Smart Documents

Marius Peter
<2020-11-08 Sun>

Contents

1 Introduction **6**

2 First-time setup **6**
2.1 **TODO** Unpacking our literate configuration . . . 7
2.2 **TODO** User details . . . 7
2.3 File system paths . . . 8

3 Early setup **8**
3.1 The first file to load . . . 8
3.2 The second file to load . . . 8
3.3 Profiling — start . . . 9
3.4 Jumping to this file . . . 9
3.5 Speeding up the next startup . . . 9
3.6 Meta-files . . . 9
3.6.1 Recently visited files . . . 10
3.6.2 Projects' bookmarks . . . 10
3.6.3 Location in previously visited file . . . 10
3.6.4 Custom file . . . 10
3.7 Backups . . . 10
3.8 Initial and default frames . . . 10
3.8.1 GNU/Linux . . . 11
3.9 Secrets . . . 11

4 Keyboard shortcuts **11**
4.1 Files . . . 11
4.1.1 Save a file . . . 11
4.1.2 Open a file . . . 12
4.1.3 List open files . . . 12
4.1.4 Open this very file . . . 12
4.1.5 Open a recently visited file . . . 12
4.1.6 Locate a file . . . 12
4.1.7 Open the agenda . . . 12
4.1.8 Open the diary . . . 12
4.1.9 Open Org mode document properties . . . 12
4.2 Windows . . . 12
4.2.1 Close window and quit . . . 12
4.3 Frame . . . 13
4.3.1 Make new frame . . . 13
4.3.2 Make only frame . . . 13
4.3.3 Delete frame or kill Emacs . . . 13
4.3.4 Open sidebar . . . 13
4.4 Text display . . . 13

4.4.1 Zoom . . . 13
4.5 Navigation . . . 14
4.5.1 Move down one line . . . 14
4.5.2 Move up one line . . . 14
4.5.3 Move left one character . . . 14
4.5.4 Move right one character . . . 14
4.6 Customizing the editor . . . 15
4.6.1 Customize a variable . . . 15
4.6.2 Customize a face . . . 15
4.7 One-click workflows . . . 15
4.7.1 Export to PDF . . . 15
4.7.2 Indent buffer . . . 15
4.7.3 Beautify Org mode buffer . . . 15

5 Packages **15**
5.1 Meta . . . 16
5.1.1 Package archives . . . 16
5.1.2 **TODO** Convenient package update . . . 16
5.1.3 `use-package` . . . 16
5.2 `evil-mode` . . . 16
5.3 Spelling, completion, and snippets . . . 17
5.3.1 Syntax checking . . . 17
5.3.2 Spelling . . . 17
5.3.3 Insert template from keyword . . . 17
5.3.4 Complete anything interactively . . . 17
5.3.5 Delete all consecutive whitespaces . . . 18
5.4 Utilities . . . 18
5.4.1 Versioning of files . . . 18
5.4.2 Navigate between projects . . . 18
5.4.3 Display keyboard shortcuts on screen . . . 18
5.4.4 Jump to symbol's definition . . . 18
5.4.5 Graphical representation of file history . . . 18
5.4.6 Auto-completion framework . . . 19
5.5 Tree . . . 19
5.6 Coding languages . . . 19
5.6.1 **TODO** Emacs Lisp . . . 19
5.6.2 Python . . . 19
5.7 File formats . . . 19
5.7.1 `csv` and Excel . . . 19
5.7.2 Interacting with PDFs . . . 20
5.7.3 Accounting . . . 20
5.7.4 Plotting & charting . . . 20
5.8 Cosmetics . . . 20
5.8.1 Start page . . . 20

5.8.2 **TODO** Sidebar . . . 21
5.8.3 Better parentheses . . . 21
5.8.4 Highlight “color keywords” in their color . . . 21
5.8.5 ~~UTF-8 bullet points in `Org mode`~~ . . . 21

6 `org-mode` 21
6.1 Introduction . . . 22
6.2 Basic customization . . . 22
6.2.1 Base folder . . . 22
6.2.2 Prevent/warn on invisible edits . . . 22
6.3 Org cosmetics . . . 22
6.3.1 Dynamic numbering of headlines . . . 23
6.3.2 Document properties . . . 23
6.3.3 Timestamps . . . 24
6.4 Programming a Smart Documents . . . 24
6.5 Agenda . . . 24
6.6 LaTeX export . . . 24
6.6.1 Exporting timestamps . . . 25
6.6.2 LaTeX packages . . . 25
6.6.3 Colored source blocks in PDF export . . . 26
6.6.4 Cleaning directory after export . . . 26
6.6.5 Chronological diary entries . . . 26
6.6.6 Table of contents . . . 26
6.7 **TODO** Org links . . . 27

