ALMOST EVERYTHING WITH EMACS

Frédéric GALLIANO

Université Paris-Saclay, Université Paris Cité, CEA, CNRS, AIM, 91191, Gif-sur-Yvette, France

February 23, 2024

Contents

1 GENERAL SETTINGS

1.1 Configuration file

1.1.1 Tangling the org setting file

The file `~/.emacs.d/init.el` is the first initialization file read when Emacs starts. In my case it looks like that:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;*
;;*                       EMACS INITIALIZATION SCRIPT
;;*
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;; Minimal settings
(require 'package)
;(package-initialize)

;; All the other settings are in the settings file
(require 'org)
(org-babel-load-file (expand-file-name "~/ownCloud/Settings/Emacs/settings.org"))
(defvar my-init-el-start-time (current-time) "Time when init.el was started")
(custom-set-variables
 ;; custom-set-variables was added by Custom.
 ;; If you edit it by hand, you could mess it up, so be careful.
 ;; Your init file should contain only one such instance.
 ;; If there is more than one, they won't work right.
 '(org-babel-load-languages
   '((emacs-lisp . t)
```

```
     (shell . t)
     (awk . t)
     (python . t)
     (fortran . t)
     (latex . t)
     (lilypond . t)
     (lua . t)))
 '(org-emphasis-alist
   '(("*" bold)
     ("/" italic)
     ("_" underline)
     ("=" org-verbatim verbatim)
     ("+"
      (:strike-through t))
     ("~"
      (:overline t)
      verbatim)))
 '(package-selected-packages
   '(gnu-elpa-keyring-update php-mode magit use-package symon rainbow-mode rainbow-delimite
 '(template-use-package t nil (template)))
(custom-set-faces
 ;; custom-set-faces was added by Custom.
 ;; If you edit it by hand, you could mess it up, so be careful.
 ;; Your init file should contain only one such instance.
 ;; If there is more than one, they won't work right.
 '(mu4e-context-face ((t (:foreground "dark green" :weight bold :background "grey"))))
 '(org-ellipsis ((t (:foreground "magenta" :underline t :overline t :weight bold))))
 '(rainbow-delimiters-depth-1-face ((t (:foreground "violet"))))
 '(rainbow-delimiters-depth-2-face ((t (:foreground "cyan"))))
 '(rainbow-delimiters-depth-3-face ((t (:foreground "green"))))
 '(rainbow-delimiters-depth-4-face ((t (:foreground "yellow"))))
 '(rainbow-delimiters-depth-5-face ((t (:foreground "orange"))))
 '(rainbow-delimiters-depth-6-face ((t (:foreground "red"))))
 '(rainbow-delimiters-depth-7-face ((t (:foreground "pink"))))
 '(rainbow-delimiters-depth-8-face ((t (:foreground "light blue"))))
 '(rainbow-delimiters-depth-9-face ((t (:foreground "light green"))))
 '(rainbow-delimiters-unmatched-face ((t (:background "gray")))))
```

This file tells Emacs to read the present file (`~/.ownCloud/Settings/Emacs/settings.org`) where all the settings are written. When Emacs reads `init.el`, the file `~/ownCloud/Settings/Emacs/settings.el` is generated from the present file, extracting the code in the `emacs-lisp` code blocks (this is called *tangling*).

1.1.2 Reload .emacs init file

```
(defun reload-init-file ()
  (interactive)
  (load-file user-init-file))
(global-set-key (kbd "C-c e") 'reload-init-file)
```

1.2 Package Management

1.2.1 Package installation with MELPA et al.

`M-x list-packages` to see all available, installed as well as built-in packages. To install a package, simply hit return when the cursor is on that package and select Install.

1.2.2 Automatically update the list of packages

```
(when (not package-archive-contents)
      (package-refresh-contents))
(require 'package)
(add-to-list 'package-archives
```

```
             '("melpa-stable" . "https://stable.melpa.org/packages/") t)
(add-to-list 'package-archives
             '("melpa" . "http://melpa.org/packages/") t)
(package-initialize)
(setq package-check-signature nil)
```

1.2.3 Location of packages installed by hand

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

1.3 File Management

1.3.1 Automatic refreshment

When a file is modified by someone else or because it is synced by `owncloud`, it is automatically refreshed.

```
(global-auto-revert-mode 1)
```

1.3.2 Opening recent files

1. Keep a list of recently opened files

   ```
   (require 'recentf)
   (recentf-mode 1)
   ```

2. Remember cursor position

   ```
   (if (version< emacs-version "25.0")
       (progn
         (require 'saveplace)
         (setq-default save-place t))
     (save-place-mode 1))
   ```

1.3.3 Dired

```
(use-package dired
  :defer t
  :bind (:map dired-mode-map
              ("C-<f1>" . show-dired-help)
              ("<f7>" . hydra-files/body)
              ("<f2>" . dired-omit-switch)
              ("<f3>" . dired-hide-details-mode)
              ("<f4>" . locate)
              ("<f5>" . find-alternate-file)
              ("<f6>" . dired-equal)
              ("<f10>" . term-here)
              ("<f9>"  . magit-status)
              ("<tab>" . dired-subtree-insert)
              ("<backtab>" . dired-subtree-remove)
              ("<backspace>" . dired-up-directory)
              ("C-c C-a" . gnus-dired-attach)
              )
  :init
    ;; dired layout
    (defun dired-layout ()
    (interactive)
    (delete-other-windows)
    (dired launch-directory)
    )
  :config
    (require 'dired-subtree)
    (require 'dired-rainbow)
    (require 'dired-x)
    ;; Auto refresh dired when file changes
    (add-hook 'dired-mode-hook 'auto-revert-mode)
    ;; If nil, don't ask whether to kill buffers visiting deleted files
```

```
    (setq dired-clean-confirm-killing-deleted-buffers nil)
    ;; Get readable file sizes, dirs first
    (setq dired-listing-switches "-aBhl  --group-directories-first")
)
```

1.4 Custom Back-Up

1.4.1 Avoid annoying ~ files

Instead store all successive back-ups in a dedicated directory (poor-man's © *Time Machine*...).

```
(setq backup-directory-alist `(("." . "~/.saves")))
(setq backup-by-copying t)
(setq delete-old-versions t
      kept-new-versions 6
      kept-old-versions 2
      version-control t)
```

1.5 Templates

Set different templates. I have prepared these templates in a directory (`python` script, `python` module, `fortran` program, `fortran` module, LaTeX report, LaTeX notes, LaTeX official letter, LaTeX personal letter, `emacs` ORG file, `shell` script, *etc.*). They are automatically opened, based on their name and extension, when I create a new file.

```
(custom-set-variables '(template-use-package t nil (template)))
(require 'template)
(template-initialize)
```

1.6 Bookmark file

File containing all the bookmarks to my most frequently visited files. Access with C-x r b.

```
(setq bookmark-default-file "~/ownCloud/Settings/Emacs/bookmarks.el")
```

1.7 On-line help

Pressing F1 displays this help. Conclusion: the only shortcut you need to remember is F1.

```
(defun show-help ()
  (interactive)
  (display-message-or-buffer "
FILES & WINDOWS
---------------
  Files
  - - -
[C-x C-f]: open new file
[C-x C-s]: save current buffer to file
[C-x C-w]: save current buffer to a new file
[C-x C-b]: open buffer containing the session file names
[C-x left]: open the previous file of the session
[C-x right]: open the next file of the session
[C-x C-c]: quit
[C-c e]: reload init file

  Windows
  - - - -
[C-x 5 2]: open a new frame
[C-x 5 0]: close the current frame
[C-x 2]: split window vertically
[C-x 3]: split window horizontally
[C-x 1]: close the other windows of the frame
[C-x 0]: close current window
[C-x o]: move cursor to the other window of the frame
[M-p f]: enable follow-mode
```

```
[M-F7]: move frame using arrows
[M-F8]: resize frame using arrows
[M-F10]: maximize frame
[C-x }]: move the vertical separation between windows to the right
[C-x {]: move the vertical separation between windows to the left
[C-u 1 C-x 3]: split window with proportions adapted to LaTeX
[F8]: winner do
[C-F8]: winner undo

  Layouts
  - - - -
[C-c l w]: work org task layout
[C-c l h]: home org task layout
[C-c l l]: LaTeX layout
[C-c l m]: Email layout
[C-c l b]: Big layout

  Bookmarks
  - - - - -
[C-x r b <bm>]: open file from bookmark
[C-x r m <bm>]: save current file to bookmark
[C-x r l]: list bookmarks

  Dired
  - - -
[C-x d]: open directory
[d]: mark for deletion
[x]: execute the actions marked for

EDITING
-------
  General
  - - - -
[C-/]: undo
[C-g]: abort command
[M-;]: comment/uncomment selection
[C-x C-u]: convert selection to upper case
[C-x C-l]: convert selection to lower case
[C-u <n> <c>]: insert <n> times the character <c>
[C-x TAB]: indent selected text using left and right arrows
[C-<n> C-x TAB]: indent selected text by <n> characters

  Spelling & Counting
  - - - - - - - - - -
[M-=\]: print word, character and line count of the whole buffer
[C-F2]: correct word spelling
[C-c d f]: switch spelling dictionnary to french
[C-c d e]: switch spelling dictionnary to english

  Moving the cursor
  - - - - - - - - -
[M-<]: go to beginning of the file
[M->]: go to end of the file
[C-v]: go to next page
[M-v]: go to previous page
[C-right]: go to next word
[C-left]: go to previous word
[C-down]: go to next paragraph
[C-up]: go to previous paragraph
[C-a]: go to beginning of the line
[C-e]: go to end of the line
[C-l]: center the window on the cursor position
```

