# Emacs Configuration

Goncalo M. V. Henriques

May 28, 2018

## Contents

## 1 Personal info

```
(setq user-full-name "Goncalo M. V. Henriques"
user-mail-address "gmv.henriques@gmail.com")
```

## 2 Use `sensible-defaults.el`

Use sensible-defaults.el for some basic settings.

```
(load-file "~/.emacs.d/sensible-defaults.el")
(sensible-defaults/use-all-settings)
(sensible-defaults/use-all-keybindings)
(sensible-defaults/backup-to-temp-directory)
```

## 3 Load-path

```
(add-to-list 'load-path "~/Dropbox/emacs/lisp")
```

## 4 Configure `use-package`

Gradually moving to use-package.

```
(unless (package-installed-p 'use-package)
        (package-refresh-contents)
        (package-install 'use-package))

(require 'use-package)
```

Always compile packages, and use the newest version available (auto-compile).

```
(use-package auto-compile
  :ensure t
  :config (auto-compile-on-load-mode))
(setq load-prefer-newer t)
```

## 5 Configure evil-mode

Evil its a vim emulator.

```
(use-package evil
  :ensure t
  :init
  (evil-mode 1))
```

Enable surround everywhere.

```
(use-package evil-surround
  :ensure t
  :config
  (global-evil-surround-mode 1))
```

Bind `C-p` to fuzzy-finding files in the current project.

```
(define-key evil-normal-state-map (kbd "C-p") 'projectile-find-file)
```

## 6 UI preferences

### 6.1 Tweak window chrome

I don't usually use the menu or scroll bar, and they take up useful space.

```
(tool-bar-mode 0)
(menu-bar-mode 0)
(when window-system
  (scroll-bar-mode -1))
```

### 6.2 Function to toggle between themes

I like the solarized-dark theme. I prefer keeping all the characters in the same side and font, though.

```
(defun gh/apply-solarized-theme ()
    (setq solarized-use-variable-pitch nil)
    (setq solarized-height-plus-1 1.0)
    (setq solarized-height-plus-2 1.0)
    (setq solarized-height-plus-3 1.0)
    (setq solarized-height-plus-4 1.0)
    (setq solarized-high-contrast-mode-line t)
    (load-theme 'solarized-dark t))
```

Define the light color theme.

```
(defcustom default-light-color-theme 'solarized-light
"default light theme")
```

Define the dark color theme.

```
(defcustom default-dark-color-theme 'solarized-dark
"default dark theme")
```

With this function I can toggle between the dark and the light theme. To solve a problem with org-bullets, ensure that when changing theme org-mode restarts.

```
(defun gh/toggle-dark-light-theme ()
(interactive)

(let ((is-light (find default-light-color-theme custom-enabled-themes)))
  (dolist (theme custom-enabled-themes)
    (disable-theme theme))
  (load-theme (if is-light default-dark-color-theme default-light-color-theme))
  (if (org-mode) (org-mode-restart))))
```

I set the key `<f12>` to toggle between themes.

```
(global-set-key (kbd "<f12>") 'gh/toggle-dark-light-theme)
```

If this code is being evaluated by emacs –daemon, ensure that each subsequent frame is themed appropriately.

```
(if (daemonp)
  (add-hook 'after-make-frame-functions
            (lambda (frame)
                (gh/apply-solarized-theme)))
(gh/apply-solarized-theme))
```

## 6.3 Disable visual bell

`sensible-defaults` replaces the audible bell with a visual one, but I really don't even want that (and my Emacs/Mac pair renders it poorly). This disables the bell altogether.

```
(setq ring-bell-function 'ignore)
```

### 6.4 Scroll conservatively

When point goes outside the window, Emacs usually recenters the buffer point. I'm not crazy about that. This changes scrolling behavior to only scroll as far as point goes.

```
(setq scroll-conservatively 100)
```

### 6.5 Highlight the current line

`global-hl-line-mode` softly highlights the background color of the line containing point. It makes it a bit easier to find point, and it's useful when pairing or presenting code.

```
(when window-system
  (global-hl-line-mode))
```