7 One-click workflows 27
7.1 **TODO** Export to PDF . . . 27
7.2 Beautify buffer . . . 28

8 Editing preferences 28
8.1 Editor . . . 28
8.1.1 Coding standards . . . 28
8.1.2 Recent files . . . 28
8.1.3 Fill column . . . 28
8.2 Frame . . . 29
8.2.1 Header & mode lines . . . 29
8.3 Window . . . 31
8.4 Buffer . . . 31
8.4.1 Save cursor location . . . 31
8.4.2 Column filling . . . 31
8.5 Text . . . 31
8.5.1 Beautiful symbols . . . 31
8.5.2 Org mode sugar . . . 31
8.5.3 Electric modes . . . 32

8.6 Minibuffer 32

9 Themes **32**
9.1 My light and dark themes 32
9.1.1 Colors 33
9.1.2 Cursors 33
9.1.3 Fonts 34
9.2 Wealthy theme 35
9.2.1 Symbol substitution 36
9.3 **TODO** `minimal` 36

10 Late setup **36**
10.1 Profiling — stop 36
10.2 Profiling — report 36

11 Conclusion **36**

List of Figures

1 Claude Garamont, an icon of font design 35

List of Tables

1 Navigation keybindings 14
2 Light and dark themes' colors 33

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 behavior 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

You can always press `f1` to access Emacs built-in help.

2 First-time setup

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

[1] For 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 Unpacking our literate configuration

```
(defvar sd-packed-p t
  "Boolean to track literate configuration packed/unpacked status.")

(defvar sd-unpack-sections (org-property-values "sd-unpack-path")
  "List of target sections in `my/literate-config' to be unpacked.")

(defun sd-unpack-sections ()
  "Unpack literate configuration into `emacs-user-directory'."
  (interactive)
  (mapcar 'sd-unpack sd-unpack-sections)
  )

(defun sd-unpack-section (&optional section)
  "Unpack SECTION into `user-emacs-directory'.
      If nil, unpack section under point.
      Make go through list of headings and unpack first matching
↪  SECTION."
  (interactive)
  (if (not section)
      (insert
       (concat
        "\nThe contents of this Section was automatically moved to\n="
        user-emacs-directory (org-entry-get nil "sd-unpack-path")
        ↪  "=.\n"
        "Use `sd-pack-section' to copy the contents back into this
        ↪  section."))))


(defun sd-pack-section ()
  "Pack SECTION into `my/literate-config'."
  (interactive)
  (message "foobar!!!"))

(global-set-key (kbd "C-t") 'sd-pack-section)

(sd-unpack "init.el")
```

2.2 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.

```
(setq user-full-name "Marius Peter")

(defun my/user-details-get ()
"Get user details."
    (setq user-full-name (read-string "Enter full user name:"))
    (setq user-mail-address (read-string "Enter user e-mail address:"))
    (message "Successfully captured user details."))

(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-details '(user-full-name
                       user-mail-address))
  (append-to-file "Foobar\n" nil "~/.emacs.d/meta/foobar"))
```

2.3 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

The contents of this Section was automatically moved to `~/.emacs.d/init.el`. Use 'sd-pack-section' to copy the contents back into this section.

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.

It can be found here: `early-init.el`

3.2 The second file to load

> 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.

[2] Download from `https://git-scm.com/download/win`

[3] This feature became available in version 27.1.

From the GNU website[4]

This file can be found here: `init.el`
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 10.1. We will then present profiling report in Section 10.2.

