

Smart Documents

Marius Peter

23 Jul. 2020 (Thu)

Contents

1	Introduction	5
2	TODO First-time setup	5
2.1	TODO User details	6
2.2	File system paths	6
3	Early setup	6
3.1	The first file to load, <code>early-init.el</code>	6
3.2	The second file to load, <code>init.el</code>	7
3.3	Profiling — start	7
3.4	TODO Emacs client	8
3.5	Jumping to this file	8
3.6	Meta-files	8
3.6.1	Recently visited files	8
3.6.2	Projects' bookmarks	8
3.6.3	Location in previously visited file	8
3.6.4	Custom file	9
3.7	Backups	9
3.8	Initial and default frames	9
3.8.1	GNU/Linux	9
3.9	Secrets	10
4	Keyboard shortcuts	10
4.1	Files	10
4.1.1	Save a file	10
4.1.2	Open a file	10
4.1.3	List open files	10
4.1.4	Open this very file	10
4.1.5	Open a recently visited file	11
4.1.6	Locate a file	11
4.2	Windows	11
4.2.1	Make new window	11
4.2.2	Make only window	11
4.2.3	Close window and quit	11
4.3	Text display	11
4.3.1	Zoom	11
4.4	Customizations	12
4.4.1	Customize a variable	12
4.4.2	Customize a face	12
4.5	One-click workflows	12
4.5.1	Export to PDF	12

5	Packages	12
5.1	Meta	12
5.1.1	Package archives	12
5.1.2	TODO Convenient package update	12
5.1.3	use-package	13
5.2	org-mode	13
5.2.1	Basic customization	13
5.2.2	Languages executable in smart documents	14
5.2.3	Prevent or warn on invisible edits	14
5.2.4	Agenda	14
5.2.5	Timestamps	14
5.2.6	LaTeX export	15
5.2.7	TODO Export	17
5.3	TODO evil-mode	17
5.4	Spelling, completion, and snippets	17
5.4.1	Syntax checking	17
5.4.2	Spelling	17
5.4.3	Insert template from keyword	18
5.4.4	Complete anything interactively	18
5.4.5	Delete all consecutive whitespaces	18
5.5	Utilities	18
5.5.1	Versioning of files	18
5.5.2	Navigate between projects	18
5.5.3	Display keyboard shortcuts on screen	19
5.5.4	Jump to symbol's definition	19
5.5.5	Graphical representation of file history	19
5.5.6	Auto-completion framework	19
5.6	File formats	19
5.6.1	csv and Excel	19
5.6.2	Interacting with PDFs	20
5.6.3	Accounting	20
5.6.4	Plotting & charting	20
5.7	Cosmetics	20
5.7.1	Start page	20
5.7.2	TODO Mode line	21
5.7.3	TODO Sidebar	21
5.7.4	Better parentheses	21
5.7.5	Highlight "color keywords" in their color	21
5.7.6	UTF-8 bullet points in Org mode	21
6	Editing preferences	21
6.1	Editor	22
6.1.1	Coding standards	22
6.1.2	Recent files	22

6.2	Frame	22
6.2.1	TODO Header & mode line	22
6.3	Window	22
6.4	Buffer	22
6.4.1	Column filling	23
6.5	Minibuffer	23
7	Themes	23
7.1	My light and dark themes	23
7.1.1	Colors	23
7.1.2	Cursors	25
7.1.3	Fonts	25
7.2	TODO <i>minimal</i>	26
8	Late setup	26
8.1	Speeding up the next startup	26
8.2	Profiling — stop	26
8.3	Profiling — report	26
9	Conclusion	26

List of Figures

List of Tables

1	Light and dark themes' colors	24
---	---	----

Abstract

The idea of *Smart Documents* came to me as I was reflecting on how to improve the document creation process in my workplace. The GNU Emacs editor had captured my imagination and I wanted to create an accessible and highly productive text editor to benefit my organization. In this paper, I'll lay out my vision for the *Smart Document*, a file containing both text destined to the reader, and code describing how to update, validate, and present this text; then, I'll weave my personal GNU Emacs customizations with a tutorial. This paper is a *Smart Document* itself!