### 6.6 Display the current column number

Display the current column.

```
(setq column-number-mode t)
```

## 7 Publishing and task management with Org-mode

### 7.1 Display preferences

I like to see an outline of pretty bullets instead of a list of asterisks.

```
(use-package org-bullets
  :ensure t
  :init
  (add-hook 'org-mode-hook #'org-bullets-mode))
```

I like seeing a little downward-pointing arrow instead of the usual ellipsis (...) that org displays when there's stuff under a header.

```
(setq org-ellipsis "")
```

Fontify code in code blocks

```
(setq org-src-fontify-natively t)
```

Make TAB act as if it were issued in a buffer of the language's major mode.

```
(setq org-src-tab-acts-natively t)
```

```
(setq org-src-window-setup 'current-window)
```

Quickly insert a block of elisp: (<el)

```
(add-to-list 'org-structure-template-alist
             '("el" "#+BEGIN_SRC emacs-lisp\n?\n#+END_SRC"))
```

Enable spell-checking in Org-mode. The quick brown fox jumps over the lazy dog.

```
(add-hook 'org-mode-hook 'flyspell-mode)
```

```
(setq org-hierarchical-todo-statistics nil)
```

## 7.2 Org-mode

Store my org files in `~/Dropbox/org/`, define the location of the index file, and archive finished tasks in `~/Dropbox/org/archive.org`.

```
(setq org-directory "~/Dropbox/org")
(defun org-file-path (filename)
        "Return the absolute address of an org file, given its relative name."
        (concat (file-name-as-directory org-directory) filename))

(setq org-index-file (org-file-path "index.org"))
(setq org-archive-location
        (concat (org-file-path "archive.org") "::* From %s"))
```

Derive my agenda from this directory:

```
(setq org-agenda-files '("~/Dropbox/org"))
```

Hitting `C-c C-x C-s` will mark a todo as done and move it to an appropriate place in the archive.

```
(defun gh/mark-done-and-archive ()
  "Mark the state of an org-mode item as DONE and archive it."
  (interactive)
  (org-todo 'done)
  (org-archive-subtree))

(define-key org-mode-map (kbd "C-c C-x C-s") 'gh/mark-done-and-archive)
```

Record the time that a todo was archived.

```
(setq org-log-done 'time)
```

### 7.2.1 Capturing tasks

Define a few common tasks as capture templates. Specifically, I frequently:

- Record ideas for future blog posts in `~/Dropbox/org/blog-ideas.org`,
- Maintain a todo list in `~/org/index.org`.
- Convert emails into todos to maintain an empty inbox.

```
(setq org-capture-templates
      '(("a" "Appointment"
         entry
         (file  "~/Dropbox/org/calendar.org" )
         "* %?\n\n%^T\n\n:PROPERTIES:\n\n:END:\n\n")

        ("b" "Blog idea"
         entry
         (file (org-file-path "blog-ideas.org"))
         "* %?\n")

        ("e" "Email" entry
         (file+headline org-index-file "Inbox")
         "* TODO %?\n\n%a\n\n")

        ("f" "Finished book"
         table-line (file "~/documents/notes/books-read.org")
         "| %^{Title} | %^{Author} | %u |")

        ("r" "Reading"
         checkitem
         (file (org-file-path "to-read.org")))

        ("s" "Subscribe to an RSS feed"
         plain
         (file "~/documents/rss/urls")
         "%^{Feed URL} \"~%^{Feed name}\"")

        ("t" "Todo"
         entry
         (file+headline org-index-file "Inbox")
         "* TODO %?\n")))
```

When I'm starting an Org capture template I'd like to begin in insert mode. I'm opening it up in order to start typing something, so this skips a step.

```
(add-hook 'org-capture-mode-hook 'evil-insert-state)
```

When refiling an item, I'd like to use ido for completion.

```
(setq org-refile-use-outline-path t)
(setq org-outline-path-complete-in-steps nil)
```

### 7.2.2 Keybindings

Bind a few handy keys.

```
(define-key global-map "\C-cl" 'org-store-link)
(define-key global-map "\C-ca" 'org-agenda)
(define-key global-map "\C-cc" 'org-capture)
```

Hit `C-c i` to quickly open up my index file.

```
(defun gh/open-index-file ()
  "Open the master org TODO list."
  (interactive)
  (find-file org-index-file)
  (flycheck-mode -1)
  (end-of-buffer))

(global-set-key (kbd "C-c i") 'gh/open-index-file)
```