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

3.4 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 (concat my/literate-config ".org")))
```

3.5 Speeding up the next startup

```
(defun byte-compile-literate-config ()
  "Byte compile our literate configuration file."
  (delete-file (concat my/literate-config ".elc"))
  (delete-file (concat my/literate-config ".el"))
  (org-babel-tangle-file (concat my/literate-config ".org"))
  (byte-compile-file (concat my/literate-config ".el")))

(add-hook 'kill-emacs-hook 'byte-compile-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 sd-meta-files-location (concat user-emacs-directory "meta/"))
```

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

3.6.1 Recently visited files

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

3.6.2 Projects' bookmarks

```
(setq projectile-known-projects-file (concat
                                      sd-meta-files-location
                                      "projectile-bookmarks.eld"))
```

3.6.3 Location in previously visited file

```
(setq save-place-file (concat
                       sd-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 "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 frame:

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

We also set the dimensions of subsequent frames:

```
(add-to-list 'default-frame-alist '(width  . 50))
(add-to-list 'default-frame-alist '(height . 30))
```

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 6). You need to create this `secrets.org` file, as it is ignored by `git` by default.

```
(let ((secrets (concat user-emacs-directory "secrets.org")))
  (when (file-exists-p secrets) (org-babel-load-file secrets)))
```

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 functions themselves yet; Emacs will accept a keybinding for any symbol and does not check if the symbol's function definition exists, until the keybinding is pressed.

4.1 Files

4.1.1 Save a file

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

[5] Common 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-c` and `Ctrl-v` keyboard shortcuts.

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 3.4)

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

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)
```

4.1.7 Open the agenda

```
(global-set-key [f5] 'org-agenda-list)
```

4.1.8 Open the diary

```
(global-set-key [f6]
                '(lambda ()
                   "Load `org-agenda-diary-file'."
                   (interactive)
                   (find-file org-agenda-diary-file)))
```

4.1.9 Open Org mode document properties

```
(global-set-key [f9] 'sd-document-properties)
```

4.2 Windows

4.2.1 Close window and quit

The following bindings lead to more natural window & frame exit behaviors.

```
(global-set-key (kbd "C-w")
                '(lambda ()
                   "Delete window; if sole window, previous buffer."
                   (interactive)
                   (if (> (length (window-list)) 1)
                       (delete-window)
                     (previous-buffer))))
```

4.3 Frame

4.3.1 Make new frame

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

4.3.2 Make only frame

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

4.3.3 Delete frame or kill Emacs

```
(global-set-key (kbd "C-q")
                '(lambda ()
                   (interactive)
                   "delete frame; if sole frame, kill Emacs."
                   (if (> (length (frame-list)) 1)
                          (delete-frame)
                     (kill-emacs))))
```

4.3.4 Open sidebar

```
(global-set-key (kbd "<left-fringe> <mouse-1>") 'sd-sidebar)
```

4.4 Text display

4.4.1 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)
```

4.5 Navigation

`Alt` (`Meta`) is the privileged key for motion in a buffer. It is followed by an optional numerical argument, and a movement command. You may navigate in a buffer by keeping `Alt` pressed, optionally inputting a number from the keypad or number row, then pressing any of the following movement keys: `j`, k, h, and `l`. You will move in that direction in the amount of the numerical argument.

Table 1: Navigation keybindings.

	Backwards	Forwards
Character	`M-h`	`M-l`
Line	`M-k`	`M-j`
Word	`M-f`	`M-b`
Paragraph	`M-a`	`M-e`

We prevent Org mode from overriding preferred navigation keys.

```
(local-unset-key (kbd "M-j"))
(local-unset-key (kbd "M-k"))
(local-unset-key (kbd "M-l"))
(local-unset-key (kbd "M-h"))
```

4.5.1 Move down one line

```
(global-set-key (kbd "M-j") 'next-line)
```

4.5.2 Move up one line

```
(global-set-key (kbd "M-k") 'previous-line)
```

4.5.3 Move left one character

```
(global-set-key (kbd "M-h") 'left-char)
```

4.5.4 Move right one character

```
(global-set-key (kbd "M-l") 'right-char)
```

4.6 Customizing the editor

4.6.1 Customize a variable

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

4.6.2 Customize a face

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

4.7 One-click workflows

A major advantage of the Emacs document production system: arbitrarily complicated functions can be assigned to very simple keybindings. This means we can automate workflows up to a pretty absurd level.

4.7.1 Export to PDF

PDF is probably the most prevalent file format for sharing static documents.

1. Document

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

2. **TODO** Presentation

4.7.2 Indent buffer

Indent buffer in every mode.

```
(global-set-key [f12] 'sd-indent-buffer)
```

4.7.3 Beautify Org mode buffer

Not only indent, but also clean up superfluous newlines.

```
(local-set-key [f12] 'sd-org-beautify)
```

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 `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.[6]

[6] For more information on `vi` keybindings, visit `https://hea-www.harvard.edu/~fine/Tech/vi.html`.