1 Introduction

GNU Emacs is most often used as a text editor. It would be unfair to say it is just that, because Emacs is capable of so much more. The utmost level of customization is afforded by enabling the user to rewrite *any* part of the source code and observe the editor's modified behaviour in real time. Since its inception in 1984, GNU Emacs has grown to be much more than a full-featured, high-productivity text editor—new *modes* have been written to interact with hundreds of file formats, including `.txt`, `.pdf`, `.jpg`, `.csv`, and `.zip` just to name a few. This paper itself was written in *Org mode*, a collection of functions enabling the harmonious mixing of code and comments in view of publication: this is the endgame of *literate programming*, and the basis of my vision for *Smart Documents*.

The following sections were laid out very deliberately. When we start Emacs, the source code blocks contained in this document are evaluated sequentially—our editing environment is constructed in real time as we execute the blocks in order. For instance, we only begin loading packages once we ensured `use-package` is working properly.¹

Customizing Emacs goes far, far beyond rewriting sections of this document—feel free to experiment and discover. Here are three commands that will help you understand all the symbols in this file, if you are browsing this paper within Emacs itself:

- `C-h f` describe function
- `C-h v` describe variable
- `C-h k` describe key

2 TODO First-time setup

The following code blocks are normally evaluated once—upon starting Emacs for the first time.

¹For more information on the detailed steps Emacs takes upon starting, refer to https://www.gnu.org/software/emacs/manual/html_node/elisp/Startup-Summary.html.

2.1 TODO User details

One advantage of working with *Smart Documents* is that they can automatically be populated with our details in the header, footer, or other appropriate element.

```
(defun my/tokenize-user-details ()
  "Tokenize user details."

  (cons 'user-full-name user-full-name))

(unless (file-exists-p (concat user-emacs-directory
                              "meta/user-details"))
  (setq user-full-name (read-string "Enter full user name:"))
  (setq user-mail-address (read-string "Enter user e-mail address:"))
  (setq user-details '(user-full-name
                      user-mail-address))
  (append-to-file "Foobar\n" nil "~/.emacs.d/meta/foobar"))
```

2.2 File system paths

In this subsection, we tell Emacs about relevant paths to resources.

On my MS Windows machine, I add the path to Portable Git.²

```
(when (string-equal system-type "windows-nt")
  (add-to-list 'exec-path "C:/Users/marius.peter/PortableGit/bin/"))
```

3 Early setup

3.1 The first file to load, `early-init.el`

This is the very first user-editable file loaded by Emacs.³ In it, we disable GUI elements that would otherwise be loaded and displayed once Emacs is ready to accept user input.

```
(setq my/early-init-file (concat user-emacs-directory "early-init.el"))

(defun my/create-early-init-file ()
  "Create `early-init.el' file in `user-emacs-directory'."
  (write-region
   "(menu-bar-mode -1)
(tool-bar-mode -1)
(menu-bar-bottom-and-right-window-divider)"
   my/early-init-file))
```

²Download from <https://git-scm.com/download/win>

³This feature became available in version 27.1.

```
(setq gc-cons-threshold 100000000)" nil my/early-init-file))

(unless (file-exists-p my/early-init-file)
  (menu-bar-mode -1)
  (tool-bar-mode -1)
  (menu-bar-bottom-and-right-window-divider)
  (setq gc-cons-threshold 100000000)
  (my/create-early-init-file))
```

3.2 The second file to load, `init.el`

Traditionally, file `~/.emacs` is used as the init file, although Emacs also looks at `~/.emacs.el`, `~/.emacs.d/init.el`, `~/.config/emacs/init.el`, or other locations.⁴

```
(setq my/init-file (concat user-emacs-directory "init.el"))

(defun my/create-init-file ()
  "Create `init.el' file in `user-emacs-directory'."
  (write-region
   "(menu-bar-mode -1)
(tool-bar-mode -1)
(menu-bar-bottom-and-right-window-divider)" nil my/init-file))

;; TODO This is a copy paste of `early.init.el'. Should be changed.
(unless (file-exists-p my/init-file)
  (menu-bar-mode -1)
  (tool-bar-mode -1)
  (menu-bar-bottom-and-right-window-divider)
  (my/create-init-file))
```

If no file is found, Emacs then loads in its purely vanilla state.