```
  Search & replace
  - - - - - - - - -
[C-s]: search word forward
[C-r]: search word backward
[M-g g <n>]: go to line number <n>
[M-%]: query replace --> 'y', 'n' or '!' for all
[M-s o]: search all lines containing a given string
[M-s h p]: highlight all occurences of a given word
[M-s h r]: highlight all occurences of a given regexp
[M-s h l]: highlight all lines containing a given regexp
[M-s h u]: turn off the highlighting

  Copy & Paste
  - - - - - - -
[C-x h]: select the whole window
[M-p l]: select the current line
[M-p b]: select the current block
[C-SPC]: set the marker
[M-w]: copy marked text
[C-w]: copy and delete marked text
[C-k]: cut text from position to the end of the line
[C-y]: paste copied text
[M-y]: after C-y, it replaces the pasted text by the previous copy
[M-DEL]: copy and cut the word on the left of the cursor
[M-d]: copy and cut the word on the right of the cursor
[C-x r t]: cut a text rectangle between the marker and the cursor
[C-x r y]: paste the rectangle
[C-x r k]: cut a rectangle
[C-x r s <id>]: copy a sequence under <id>
[C-x r i <id>]: paste the <id> sequence

  Parentheses
  - - - - - -
[C-M-SPC]: if cursor is on a left parenthesis => select the text
 up to the closing parenthesis
[C-M-(]: create a pair of () around the selected text
[C-M-[]: create a pair of [] around the selected text
[C-M-{]: create a pair of {} around the selected text
[C-M-']: create a pair of '' around the selected text
[M-p s]: select the text between two delimiters

MAJOR MODES
-----------
  Org Mode
  - - - -
[TAB]: show/hide current setion
[S-TAB]: show/hide all sections
[C-c C-n]: go to next section
[C-c C-p]: go to previous section
[C-c C-f]: go to next section of the same level
[C-c C-b]: go to previous section of the same level
[M-up]: move current section/item above
[M-down]: move current section/item below
[M-right]: decrease the level of the current section/item
[M-left]: increase the levle of the current section/item
[M-RET]: create the next section/item
[S-left/right]: cycle through bullet styles
[C-t t]: create a TODO
[C-c C-s]: open calendar to schedule a task
[C-c C-d]: open calendar to schedule a deadline
[C-c a a]: open agenda
```

```
[C-c C-e t u]: export ORG to UTF-8
[C-c C-e h h]: export ORG to HTML
[C-c C-e l p]: export ORG to PDF
[C-c C-e m m]: export ORG to MD

  LaTeX
  - - -
[C-c =]: open a top buffer containing the section list
[C-c C-c l TAB]: compile
[C-c C-c b TAB]: compile the bibliography
[C-c C-c v TAB]: show PDF
[C-c C-g]: shows position in PDF corresponding to cursor position

  BibTeX
  - - -
[C-x b]: open helm-bibtex

  PDF-tools
  - - - - -
[C-c C-a t]: add text annotation to a selection
[C-c C-a h]: highlight a selection

MAILS
-----
[C-x m]: open mu4e
[j]: jump to a mail directory
[s]: search
[C]: compose
[U]: check for emails
[q]: quit

  Reading
  - - - -
[d]: mark for deletion
[D]: delete
[R]: reply
[F]: forward
[E]: edit draft
[aV]: open in web browser (in email view mode)
[ax]: search for other emails from the same sender (in email view mode)
[a<n>o]: open attachment number <n> (with xdg-open settings)
[g <n>]: follow link <n>
[af]: copy the name of the local email file
[ar]: remove all the attachment (without saving them)

  Composition
  - - - - - -
[<alias> SPC]: use an email address alias (individual or list)
[C-c C-f C-c]: add CC
[C-c C-f C-b]: add BCC
[C-c C-k]: do not send message (abort)
[C-c C-w]: add signature (if not in org-msg-mode)
[C-c C-a]: attach a file
[C-c RET C-a]: attach files marked with [m] in dired (C-x d)
[C-c C-e]: display the HTML rendering of the email in a browser (org-msg)
[C-c C-c]: send and exit

DOCUMENTATION
-------------
[F1]: this help
[C-h k <key>]: help on keybinding
[C-h f <fun>]: help on function
```

```
[C-h v <var>]: help on variable
[C-h o <sym>]: help on symbol (e.g. face)
[C-h w <key>]: help on keyword

MISCELLANEOUS
-------------

  Date
  - -
[M-p d]: insert the date at the current location
"))
(global-set-key (read-kbd-macro "<f1>") 'show-help)
```

1.8 Guide popup for shortcuts

1.8.1 Interactive guide

```
(use-package guide-key
  :init
   (setq guide-key/guide-key-sequence t) ;;'("C-x r" "C-x 4")
   (guide-key-mode 1)  ; Enable guide-key-mode
   (setq guide-key/idle-delay 2)
)
```

1.8.2 Laziness

Change "yes or no" to "y or n":

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

Don't ask for confirmation for "dangerous" commands:

```
(put 'erase-buffer 'disabled nil)
(put 'narrow-to-page 'disabled nil)
(put 'upcase-region 'disabled nil)
(put 'narrow-to-region 'disabled nil)
(put 'downcase-region 'disabled nil)
(put 'scroll-left 'disabled nil)
(put 'scroll-right 'disabled nil)
(put 'set-goal-column 'disabled nil)
```

Large file warning:

```
(setq large-file-warning-threshold (* 15 1024 1024))
```

---

2 APPEARANCE

2.1 General

Dark background color theme and main font:

```
(setq column-number-mode t)
(load-theme 'manoj-dark)
(set-frame-font "Monospace 10" nil t)
```

2.2 Start-up

```
(setq inhibit-startup-screen t)
(setq inhibit-startup-message t)
(setq inhibit-startup-echo-area-message t)
(setq initial-scratch-message nil)
(setq org-set-startup-visibility "showall")
(defun display-startup-echo-area-message () (message ""))
```

2.3 Window size

I come from an era where the recommended window size was 80 character wide, because it was the maximum line width that postscript printers could handle. Besides, it is more difficult to read wide text files.

```
(add-to-list 'default-frame-alist '(height . 55))
(add-to-list 'default-frame-alist '(width . 81))
```

2.4 Hide toolbars

We are being serious about not using GUI.

```
(menu-bar-mode -1)
(tool-bar-mode -1)
```

2.5 Indicate empty lines

```
(setq-default indicate-empty-lines t)
(when (not indicate-empty-lines) (toggle-indicate-empty-lines))
```

2.6 Window splitting

2.6.1 Custom sizes

To split windows unevenly C-u <num> C-x 3 will split windows horizontally, with a larger window on the right side.

```
(defvar my-ratio-dict
  '((1 . 1.47)           ;; adapted to LaTeX on sappcw100
    (2 . 1.61803398875)) ;; golden number ratio
  "The ratio dictionary.")

(defun my-split-window-horizontally (&optional ratio)
  "Split window horizontally and resize the new window.
   'C-u number M-x my-split-window-horizontally' uses pre-defined
   ratio from `my-ratio-dict'.
   Always focus on bigger window."
  (interactive "P")
  (let* (ratio-val)
    (cond
     (ratio
      (setq ratio-val (cdr (assoc ratio my-ratio-dict)))
      (split-window-horizontally (floor (/ (window-body-width)
                                           (1+ ratio-val)))))
     (t
      (split-window-horizontally)))
    (set-window-buffer (next-window) (current-buffer))
    (if (or (not ratio-val)
            (>= ratio-val 1))
        (windmove-right))))

(defun my-split-window-vertically (&optional ratio)
  "Split window vertically and resize the new window.
   'C-u number M-x my-split-window-vertically' uses pre-defined
   ratio from `my-ratio-dict'.
   Always focus on bigger window."
  (interactive "P")
  (let* (ratio-val)
    (cond
     (ratio
      (setq ratio-val (cdr (assoc ratio my-ratio-dict)))
      (split-window-vertically (floor (/ (window-body-height)
                                         (1+ ratio-val)))))
     (t
```

```
      (split-window-vertically)))
    ;; open another window with current-buffer
    (set-window-buffer (next-window) (current-buffer))
    ;; move focus if new window bigger than current one
    (if (or (not ratio-val)
            (>= ratio-val 1))
        (windmove-down))))

(global-set-key (kbd "C-x 2") 'my-split-window-vertically)
(global-set-key (kbd "C-x 3") 'my-split-window-horizontally)
```

2.6.2 Follow mode

This allows to split the same file in two different windows of the same frame, having the bottom of the left window corresponding to the top of the right window.

```
(global-set-key (kbd "M-p f") 'follow-mode)
```

2.6.3 Winner mode

Winner undo/redo to restore configuration:

```
(when (fboundp 'winner-mode)
      (winner-mode 1))
(global-set-key [f8] 'winner-undo)
(global-set-key [C-f8] 'winner-redo)
```

2.7 Visual-line-mode for text files

Text wrapping when we are not coding.