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

5.3 Spelling, completion, and snippets

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

5.3.1 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)
```

5.3.2 Spelling

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

5.3.3 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).[7]

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

```
#+BEGIN_SRC emacs-lisp :tangle yes

#+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)
```

5.3.4 Complete anything interactively

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

[7] <backtab> is synonymous with pressing shift-tab.

5.3.5 Delete all consecutive whitespaces

```
(use-package hungry-delete)
(global-hungry-delete-mode)
```

5.4 Utilities

5.4.1 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))
```

5.4.2 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))
```

5.4.3 Display keyboard shortcuts on screen

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

5.4.4 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)
```

5.4.5 Graphical representation of file history

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

5.4.6 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

   And he will be called Wonderful **Counselor**, Mighty God, Everlasting Father, Prince of Peace.

   ```
   (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)))
   ```

5.5 Tree

```
(use-package treemacs)
```

5.6 Coding languages

5.6.1 TODO Emacs Lisp

5.6.2 Python

Python is included by default on most Linux distributions.

```
(use-package py-yapf)
(add-hook 'python-mode-hook 'py-yapf-enable-on-save)
```

5.7 File formats

5.7.1 csv and Excel

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

5.7.2 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)
```

5.7.3 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.[8]

```
(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 --depth 2 --invert")
    ("payee" "%(binary) -f %(ledger-file) reg @%(payee)")
    ("account" "%(binary) -f %(ledger-file) reg %(account)")
    ("budget" "%(binary) -f %(ledger-file) budget --exchange USD")))
```

5.7.4 Plotting & charting

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

5.8 Cosmetics

5.8.1 Start page

We replace the standard welcome screen with our own.

[8] For more information, visit `https://www.ledger-cli.org/`.

```
(setq inhibit-startup-message t)
(use-package dashboard
  :config
  (dashboard-setup-startup-hook)
  (setq dashboard-startup-banner (concat user-emacs-directory
  ↪  "img/icons/Safran_logo.svg"))
  (setq dashboard-items '((recents  . 5)
                          (projects . 5)))
  (setq dashboard-banner-logo-title "A modern professional text
  ↪  editor."))
```

5.8.2 TODO Sidebar

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

```
(defun sd-sidebar ()
  (interactive)
  (split-window-right))
```

5.8.3 Better parentheses

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

5.8.4 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))
```

5.8.5 ~~UTF-8 bullet points in `Org` mode~~

This section was removed, as it is more explicit to display the headline character for what it is: a collection of at least one asterisk.

6 `org-mode`

Org mode is so significant that this section of the paper deserves its own introduction.

6.1 Introduction

Phew, after all this initialization, I can finally introduce Org mode! I am so **excited**.

Org mode replaces aword processor, a presentation creator, and a spreadsheet editor. 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!

Check out how much information Org mode keeps concerning the most recent header:

```
(save-excursion
  (org-previous-visible-heading 1)
  (org-entry-properties))
```

```
(("CATEGORY" . "smart-documents")
 ("BLOCKED" . "")
 ("FILE" . "/home/blendux/.emacs.d/smart-documents.org")
 ("PRIORITY" . "B")
 ("ITEM" . "Introduction"))
```

6.2 Basic customization

6.2.1 Base folder

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

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

6.2.2 Prevent/warn on invisible edits

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

6.3 Org cosmetics

First, we ensure the display of markup symbols for **bold**, *italic*, underlined and ~~strikethrough~~ text, and ensure our document appears indented upon loading.[9]

We then set values for many other Org-related cosmetic symbols.

```
(setq org-hide-emphasis-markers nil
      org-startup-indented t
      org-src-preserve-indentation nil
```

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

```
      org-edit-src-content-indentation 2
      org-ellipsis " ▾ ") ; folding symbol
```

6.3.1 Dynamic numbering of headlines

We enable the dynamic numbering of headlines in an Org buffer. We also set the numbering face to `org-special-keyword`, which specifies a `:background white` attribute. This is necessary because otherwise, the background of the numbering may be overridden by the TODO face attribute `:background coral`.

```
(add-hook 'org-mode-hook 'org-num-mode)
(setq org-num-face 'org-special-keyword)
```

By default, we hide Org document properties such as #+TITLE, #+AUTHOR, and #+DATE, because those keywords are defined when the document template is populated. We can nevertheless always access those properties and edit them manually, with a simple keyboard shortcut (cf. Section 4.1.9).