3.3 Profiling — start

We start the profiler now , and will interrupt it in Section 8.2. We will then present profiling report in Section 8.3.

```
; (profiler-start)
```

⁴https://www.gnu.org/software/emacs/manual/html_node/emacs/Init-File.html

3.4 TODO Emacs client

Makes opening emacs faster for following instances.

```
; (setq initial-buffer-choice (lambda () (get-buffer "*dashboard*")))
```

3.5 Jumping to this file

We begin by defining a function to open this very file.

```
(defun my/find-literate-config ()  
  "Jump to this very file."  
  (interactive)  
  (find-file my/literate-config))
```

3.6 Meta-files

In this section, we'll be tidying up the `.emacs.d/` directory—by default, many Emacs packages create files useful for themselves in our `user-emacs-directory`. This leads to undesirable clutter. Certain packages create files that log recently visited files (3.6.1); log location of known projects (3.6.2); log location in recently visited files (3.6.3) The commonality between all these files is that they tend to reference... other files. Thus, I decided to refer to them as meta-files. First, let's designate a folder to collect our meta-files together:

```
(setq my/meta-files-location (concat user-emacs-directory "meta/"))
```

3.6.1 Recently visited files

```
(setq recentf-save-file (concat  
                          my/meta-files-location  
                          "recentf"))
```

3.6.2 Projects' bookmarks

```
(setq projectile-known-projects-file (concat  
                                      my/meta-files-location  
                                      "projectile-bookmarks.el"))
```

3.6.3 Location in previously visited file

```
(setq save-place-file (concat  
                       my/meta-files-location  
                       "places"))
```


3.6.4 Custom file

Load settings created automatically by GNU Emacs Custom. (For example, any clickable option/toggle is saved here.) Useful for fooling around with `M-x customize-group <package>`.

```
(setq custom-file (concat user-emacs-directory "init-custom.el"))  
(load custom-file)
```

3.7 Backups

Backups are very important!

```
(setq backup-directory-alist `(("*" . ,temporary-file-directory))  
      auto-save-file-name-transforms `(("*" ,temporary-file-directory t))  
      backup-by-copying t      ; Don't delink hardlinks  
      version-control t        ; Use version numbers on backups  
      delete-old-versions t    ; Automatically delete excess backups  
      kept-new-versions 20     ; how many of the newest versions to keep  
      kept-old-versions 5      ; and how many of the old  
      )
```

3.8 Initial and default frames

We set the dimensions of the initial and default frames.

```
(add-to-list 'default-frame-alist '(width . 100))  
(add-to-list 'default-frame-alist '(height . 50))  
  
(add-to-list 'initial-frame-alist '(width . 100))  
(add-to-list 'initial-frame-alist '(height . 50))
```

3.8.1 GNU/Linux

These settings affect the first and subsequent frames spawned by Emacs in GNU/Linux. Frame transparency increases when focus is lost.

```
(when (and (display-graphic-p) (string-equal system-type "gnu/linux"))  
  (set-frame-parameter (selected-frame) 'alpha '(90 . 50))  
  (add-to-list 'default-frame-alist '(alpha . (90 . 50))))
```

3.9 Secrets

The code contained in the `secrets.org` file is loaded by Emacs, but not rendered in this PDF for the sake of privacy. It contains individually identifying information such as names and e-mail addresses, which are used to populate Org templates (Section 5.2). You need to create this `secrets.org` file, as it is ignored by `git` by default.

```
(org-babel-load-file "~/emacs.d/secrets.org")
```

4 Keyboard shortcuts

The following bindings strive to further enhance CUA mode.⁵

```
(cua-mode)
```

What follows are the most useful keybindings, as well as the keybindings to the functions we defined ourselves. It doesn't matter if we haven't defined the function itself yet; Emacs will accept a keybinding for any symbol and does not check if the symbol exists until the keybinding is pressed.

4.1 Files

4.1.1 Save a file

```
(global-set-key (kbd "C-s") 'save-buffer)
```

4.1.2 Open a file

```
(global-set-key (kbd "C-o") 'find-file)
```

4.1.3 List open files

```
(global-set-key (kbd "C-b") 'ivy-switch-buffer)
```

4.1.4 Open this very file

(Function defined in Section 3.5)

```
(global-set-key (kbd "C-c c") 'my/find-literate-config)
```

⁵Common User Access. This is a term coined by IBM which has influenced user navigation cues on all modern desktop OSes. From IBM's CUA, we get the `Ctrl-v` and `Ctrl-v` keyboard shortcuts.