```
(add-hook 'text-mode-hook #'turn-on-visual-line-mode)
```

2.8 Turn-off annoying auto-indentation

```
(when (fboundp 'electric-indent-mode) (electric-indent-mode -1))
```

2.9 Rainbow mode

```
(use-package rainbow-mode
  :diminish rainbow-mode
  :config
  (add-hook 'emacs-lisp-mode-hook 'rainbow-mode)
  (add-hook 'css-mode-hook 'rainbow-mode)
  (add-hook 'html-mode-hook 'rainbow-mode)
  (add-hook 'js2-mode-hook 'rainbow-mode))
```

2.10 Turn-off extra annoying scratch buffer at start

2.10.1 Removes scratch from buffer after the mode has been set

```
(defun remove-scratch-buffer ()
  (if (get-buffer "*scratch*")
      (kill-buffer "*scratch*")))
(add-hook 'after-change-major-mode-hook 'remove-scratch-buffer)
```

2.10.2 Removes Completions from buffer after you've opened a file

```
(add-hook 'minibuffer-exit-hook
      #'(lambda ()
          (let ((buffer "*Completions*"))
            (and (get-buffer buffer)
                 (kill-buffer buffer)))))
```

2.10.3 Don't show Buffer list when opening multiple files at the same time

```
(setq inhibit-startup-buffer-menu t)
```

2.10.4 Show only one active window when opening multiple files at the same time

```
(add-hook 'window-setup-hook 'delete-other-windows)
```

2.10.5 Save initial launch directory

```
(setq launch-directory default-directory)
```

2.10.6 Turn on visible bell

```
(setq visible-bell t)
(setq ring-bell-function 'ignore)
```

2.11 Bars

2.11.1 Top bar

This is what is printed on top of the `emacs` window.

```
(setq frame-title-format '(buffer-file-name "%F: %f" ("%F: %b")))
```

2.11.2 Bottom bar

System monitor below bottom bar:

```
;; (use-package symon
;;   :defer 3
;;   :ensure t
;;   :config
;;   (setq symon-sparkline-type 'bounded)
;;   (define-symon-monitor symon-current-date-time-monitor
;;     :interval 5
;;     :display (propertize
;;               (format-time-string "%a %d %b %Y [%k:%M] |")
;;               'face 'font-lock-type-face))
;;   (setq symon-monitors
;;       '(symon-current-date-time-monitor
;;         symon-linux-memory-monitor
;;         symon-linux-cpu-monitor))
;;   (symon-mode)
;;   )
```

2.12 Layouts

2.12.1 ORG agenda

Automatically sets the layout for managing tasks in the org agenda:

```
(defun coding-layout ()
  (interactive)
  (delete-other-windows)
  (set-frame-size (selected-frame) 81 55 nil)
  )
(defun big-layout ()
  (interactive)
  (delete-other-windows)
  (set-frame-size (selected-frame) 166 55 nil)
  )
(defun org-task-layout-work ()
  (interactive)
  (delete-other-windows)
  (set-frame-size (selected-frame) 166 55 nil)
```

```
  (find-file "~/ownCloud/Organisation/boulot.org")
  (split-window-horizontally)
  (other-window 1)
  (org-agenda "a" "a")
  (other-window 1)
  )
(defun org-task-layout-home ()
  (interactive)
  (delete-other-windows)
  (set-frame-size (selected-frame) 166 55 nil)
  (find-file "~/ownCloud/Organisation/guitare.org")
  (split-window-horizontally)
  (other-window 1)
  (org-agenda "a" "a")
  (other-window 1)
  )
```

2.12.2 LaTeX

```
(defun latex-layout ()
  (interactive)
  (delete-other-windows)
  (set-frame-size (selected-frame) 166 55 nil)
  (my-split-window-horizontally 1)
  (other-window 1)
  )
```

2.12.3 Emails

```
(defun email-layout ()
  (interactive)
  (delete-other-windows)
  (set-frame-size (selected-frame) 166 55 nil)
  (mu4e)
  )
```

2.12.4 Shortcuts

```
(global-set-key (kbd "\C-cl")
                (defhydra hydra-layouts
                  (:exit t)
                  "Layouts"
                  ("c" coding-layout "Plain coding")
                  ("w" org-task-layout-work "Work tasks")
                  ("h" org-task-layout-home "Home tasks")
                  ("l" latex-layout "LaTeX")
                  ("m" email-layout "Emails")
                  ("b" big-layout "Big")
                  ))
```

---

3 TEXT EDITION

3.1 Text insertion

3.1.1 Overwrite text in active region

```
(delete-selection-mode 1)
```

3.1.2 Emacs clipboard available to other apps

```
(setq x-select-enable-clipboard t)
```

3.2 Changing the font cases

This commands turns on the C-x C-u (C-x C-l) command to convert a selection to upper (lower) case.

```
(put 'upcase-region 'disabled nil)
(put 'downcase-region 'disabled nil)
```

3.3 Transparently open compressed files

```
(auto-compression-mode t)
```

3.4 Parentheses

3.4.1 Highlighting

Show matching parentheses:

```
(setq show-paren-delay 0)
(show-paren-mode 1)
(setq show-paren-style 'expression) ;; or 'parenthesis (default)
(setq show-paren-highlight-openparen t
      show-paren-when-point-inside-paren t)
;; To change the color/face:
(require 'paren)
(set-face-background 'show-paren-match "#505050")
(set-face-attribute 'show-paren-match nil :weight 'extra-bold)
```

One color per set of parenthesis

```
(use-package rainbow-delimiters
  :ensure t
  :config
  (add-hook 'prog-mode-hook 'rainbow-delimiters-mode))
(custom-set-faces
 '(rainbow-delimiters-depth-1-face ((t (:foreground "violet"))))
 '(rainbow-delimiters-depth-2-face ((t (:foreground "cyan"))))
 '(rainbow-delimiters-depth-3-face ((t (:foreground "green"))))

 '(rainbow-delimiters-depth-4-face ((t (:foreground "yellow"))))
 '(rainbow-delimiters-depth-5-face ((t (:foreground "orange"))))
 '(rainbow-delimiters-depth-6-face ((t (:foreground "red"))))

 '(rainbow-delimiters-depth-7-face ((t (:foreground "pink"))))
 '(rainbow-delimiters-depth-8-face ((t (:foreground "light blue"))))
 '(rainbow-delimiters-depth-9-face ((t (:foreground "light green"))))
 '(rainbow-delimiters-unmatched-face ((t (:background "gray"))))
 )
```

3.4.2 Electric bracket pairs

```
(defun my-insert-bracket-pair (@left-bracket @right-bracket &optional @wrap-method)
  "Insert brackets around selection, word, at point, and maybe move cursor in between.

  *left-bracket and *right-bracket are strings. *wrap-method must be either 'line or 'block

  • if there's a region, add brackets around region.
  • If *wrap-method is 'line, wrap around line.
  • If *wrap-method is 'block, wrap around block.
  • if cursor is at beginning of line and its not empty line and contain at least 1 space, w
  • If cursor is at end of a word or buffer, one of the following will happen:
   xyz → xyz()
   xyz → (xyz)        if in one of the lisp modes.
  • wrap brackets around word if any. e.g. xyz → (xyz). Or just ()

  URL `http://ergoemacs.org/emacs/elisp_insert_brackets_by_pair.html'
```

```
  Version 2017-01-17"
  (if (use-region-p)
      (progn ; there's active region
        (let (
              ($p1 (region-beginning))
              ($p2 (region-end)))
          (goto-char $p2)
          (insert @right-bracket)
          (goto-char $p1)
          (insert @left-bracket)
          (goto-char (+ $p2 2))))
    (progn ; no text selection
      (let ($p1 $p2)
        (cond
         ((eq @wrap-method 'line)
          (setq $p1 (line-beginning-position) $p2 (line-end-position))
          (goto-char $p2)
          (insert @right-bracket)
          (goto-char $p1)
          (insert @left-bracket)
          (goto-char (+ $p2 (length @left-bracket))))
         ((eq @wrap-method 'block)
          (save-excursion
            (progn
              (if (re-search-backward "\n[ \t]*\n" nil 'move)
                  (progn (re-search-forward "\n[ \t]*\n")
                         (setq $p1 (point)))
                (setq $p1 (point)))
              (if (re-search-forward "\n[ \t]*\n" nil 'move)
                  (progn (re-search-backward "\n[ \t]*\n")
                         (setq $p2 (point)))
                (setq $p2 (point)))))
            (goto-char $p2)
            (insert @right-bracket)
            (goto-char $p1)
            (insert @left-bracket)
            (goto-char (+ $p2 (length @left-bracket)))))
         ( ;  do line. line must contain space
          (and
           (eq (point) (line-beginning-position))
           ;; (string-match " " (buffer-substring-no-properties (line-beginning-position) (
           (not (eq (line-beginning-position) (line-end-position))))
          (insert @left-bracket )
          (end-of-line)
          (insert  @right-bracket))
         ((and
           (or ; cursor is at end of word or buffer. i.e. xyz
            (looking-at "[^-_[:alnum:]]")
            (eq (point) (point-max)))
           (not (or
                 (string-equal major-mode "my-elisp-mode")
                 (string-equal major-mode "emacs-lisp-mode")
                 (string-equal major-mode "lisp-mode")
                 (string-equal major-mode "lisp-interaction-mode")
                 (string-equal major-mode "common-lisp-mode")
                 (string-equal major-mode "clojure-mode")
                 (string-equal major-mode "my-clojure-mode")
                 (string-equal major-mode "scheme-mode"))))
          (progn
            (setq $p1 (point) $p2 (point))
            (insert @left-bracket @right-bracket)
            (search-backward @right-bracket )))
```

```
          (t (progn
               ;; wrap around "word". basically, want all alphanumeric, plus hyphen and unde
               ;;
               (skip-chars-backward "-_[:alnum:]")
               (setq $p1 (point))
               (skip-chars-forward "-_[:alnum:]")
               (setq $p2 (point))
               (goto-char $p2)
               (insert @right-bracket)
               (goto-char $p1)
               (insert @left-bracket)
               (goto-char (+ $p2 (length @left-bracket)))))))))))
(defun my-insert-paren ()
  (interactive)
  (my-insert-bracket-pair "(" ")") )
(defun my-insert-bracket ()
  (interactive)
  (my-insert-bracket-pair "[" "]") )
(defun my-insert-brace ()
  (interactive)
  (my-insert-bracket-pair "{" "}") )
(defun my-insert-ascii-double-quote () (interactive) (my-insert-bracket-pair "\"" "\"") )
(defun my-insert-ascii-single-quote () (interactive) (my-insert-bracket-pair "'" "'") )
(global-set-key (kbd "C-M-{") 'my-insert-brace) ; {}
(global-set-key (kbd "C-M-(") 'my-insert-paren) ; ()
(global-set-key (kbd "C-M-[") 'my-insert-bracket) ; []
(global-set-key (kbd "C-M-'") 'my-insert-ascii-single-quote) ; ''
(global-set-key (kbd "C-M-\"") 'my-insert-ascii-double-quote) ; ""
```

3.5 Power selection

3.5.1 Between delimiters

```
(defun my-select-text-in-quote ()
  "Select text between the nearest left and right delimiters.
   Delimiters here includes the following chars: \"<>(){}[]“”‘’‹›«»
   This command select between any bracket chars, not the inner text of a bracket. For examp

   (a(b)c)

   the selected char is “c”, not “a(b)c”.