6.3.2 Document properties

```
(defun org-property-value (property)
"Return the value of a given Org document property."
(interactive)
(save-excursion
              (goto-char (point-min))
           (re-search-forward
                     (concat
                      "^[[:space:]]*#\\+"
                      property
                      ":[[:space:]]*\\(.*?\\)[[:space:]]*$")
                     nil t)
(nth 3 (car (cdr (org-element-at-point))))))

(defun sd-document-properties ()
  "Open separate buffer to edit Org mode properties."
  (interactive)
  (let ((title (car (org-property-value "TITLE")))
          (date (org-property-value "DATE")))
    (with-output-to-temp-buffer "Smart Document Properties"
      (print title)
      (print date))))

(add-hook 'org-src-mode-hook
          '(lambda ()
             "Disable flycheck for `emacs-lisp-mode'."
```

```
            (setq-local flycheck-disabled-checkers
                        '(emacs-lisp-checkdoc))))
```

6.3.3 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"))
```

6.4 Programming a Smart Documents

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

```
(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)))
```

6.5 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`.[10]

We open the agenda in a separate window.

```
(setq org-agenda-window-setup 'other-frame)
```

6.6 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.

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

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-excape` 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"))
```

6.6.1 Exporting timestamps

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}")
```

6.6.2 LaTeX packages

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

```
(setq org-latex-packages-alist
      '(("AUTO" "babel" t
         ("pdflatex"))
        ("AUTO" "polyglossia" t ; Babel replacement for LuaLaTeX
         ("xelatex" "lualatex"))
        ("" "booktabs" t ; Publication quality tables in LaTeX
         ("pdflatex"))
        ("table,svgnames" "xcolor" t ; svgnames opens up ~150 color
        ↪  keywords
         ("pdflatex"))
        ("skip=0.5\\baselineskip, width=0.618\\textwidth" "caption" t
        ↪  ; Increase space between floats and captions
         ("pdflatex" "lualatex"))))
```

6.6.3 Colored source blocks in PDF export

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")
                                   ("breaklines" "true")
                                   ("breakanywhere" "true"))))
```

6.6.4 Cleaning directory after export

Now, we set the files to be deleted when a LaTeX → 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"))
```

6.6.5 Chronological diary entries

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 is an additional document class structures that can be exported in LaTeX.

```
;; (add-to-list 'org-latex-classes
;;              '("book-blendoit"
;;                "\\documentclass[12pt]{book}"
;;                ("\\chapter{%s}" . "\\chapter*{%s}")
;;                ("\\section{%s}" . "\\section*{%s}")
;;                ("\\subsection*{%s}" . "\\subsection*{%s}")
;;                ("\\subsubsection*{%s}" . "\\subsubsection*{%s}")))
```

6.6.6 Table of contents

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!

6.7 TODO Org links

This is a mind-bending capacity of Org mode: we can assign arbitrary functions to be executed when a user follows an Org link. Org links appear like hyperlinks both in buffers and PDF exports — e.g. the following link to this very section, Section 6.7 — but their in-buffer behavior can be arbitrarily assigned.

```
(org-add-link-type
 "tag" 'endless/follow-tag-link)

(defun endless/follow-tag-link (tag)
  "Display a list of TODO headlines with tag TAG.
With prefix argument, also display headlines without a TODO keyword."
  (org-tags-view (null current-prefix-arg) tag))

[[tag:work+phonenumber-boss][Optional Description]]
```

7 One-click workflows

In this section, we'll implement useful one-click workflows. It comes later keybinding definitions for two reasons:

1. To a new user, keybindings are more important than the precise implementation of the bound function — it is more important to know how to drive a car than how a car works.
2. If the following subsections share the same name as the keybinding subsection (Section 4), the links will resolve to the earliest heading in the document, i.e. the keybinding subsection and not the subsection describing the 'one-click workflow'.

7.1 TODO Export to PDF

This reimplements the most common Org mode export: Org → LaTeX → PDF. The binding is defined in Section 4.7.1.

```
(defun sd-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)))
```

7.2 Beautify buffer

Binding defined in Section 4.7.2.