4.1.5 Open a recently visited file

```
(global-set-key (kbd "C-r") 'counsel-recentf)
```

4.1.6 Locate a file

```
(global-set-key (kbd "C-c l") 'counsel-locate)
```

4.2 Windows

4.2.1 Make new window

```
(global-set-key (kbd "C-n") 'make-frame)
```

4.2.2 Make only window

```
(global-set-key (kbd "C-`") 'delete-other-windows)
```

4.2.3 Close window and quit

```
(defun my/delete-window-or-previous-buffer ()  
  "Delete window; if sole window, previous buffer."  
  (interactive)  
  (if (> (length (window-list)) 1)  
      (delete-window)  
      (previous-buffer)))
```

The following bindings lead to more natural exit behaviors.

```
(global-set-key (kbd "C-w") 'my/delete-window-or-previous-buffer)  
(global-set-key (kbd "C-q") 'save-buffers-kill-terminal)
```

4.3 Text display

4.3.1 Zoom

The typical binding on both GNU/Linux and MS Windows is adequate here: C-= to zoom in, C-- to zoom out.

It seems that starting with Emacs 27.1, Control + mousewheel works.

```
(global-set-key (kbd "C--") 'text-scale-decrease)  
(global-set-key (kbd "C-=") 'text-scale-increase)  
(global-set-key (kbd "C-+") 'text-scale-increase)
```

4.4 Customizations

4.4.1 Customize a variable

```
(global-set-key (kbd "C-c v") 'customize-variable)
```

4.4.2 Customize a face

```
(global-set-key (kbd "C-c f") 'customize-face)
```

4.5 One-click workflows

4.5.1 Export to PDF

```
(global-set-key (kbd "C-p") 'my/org-quick-export)
```

5 Packages

Packages are collections of `.el` files providing added functionality to Emacs.

5.1 Meta

How do we bootstrap packages? First, let's figure out:

1. Where we get our packages from
2. How we upgrade packages
3. How we ensure our required packages are installed

5.1.1 Package archives

List of package archives.

```
(require 'package)
(add-to-list 'package-archives '("melpa" . "https://melpa.org/packages/") t)
(add-to-list 'package-archives '("org" . "https://orgmode.org/elpa/") t)
(package-initialize)
```

5.1.2 TODO Convenient package update

One-function rollup of upgradeable package tagging, download and lazy install.

5.1.3 use-package

We ensure `use-package` is installed, as well as all packages described in this configuration file.

```
(unless (package-installed-p 'use-package)
  (package-refresh-contents)
  (package-install 'use-package)
  (eval-when-compile (require 'use-package)))
(setq use-package-always-ensure t)
(require 'use-package)
(require 'bind-key)
```

5.2 org-mode

Phew, I can finally introduce Org mode! I am so **excited**.

Org mode replaces a word processor, a presentation creator, and a spreadsheet editor. IMHO, the spreadsheet ability captures more than 80% use cases wherein one wishes to include a table in a text document destined for physical publication. (It is clear that Excel spreadsheets are *not* destined for physical publication—simply attempt to print an Excel spreadsheet with the default settings.) In my opinion, Org mode matches all *useful* features of the Microsoft Office suite 1-to-1.

What follows are customizations designed to make Org mode behave more like Microsoft Word. The end goal is, once again, to draw as many new users to Emacs as possible!

5.2.1 Basic customization

Org base directory is in user home on GNU/Linux, or in `AppData` in MS Windows.

```
(setq org-directory (concat user-emacs-directory "~/org"))
```

First, we hide markup symbols for **bold**, *italic*, underlined and ~~striketrough~~ text, and ensure our document appears indented upon loading:⁶

For the time being, I will in fact display emphasis markers, because hiding them corrupts tables.

```
(setq org-hide-emphasis-markers nil
      org-startup-indented t
      org-src-preserve-indentation nil
      org-edit-src-content-indentation 0)
```

⁶It *appears* indented, but the underlying plaintext file does not contain tab characters!