## 7.3 Sync Org-mode with Google Calendar

I use org-gcal to sync my Google calendar.

```
(setq package-check-signature nil)
```

```
(use-package org-gcal
  :ensure t
  :config
(setq org-gcal-client-id "107011808994-g9s382a66p4d3f78ibkccl15sjgh7a9n.apps.googleuser
          org-gcal-client-secret "Gjfci0moPki0d_APpcqEL3WF"
          org-gcal-file-alist '(("gmv.henriques@gmail.com" .  "~/Dropbox/org/calendar.
```

I use these two hooks to sync things semi-automatically. The first hook syncs whenever I load the agenda. Since this happens in the background, if I just added something to my calendar, I might have to reload the agenda by hitting r in the agenda view. The second hook syncs with my Google calendar when I capture.

```
(add-hook 'org-agenda-mode-hook (lambda () (org-gcal-sync) ))
(add-hook 'org-capture-after-finalize-hook (lambda () (org-gcal-sync) ))
```

Calfw it's a nice tool to view calendars in Google.

```
  (use-package calfw
    :ensure t
    :config
    (use-package calfw-ical
    :ensure t
    :config
    (use-package calfw-org
    :ensure t
    :config
    (setq cfw:display-calendar-holidays nil)
    (defun mycalendar ()
      (interactive)
      (cfw:open-calendar-buffer
       :contents-sources
       (list
        (cfw:org-create-source "Green")
; (cfw:ical-create-source "Gcal" "https://calendar.google.com/calendar/ical/gmv.henriqu
        (cfw:ical-create-source "Feriados" "https://calendar.google.com/calendar/ical/p
        )))
  )
  )
  )
```

# 8 Editing Settings

## 8.1 Quickly visit Emacs configuration

I futz around with my dotfiles a lot. This binds `C-c e` to quickly open my Emacs configuration file.

```
(defun gh/visit-emacs-config ()
  (interactive)
  (find-file "~/Dropbox/emacs/configuration.org"))

(global-set-key (kbd "C-c e") 'gh/visit-emacs-config)
```

## 8.2 Always kill current buffer

Assume that I always want to kill the current buffer when hitting `C-x k`.

```
(global-set-key (kbd "C-x k") 'gh/kill-current-buffer)
```

## 8.3 Use `company-mode` everywhere

```
(use-package company
:ensure t
:init
(add-hook 'after-init-hook 'global-company-mode)
)
```

## 8.4 `Saveplace`

Purpose: When you visit a file, point goes to the last place where it was when you previously visited the same file.

```
(use-package saveplace
 :ensure t
 :init
 (save-place-mode 1)
)
```

Save Place

### 8.5 Always indent with spaces

Never use tabs. Tabs are the devil's whitespace.

```
(setq-default indent-tabs-mode nil)
```

### 8.6 Configure `yasnippet`

I keep my snippets in `~/Dropbox/emacs/snippets/text-mode`, and I always want `yasnippet` enabled.

```
(use-package yasnippet
:ensure t
:init
  (setq yas-snippet-dirs '("~/Dropbox/emacs/snippets/text-mode"))
  (yas-global-mode 1))
```

I don't want `ido` to automatically indent the snippets it inserts. Sometimes this looks pretty bad (when indenting org-mode, for example, or trying to guess at the correct indentation for Python).

```
(setq yas/indent-line nil)
```

### 8.7 Configure `ido`

```
(use-package ido
  :ensure t
  :init
  (setq ido-enable-flex-matching t)
  (setq ido-everywhere t)
  (ido-mode 1)

  (use-package flx-ido
    :ensure t
    :init
    (flx-ido-mode 1) ; better/faster matching
  )

(setq ido-create-new-buffer 'always) ; don't confirm to create new buffers

  (use-package ido-vertical-mode
    :ensure t
    :init
    (ido-vertical-mode 1)
    (setq ido-vertical-define-keys 'C-n-and-C-p-only)
  )
)
```

ido flx-ido ido-vertical-mode

### 8.8 Electric pair

Typing any left bracket automatically insert the right matching bracket.

```
(electric-pair-mode 1)
```