```
(defun sd-indent-buffer ()
  "Indent entire buffer."
  (interactive)
  (save-excursion
    (indent-region (point-min) (point-max) nil)))

(defun sd-org-beautify ()
  "Beautify Org mode buffer."
  (interactive)
  (when (eq major-mode 'org-mode)
    (sd-indent-buffer)))
```

8 Editing preferences

These customizations enhance editor usability. They also encompass cosmetic changes not brought about a specific package.

8.1 Editor

8.1.1 Coding standards

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

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

8.1.2 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)
```

8.1.3 Fill column

```
(setq fill-column 66)
```

8.2 Frame

8.2.1 Header & mode lines

1. **TODO** Icons We start by defining some icons we wish to include in our user interface. Emacs allows the usage of GIF images — this paves the way for UI elements which may be animated.

```
  (defvar sd-icons-blue-ellipsis (create-image
                                  (concat user-emacs-directory
                                  ↪  "img/icons/ellipsis.gif")
                                  'gif nil
                                  :scale 0.4) "A blue loading
                                  ↪  ellipsis.")

(defun sd-icons-blue-ellipsis ()
    "Insert an animated blue ellipsis."
    (insert-image sd-icons-blue-ellipsis)
    (image-animate sd-icons-blue-ellipsis 0 t))
```

(sd-icons-blue-ellipsis)

2. **TODO** Header line In Org mode, the document header line will be the title of the document we are working on currently.

```
(setq sd-header-gnu-linux
      (list
       (propertize "..." 'display sd-icons-blue-ellipsis)
       '(:eval
         (list
          (if (eq (length (window-list)) 1)
              (propertize " ↤ " 'mouse-face 'highlight
                          'face 'org-special-keyword
                          'local-map 'previous-buffer
                          'help-echo "Return to previous window.")
            (list (propertize " □ " 'mouse-face 'org-todo
                              'face 'org-special-keyword
                              'help-echo "Close this window.")
                  (propertize " ⇱" 'mouse-face 'highlight
                              'face 'org-special-keyword
                              'help-echo "Maximize this window.")
                  (propertize "⇲ " 'mouse-face 'highlight
                              'face 'org-special-keyword
                              'help-echo "Minimize this window.")))
          (if (string-equal major-mode "org-mode")
```

```
                (org-property-value "TITLE")
              (buffer-name))))))

  (setq sd-header-windows-nt
        (list
         '(:eval
           (list
            (if (eq (length (window-list)) 1)
                (propertize " ↔ " 'mouse-face 'highlight
                            'face 'org-special-keyword
                            'local-map 'previous-buffer
                            'help-echo "Return to previous window.")
              (list (propertize "  " 'mouse-face 'org-todo
                                'face 'org-special-keyword
                                'help-echo "Close this window.")
                    (propertize " ⇱" 'mouse-face 'highlight
                                'face 'org-special-keyword
                                'help-echo "Maximize this window.")
                    (propertize "⇲ " 'mouse-face 'highlight
                                'face 'org-special-keyword
                                'help-echo "Minimize this window.")))
            (if (string-equal major-mode "org-mode")
                (org-property-value "TITLE")
              (buffer-name))))))

  (cond ((string-equal system-type "windows-nt")
         (setq header-line-format sd-header-windows-nt))
        ((string-equal system-type "gnu'linux")
         (setq header-line-format sd-header-gnu-linux)))

  ;; Ensure our icons are animated on start
  (image-animate sd-icons-blue-ellipsis 0 t)
```

3. Mode line

```
  (setq-default mode-line-format
        (list
         '(:eval
           (list
            "    "
            (if buffer-read-only "" "")
            (propertize " %b "
                        'help-echo (buffer-file-name)
                        )
```

```
            (if (buffer-modified-p) "" "")))))
```

8.3 Window

8.4 Buffer

8.4.1 Save cursor location

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

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

8.4.2 Column filling

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

```
(setq-default fill-column 79)
(add-hook 'org-mode-hook
          'turn-on-auto-fill) ; Automatically break lines longer than
                              ; =fill-column=.
```

8.5 Text

8.5.1 Beautiful symbols

We want the Emacs Lisp keyword `lambda` to be rendered as λ within the editor. This is mostly for a subjective “cool” factor.

```
(global-prettify-symbols-mode 1)
```

8.5.2 Org mode sugar

Let’s pimp out the appearance of our text in Org mode. First, we prettify checkbox lists.

```
(when (string-equal system-type "gnu/linux")
  (add-hook 'org-mode-hook
            (lambda ()
              "Beautify Org symbols."
              (push '("[ ]" .  "○") prettify-symbols-alist) ;
               ↪  Unchecked item
              (push '("[X]" . "◉" ) prettify-symbols-alist) ; Checked
               ↪  item
              (push '("[-]" . "◎" ) prettify-symbols-alist) ;
               ↪  Partially checked item
              (push '("-" . "" ) prettify-symbols-alist) ; DONE
               ↪  headings
              (prettify-symbols-mode))))
```