5.2.2 Languages executable in smart documents

The following languages can be written inside SRC blocks, in view of being executed by the Org Babel backend.

```
(setq org-babel-load-languages
'((shell . t)
  (python . t)
  (plantuml . t)
  (emacs-lisp . t)
  (awk . t)
  (ledger . t)
  (gnuplot . t)
  (latex . t)))

(org-babel-do-load-languages
 'org-babel-load-languages '((C . t) (shell . t) (gnuplot . t)))
```

5.2.3 Prevent or warn on invisible edits

```
(setq org-catch-invisible-edits t)
```

5.2.4 Agenda

The agenda displays a chronological list of headings across all agenda files for which the heading or body contain a matching `org-time-stamp`.⁷

```
(global-set-key (kbd "C-c a") 'org-agenda-list)

(defun my/find-diary-file ()
  "Load `org-agenda-diary-file'."
  (interactive)
  (find-file org-agenda-diary-file))

(global-set-key (kbd "C-c d") 'my/find-diary-file)
```

5.2.5 Timestamps

More literary timestamps are exported to \LaTeX using the following custom format:

```
(setq org-time-stamp-custom-formats
'("%d %b. %Y (%a)" . "%d %b. %Y (%a), at %H:%M"))
```

⁷An `org-time-stamp` can be inserted with `C-c .` (period)

5.2.6 \LaTeX export

We'll be compiling our documents with LuaTeX. This will afford us some future-proofing, since it was designated as the successor to pdfTeX by the latter's creators.

First, we define the command executed when an Org file is exported to \LaTeX . We'll use `latexmk`, the Perl script which automagically runs binaries related to \LaTeX in the correct order and the right amount of times.

Options and why we need them:

-shell-escape required by minted to color source blocks

-pdflatex=lualatex we use lualatex to generate our PDF

-interaction=nonstopmode go as far as possible without prompting user for input

```
(setq org-latex-pdf-process
      '("latexmk -pdf -f \
-pdflatex=lualatex -shell-escape \
-interaction=nonstopmode -outdir=%o %f"))
```

We customize the format for org time stamps to make them appear monospaced in our exported \LaTeX documents. This makes them visually distinguishable from body text.

```
(setq org-latex-active-timestamp-format
      "\\texttt{%s}")
(setq org-latex-inactive-timestamp-format
      "\\texttt{%s}")
```

The following packages are loaded for every time we export to \LaTeX .

```
(setq org-latex-packages-alist
      '(("AUTO" "polyglossia" t
         ("xelatex" "lualatex"))
        ("AUTO" "babel" t
         ("pdflatex"))
        (" " "booktabs" t
         ("pdflatex"))
        ("table,svgnames" "xcolor" t
         ("pdflatex"))))
```

Little bonus for GNU/Linux users: syntax highlighting for source code blocks in \LaTeX exports.

```
(when (string-equal system-type "gnu/linux")
  (add-to-list 'org-latex-packages-alist '("AUTO" "minted" t
                                           ("pdflatex" "lualatex"))))

(setq org-latex-listings 'minted)
(setq org-latex-minted-options
      '(("style" "friendly") ())))
```

Now, we set the files to be deleted when a \LaTeX \rightarrow PDF compilation occurs. We only care about two files, in the end: the Org mode file for edition, and the PDF for distribution.

```
(setq org-latex-logfiles-extensions
  '("aux" "bcf" "blg" "fdb_latexmk"
    "fls" "figlist" "idx" "log" "nav"
    "out" "ptc" "run.xml" "snm" "toc" "vrb" "xdv"
    "tex" "lot" "lof"))
```

By default, Org agenda inserts diary entries as the first under the selected date. It is preferable to insert entries in the order that they were recorded, i.e. chronologically.

```
(setq org-agenda-insert-diary-strategy 'date-tree-last)
```

