Basecamp

Jeffrey Szymanski

CONTENTS

Last compiled November 14, 2024.

1 Setup And Configuration

1.1 General Information And Good Practice For Base Computing Setup

1. Bash Configuration .bashrc is used for interactive shell customizations and .profile is used for setting up environment variables and system-wide settings needed during the login process.

   (a) Bashrc

   - The default bashrc lives at *etc/skel*.bashrc.
   - Other programs such as singularity or conda may write directly to bashrc.
   - Otherwise, bashrc is not altered automatically, as through Org-mode tangling. Instead, additional code is sourced though a simple if else statement. I source my basecamp shell libraries as follows:

```
for file in "${HOME}"/repos/basecamp/lib/*.sh; do
    if [ -f "$file" ]; then
        source "$file"
    fi
done
```

   - Use functions instead of aliases. Functions are more flexible, and can be debugged.

1.2 An Opinionated And Incomplete Base Computing Configuration

1. Base operating system

   My base operating system is linux Ubuntu using a long-term support (LTS) version.

2. Bash

3. Git repositories Git repos live in ${HOME}/repos.

   Git repos are modular using the git submodule structure.

   Git repos are pushed to GitHub.

   Git version control follows a simple trunk-based development strategy, with a single active branch: master. Major changes can be spun out into separate branches, but should be quickly merged back to master after changes are validated. Stable, validated versions are occasionally saved as git tags.

   (a) Repositories where I am the main or sole author

   Repository code is tangled from a single Org-mode file. Updates from others are through pull requests as tangling would overwrite any direct commits.

4. Core directory structure – /

5. Data governance

   (a) Security

   (b) Architecture

   (c) Lifecycle

   (d) Management

      i. Backup

6. Core applications core components (incomplete)

   - installed via apt
     - syncthing
     - i3
     - emacs
     - texlive
     - texlive
     - rclone
     - okular

- libreoffice
- minor
  - tree

- R

(a) R

- [ ] convert python and R to argparse https://www.google.com/search?q=r+argparse+optparse&oq=r+argparse+optparse&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDg2MDFqMGo0qAIAsAIA&sourceid=chrome-mobile&ie=UTF-8

(b) Python

- [ ] convert python and R to argparse https://www.google.com/search?q=r+argparse+optparse&oq=r+argparse+optparse&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDg2MDFqMGo0qAIAsAIA&sourceid=chrome-mobile&ie=UTF-8

(c) Emacs

i. Initialization
When my Emacs loads, several configuration files and directories are references by symbolic link from git repositories to the default ~/.emacs.d directory:

- init.el
- public_emacs_config.el
- private_emacs_config.el
- public_yasnippets
- private_yasnippets

These are loaded as optional files and if not found, they will throw a warning on initialization. These files are generated by Org-mode tangle.

The init.el file is compiled from the Org-mode header a simple wrapper for other Elisp files.

My public customized Emacs initialization is compiled from the Org-mode header basecamp.org::Public configuration and tangles to ./config/public_emacs_config.el.

A. Style and good practice
Use-package is my preferred package management system. By default, use-package loads are structured as:

```
(use-package package2
:ensure t
:init
:config
)
```

Each use-package keyword will expect a list or needs to be removed. Other keywords exist (see documentation).

- :ensure t will install any missing package

- :init is evaluated before package loading
- :config is evaluated after package loading

ii. Org-mode
   A. Policy, style, and good practice
   B. Block structures, source code, and tangling
      Lowercase is preferred for all block notation, *e.g.*

      instead of

      At the file level, tangled code should reference it's location in orgmode files.
      tangle defaults to ./

iii. Templating with YASnippets My public YASnippets are compiled from source code blocks across my Org-mode agenda files. These tangle into appropriate subdirectories of ./emacs/public_yasnippets. Private snippets are tangled into appropriate subdirectories of org/emacs/private_yasnippets.

2 Pan-computing Good Practive And General Style Guide

- Directory names should be in hyphen-case, using only lowercase letters (a-z), digits (0-9), and hyphens (-) as separators. Use as few words as possible
  - Can use more human-friendly title case and spaces with symlinks and shortcuts
- Prefer single-word file and directory names
- For multi-word file and directory names, prefer a dash (-) separator (*e.g.* a-longer-file-name.txt)

3 Conda

For repositories and projects with heavy use of Python and R, software should be managed through the Conda package manager.

3.1 My Basecamp Conda Environment

A basecamp conda environment is stored in this repository at basecamp_env.yaml.

4 Emacs

4.1 Setup, Configuration, And Customization

My setup assumes normal emacs initialization and package structure at ~/.emacs.d. The ~/.emacs.d/init.el ensures package management and then conditionally loads other lisp files, if they exist, in the following order:

1. ~/.emacs.d/load-early.el

2. ~/.emacs.d/config/

3. ~/.emacs.d/load-late.el

The ~/.emacs.d/config directory can contain symlinked lisp files from other locations. There is no assumed order within that directory.