  URL `http://ergoemacs.org/emacs/modernization_mark-word.html'
  Version 2016-12-18"
  (interactive)
  (let (
        ($skipChars
         (if (boundp 'my-brackets)
             (concat "^\"" my-brackets)
           "^\"<>(){}[]“”‘’‹›«»"))
        $pos
        )
    (skip-chars-backward $skipChars)
    (setq $pos (point))
    (skip-chars-forward $skipChars)
    (set-mark $pos)))
(global-set-key (kbd "M-p s") 'my-select-text-in-quote)
```

3.5.2 Current line

```
(defun my-select-line ()
  "Select current line. If region is active, extend selection downward by line.
  URL `http://ergoemacs.org/emacs/modernization_mark-word.html'
```

```
  Version 2017-11-01"
  (interactive)
  (if (region-active-p)
      (progn
        (forward-line 1)
        (end-of-line))
    (progn
      (end-of-line)
      (set-mark (line-beginning-position)))))
(global-set-key (kbd "M-p l") 'my-select-line)
```

3.5.3 Current text block

```
(defun my-select-block ()
  "Select the current/next block of text between blank lines.
  If region is active, extend selection downward by block.

  URL `http://ergoemacs.org/emacs/modernization_mark-word.html'
  Version 2019-12-26"
  (interactive)
  (if (region-active-p)
      (re-search-forward "\n[ \t]*\n" nil "move")
    (progn
      (skip-chars-forward " \n\t")
      (when (re-search-backward "\n[ \t]*\n" nil "move")
        (re-search-forward "\n[ \t]*\n"))
      (push-mark (point) t t)
      (re-search-forward "\n[ \t]*\n" nil "move"))))
(global-set-key (kbd "M-p b") 'my-select-block)
```

3.6 Convert paragraph to single line

```
(defun unfill-paragraph (&optional region)
  "Takes a multi-line paragraph and makes it into a single line of text."
  (interactive (progn (barf-if-buffer-read-only) '(t)))
  (let ((fill-column (point-max))
        ;; This would override `fill-column' if it's an integer.
        (emacs-lisp-docstring-fill-column t))
    (fill-paragraph nil region)))
```

3.7 Time Formatting

Inserting the date in a file (https://stackoverflow.com/questions/251908/how-can-i-insert-current-date

```
(defvar current-date-time-format "%a %b %d %Y-%m-%d (%H:%M:%S)"
  "Format of date to insert with `insert-current-date-time' func
  See help of `format-time-string' for possible replacements")
(defun insert-current-date-time ()
  "insert the current date and time into current buffer.
  Uses `current-date-time-format' for the formatting the date/time."
       (interactive)
       (insert (format-time-string current-date-time-format (current-time)))
       (insert "\n")
       )
(global-set-key (kbd "M-p d") 'insert-current-date-time)
```

3.8 Spell Checking & Word Counting

3.8.1 Spell Checking

Use hunspell (https://manuel-uberti.github.io/emacs/2016/06/06/spellchecksetup/):

```
(add-hook 'text-mode-hook 'flyspell-mode)
(add-hook 'prog-mode-hook 'flyspell-prog-mode)
```

```
(with-eval-after-load "ispell"
  (setq ispell-check-comments t)
  (setq ispell-really-hunspell t)
  (setq ispell-program-name (executable-find "hunspell")
        ispell-dictionary "en_US"))
```

Automatically detect language for Flyspell for current paragraph! Also reruns flyspell, just on the paragraph:

```
(use-package guess-language
  :ensure t
  :defer t
  :init (add-hook 'text-mode-hook #'guess-language-mode)
  :config
  (setq guess-language-langcodes '((en . ("en_US" "English"))
                                   (fr . ("fr_FR" "French")))
        guess-language-languages '(en fr)
        guess-language-min-paragraph-length 45) ;minimal length that a paragraph needs to ha
  :diminish guess-language-mode)
```

Hydra to switch dictionnary:

```
(global-set-key (kbd "\C-cd")
  (defhydra hydra-dict ()
   "dict"
   ("f" (lambda () (interactive) (ispell-change-dictionary "fr_FR") (flyspell-buffer)) "fren
   ("e" (lambda () (interactive) (ispell-change-dictionary "en_US") (flyspell-buffer)) "engl
   ("q" nil "cancel")))
(global-set-key (kbd "<C-f2>") 'ispell-word)
(global-set-key (kbd "<C-f3>") 'helm-flyspell-correct)
```

French grammar checking:

```
(with-eval-after-load 'flycheck
  (flycheck-grammalecte-setup))
```

3.8.2 Word Counting

Use count-words instead of count-words-region as it works on buffer if no region is selected:

```
(global-set-key (kbd "M-=") 'count-words)
```

3.9 Printing

Use M-x print-buffer:

```
(require 'printing)
(setq ps-lpr-command "print_preview")
```

3.10 Coding

3.10.1 Python

```
(use-package highlight-indent-guides
  :hook
  (python-mode . highlight-indent-guides-mode)
  :custom
  (highlight-indent-guides-method 'character)
  (highlight-indent-guides-responsive 'top)
  (highlight-indent-guides-delay 0)
  )
(add-hook 'prog-mode-hook 'highlight-indent-guides-mode)
```

3.10.2 Tangling

One of the coolest tricks. . .

```
(custom-set-variables
'(org-babel-load-languages
(quote
((emacs-lisp . t)
 (shell . t)
 (awk . t)
 (python . t)
 (fortran . t)
 (latex . t)
 (lilypond . t)
 (lua . t)
))))
(setq org-babel-python-command "python3"
      org-confirm-babel-evaluate nil)
```

---

4 LATEX

4.1 AucTeX

```
(require 'tex)
(require 'latex)
(setq TeX-engine "luatex")
(setq TeX-auto-save t)
(setq-default TeX-master nil)
(setq TeX-parse-self t)
(setq TeX-save-query nil)
(setq TeX-PDF-mode t)
(setq reftex-plug-into-AUCTeX t)
(setq-default TeX-auto-local "~/.auctex-auto")
```

4.1.1 RefTex

```
(autoload 'reftex-mode "reftex" "RefTeX Minor Mode" t)
(autoload 'turn-on-reftex "reftex" "RefTeX Minor Mode" nil)
(autoload 'reftex-citation "reftex-cite" "Make citation" nil)
(autoload 'reftex-index-phrase-mode "reftex-index" "Phrase Mode" t)
(add-hook 'LaTeX-mode-hook 'turn-on-reftex)
(setq reftex-plug-into-AUCTeX t)
```

4.1.2 Miscellaneous

```
(setq TeX-parse-self t
 LaTeX-section-hook
 '(LaTeX-section-heading
   LaTeX-section-title
   LaTeX-section-toc
   LaTeX-section-section
   LaTeX-section-label))
```

4.1.3 Enable synctex correlation

Just press Shift + Left click to go to the good line:

```
(setq TeX-source-correlate-mode t
 TeX-source-correlate-start-server t
 TeX-source-correlate-method 'synctex)
```

4.2 PDF viewer

4.2.1 pdf-tools

```
(with-eval-after-load 'tex
  (pdf-tools-install)
  (setq TeX-view-program-selection '((output-pdf "PDF Tools")))
  (setq TeX-view-program-list '(("pdf-tools" "TeX-pdf-tools-sync-view"))))
```

4.2.2 Update PDF buffers after successful LaTeX runs

```
;(setq TeX-show-compilation t)
(add-hook 'LaTeX-mode-hook 'TeX-PDF-mode)
(add-hook 'LaTeX-mode-hook 'TeX-source-correlate-mode)
(add-hook 'TeX-after-compilation-finished-functions
          #'TeX-revert-document-buffer)
```

4.3 BibTeX

4.3.1 Helm-bibtex

```
(use-package helm-bibtex
   :defer t
   :bind (:map helm-map ("C-<f1>" . show-helm-bibtex-help))
   :init
   (setq bibtex-completion-bibliography
    '("~/ownCloud/TeXstyle/references.bib"))
   (setq bibtex-completion-library-path "~/ownCloud/References/Copies")
   (setq bibtex-completion-notes-path "~/ownCloud/References/Notes")
  )
```

4.3.2 Searching default fields

Fields used for searching are: author, title, year, BibTeX key, entry type (article, inproceedings, *etc.*)

```
(setq bibtex-completion-additional-search-fields '(firstauthor tags))
(setq bibtex-completion-display-formats
     '((t . "${author:30}|${year:4}|${title:*}|${=has-pdf=:1}| ${=has-note=:1}|${=type=:7}"
  (advice-add 'bibtex-completion-candidates
              :filter-return 'reverse)