What follows are the document class structures that can be exported in \LaTeX .

```
(setq org-latex-classes
  '(("article" "\\documentclass[11pt]{article}"
    ("\\section{%s}" . "\\section*{%s}")
    ("\\subsection{%s}" . "\\subsection*{%s}")
    ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
    ("\\paragraph{%s}" . "\\paragraph*{%s}")
    ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))
    ("report" "\\documentclass[11pt]{report}"
    ("\\part{%s}" . "\\part*{%s}")
    ("\\chapter{%s}" . "\\chapter*{%s}")
    ("\\section{%s}" . "\\section*{%s}")
    ("\\subsection{%s}" . "\\subsection*{%s}")
    ("\\subsubsection{%s}" . "\\subsubsection*{%s}"))
    ("book" "\\documentclass[12pt]{book}"
    ("\\part{%s}" . "\\part*{%s}")
    ("\\chapter{%s}" . "\\chapter*{%s}")
    ("\\section{%s}" . "\\section*{%s}")
    ("\\subsection{%s}" . "\\subsection*{%s}")
    ("\\subsubsection{%s}" . "\\subsubsection*{%s}"))
    ("book-blendit" "\\documentclass[12pt]{book}"
    ("\\chapter{%s}" . "\\chapter*{%s}")
    ("\\section{%s}" . "\\section*{%s}")
    ("\\subsection*{%s}" . "\\subsection*{%s}")
    ("\\subsubsection*{%s}" . "\\subsubsection*{%s}"))))
```

By default, body text can immediately follow the table of contents. It is however cleaner to separate table of contents with the rest of the work.

```
(setq org-latex-toc-command "\\tableofcontents\\clearpage")
```

The following makes TODO items appear red and CLOSED items appear green in Org's \LaTeX exports. Very stylish, much flair!

5.2.7 TODO Export

This creates a shorter binding for the most common Org export: $\text{Org} \rightarrow \text{\LaTeX} \rightarrow \text{PDF}$. The binding is defined in Section 4.5.1.

```
(defun my/org-quick-export ()  
  "Org async export to PDF and open.  
  This basically reimplements `C-c C-e C-a l o'."  
  (interactive)  
  (org-open-file (org-latex-export-to-pdf)))
```

5.3 TODO evil-mode

Forgive me, for I have sinned.

This is the 2nd most significant customization after `org-mode`. Enabling `evil-mode` completely changes editing keys.⁸

```
(use-package evil)  
; (setq evil-toggle-key "C-c d") ; devil...  
; (evil-mode 1)
```

5.4 Spelling, completion, and snippets

The following customizations open the doors to vastly increased typing speed and accuracy.

5.4.1 Syntax checking

We require a package to highlight syntax errors and warnings. The `flycheck` package ensures we are aware of all our code's syntactical shortcomings.

```
(use-package flycheck)  
(global-flycheck-mode)
```

5.4.2 Spelling

```
(use-package flyspell)  
(add-hook 'text-mode-hook 'flyspell-mode)
```

⁸For more information on `vi` keybindings, visit <https://hea-www.harvard.edu/~fine/Tech/vi.html>.

5.4.3 Insert template from keyword

Thanks to `yasnippet`, we can type certain keywords, then press `TAB`, to automatically insert a predefined text snippet. We can then navigate through the snippet by using `<tab>` (next field) and `<backtab>` (previous field).⁹

For instance: typing `src` then pressing `TAB` will expand the keyword to the following text:

```
#+BEGIN_SRC emacs-lisp
```

```
#+END_SRC
```

We notice that `emacs-lisp` is highlighted—this is the first modifiable field. Many clever programming tricks can be performed with `yasnippet` to save us a ton of time with boilerplate text!

```
(use-package yasnippet)
(yas-global-mode 1)
```

5.4.4 Complete anything interactively

```
; (add-hook 'after-init-hook 'global-company-mode)
```

5.4.5 Delete all consecutive whitespaces

```
(use-package hungry-delete
:init (hungry-delete-mode))
```

5.5 Utilities

5.5.1 Versioning of files

Wonderful Git porcelain for Emacs. Enables the administration of a Git repository in a pain-free way.

```
(use-package magit
:bind ("C-c g" . magit-status))
```

5.5.2 Navigate between projects

This enables us to better manage our `.git` projects.

```
(use-package projectile
:bind ("C-c p" . 'projectile-command-map)
:init (projectile-mode 1)
      (setq projectile-completion-system 'ivy))
```

⁹`<backtab>` is synonymous with pressing `shift-tab`.