All files mentioned here can be found in the Lisp Files appendix below.

load-path is a list of directories that Emacs searches when you use functions like require, load, or when Emacs needs to find a library or package. When you add a directory to load-path, you're telling Emacs where to look for Emacs Lisp files when they need to be loaded. My load path is simply the default package location ~/.emacs.d/elpa and also all files anywhere within ~/.emacs.d/lisp

4.2 Simple Emacs Lisp Tutorial

Alist

alist: association list, stores key-value pairs

```
(setq my-alist '((key1 . value1)
                 (key2 . value2)
                 (key3 . value3)))

(assoc 'key2 my-alist)  ;; Returns (key2 . value2)
(cdr (assoc 'key2 my-alist))  ;; Returns value2

(setq my-alist (cons '(key4 . value5) my-alist))  ;; Adds (key4 . value4) to the front of the alist
(setq my-alist (assoc-delete-all 'key2 my-alist))  ;; Removes the element with key 'key2'
```

```
(setq my-alist '((key1 . value1)
                 (key2 . value2)
                 (key3 . value3)))

(prin1 (assoc 'key2 my-alist))  ;; Display (key2 . value2)
(princ "\n")  ;; Add a newline for readability
(prin1 (cdr (assoc 'key2 my-alist)))  ;; Display value2
(princ "\n")  ;; Add a newline for readability

(setq my-alist (cons '(key4 . value4) my-alist))  ;; Adds (key4 . value4) to the front of the alist
(prin1 my-alist)  ;; Display the updated alist
(princ "\n")  ;; Add a newline for readability

(setq my-alist (assoc-delete-all 'key2 my-alist))  ;; Removes the element with key 'key2'
(prin1 my-alist)  ;; Display the final alist
```

progn: special form to evaluate sequence of expressions

```
(progn
  (message "First")
  (message "Second"))
```

4.3 Org-mode

If you place an asterisk at the beginning of your search, Org-mode will search only headlines (and not entry text). E.g., to find all entries with ”emacs” in the headline, you could type:

```
C-c a s
[+-]Word/{Regexp} ...: *+emacs
```

4.4 Literate Programming With Emacs Org-mode

Generally comments should reside within the Org-mode structure and outside of code blocks. Tangling with a header argument :comments org will include both the header text and text between the header and code block. For example:

(property drawers are excluded from tangling with custom-org-remove-properties-drawer)

1. For example:

   (a) This header will be a comment This text in org and below the header will be a comment

   ```
   # This text within the code block will be a comment

   ls
   ```

   This comment will not appear in the tangled code

      i. Child headers are not comments unless they contain more code blocks

   (b) Result:

   ls

4.5 YASnippets Style Guide And Good Practice

- – snippets are associated with a specific major mode, use of bash mode is discouraged
  – As snippets are associated with major modes, single word keywords are encouraged (e.g. function instead of bash.function)
  – snippets expand on tab
  – for modes with extensive snippet libraries, prefixes followed by a single period are preferred (e.g. mod.meeting)
  – prefixed snippets keywords are not likely to be confused with non-snippet terms, and they should expand without trigger
  – Snippet creation and storage
    * Snippets are created in org mode bash source code blocks with formatting as in the snippet

- Each snippet resides under it's own terminal org mode header. The header consists of only the snippet keyword and tags including the :yas: tag (i.e. * <SNIPPET KEYWORD> :yas:)

- The :yas: tag is reserved exclusively for yas block headers and does not have tag inheritance.

* Snippets are stored in the main org repository under the directory./snippets which is symlinked to .emacs.d. Snippets are therefore under org version control.

4.6 Notes

See also Emacs and Org-mode use in my LaTeX repository (GitHub, local buffer).

5 Appendices

5.1 Lisp Files

```
;;-*- mode: elisp -*-

;; Package Management Setup

(require 'package)

(add-to-list 'package-archives '("melpa" . "https://melpa.org/packages/") t)

;; Ensure 'use-package' is installed
(unless (package-installed-p 'use-package)
  (package-refresh-contents)
  (package-install 'use-package))

(require 'use-package)
(setq use-package-always-ensure t)

(defun add-to-load-path-if-exists (dir)
  "Add DIR and its subdirectories to the Emacs load-path if DIR exists."
  (if (file-directory-p dir)
      (let ((default-directory dir))
        (message "Adding %s and its subdirectories to load-path" dir)
        (normal-top-level-add-to-load-path '("."))
        (normal-top-level-add-subdirs-to-load-path))
    (message "Directory %s does not exist, skipping." dir)))

(add-to-load-path-if-exists "~/.emacs.d/lisp/")

;; Function to safely load a file if it exists
(defun safe-load-file-if-exists (filepath)
  "Safely load the Emacs Lisp file at FILEPATH if it exists."
  (when (file-exists-p filepath)
    (condition-case err
        (load (file-name-sans-extension filepath))
      (error (message "Error loading %s: %s" filepath err)))))

;; Load early configuration
(safe-load-file-if-exists "~/.emacs.d/load-first.el")

;; Define the path to your configuration directory
```

```
(defvar my-config-dir "~/.emacs.d/config/"
  "Directory containing personal Emacs configuration files.")

;; Load all .el files in the config directory
(when (file-directory-p my-config-dir)
  (dolist (file (directory-files my-config-dir t "\\.el$"))
    (condition-case err
        (load (file-name-sans-extension file))
      (error (message "Error loading %s: %s" file err)))))

;; Load late configuration
(safe-load-file-if-exists "~/.emacs.d/load-last.el")

(add-to-list 'exec-path "/usr/local/bin")

;(cua-selection-mode t)
;(setq mark-even-if-inactive t) ;; Keep mark active even when buffer is inactive
;(transient-mark-mode 1) ;; Enable transient-mark-mode for visual selection
(scroll-bar-mode 'right) ;; Place scroll bar on the right side

(setq org-export-backends '(ascii html latex odt icalendar md org)) ; This variable needs to be set before o

(run-with-idle-timer
 1 nil
 (lambda ()
   (when (member "Hack" (font-family-list))
     (set-face-attribute 'default nil
                         :family "Hack"
                         :height 114
                         :weight 'light)
     (message "Font set to Hack"))))


(custom-set-faces
 '(default ((t (:family "Hack" :height 114 :weight light)))))

(add-hook 'emacs-startup-hook
          (lambda ()
            (load-theme 'manoj-dark t)))

(load-theme 'manoj-dark t)
```