(defun helm-bibtex-my-publications (&optional arg)
    "Search BibTeX entries authored by "Jane Doe".
    With a prefix ARG, the cache is invalidated and the bibliography reread."
    (interactive "P")
    (helm-bibtex arg nil "Xxx"))
;; Bind
(global-set-key (kbd "C-x b") 'helm-bibtex)
```

---

5 ORG MODE

5.1 Org mode settings

```
(require 'org)
(setq org-log-done t)
(setq org-list-description-max-indent 5)
(setq org-adapt-indentation nil)
(setq org-startup-folded nil)
(setq org-startup-indented t)
(setq org-return-follows-link t)
(setq org-hide-emphasis-markers t)
(setq org-pretty-entities t)
```

```
(setq org-list-allow-alphabetical t)
(global-set-key "\C-cs" 'org-store-link)
(global-set-key "\C-ca" 'org-agenda)
(global-set-key "\C-cc" 'org-capture)
(global-set-key "\C-cb" 'org-switchb)
```

Customize the appearance of ellipsis (when sections are folded):

```
(setq org-ellipsis "  ")
(custom-set-faces '(org-ellipsis ((t (:foreground "magenta" :underline t :overline t :weight
```

5.2 Rich text format

5.2.1 Font weights

```
(custom-set-variables
 '(org-emphasis-alist
   (quote
    (("*" bold)
     ("/" italic)
     ("_" underline)
     ("=" org-verbatim verbatim)
     ("+" (:strike-through t))
     ("~" (:overline t) verbatim)))))
```

5.2.2 Bullets

```
(require 'org-bullets)
(add-hook 'org-mode-hook (lambda () (org-bullets-mode 1)))
```

5.3 Date format

```
(setq-default org-display-custom-times t)
(setq org-time-stamp-custom-formats '("<%a %d %b %Y>" . "<%a %d %b %Y, %H:%M>"))
```

5.4 Inline image display

```
;; (setq org-display-inline-images t) (setq org-redisplay-inline-images t) (setq org-startu
```

5.5 Inheritance of TODO subsections

In order for it to work, the top section needs to have a [/] or [%]. If subsections contain TODO, when they are all in DONE, the top section automatically toggle to DONE. Source: https://christiantietze.de/posts/2021/02/emacs-org-todo-doing-done-checkbox-cycling/

```
(defun org-todo-if-needed (state)
  "Change header state to STATE unless the current item is in STATE already."
  (unless (string-equal (org-get-todo-state) state)
    (org-todo state)))

(defun org-summary-todo (n-done n-not-done)
    "Switch header state to DONE when all subentries are DONE, to TODO when none are DONE, a
    (let (org-log-done org-log-states)
      (org-todo-if-needed (cond ((= n-done 0)
                                 "À-FAIRE")
                                ((= n-not-done 0)
                                 "FAIT")
                                (t
                                 "EN-COURS")))))
  (add-hook 'org-after-todo-statistics-hook 'org-summary-todo)

(defun org-summary-checkbox-cookie ()
  "Switch header state to DONE when all checkboxes are ticked, to TODO when none are ticked
  (let (beg end)
```

```
    (unless (not (org-get-todo-state))
      (save-excursion
        (org-back-to-heading t)
        (setq beg (point))
        (end-of-line)
        (setq end (point))
        (goto-char beg)
        ;; Regex group 1: %-based cookie
        ;; Regex group 2 and 3: x/y cookie
        (if (re-search-forward "\\[\\([0-9]*%\\)\\]\\|\\[\\([0-9]*\\)/\\([0-9]*\\)\\]"
                               end t)
            (if (match-end 1)
                ;; [xx%] cookie support
                (cond ((equal (match-string 1) "100%")
                       (org-todo-if-needed "FAIT"))
                      ((equal (match-string 1) "0%")
                       (org-todo-if-needed "À-FAIRE"))
                      (t
                       (org-todo-if-needed "EN-COURS")))
              ;; [x/y] cookie support
              (if (> (match-end 2) (match-beginning 2)) ; = if not empty
                  (cond ((equal (match-string 2) (match-string 3))
                         (org-todo-if-needed "FAIT"))
                        ((or (equal (string-trim (match-string 2)) "")
                             (equal (match-string 2) "0"))
                         (org-todo-if-needed "À-FAIRE"))
                        (t
                         (org-todo-if-needed "EN-COURS")))
                (org-todo-if-needed "EN-COURS")))))))
(add-hook 'org-checkbox-statistics-hook #'org-summary-checkbox-cookie)
```

Make sure that once all the list have been chekec the top section toggles to DONE.

Dependencies between TODOs and checkboxes. Credits: https://orgmode.org/worg/org-hacks.html.

```
(with-eval-after-load 'org-list
  (add-hook 'org-checkbox-statistics-hook #'checkbox-list-complete))

(defun checkbox-list-complete ()
  (save-excursion
    (when (ignore-errors (org-back-to-heading t))
      (let ((beg (point))(current-state (org-get-todo-state)) end)
        (end-of-line)
        (setq end (point))
        (goto-char beg)
        (if (re-search-forward "\\[\\([0-9]*%\\)\\]\\|\\[\\([0-9]*\\)/\\([0-9]*\\)\\]" end t
            (if (match-end 1)
                (if (equal (match-string 1) "100%")
                    ;; all done - do the state change
                    (org-todo 'done)
                  (when (and current-state
                             (string= current-state "FAIT"))
                    (org-todo 'todo)))
              (if (and (> (match-end 2) (match-beginning 2))
                       (equal (match-string 2) (match-string 3)))
                  (org-todo 'done)
                (when (and current-state
                           (string= current-state "FAIT"))
                  (org-todo 'todo)))))))))
```

5.6 Exporting

Default settings for exporting ORG files:

```
(setq user-full-name "Frédéric GALLIANO"
      user-mail-address "xxxxxxxx.xxxxxxxx@cea.fr"
      org-export-default-language "en"
      org-export-with-email t)
```

5.6.1 Customize the LaTeX exportation

1. Preamble

```
(require 'ox-latex)
(unless (boundp 'org-latex-classes)
  (setq org-latex-classes nil))
(add-to-list 'org-latex-classes
             '("notes_english"
               "\\documentclass[10pt,a4paper]{article}
                [NO-DEFAULT-PACKAGES]
                [NO-EXTRA]
                \\usepackage[english]{babel}
                \\usepackage[cachedir=/tmp/mint]{minted}
                \\input{style_org}"
               ("\\section{%s}" . "\\section*{%s}")
               ("\\subsection{%s}" . "\\subsection*{%s}")
               ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
               ("\\paragraph{%s}" . "\\paragraph*{%s}")
               ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
(add-to-list 'org-latex-classes
             '("notes_french"
               "\\documentclass[10pt,a4paper]{article}
                [NO-DEFAULT-PACKAGES]
                [NO-EXTRA]
                \\usepackage[french]{babel}
                \\usepackage[cachedir=/tmp/mint]{minted}
                \\input{style_org}"
               ("\\section{%s}" . "\\section*{%s}")
               ("\\subsection{%s}" . "\\subsection*{%s}")
               ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
               ("\\paragraph{%s}" . "\\paragraph*{%s}")
               ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
(add-to-list 'org-latex-classes
             '("notes_music"
               "\\documentclass[10pt,a4paper]{article}
                [NO-DEFAULT-PACKAGES]
                [NO-EXTRA]
                \\usepackage[english]{babel}
                \\usepackage[cachedir=/tmp/mint]{minted}
                \\input{style_org}
                \\lfoot[\\textcolor{colhead}{\\myhtml}]%
                    {\\textcolor{colhead}{\\textsc{F.\\ Galliano's musical stuff}}}
                \\rfoot[\\textcolor{colhead}{\\textsc{F.\\ Galliano's musical stuff}}]%
                    {\\textcolor{colhead}{\\myhtml}}"
               ("\\section{%s}" . "\\section*{%s}")
               ("\\subsection{%s}" . "\\subsection*{%s}")
               ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
               ("\\paragraph{%s}" . "\\paragraph*{%s}")
               ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
(add-to-list 'org-latex-classes
             '("beamer"
               "\\documentclass[8pt]{beamer}
                [NO-DEFAULT-PACKAGES]
                [NO-EXTRA]
                \\input{style_slides}"
               ("\\section{%s}" . "\\section*{%s}")
               ("\\subsection{%s}" . "\\subsection*{%s}")
```

```
                ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                ("\\paragraph{%s}" . "\\paragraph*{%s}")
                ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
```

2. Colors in source code Use Minted to highlight the syntax of source code blocks, when exporting to LaTeX (`python-pygments` need to be installed with `apt`).

```
(require 'ox-latex)
(add-to-list 'org-latex-packages-alist '("" "minted"))
(setq org-latex-listings 'minted)
(setq org-latex-pdf-process
      '("pdflatex -shell-escape -interaction nonstopmode -output-directory %o %f"
        "pdflatex -shell-escape -interaction nonstopmode -output-directory %o %f"
        "pdflatex -shell-escape -interaction nonstopmode -output-directory %o %f"))
```

5.6.2 Customize HTML export

1. Replace "Created" by "Last update"

```
(defun my-org-html-postamble (plist)
   (format "<p style='color:grey'><u>Author:</u> F. Galliano <br><u>Last update:</u> %s
(setq org-html-postamble 'my-org-html-postamble)
```

2. For embedding figures

```
(setq org-html-html5-fancy t
      org-html-doctype "html5")
```

5.6.3 Allow markdown export

Useful for Gitlab.