### 8.9 Rainbow-delimiters

```
(use-package rainbow-delimiters
  :ensure t
  :commands rainbow-delimiters-mode
  :init
  (add-hook 'prog-mode-hook #'rainbow-delimiters-mode)
  (add-hook 'LaTex-mode-hook #'rainbow-delimiters-mode)
  (add-hook 'org-mode-hook 'rainbow-delimiters-mode))
```

### 8.10 Use `smex` to handle `M-x` with `ido`

```
(use-package smex
  :ensure t
  :init
  (smex-initialize)
)
```

```
(global-set-key (kbd "M-x") 'smex)
(global-set-key (kbd "M-X") 'smex-major-mode-commands)
```

### 8.11 Switch and rebalance windows when splitting

When splitting a window, I invariably want to switch to the new window. This makes that automatic.

```
 (defun gh/split-window-below-and-switch ()
  "Split the window horizontally, then switch to the new pane."
  (interactive)
  (split-window-below)
  (balance-windows)
  (other-window 1))

(defun gh/split-window-right-and-switch ()
  "Split the window vertically, then switch to the new pane."
  (interactive)
  (split-window-right)
  (balance-windows)
  (other-window 1))

(global-set-key (kbd "C-x 2") 'gh/split-window-below-and-switch)
(global-set-key (kbd "C-x 3") 'gh/split-window-right-and-switch)
```

## 9 Writing

### 9.1 Change dictionary

Change dictionary to `PT-preao`

```
(global-set-key
[f3]
(lambda ()
    (interactive)
    (ispell-change-dictionary "pt_PT-preao")))
```

Change dictionary to `En`

```
(global-set-key
[f4]
(lambda ()
    (interactive)
    (ispell-change-dictionary "en")))
```

### 9.2 Wrap paragraphs automatically

`AutoFillMode` automatically wraps paragraphs.

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

Sometimes, though, I don't wanna wrap text. This toggles wrapping with `C-c q`:

```
(global-set-key (kbd "C-c q") 'auto-fill-mode)
```

### 9.3 `Flyspell-popup`

Call flyspell-popup-correct to correct misspelled word at point with a Popup Menu. You might want to bind it to a short key, for example:

```
(use-package flyspell-popup
:ensure t
:init
(define-key flyspell-mode-map (kbd "C-;") #'flyspell-popup-correct))
```

### 9.4 Darkroom

```
(use-package darkroom
   :ensure t)
```

## 10 Email with `mu4e`

### 10.1 Evil

Use the evil bindings for navigation. They're very similar to the mutt bindings, which matches my muscle memory nicely. =)

```
(require 'evil-mu4e)
```

### 10.2 Where's my mail? Who am I?

I keep my mail in `~/.mail`. The default mail directory would be `~/Maildir`, but I'd rather hide it; I don't poke around in there manually very often.

This setting matches the paths in my mbsync configuration.

```
(setq mu4e-maildir "~/.mail")
```

I only have one context at the moment. If I had another email account, though, I'd define it in here with an additional `make-mu4e-context` block.

My full name is defined earlier in this configuration file.

```
(setq mu4e-contexts
      `(,(make-mu4e-context
          :name "gmail"
          :match-func (lambda (msg)
                        (when msg
                          (string-prefix-p "/gmail" (mu4e-message-field msg :maildir))
          :vars '((user-mail-address . "gmv.henriques@gmail.com")
                  (mu4e-trash-folder . "/gmail/archive")
                  (mu4e-refile-folder . "/gmail/archive")
                  (mu4e-sent-folder . "/gmail/sent")
                  (mu4e-drafts-folder . "/gmail/drafts")))))
```

### 10.3 Fetching new mail

I fetch my email with `mbsync`. I've also bound "o" to fetch new mail.

```
(setq mu4e-get-mail-command "killall --quiet mbsync; mbsync inboxes")

(define-key mu4e-headers-mode-map (kbd "o") 'mu4e-update-mail-and-index)
```

Rename files when moving them between directories. `mbsync` supposedly prefers this; I'm cargo-culting.

```
(setq mu4e-change-filenames-when-moving t)
```

Poll the server for new mail every 1 minute.

```
(setq mu4e-update-interval 60)
```

### 10.4 Viewing mail

I check my email pretty often! Probably more than I should. This binds `C-c m` to close any other windows and open my personal inbox.