5.5.3 Display keyboard shortcuts on screen

```
(use-package which-key
  :init (which-key-mode))
```

5.5.4 Jump to symbol's definition

`dumb-jump` is a reliable symbol definition finder. It uses different matching algorithms and heuristics to provide a very educated guess on the location of a symbol's definition.

```
(use-package dumb-jump)
(add-hook 'xref-backend-functions #'dumb-jump-xref-activate)
```

5.5.5 Graphical representation of file history

```
(use-package undo-tree)
(global-undo-tree-mode)
```

5.5.6 Auto-completion framework

```
(use-package ivy
  :config (setq ivy-use-virtual-buffers t
                ivy-count-format "%d/%d "
                enable-recursive-minibuffers t))
(ivy-mode t)
```

1. Smartly suggesting interactive search matches

Wonderful counsellor!

```
(use-package counsel
  :bind ("M-x" . counsel-M-x)
  :config (counsel-mode t))

(global-set-key (kbd "C-f") 'counsel-grep-or-swiper)
```

2. Searching for items

```
(use-package swiper
  :bind (("C-f" . counsel-grep-or-swiper)))
```

5.6 File formats

5.6.1 csv and Excel

```
(use-package csv-mode)
```

5.6.2 Interacting with PDFs

Org mode shines particularly when exporting to PDF—Org files can reliably be shared and exported to PDF in a reproducible fashion.

```
(use-package pdf-tools)
;; (pdf-tools-install)
```

5.6.3 Accounting

Ledger is a creation of John Wiegley's. It enables double-entry accounting in a simple plaintext format, and reliable verification of account balances through time.¹⁰

```
(use-package ledger-mode
  :bind
  ("C-c r" . ledger-report)
  ("C-c C" . ledger-mode-clean-buffer))
```

These reports can be generated within Emacs. It is quite useful to pipe their output to an automated "smart document".

```
(setq ledger-reports
  '(("bal" "%(binary) -f %(ledger-file) bal")
    ("bal-USD" "%(binary) -f %(ledger-file) bal --exchange USD")
    ("reg" "%(binary) -f %(ledger-file) reg")
    ("net-worth" "%(binary) -f %(ledger-file) bal ^Assets ^Liabilities --exchange USD")
    ("net-income" "%(binary) -f %(ledger-file) bal ^Income ^Expenses --exchange USD")
    ("payee" "%(binary) -f %(ledger-file) reg @%(payee)")
    ("account" "%(binary) -f %(ledger-file) reg %(account)")
    ("budget" "%(binary) -f %(ledger-file) budget --exchange USD")))
```

5.6.4 Plotting & charting

```
(use-package gnuplot)
```

5.7 Cosmetics

5.7.1 Start page

We replace the standard welcome screen with our own.

```
(setq inhibit-startup-message t)
(use-package dashboard)
```

¹⁰For more information, visit <https://www.ledger-cli.org/>.

```
:config
(dashboard-setup-startup-hook)
(setq dashboard-startup-banner (concat user-emacs-directory "img/Safran_logo.s
(setq dashboard-items '((recents . 5)
                        (projects . 5)))
(setq dashboard-banner-logo-title "A modern professional text editor."))
```

5.7.2 TODO Mode line

```
(use-package powerline)
```

5.7.3 TODO Sidebar

Get inspiration from `ibuffer-sidebar` and create a better sidebar.

```
;; (load-file)
```

5.7.4 Better parentheses

```
(use-package rainbow-delimiters
:config (add-hook 'prog-mode-hook #'rainbow-delimiters-mode))
(electric-pair-mode)
(show-paren-mode 1)
```

5.7.5 Highlight “color keywords” in their color

This highlights hexadecimal numbers which look like colors, in that same color.

```
(use-package rainbow-mode
:init
(add-hook 'prog-mode-hook 'rainbow-mode))
```

5.7.6 UTF-8 bullet points in Org mode

```
(use-package org-bullets
:config
(when (string-equal system-type "gnu/linux")
(add-hook 'org-mode-hook (lambda () (org-bullets-mode 1)))))
```

6 Editing preferences

These customizations enhance editor usability. They are not brought about

6.1 Editor

6.1.1 Coding standards

This is just a better default. Don't @ me.

```
(setq c-default-style "linux"
      c-basic-offset 4)
```

6.1.2 Recent files

The keybinding for opening a recently visited file is described in paragraph 4.1.5.

```
(recentf-mode 1)
(setq recentf-max-menu-items 25)
(setq recentf-max-saved-items 25)
(run-at-time nil (* 5 60) 'recentf-save-list)
```

6.2 Frame

6.2.1 TODO Header & mode line

Complete mode line rewrite. Might require new package.

Top of the buffer is more intuitive for buffer info, bottom is more intuitive for buffer action.

