

Sonic Emacs Literate Programming

PBS

2026

Contents

I Sonic Emacs Config in Literate Programming	3
0.1 Package sources and intialization	5
0.2 UI	5
0.3 Theme and Dashboard	5
0.4 Autocompletion and Discovery	6
0.5 Snippets	7
0.6 Org mode and Literate Programming	7
0.7 Latex Support	8
0.8 Org Roam	9
0.9 Boilerplate	9
0.10 Treemacs and Vterm	9

Part I

Sonic Emacs Config in Literate Programming

This code initializes the package sources to install, while the Bootstrap part ensures that if you are in a new computer it will refresh contents to install the packages, the "setq use-package-always-ensure" ensures that a package is installed if it is missing.

0.1 Package sources and initialization

```
(require 'package)
(setq package-archives '(("melpa" . "https://melpa.org/packages/")
                        ("elpa" . "https://elpa.gnu.org/packages/")
                        ("nongnu" . "https://elpa.nongnu.org/nongnu/")))
(package-initialize)

;; Bootstrap 'use-package'
(unless (package-installed-p 'use-package)
  (package-refresh-contents)
  (package-install 'use-package))
(require 'use-package)
(setq use-package-always-ensure t)
```

0.2 UI

This bit involves some UI readjustments for optimal discovery and movement within the editor.

- fido-vertical-mode is disabled so vertico can take over (those show matching options when you run something like M-x).
- "global-visual-line-mode t" soft-wraps the lines instead of cutting them off or moving the buffer to the right.
- "display-line-numbers-type 'relative" puts relative lines, which are good for users that like to use Emacs commands to jump lines or pages.
- "global-display-line-numbers-mode" allows the person to view the line number

```
;; EMACS UI
(fido-vertical-mode -1)
(global-visual-line-mode t)
(setq display-line-numbers-type 'relative)
(global-display-line-numbers-mode 1)
```

0.3 Theme and Dashboard

This bit is about the aesthetic, it sets the theme and the dashboard Tardis logo

- The first part just sets the ef-deuteranopia-dark theme
- The second part calls for a file named tardis.png to be used instead of the Emacs logo in the dashboard, it also sets a size for it to fit properly

```
;; Theme: TARDIS Blue / Deuteranopia dark
(use-package ef-themes
  :ensure t
  :config
  (load-theme 'ef-deuteranopia-dark t))

;; Startup Dashboard
(use-package dashboard
  :config
  (dashboard-setup-startup-hook)
  (setq dashboard-startup-banner (expand-file-name "~/tardis.png"))
  (setq dashboard-image-banner-max-height 300)
  (setq dashboard-center-content t)
  (setq dashboard-items '((recents . 5)
                         (projects . 5))))
```

0.4 Autocompletion and Discovery

This part involves packages and configurations to ease the use of Emacs, allowing command and shortcut discovery and text completion.