```
(eval-after-load "org"
  '(require 'ox-md nil t))
```

5.7 Customize the task status

I changed TODO/DONE to a collection of French words:

```
(setq org-todo-keywords
      '((type "À-FAIRE(t)" "EN-COURS(p)" "EN-ATTENTE(w)" "UN-JOUR(s)"
              "|" "FAIT(d)" "ANNULÉ(c)")))
(face-spec-set 'org-todo '((t (:overline t))))
(face-spec-set 'org-done '((t (:overline t))))
```

5.8 Agenda

```
(require 'calfw)
(require 'calfw-org)
(setq org-agenda-files (list "~/ownCloud/Organisation/"))
(setq org-agenda-include-diary nil)
(setq org-agenda-span 7)
(setq calendar-week-start-day 1)
(setq org-agenda-start-on-weekday nil)
(add-hook 'org-agenda-mode-hook
          (lambda ()
            (visual-line-mode t)))
```

5.8.1 Set-up colors and fonts of agenda

```
(setq org-agenda-format-date
  "\n_____%n%n%^A (%d %B %Y)%n_____")
(defun my-org-agenda-day-face-fn (date)
  "Return the face DATE should be displayed with."
  (let ((day-of-week (calendar-day-of-week date)))
    (cond
      ((or (= day-of-week 0) (= day-of-week 6))
```

```
        '(:foreground "grey24" :weight ultra-bold))
       ((or (= day-of-week 1) (= day-of-week 2) (= day-of-week 3)
            (= day-of-week 4) (= day-of-week 5))
        '(:foreground "yellow1" :weight ultra-bold)))))
(setq org-agenda-day-face-function 'my-org-agenda-day-face-fn)
(setq org-agenda-deadline-faces
   '((1.0 . (:foreground "magenta"))
     (0.5 . (:foreground "cyan"))
     (0.0 . (:foreground "red"))))
```

5.8.2 Format the line showing the tasks in agenda view

```
(setq org-agenda-hide-tags-regexp "\\|sometag")
(setq org-agenda-prefix-format '((agenda  . "  • %i %?-12t% s")))
```

5.8.3 Super-agenda settings

```
(require 'org-super-agenda)
(use-package org-super-agenda
  :ensure t
  :after (org org-agenda)
  :config
  (org-super-agenda-mode t)
  (setq org-super-agenda-groups
   '((:name "---------- IMPORTANT/URGENT ----------"   :tag "IMPORTANT"
      :face (:foreground "red" :weight bold))
     (:name "---------- ÉCHÉANCES ----------"          :category "GC-deadline"
      :face (:foreground "magenta" :weight bold))
     (:name "---------- COURRIELS ----------"         :tag "COURRIEL"        )
     (:name "---------- PROJETS ----------"            :tag "PROJET"        )
     (:name "---------- COLLABORATIONS ----------"     :tag "COLLABORATION" )
     (:name "---------- ENCADREMENT ----------"        :tag "ENCADREMENT"   )
     (:name "---------- DÉVELOPPEMENT ----------"      :tag "DEVELOPPEMENT" )
     (:name "---------- INFORMATIQUE ----------"       :tag "INFORMATIQUE"  )
     (:name "---------- SÉMINAIRES ----------"         :tag "SEMINAIRES"    )
     (:name "---------- BUREAUCRATIE ----------"       :tag "BUREAUCRATIE"  )
     (:name "---------- CALENDRIERS ----------"        :tag "CALENDRIERS"
      :category ( "GC-work" "GC-SFL" "GC-DAp" ) )
     (:name "---------- EXTRA-BOULOTIQUE ----------"   :tag "PERSONNEL"
      :category "GC-perso")
     (:name "---------- GUITARE ----------"            :tag "GUITARE"  )
     (:auto-group t :time-grid t) ;; refers to the agenda-group property in the orgs
     (:auto-category t :time-grid t) ;; category is the file name minus .org
    )
  )
  (org-agenda-list)
  (org-agenda nil "a")
  ;; Shows SOMEDAY with C-c a l
  (setq org-agenda-custom-commands
    '(("l" "LONGUE DURÉE" todo "UN-JOUR"
       ((org-agenda-todo-ignore-with-date t))))
  )
)
```

5.8.4 Automatically refresh the agenda

The agenda (C-a a) is automatically refreshed every 5 minutes (300 s):

```
(run-with-idle-timer 300 t (lambda () (org-agenda-maybe-redo)) )
```

5.9 Lilypond

Music in ORG mode, with Lilypond. With this module, you can write musical snippets or whole score in ORG and export them to HTML, PDF or MIDI. . .

```
(require 'ob-lilypond)
(autoload 'LilyPond-mode "lilypond-mode" "LilyPond Editing Mode" t)
(add-to-list 'auto-mode-alist '("\\.ly$" . LilyPond-mode))
(add-hook 'LilyPond-mode-hook (lambda () (turn-on-font-lock)))
(add-to-list 'auto-mode-alist '("\\.ily$" . LilyPond-mode))
```

---

6 EMAILS

6.1 External Softwares and Settings

Before having the possibility to do emails with `emacs`, there are a few things to do. First, install the following software.

- `sudo apt instal isync` (for mbsync, the software retrieving the mails from the server)
- `sudo apt install altermime` (with default settings in the dialog)
- `sudo apt install html2text`

6.1.1 Mu (powerful email search tool)

- Prior to that, you need to install `meson` and other libraries (look at the Github page). However, do not do that using the `--user` option in `pip3`. Do it as `sudo`, otherwise it won't work when you will do `sudo make install`
- `sudo apt install libssl-dev`
- Install `cmake` from the repo (not using `apt`; the version is not high enough)
- `sudo apt install guile-3.0`
- Download tarball from https://github.com/djcb/mu
- Follow the instruction to manually install `mu`
- `mkdir ~/mail`
- `mkdir ~/mail/CNRS`
- `mu init --maildir=~/mail --my-address=xxxxxxxx.xxxxxxxx@cea.fr --my-address=xxxxxxxx.xx`
- `mu index`

It can be used on the command line to search emails, *e.g.* `mu find coucou`. To list the contacts: `mu cfind`.

6.1.2 Create a certificate

For the CNRS account:

```
mkdir -p ~/.cert
openssl s_client -connect imap.cnrs.fr:993 -showcerts 2>&1 < /dev/null | sed -ne '/-BEGIN C
```

6.1.3 Configuration of mbsync

1. Create the `~/.mbsyncrc` file (always the same)
2. Create a crypted password file with gpg2:
   a) Create `~/tmp` file with:
   ```
   machine imap.cnrs.fr login xxxxxxxx.xxxxxxxx@ods.services password XXX
   machine smtp.cnrs.fr login xxxxxxxx.xxxxxxxx@ods.services password XXX
   machine imap.extra.cea.fr login fgallian password XXX
   machine mx.extra.cea.fr login fgallian password XXX
   machine imap.sfr.fr login xxxxxxxx.xxxxxxxx@neuf.fr password XXX
   machine smtp.sfr.fr login xxxxxxxx.xxxxxxxx@neuf.fr password XXX
   ```
   b) `gpg2 --output .authinfo.gpg --symmetric tmp`
   c) `rm ~/tmp`
3. It can be used on the command line, without going through emacs: `mbsync -aV` ⇒ sync all mailboxes. This is what emacs spawns.

6.2 General configuration

6.2.1 Set mu4e

```
(add-to-list 'load-path "/usr/local/share/emacs/site-lisp/mu4e/")
(setq mu4e-mu-binary "/usr/local/bin/mu")
```

```
(setq auth-sources '("~/.authinfo.gpg"))
(require 'mu4e)
```

6.2.2 Basic settings

```
(use-package mu4e
 :defer t
 :config
 (require 'gnus)
 ;; Make mu4e the default emacs app, opened with C-x m
 (setq mail-user-agent 'mu4e-user-agent)
 (set-variable 'read-mail-command 'mu4e)
 ;; For some reason it does not work so enforce it
 (global-set-key (kbd "C-x m") 'mu4e)
 ;; Don't keep message buffers around
 (setq message-kill-buffer-on-exit t)
 ;; No need to confirm
 (setq mu4e-confirm-quit nil)
 ;; Use UTF-8 characters
 (setq mu4e-use-fancy-chars t)
)
```

6.2.3 Addresses

```
(use-package mu4e
 :defer t
 :config
 ;; General emacs mail settings; used when composing e-mail
 ;; the non-mu4e-* stuff is inherited from emacs/message-mode
 ;; later redefined in contexts
 (setq user-mail-address "xxxxxxxx.xxxxxxxx@cnrs.fr"
       user-full-name "Frédéric Galliano"
       mu4e-reply-to-address "xxxxxxxx.xxxxxxxx@cnrs.fr"
       mu4e-compose-reply-to-address "xxxxxxxx.xxxxxxxx@cnrs.fr")
;; To determine whether a message was sent by you, mu4e uses the variable mu4e-user-mail-ad
(setq mu4e-user-mail-address-list '("xxxxxxxx.xxxxxxxx@cea.fr"  "xxxxxxxx.xxxxxxxx@gmail.co
;; Mail aliases for lists (address lists)
(setq mail-personal-alias-file (expand-file-name  "~/ownCloud/Settings/Emacs/mailing-lists.t
)
```

6.2.4 Reception

```
(use-package mu4e
:defer t
:config
;; get mail
(setq mu4e-get-mail-command "/usr/bin/mbsync -aV"
      mu4e-update-interval 300 ;; every 5 minutes
      mu4e-headers-auto-update t
      mu4e-view-show-images t
      mu4e-show-images t
      )
)
```

6.2.5 Répertoires

```
(use-package mu4e
 :defer t
 :config
 ;; Attachments
 (setq mu4e-attachment-dir "~/Downloads")
 ;; Needed for mbsync
```

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

6.3 View

6.3.1 Wrapping long lines