This is pretty much a gutted out powerline.

1. Header line

```
(setq header-line-format "%b")
```

2. Mode line

```
(setq mode-line-format nil)
```

6.3 Window

6.4 Buffer

Save cursor location in visited buffer after closing it or Emacs.

```
(save-place-mode 1)
```

6.4.1 Column filling

A line of text is considered “filled” when it reaches 79 characters in length.

```
(setq-default fill-column 79)
```

Automatically break lines longer than `fill-column`.

```
(add-hook 'org-mode-hook 'turn-on-auto-fill)
```

6.5 Minibuffer

We replace the longer `yes-or-no-p` questions with more convenient `y-or-n-p`.

```
(defalias 'yes-or-no-p 'y-or-n-p)
```

Disable minibuffer scroll bar.

```
(set-window-scroll-bars (minibuffer-window) nil nil)
```

7 Themes

Without a carefully designed theme, our editor could become unusable. Thus, we describe two themes that were developed purposefully and iteratively.

```
(setq custom-theme-directory (concat user-emacs-directory "themes/"))  
(load-theme 'blendoit-light)  
; (load-theme 'blendoit-dark)
```

7.1 My light and dark themes

A highly legible, unambiguous, and classic theme.

7.1.1 Colors

The default face is a black foreground on a white background, this matches MS Word. We are striving for a simple, intuitive color scheme.

Most of the visual cues derived from color are identical in both light and dark themes (Table 1).

Table 1: Light and dark themes' colors.

Color	blendoit-light	blendoit-dark
Black	default text	default background
Lighter shades	lesser headers	<i>n/a</i>
White	default background	default text
Darker shades	<i>n/a</i>	lesser headers
Red	negative	<i>same</i>
Tomato	timestamp 'TODO'	<i>same</i>
Green	positive	<i>same</i>
ForestGreen	timestamp 'DONE'	<i>same</i>
Blue	interactable content; links	<i>same</i>
SteelBlue	anything Org mode; anchor color	<i>same</i>
DeepSkyBlue	highlight	<i>same</i>
DodgerBlue	isearch	<i>same</i>
Purple		

7.1.2 Cursors

In order to imitate other modern text editors, we resort to a blinking bar cursor. We choose red, the most captivating color, because the cursor is arguably the region on our screen:

1. most often looked at;
2. most often searched when lost.

In files containing only **fixed-pitch** fonts (i.e. files containing only code), the cursor becomes a high-visibility box.

In files containing a mix of **variable-pitch** and **fixed-pitch** fonts, the cursor is a more MS Word-like bar.

```
(setq-default cursor-type 'bar)
```

7.1.3 Fonts

Hack default and fixed-pitch

- Legible, modern monospace font
- Strict, sharp, uncompromising

Liberation Sans variable-pitch

- Libre alternative to Arial
- Unoffensive

Hermit org-black, anything Org/meta in general

- Slightly wider than Hack
- More opinionated shapes
- Very legible parentheses

1. Using proportional fonts when needed

We use **variable-pitch-mode** for appropriate modes.

```
(add-hook 'org-mode-hook 'variable-pitch-mode)  
(add-hook 'info-mode-hook 'variable-pitch-mode)
```

2. **TODO** Default font size

Make default font size larger on displays of which the resolution is greater than 1920×1080.

7.2 TODO minimal

8 Late setup

At this point, our editor is almost ready to run. Phew! All that's left to do is to prepare for the next startup, interrupt our profiling activities, and smartly store the result of our profiling.

8.1 Speeding up the next startup

```
(defun byte-compile-literate-config ()  
  "Byte compile our literate configuration file."  
  (delete-file (concat  
                (file-name-sans-extension my/literate-config)  
                ".elc"))  
  (org-babel-tangle-file my/literate-config)  
  (byte-compile-file (concat  
                      (file-name-sans-extension my/literate-config)  
                      ".el"))  
  (message "c'mon work fucker"))  
  
(add-hook 'kill-emacs-hook 'byte-compile-literate-config)
```

8.2 Profiling — stop

```
;; (profiler-stop)
```

8.3 Profiling — report

```
;; (profiler-report)
```

9 Conclusion

In this configuration file, we described a series of customization steps taken to make Emacs more palatable to modern IDE users.