- which-key allows the discovery of all the possible shortcut combinations when you do a certain shortcut like C-c, it is also set to show after a delay of 0.5 seconds
- Orderless allows "fuzzy" matching (e.g., "func my" matches if you type "my func"), allowing ease of discovery and completion
- Corfu is a package that shows a popup UI for auto-completion, in this case "corfu-auto t" enables auto-completion, while the delay and prefix options makes the popup appear after 0.1 seconds and after 2 characters typed. "corfu-cycle t" makes that when you reach the end of the list if you press down it will go back the start. "corfu-preselect 'prompt" prevents Corfu from automatically picking a candidate when you have not moved the selection (typed an arrow or dragged the mouse towards the popup)
- Eglot is a built-in LSP client in Emacs, what the configuration is doing is ensuring that eglot starts in any programming mode if a LSP server is found, so say you open a .c file, it will automatically start eglot C LSP.
- Cape allows multiple backends for autocompletion so for example it can use, so for example words already typed in a file + LSP + filenames, etc. "dabbrev" allows autocompletion of words already written elsewhere in your open files. "cape-file" allows autocompletion for file paths, so if you type something like /home/user/file.something it will automatically show. "cape-elisp-block" allows some elisp autocompletion and "cape-keyword" shows some keywords for programming languages, like "if", "while", etc. "(list (cape-capf-super #'eglot-completion-at-point #'cape-dabbrev #'cape-keyword))" merges all the backends together in the autocompletion popup.
- Vertico is a bar that appear when you run something like M-x, like Corfu, but for commands (as far as I know), the settings are similar to corfu, cycling lists, prevention of automatic selection.
- "tab-always-indent" will first indent if a line is not indented properly, then select something like an autocompletion

```
;; HELP, DISCOVERY AND COMPLETION
(use-package which-key
  :init (which-key-mode)
  :config (setq which-key-idle-delay 0.5))

;; Orderless: Allows "fuzzy" matching (e.g., "func my" matches "my func")
(use-package orderless
  :ensure t
  :custom
  (completion-styles '(orderless basic))
  (completion-category-overrides '((file (styles basic partial-completion)))))

;; Corfu: The popup UI for auto-completion
(use-package corfu
  :ensure t
  :custom
  (corfu-auto t)
  (corfu-auto-delay 0.1)
  (corfu-auto-prefix 2)
  (corfu-cycle t)
  (corfu-preselect 'prompt)
  :init
  (global-corfu-mode 1))

;; Eglot: The built-in LSP client
(use-package eglot
  :ensure t
  :hook
  (prog-mode . eglot-ensure))

;; Cape: Extends completion backends
(use-package cape
  :ensure t)
```

```
:init
(add-to-list 'completion-at-point-functions #'cape-dabbrev)
(add-to-list 'completion-at-point-functions #'cape-file)
(add-to-list 'completion-at-point-functions #'cape-elisp-block)
(add-to-list 'completion-at-point-functions #'cape-keyword)
:config
(add-hook 'eglot-managed-mode-hook
  (lambda ()
    (setq-local completion-at-point-functions
      (list (cape-capf-super
        #'eglot-completion-at-point
        #'cape-dabbrev
        #'cape-keyword))))))

;; VERTICO provides a vertical bar minibuffer for autocompleting commands

(use-package vertico
:ensure t
:custom
(vertico-preselect 'prompt)
(vertico-cycle t)
:init
(vertico-mode 1))

;; Emacs core completion settings
(use-package emacs
:custom
(tab-always-indent 'complete))
```

0.5 Snippets

This part allows yasnippet, which allows you to make and use snippets (I do not really use it, but it can be handy).

;; Snippets

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

0.6 Org mode and Literate Programming

- “org-display-inline-images” allows inline images by default in org mode so you can view it when you add an image to an org file
- “org-bullets” makes some things prettier, for example if you type “*” it will auto indent, change the color of the characters in that line and have different symbols for each asterisk
- “org-babel-load-languages” adds support for some languages in org babel, if needed other languages can be added or removed
- “org-tempo” allows you to type “<s” + TAB to insert a code block
- “org-confirm-babel-evaluate nil” removes the popup to run the code when you run C-c C-c in a code block
- “org-src-fontify-natively t” allows syntax highlighting in code blocks
- “org-src-tab-acts-natively t” allows language specific TAB behavior
- the “org-src-mode-hook” and “eglot-ensure” part ensures that if you open a code block, like in C-c ‘ then eglot will start
- the last part about indentation ensures that code formatting is clean and no new lines or indents are added, which is important for languages like python that rely on indentation for proper functioning.

; ;ORG MODE AND LITERATE PROGRAMMING

```
(use-package org
  :hook (org-mode . org-display-inline-images)
  :config
(use-package org-bullets
  :config (add-hook 'org-mode-hook (lambda () (org-bullets-mode 1)))))

;; Babel: Language Support
(org-babel-do-load-languages
 'org-babel-load-languages
 '((emacs-lisp . t) (python . t) (C . t) (latex . t) (haskell . t)))

;; Literate Programming & Source Block Behavior
(with-eval-after-load 'org
  (require 'org-tempo) ;; Typing "<s" + TAB inserts code blocks
  (setq org-confirm-babel-evaluate nil)
  (setq org-src-fontify-natively t)
  (setq org-src-tab-acts-natively t))

(add-hook 'org-src-mode-hook #'eglot-ensure)

(setq org-src-preserve-indentation t
      org-src-tab-acts-natively t
      org-edit-src-content-indentation 0)
```

This part of the code has the part tailored for my preferred type of L^AT_EX document, if needed a person can change it, though what I do mostly (beyond the basics) is add packages.