```
(use-package mu4e
 :defer t
 :config
  (add-hook 'mu4e-view-mode-hook #'turn-on-visual-line-mode)
 )
```

6.3.2 View special formats

When the email is too HTMLy... ⇒ use [aV] when the message is open

```
(use-package mu4e
 :defer t
 :config
 ;; View in browser
 (add-to-list 'mu4e-view-actions
  '("ViewInBrowser" . mu4e-action-view-in-browser) t)
 ;; Images
 (setq mu4e-view-image-max-width 800
       mu4e-image-max-width 800)
 (when (fboundp 'imagemagick-register-types)
       (imagemagick-register-types))
 ;; HTML
 (setq mu4e-html2text-command 'mu4e-shr2text)
 (setq shr-color-visible-luminance-min 80)
 (setq shr-color-visible-distance-min 5)
 (setq mu4e-view-prefer-html nil)
 (add-hook 'mu4e-view-mode-hook
   (lambda()
    (local-set-key (kbd "<tab>") 'shr-next-link)
    (local-set-key (kbd "<backtab>") 'shr-previous-link)))
 (setq gnus-unbuttonized-mime-types nil)
)
```

6.3.3 Headers

```
(use-package mu4e
  :defer t
  :config
  (setq mu4e-headers-fields
   '( (:human-date . 25)
      (:from-or-to . 25)
      (:flags . 5)
      (:size . 8)
      (:recipnum . 6)
      (:subject . 90)
      ))
  (setq mu4e-headers-date-format "%a %e %b %Y")
  (setq mu4e-headers-time-format "Aujourd'hui, %H:%M")
  (setq mu4e-headers-show-threads t)
)
```

6.3.4 Fonts

```
 (use-package mu4e
  :defer t
  :config
  ;; Display is definitely nicer with these
```

```
  (setq mu4e-use-fancy-chars t)
  (setq mu4e-headers-precise-alignment t)
  ;; We wanna get the Mono font for the font-lock faces for mu4e-columns-faces
  (set-face-attribute 'font-lock-type-face nil :font (face-attribute 'default :font) :height
  (set-face-attribute 'font-lock-keyword-face nil :font (face-attribute 'default :font) :hei
  (set-face-attribute 'font-lock-string-face nil :font (face-attribute 'default :font) :heig
  (set-face-attribute 'font-lock-variable-name-face nil :font  (face-attribute 'default :fo
  (set-face-attribute 'font-lock-doc-face nil :font (face-attribute 'default :font) :height
  (set-face-attribute 'font-lock-function-name-face nil :font (face-attribute 'default :font
  (set-face-attribute 'font-lock-constant-face nil :font (face-attribute 'default :font) :he
  (set-face-attribute 'mu4e-header-face nil :font (face-attribute 'default :font) :height 1
  ;; Default faces (may be overridden with header filter, see below)
  (set-face-attribute 'mu4e-header-face nil :height 100 :foreground "light cyan")
  (set-face-attribute 'mu4e-unread-face nil :height 100 :background "firebrick" :overline t
  (set-face-attribute 'mu4e-draft-face nil :height 100 :background "dark blue" :foreground
  (set-face-attribute 'mu4e-replied-face nil :height 100 :background "dark magenta" :foregro
  (set-face-attribute 'mu4e-forwarded-face nil :height 100 :foreground "dark green" :foregro
  (set-face-attribute 'mu4e-flagged-face nil :height 100 :foreground "orange")
  (set-face-attribute 'mu4e-header-highlight-face nil :height 100 :background "orange" :fore
  (set-face-attribute 'mu4e-modeline-face nil :height 100 :background "dark red" :foreground
  (set-face-attribute 'mu4e-footer-face nil :height 100 :background "dark gray" :foreground
  (set-face-attribute 'mu4e-compose-separator-face nil :height 100 :background "light green'
  (set-face-attribute 'mu4e-cited-1-face nil :foreground "#0077ff" :slant 'italic)
  (set-face-attribute 'mu4e-cited-2-face nil :foreground "#007788" :slant 'italic)
  (set-face-attribute 'mu4e-cited-3-face nil :foreground "#007744" :slant 'italic)
  (set-face-attribute 'mu4e-cited-4-face nil :foreground "#007700" :slant 'italic)
  (set-face-attribute 'mu4e-cited-5-face nil :foreground "#0000ff" :slant 'italic)
  (set-face-attribute 'mu4e-cited-6-face nil :foreground "#000088" :slant 'italic)
  (set-face-attribute 'mu4e-cited-7-face nil :foreground "#000044" :slant 'italic)
  ;; Threads
  (setq mu4e-headers-thread-child-prefix '(" L  " . "   ")
        mu4e-headers-thread-connection-prefix '(" |  " . "   ")
        mu4e-headers-thread-duplicate-prefix '(" =  " . "   ")
        mu4e-headers-thread-first-child-prefix '(" L  " . "   ")
        mu4e-headers-thread-last-child-prefix '("  " . "  "))
  ;; Flag
  (setq mu4e-headers-flagged-mark   `("F" . ""))
  (setq mu4e-headers-trashed-mark   `("T" . ""))
  (setq mu4e-headers-attach-mark    `("a" . ""))
  (setq mu4e-headers-encrypted-mark `("x" . ""))
  (setq mu4e-headers-signed-mark    '("s" . ""))
  (setq mu4e-headers-unread-mark    `("u" . ""))
  (setq mu4e-headers-new-mark       '("N" . ""))
  (setq mu4e-headers-draft-mark     '("D" . ""))
  (setq mu4e-headers-passed-mark    '("P" . "↪"))
  (setq mu4e-headers-replied-mark   '("R" . ""))
  (setq mu4e-headers-seen-mark      '("S" . ""))
)
```

6.3.5 Searching

```
(use-package mu4e
 :defer t
 :config
 (setq mu4e-maildir-shortcuts
    '((:maildir "/CNRS/Inbox" :key ?i)
      (:maildir "/CNRS/SentMail" :key ?s)
      (:maildir "/CNRS/Trash" :key ?t)
      (:maildir "/CNRS/Drafts" :key ?d)
      (:maildir "/CNRS_archive/Inbox" :key ?j)
      (:maildir "/CNRS_archive/SentMail" :key ?r)
      (:maildir "/CEA_archive/Inbox" :key ?J)
```

```
      (:maildir "/CEA_archive/SentMail" :key ?R)
      (:maildir "/SFR/Inbox" :key ?I)
      (:maildir "/SFR/SentMail" :key ?S))
  )
;; Search for sender (shortcut x)
(defun search-for-sender (msg)
  "Search for messages sent by the sender of the message at point."
  (mu4e-headers-search
   (concat "from:" (cdar (mu4e-message-field msg :from)))))
  (add-to-list 'mu4e-view-actions
   '("xsearch for sender" . search-for-sender) t)
;; Add action to show the local filename of the message
(defun my-show-filename (msg)
  (kill-new (mu4e-message-field msg :path))
  )
(add-to-list 'mu4e-view-actions
             '("filename in kill-ring" . my-show-filename))
(add-to-list 'mu4e-headers-actions
             '("filename in kill-ring" . my-show-filename))
 )
```

6.3.6 Bookmarks

```
(use-package mu4e
 :defer t
 :config
  (makunbound 'mu4e-bookmarks)
  ;; CEA/CNRS bookmarks
  (defvar mu4e-bookmarks
    `( ,(make-mu4e-bookmark
     :name  "Inbox @Home (SFR)"
     :query "maildir:/SFR/Inbox OR maildir:/SFR_archive/Inbox"
     :key ?I)))
  (add-to-list 'mu4e-bookmarks
    (make-mu4e-bookmark
     :name  "Last 7 days @Home (SFR)"
     :query "maildir:/SFR/Inbox AND date:7d..now"
     :key ?W) )
  (add-to-list 'mu4e-bookmarks
    (make-mu4e-bookmark
     :name  "Unread @Home (SFR)"
     :query "(maildir:/SFR/Inbox OR maildir:/SFR_archive/Inbox) AND flag:unread"
     :key ?U) )
  (add-to-list 'mu4e-bookmarks
    (make-mu4e-bookmark
     :name  "Big messages @Work (CEA, CNRS)"
     :query "(maildir:/CNRS/Inbox OR maildir:/CNRS_archive/Inbox) AND size:5M..500M"
     :key ?b) )
  (add-to-list 'mu4e-bookmarks
    (make-mu4e-bookmark
     :name  "With attachment @Work (CEA, CNRS)"
     :query "(maildir:/CNRS/Inbox OR maildir:/CNRS_archive/Inbox) AND flag:attach"
     :key ?a) )
  (add-to-list 'mu4e-bookmarks
    (make-mu4e-bookmark
     :name  "Last 7 days @Work (CEA, CNRS)"
     :query "maildir:/CNRS/Inbox AND date:7d..now"
     :key ?w) )
  (add-to-list 'mu4e-bookmarks
    (make-mu4e-bookmark
     :name  "Inbox @Work (CEA, CNRS)"
     :query "maildir:/CNRS/Inbox"
```

```
     :key ?i) )
  (add-to-list 'mu4e-bookmarks
    (make-mu4e-bookmark
     :name  "Today @Work (CEA, CNRS)"
     :query "maildir:/CNRS/Inbox AND date:today..now"
     :key ?t) )
  (add-to-list 'mu4e-bookmarks
    (make-mu4e-bookmark
     :name  "Unread @Work (CEA, CNRS)"
     :query "maildir:/CNRS/Inbox AND flag:unread"
     :key ?u) )
  )
```

6.4 Attachments

6.4.1 Function to remove attachments

It spawns altermime. The attachment will be erased. If you want it to be saved, save it before.