In practice, I keep an **mu4e-headers** buffer in its own frame, full-screen, on a dedicated i3 workspace.

```
(defun gh/visit-inbox ()
  (interactive)
  (delete-other-windows)
  (mu4e~headers-jump-to-maildir "/gmail/inbox"))

(global-set-key (kbd "C-c m") 'gh/visit-inbox)
```

Open my inbox and sent messages folders with `J-i` and `J-s`, respectively. These are the only two folders I visit regularly enough to warrant shortcuts.

```
(setq mu4e-maildir-shortcuts '(("/gmail/inbox" . ?i)
                               ("/gmail/sent" . ?s)))
```

`mu4e` starts approximately instantaneously, so I don't know why I'd want to reconsider quitting it.

```
(setq mu4e-confirm-quit nil)
```

### 10.5 Composing a new message

When I'm composing a new email, default to using the first context.

```
(setq mu4e-compose-context-policy 'pick-first)
```

Compose new messages (as with `C-x m`) using `mu4e-user-agent`.

```
(setq mail-user-agent 'mu4e-user-agent)
```

Enable Org-style tables and list manipulation.

```
(add-hook 'message-mode-hook 'turn-on-orgtbl)
(add-hook 'message-mode-hook 'turn-on-orgstruct++)
```

Check my spelling while I'm writing.

```
(add-hook 'mu4e-compose-mode-hook 'flyspell-mode)
```

Once I've sent an email, kill the associated buffer instead of just burying it.

```
(setq message-kill-buffer-on-exit t)
```

## 10.6 Reading an email

Display the sender's email address along with their name.

```
(setq mu4e-view-show-addresses t)
```

Save attachments in my ~/downloads directory, not my home directory.

```
(setq mu4e-attachment-dir "~/downloads")
```

Hit C-c C-o to open a URL in the browser.

```
(define-key mu4e-view-mode-map (kbd "C-c C-o") 'mu4e~view-browse-url-from-binding)
```

While HTML emails are undeniably sinful, we often have to read them. That's sometimes best done in a browser. This effectively binds a h to open the current email in my default Web browser.

```
(add-to-list 'mu4e-view-actions '("html in browser" . mu4e-action-view-in-browser) t)
```

## 10.7 Encryption

If a message is encrypted, my reply should always be encrypted, too.

```
(defun gh/encrypt-responses ()
  (let ((msg mu4e-compose-parent-message))
    (when msg
      (when (member 'encrypted (mu4e-message-field msg :flags))
        (mml-secure-message-encrypt-pgpmime)))))

(add-hook 'mu4e-compose-mode-hook 'gh/encrypt-responses)
```

## 10.8 Sending mail over SMTP

I send my email through `msmtp`. It's very fast, and I've already got it configured from using `mutt`. These settings describe how to send a message:

- Use a sendmail program instead of sending directly from Emacs,
- Tell `msmtp` to infer the correct account from the `From:` address,
- Don't add a "`-f username`" flag to the `msmtp` command, and

- Use `/usr/bin/msmtp`!

```
(setq message-send-mail-function 'message-send-mail-with-sendmail)
(setq message-sendmail-extra-arguments '("--read-envelope-from"))
(setq message-sendmail-f-is-evil 't)
(setq sendmail-program "msmtp")
```

### 10.9 Org integration

`org-mu4e` lets me store links to emails. I use this to reference emails in my TODO list while keeping my inbox empty.

```
(require 'org-mu4e)
```

When storing a link to a message in the headers view, link to the message instead of the search that resulted in that view.

```
(setq org-mu4e-link-query-in-headers-mode nil)
```

### 10.10 Configure BBDB with mu4e

Use BBDB to handle my address book.

```
(require 'bbdb-mu4e)
```

Don't try to do address completion with mu4e. Use BBDB instead:

```
(setq mu4e-compose-complete-addresses nil)
```