- ox-latex is the org export library for latex, while the next line adds the compiler lualatex, which I think has the broadest support.
- org-format-latex-options sets math preview scale to 1.5 so it can be easier to read
- the rest just adds the packages and defines that for example one "" will be equivalent to a part in the document while two will be a chapter. Also it sets a custom-book class, if needed you can create other classes, then in the beginning of the org file you choose the class you want like: "#+LATEX_CLASS: custom-book"

0.7 Latex Support

```
;; LATEX SUPPORT
;; Add these to the beginning of the document:
;; #+TITLE: Your Title
;; #+AUTHOR: PBS
;; #+DATE: 2026
;; #+LATEX_CLASS: custom-book
;;
;; If you want to add native LaTeX you can for example do \newpage, but you can also do something like:
;;#+BEGIN_EXPORT latex
;;\begin{multicols}{2}
;;#+END_EXPORT

(require 'ox-latex)
(setq org-latex-compiler "lualatex")

;; Adjust Preview Scale
(setq org-format-latex-options (plist-put org-format-latex-options :scale 1.5))

;; Define the Custom Class
(add-to-list 'org-latex-classes
  ('("custom-book"))
```

0.8 ORG ROAM

```

"\\documentclass{book}
\\usepackage[multicol]
\\usepackage{float}
\\usepackage{fontspec} % Required for XeLaTeX font selection
\\usepackage{graphicx}
\\usepackage{color}
\\usepackage[a4paper, total={8in, 10in}]{geometry}
\\usepackage{lmodern} % necessary for small font
\\usepackage{fix-cm} % necessary for small font
\\usepackage{enumitem} % removes whitespaces between lists and others
\\usepackage{parskip} %removes whitespaces between lines
\\usepackage{blindtext} % allows clearing double pages (I think)
[DEFAULT-PACKAGES]
[PACKAGES]
[EXTRA]
\\graphicspath{{images/}}
\\let\\cleardoublepage=\\clearpage"
("\\part[%s]" . "\\part*[%s]")
("\\chapter[%s]" . "\\chapter*[%s]")
("\\section[%s]" . "\\section*[%s]")
("\\subsection[%s]" . "\\subsection*[%s]")
("\\subsubsection[%s]" . "\\subsubsection*[%s"]))

```

0.8 Org Roam

This part adds org roam and sets its directories, org-notes for its base directory and daily for daily capture. The last line ensures that whenever you write some change in a org file it will sync again the links and files.

```

;; ORG ROAM
(use-package org-roam
  :custom
  (org-roam-directory (file-truename "~/Documents/org-notes"))
  (org-roam-dailies-directory "daily/")
  :config
  (org-roam-db-autosync-mode))

```

0.9 Boilerplate

Emacs autogenerates this, not much to explain.

```

;; SET FACES BOILERPLATE

(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.
)

```

0.10 Treemacs and Vterm

This bit adds Treemacs a package for file tree view and Vterm, a terminal emulator (more capabilities than eshell)

- “:defer t” and “:commands” on both of them ensures they do not initialize before requested, which avoids slow down of Emacs.
- The treemacs-width 30, treemacs-is-never-other-window and treemacs-follow-after-init nil ensures the bar will have 30 characters width, files and buffers won’t open on the sidebar window and it will also not move unless requested.
- One thing to note is that Vterm (maybe some other packages) require certain dependencies to compile, like Cmake and libtool

```
; ; TREEMACS AND VTERM

(use-package treemacs
  :ensure t
  :defer t
  :commands (treemacs treemacs-toggle)
  :config
  (setq treemacs-width 30
        treemacs-is-never-other-window t
        treemacs-follow-after-init nil))

(use-package vterm
  :ensure t
  :defer t
  :commands vterm
  :config
  )
```