- ☐ This first item is unticked
- ⊟ This second item is partially completed
  - ☒ This first sub-item is ticked
  - ☐ This sub-item is not ticked
- ☐ This third item is ticked

8.5.3 Electric modes

Electricity is a very important technology. In Emacs jargon, “electric” modes tend to automate behaviors or present some elegant simplification to a workflow.[11]

```
(electric-pair-mode) ; Certain character pairs are automatically
↪  completed.
(electric-indent-mode) ; Newlines are always intelligently indented.
```

8.6 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)
```

9 Themes

Without a carefully designed theme, our editor would 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)
```

9.1 My light and dark themes

A highly legible, unambiguous, and classic theme.

[11] More information can be found at `https://www.emacswiki.org/emacs/Electricity`.

9.1.1 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 2).

Table 2: Light and dark themes' colors.

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

9.1.2 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)
```

9.1.3 Fonts

1. Currently used *chad fonts*

   Hack[12] `default` and `fixed-pitch`, default code font
   - Legible, modern monospace font
   - Strict, sharp, uncompromising

   Public Sans[13] `variable-pitch`, default body text font
   - Very modern yet neutral
   - Designed for the U.S. government
   - Exceptional color on screen

   Hermit[14] `org-block`, anything Org/meta in general
   - Slightly wider than Hack
   - More opinionated shapes
   - Very legible parentheses, very useful for Emacs Lisp!

   Jost[15] `org-document-title` and `org-level-1`
   - Ultra-modern
   - Tasteful amount of geometric inspiration

2. Previously used *virgin fonts*

   ~~**Liberation Sans**~~[16] ~~`variable-pitch`~~
   - Metrically compatible with *Arial* (ugh)
   - Unoffensive, unambitious forms
   - Pretty angular letters, it's like you're trying to read squares

   ~~**Open Sans**~~[17] ~~`variable-pitch`~~
   - Ooh geometric Bauhaus influences, look at me
   - Tall leading height is `h a r m o n i o u s`

3. 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)
   ```

[12] `https://sourcefoundry.org/hack/`
[13] `https://public-sans.digital.gov/`
[14] `https://pcaro.es/p/hermit/`
[15] `https://indestructibletype.com/Jost.html`
[16] `https://en.wikipedia.org/wiki/Liberation_fonts`
[17] `https://www.opensans.com/`

4. **TODO** Default font size

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

   ```
   (if (< screen-width 1920)
       (default-font)
     else)
   ```

9.2 Wealthy theme

Figure 1: Claude Garamont, an icon of font design. World-renowned for his elegant typefaces, which inspired many generations of typographers.

Good golly, nobody wishes for a *pedestrian* theme! Let your entourage know that you're rocking an editor fit for a king with this finely crafted `wealthy' theme. Selecting it shall enable the following fancitudes: □

1. The default font shall be sublimed in the form of *EB Garamond*
2. Bullets will be tastefully replaced with pointing fingers
3. Heading stars will be replaced with Black Queen chess pieces

Claude Garamont (c. 1510--1561), known commonly as **Claude Garamond**, was a French type designer, publisher and punch-cutter based in Paris. Garamond worked as an engraver of punches, the masters used to stamp matrices, the moulds used to cast metal type. He worked in the tradition now called old-style serif design, which produced letters with a relatively organic structure resembling handwriting with a pen but with a slightly more structured and upright design. Considered one of the leading type designers of all time, he is recognised to this day for the elegance of his typefaces. Many old-style serif typefaces are collectively known as Garamond, named after the designer.

From `https://en.wikipedia.org/wiki/Claude_Garamond`

9.2.1 Symbol substitution

```
(defun sd-wealthy ()
  "Beautify symbols for our wealthy theme."
  (push '("-" . "□" ) prettify-symbols-alist) ; unnumbered bullets
  (push '("*" . "□" ) prettify-symbols-alist) ; headings
  (prettify-symbols-mode))
```

9.3 TODO `minimal`

10 Late setup

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

10.1 Profiling — stop

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

10.2 Profiling — report

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

11 Conclusion

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