markdown-mode))
(setq markdown-open-command "/usr/local/bin/mark"
      markdown-command "/usr/local/bin/multimarkdown"
      markdown-enable-math t
      markdown-fontify-code-blocks-natively t
      markdown-hide-markup t
      markdown-gfm-uppercase-checkbox t
      markdown-list-item-bullets '("" "" "" "" "" "" "•")
      markdown-header-scaling-values '(1.1 1.0 1.0 1.0 1.0 1.0))
```

Org

Latex et Bibtex

Gestion des modes de programmation

Introduction au mode `progn`

En plus de son support amélioré pour les différents modes texte (texte brut, Markdown, Org et bien d'autres), Emacs permet d'éditer du code dans presque n'importe quel langage de programmation. Les plus connus sont bien évidemment les modes pour Emacs Lisp et C, mais on verra d'autres langages, en particulier Python, Clojure, Scheme, la manière de configurer les modes associés et les principaux outils disponibles sous Emacs pour interagir avec le code dans ces modes. Indépendemment du langage, Emacs offre un ensemble de fonctionnalités commune à tous les modes de programmation, encore appelé `progn-mode`. Cela comprend la gestion automatique des parenthèses, de l'indentation, des commentaires, etc.

Le package lsp-mode est très utile pour enrichir les fonctionnalités de base d'Emacs pour l'édition de code source. Il s'intègre de manière non obstructive aux autres outils (make, GDB, etc.) et peut remplacer les fonctions `xref-find-*` classiques pour naviguer entre les références et les définitions. Avec Doom, le raccourci `K` permet d'accéder à la documentation de l'objet sous le curseur, et l'aide en ligne est affichée via Eldoc.

Emacs Lisp

Même si vous ne programmez pas en ELisp, ce mode nous servira de base pour la présentation des autres modes.

Il est possible de lancer un shell interactif, appelé iElm, qui n'est rien d'autre qu'un mode mineur pour Emacs Lisp, en tapant `M-x ielm`.

Dans ce qui suit, on présentera plus en détails le mode Lisp avec en particulier Slime.

Lisp et Slime

Clojure et Cider

References

Petersen, Mickey (2015). *Mastering Emacs*. URL: https://www.masteringemacs.org/.