### 10.11 Try to display images in mu4e

```
(setq
 mu4e-view-show-images t
 mu4e-view-image-max-width 800)
```

## 11 My `latex` environment

### 11.1 `Auctex`

```
(use-package tex-site
  :ensure auctex
  :mode ("\\.tex\\'" . latex-mode)
  :config
  (setq TeX-auto-save t)
  (setq TeX-parse-self t)
  (setq-default TeX-master nil)
  (add-hook 'LaTeX-mode-hook
            (lambda ()
              (rainbow-delimiters-mode)
              (company-mode)
              (smartparens-mode)
              (turn-on-reftex)
              (setq reftex-plug-into-AUCTeX t)
              (reftex-isearch-minor-mode)
              (setq TeX-PDF-mode t)
              (setq TeX-source-correlate-method 'synctex)
              (setq TeX-source-correlate-start-server t)))

;; Update PDF buffers after successful LaTeX runs
(add-hook 'TeX-after-TeX-LaTeX-command-finished-functions
           #'TeX-revert-document-buffer)

;; to use pdfview with auctex
(add-hook 'LaTeX-mode-hook 'pdf-tools-install)

;; to use pdfview with auctex
(setq TeX-view-program-selection '((output-pdf "pdf-tools"))
       TeX-source-correlate-start-server t)
(setq TeX-view-program-list '(("pdf-tools" "TeX-pdf-tools-sync-view"))))
```

### 11.2 Reftex

```
(use-package reftex
  :ensure t
  :defer t
  :config
  (setq reftex-cite-prompt-optional-args t)); Prompt for empty optional arguments in c
```

### 11.3 Ivy-bibtex

### 11.4 Pdf-tools

```
(use-package pdf-tools
  :ensure t
  :mode ("\\.pdf\\'" . pdf-tools-install)
  :bind ("C-c C-g" . pdf-sync-forward-search)
;  :defer t
  :config
  (setq mouse-wheel-follow-mouse t)
  (setq pdf-view-resize-factor 1.10))
```

### 11.5 magic-latex-buffer

```
(use-package magic-latex-buffer
  :ensure t)
(add-hook 'LaTeX-mode-hook 'magic-latex-buffer)
```

## 12 Elfeed

```
(use-package elfeed-org
  :ensure t
  :config
  (progn
    (elfeed-org)
    (setq rmh-elfeed-org-files (list "~/Dropbox/org/feeds.org"))))
```

```
(use-package elfeed)
  :ensure t
```

## 13 Header

Time-stamp!

```
(add-hook 'before-save-hook 'time-stamp)
```

```
(setq
  time-stamp-pattern nil
  time-stamp-active t          ; do enable time-stamps
  time-stamp-line-limit 12     ; check first 10 buffer lines for Time-stamp:
  time-stamp-format "%04y-%02m-%02d %02H:%02M:%02S (%u)") ; date format
```