```
(use-package mu4e
 :defer t
 :config
 (setq mu4e-save-multiple-attachments-without-asking t)
 ;; Attachment cleaning function
 (defun mu4e-remove-attachments (msg)
   "Remove attachment without saving them"
   (interactive)
   (let* ((path (shell-quote-argument (mu4e-message-field msg :path)))
          (command (format "altermime --input=%s --removeall" path)))
    (shell-command command))
    ;;(mu4e-mark-at-point 'something nil)
  )
  (add-to-list 'mu4e-headers-actions
   '("remove-all-attachments" . mu4e-remove-attachments) t)
)
```

6.4.2 Attach files from dired

Need a special version of the gnus-dired-mail-buffers function so it understands mu4e buffers as well. Make the `gnus-dired-mail-buffers` function also work on `message-mode` derived modes, such as `mu4e-compose-mode`:

```
(require 'gnus-dired)
(defun gnus-dired-mail-buffers ()
  "Return a list of active message buffers."
  (let (buffers)
    (save-current-buffer
      (dolist (buffer (buffer-list t))
        (set-buffer buffer)
        (when (and (derived-mode-p 'message-mode)
                   (null message-sent-message-via))
          (push (buffer-name buffer) buffers))))
    (nreverse buffers)))
(setq gnus-dired-mail-mode 'mu4e-user-agent)
(add-hook 'dired-mode-hook 'turn-on-gnus-dired-mode)
```

In dired: C-c RET C-a.

6.4.3 Warning

Warn if no attachments are present, but if the text talks about attachments:

```
  (defun message-attachment-present-p ()
    "Return t if an attachment is found in the current message."
    (save-excursion
      (save-restriction
```

```
        (widen)
        (goto-char (point-min))
        (when (search-forward "<#part" nil t) t))))
  (defcustom message-attachment-intent-re
    (regexp-opt '("attach"
                  "attached"
                  "joint"
                  "joins"
                  "PDF"
                  "attachment"))
    "A regex which - if found in the message, and if there is no
attachment - should launch the no-attachment warning.")
  (defcustom message-attachment-reminder
    "Are you sure you want to send this message without any attachment? "
    "The default question asked when trying to send a message
containing `message-attachment-intent-re' without an
actual attachment.")
  (defun message-warn-if-no-attachments ()
    "Ask the user if s?he wants to send the message even though
there are no attachments."
    (when (and (save-excursion
                 (save-restriction
                   (widen)
                   (goto-char (point-min))
                   (re-search-forward message-attachment-intent-re nil t)))
               (not (message-attachment-present-p)))
      (unless (y-or-n-p message-attachment-reminder)
        (keyboard-quit))))
  ;; add hook to message-send-hook (so also works with gnus)
  (add-hook 'message-send-hook #'message-warn-if-no-attachments)
```

6.5 Sending

6.5.1 Compose

```
(use-package mu4e
  :defer t
  :config
  ;; Allows reading other emails while composing
  (setq mu4e-compose-in-new-frame t)
  ;; Please don't ever include me when I reply...
  (setq mu4e-compose-dont-reply-to-self t)
  ;; Signature
  (setq mu4e-compose-signature-auto-include nil)
  ;; Don't save message to Sent Messages, IMAP takes care of this
  (setq mu4e-sent-messages-behavior 'delete)
  (add-hook 'mu4e-compose-mode-hook #'(lambda () (auto-save-mode -1)))
  ;; Citation
  (setq message-citation-line-function 'message-insert-formatted-citation-line)
  ;; Confirmation before sending
  (add-hook 'message-send-hook
    (lambda ()
      (unless (yes-or-no-p "Sure you want to send this?")
        (signal 'quit nil))))
  ;; Spell check
  (add-hook 'mu4e-compose-mode-hook
    (defun my-do-compose-stuff ()
     "My settings for message composition."
     (set-fill-column 80)
     (flyspell-mode)))
)
```

6.5.2 SMTP settings

```
(use-package mu4e
 :defer t
 :config
 (require 'smtpmail)
 (setq
    message-send-mail-function 'smtpmail-send-it
    smtpmail-auth-credentials (expand-file-name "~/.authinfo.gpg")
    smtpmail-stream-type 'starttls
    smtpmail-smtp-service 587
    ;; Errors
    smtpmail-debug-info t
    auth-source-debug t
    auth-source-do-cache nil
  )
)
```

6.5.3 Accounts and contexts

You can put any variable you want in the account lists, just make sure that you put in all the variables that differ for each account. Variables that do not differ need not be included. Below, I disabled `org-msg` on Birdland, as I am only running Emacs 25.

```
   (use-package mu4e
    :defer t
    :config
    (setq mu4e-context-policy 'pick-first)
    (if (string= (system-name) "Birdland") () (require 'org-msg))
    ;; Work (in English)
    (setq mu4e-contexts
       `( ,(make-mu4e-context
            :name "Work"
            :enter-func (lambda () (mu4e-message "Work (EN) context")
                          (setq mu4e-sent-messages-behavior 'sent))
            :leave-func (lambda ()
              (setq mu4e-maildir-list nil))
            :vars
              '((user-mail-address . "xxxxxxxx.xxxxxxxx@cnrs.fr")
                (user-full-name . "Frédéric Galliano")
                (mu4e-reply-to-address . "xxxxxxxx.xxxxxxxx@cnrs.fr")
                (mu4e-compose-reply-to-address . "xxxxxxxx.xxxxxxxx@cnrs.fr")
                (mu4e-sent-folder .  "/CNRS/SentMail")
                (mu4e-drafts-folder . "/CNRS/Drafts")
                (mu4e-trash-folder . "/CNRS/Trash")
                (smtpmail-default-smtp-server . "smtp.cnrs.fr")
                (smtpmail-local-domain . "cnrs.fr")
                (smtpmail-smtp-user . "xxxxxxxx.xxxxxxxx@ods.services")
                (smtpmail-smtp-server . "smtp.cnrs.fr")
                (message-citation-line-format . "On %a %d %b %Y à %R, %n wrote:\n")
                (message-signature-file . "~/ownCloud/Settings/Emacs/email_signature_CEA.txt
                (org-msg-greeting-fmt . "\nDear%s,\n\n")
                (org-msg-signature . "\n\nBest regards,\n\n#+begin_signature\n-----\n*Frédé
                ))
        ;; Travail (in French)
        ,(make-mu4e-context
            :name "Travail"
            :enter-func (lambda () (mu4e-message "Work (FR) context")
                          (setq mu4e-sent-messages-behavior 'sent))
            :leave-func (lambda ()
              (setq mu4e-maildir-list nil))
            :vars
              '((user-mail-address . "xxxxxxxx.xxxxxxxx@cnrs.fr")
```

```
                (user-full-name . "Frédéric Galliano")
                (mu4e-reply-to-address . "xxxxxxxx.xxxxxxxx@cnrs.fr")
                (mu4e-compose-reply-to-address . "xxxxxxxx.xxxxxxxx@cnrs.fr")
                (mu4e-sent-folder .  "/CNRS/SentMail")
                (mu4e-drafts-folder . "/CNRS/Drafts")
                (mu4e-trash-folder . "/CNRS/Trash")
                (smtpmail-default-smtp-server . "smtp.cnrs.fr")
                (smtpmail-local-domain . "cnrs.fr")
                (smtpmail-smtp-user . "xxxxxxxx.xxxxxxxx@ods.services")
                (smtpmail-smtp-server . "smtp.cnrs.fr")
                (message-citation-line-format . "Le %a %d %b %Y à %R, %n a écrit:\n")
                (message-signature-file . "~/ownCloud/Settings/Emacs/email_signature_CEA.txt
                (org-msg-greeting-fmt . "\nBonjour%s,\n\n")
                (org-msg-signature . "\n\nSincèrement,\n\n#+begin_signature\n-----\n*Frédéri
               ))
          ;; Home
          ,(make-mu4e-context
             :name "Maison"
             :enter-func (lambda () (mu4e-message "Home context")
                           (setq mu4e-sent-messages-behavior 'sent))
             :leave-func (lambda ()
               (setq mu4e-maildir-list nil)) :vars
               '((user-mail-address . "xxxxxxxx.xxxxxxxx@neuf.fr")
                 (user-full-name . "Frédéric Galliano")
                 (mu4e-reply-to-address . "xxxxxxxx.xxxxxxxx@neuf.fr")
                 (mu4e-compose-reply-to-address . "xxxxxxxx.xxxxxxxx@neuf.fr")
                 (mu4e-sent-folder .  "/SFR/SentMail")
                 (mu4e-drafts-folder . "/SFR/Drafts")
                 (mu4e-trash-folder . "/SFR/Trash")
                 (smtpmail-default-smtp-server . "smtp.sfr.fr")
                 (smtpmail-local-domain . "sfr.fr")
                 (smtpmail-smtp-user . "xxxxxxxx.xxxxxxxx@neuf.fr")
                 (smtpmail-smtp-server . "smtp.sfr.fr")
                 (message-citation-line-format . "Le %a %d %b %Y à %R, %n a écrit:\n")
                 (message-signature-file . "~/ownCloud/Settings/Emacs/email_signature_SFR.txt
                 (org-msg-greeting-fmt . "\nBonjour%s,\n\n")
                 (org-msg-signature . "\n\nSincèrement,\n\n#+begin_signature\n-----\n*Frédéri
                ))))
      (custom-set-faces
       '(mu4e-context-face ((t (:foreground "dark green" :weight bold
                                :background "grey")))))
;;      (if (string= (system-name) "Birdland") () (org-msg-mode))
      (require 'smtpmail)
      (setq mu4e-compose-context-policy nil)
   )
```

This is the end of the file, but `emacs` customization never ends. . .