Header2.el!

```
(use-package header2
  :config
  (progn

    (defconst gh/header-sep-line-char ?-
      "Character to be used for creating separator lines in header.")

    (defconst gh/header-border-line-char ?=
      "Character to be used for creating border lines in header.")

    (defconst gh/auto-headers-hooks '(latex-mode-hook
                                      LaTeX-mode-hook)
      "List of hooks of major modes in which headers should be auto-inserted.")

    (defvar gh/header-timestamp-cond (lambda () t)
      "This variable should be set to a function that returns a non-nil
      value only when the time stamp is supposed to be inserted. By default, it's
      a `lambda' return `t', so the time stamp is always inserted.")

    (defun gh/turn-on-auto-headers ()
      "Turn on auto headers only for specific modes."
      (interactive)
      (dolist (hook gh/auto-headers-hooks)
        (add-hook hook #'auto-make-header)))

    (defun gh/turn-off-auto-headers ()
      "Turn off auto headers only for specific modes."
      (interactive)
      (dolist (hook gh/auto-headers-hooks)
        (remove-hook hook #'auto-make-header)))


    (defsubst gh/header-sep-line ()
      "Insert separator line"
      (insert header-prefix-string)
      (insert-char gh/header-sep-line-char (- fill-column (current-column)))
      (insert "\n"))

    (defsubst gh/header-border-line ()
      "Insert separator line"
      (insert header-prefix-string)
      (insert-char gh/header-border-line-char (- fill-column (current-column)))
      (insert "\n"))


    (defsubst gh/header-file-name ()
      "Insert \"File Name\" line, using buffer's file name."
      (insert header-prefix-string "File Name          : "
              (if (buffer-file-name)
                  (file-name-nondirectory (buffer-file-name))
                (buffer-name))
              "\n"))

    (defsubst gh/header-author ()
      "Insert current user's name (`user-full-name') as this file's author."
      (insert header-prefix-string "Author             : "
              (user-full-name)
              "\n"))

    (defsubst gh/header-mail ()
      "Insert current user's name (`user-mail-address') as this file's author."
      (insert header-prefix-string "Author e-mail      : "
              user-mail-address
              "\n"))

    (defsubst gh/header-description ()
      "Insert \"Description\" line."
      (insert header-prefix-string "Description        : \n"))

    (defsubst gh/header-creation-date ()
      "Insert todays date as the time of last modification."
      (insert header-prefix-string "Created            : "
              (header-date-string)
              "\n"))
```

```
    (defsubst gh/header-timestamp ()
      "Insert field for time stamp."
      (when (funcall gh/header-timestamp-cond)
      (insert header-prefix-string "Time-stamp: <>\n")))

    (defsubst gh/header-modification-date ()
      "Insert todays date as the time of last modification.
       This is normally overwritten with each file save."
      (insert header-prefix-string "Last-Updated       :"
             "\n"))


    (defsubst gh/header-position-point ()
      "Position the point at a particular point in the file.
Bring the point 2 lines below the current point."
      (forward-line 0)
      (newline 2))


    (setq make-header-hook '(gh/header-border-line
                             header-blank
                             gh/header-file-name
                             gh/header-author
                             gh/header-mail
                             gh/header-creation-date
                             header-blank
                             gh/header-sep-line
                             header-blank
                             gh/header-timestamp
                             header-blank
                             gh/header-sep-line
                             header-blank
                             gh/header-description
                             header-modification-date
                             header-blank
                             gh/header-border-line
                             gh/header-position-point))
    (gh/turn-on-auto-headers)
    ))
```

## 14 Dired

Load up the assorted `dired` extensions.

```
(use-package dired-details)
(use-package dired+)
```

Open media with the appropriate programs.

```
(use-package dired-open
  :config
  (setq dired-open-extensions
        '(("pdf" . "evince")
          ("mkv" . "vlc")
          ("mp4" . "vlc")
          ("avi" . "vlc"))))
```

These are the switches that get passed to `ls` when `dired` gets a list of files. We're using:

- `l`: Use the long listing format.
- `h`: Use human-readable sizes.
- `v`: Sort numbers naturally.
- `A`: Almost all. Doesn't include "`.`" or "`..`".

```
(setq-default dired-listing-switches "-lhvA")
```

Use "j" and "k" to move around in `dired`.

```
(evil-define-key 'normal dired-mode-map (kbd "j") 'dired-next-line)
(evil-define-key 'normal dired-mode-map (kbd "k") 'dired-previous-line)
```

Kill buffers of files/directories that are deleted in `dired`.

```
(setq dired-clean-up-buffers-too t)
```

Always copy directories recursively instead of asking every time.

```
(setq dired-recursive-copies 'always)
```

Ask before recursively *deleting* a directory, though.

```
(setq dired-recursive-deletes 'top)
```

Open a file with an external program (that is, through `xdg-open`) by hitting `C-c C-o`.

```
(defun dired-xdg-open ()
  "In dired, open the file named on this line."
  (interactive)
  (let* ((file (dired-get-filename nil t)))
    (call-process "xdg-open" nil 0 nil file)))

(define-key dired-mode-map (kbd "C-c C-o") 'dired-xdg-open)
```

## 15 Projectile

```
(use-package projectile
  :ensure t
  :config
  (projectile-global-mode))
```

## 16 Hydra

```
(use-package hydra
  :ensure t)
```

## 17 Engine-mode

Enable engine-mode and define a few useful engines.

```
(use-package engine-mode
 :ensure t
 :config
 (engine/set-keymap-prefix (kbd "C-s"))
 (progn
    (defengine priberam
    "https://www.priberam.pt/dlpo/%s"
```

```
   :keybinding "p")

(engine-mode t)))
```