1. README

   ID: org_gcr_2017-05-12_mara:3DBC61FF-D790-471A-904F-DABFB0DABA1F

   Configure EMACS to for everything defined within this monolithic system.

   Sysop is likely to use this constantly.

   Start EMACS with this command:

   ```
   emacs --debug-init --no-init-file --no-splash --background-color white --foreground-color bl
   ```

   EMACS values dynamic-scoping for now and in the future.

   EMACS values lexical-scoping in the future.

   There are pecuilar interactions between lexically and dynamically scope closures and special variables.

   With the future in mind, make the switch now.

   It is enabled with a non-nil buffer-local variable `lexical-binding`. The variable is inserted only here because it loads each of the child configurations. Web.

   Never compile this.

   ```
   ;; -*- lexical-binding: t; no-byte-compile: t; -*-
   ```

   ```
   ;; Added by Package.el.  This must come before configurations of
   ;; installed packages.  Don't delete this line.  If you don't want it,
   ;; just comment it out by adding a semicolon to the start of the line.
   ;; You may delete these explanatory comments.
   ;(package-initialize)

   (load-file "~/src/help/.org-mode-contribute.emacs.el")
   ```

   (a) HELP Enables Literate Programming

      ID: org_gcr_2017-05-12_mara:73599C3B-39B0-4979-86C2-E4AC068AAC79

      Setup

      i. Clone Org-Mode to `~/src/`

      ii. Go there.

         ```
         cd ~/src/org-mode
         ```

      iii. Build it

         A. Without `Make`: Generating autoloads and Compiling Org without make

         B. As of *<2016-01-19 Tue>* this means doing this:

            ```
            emacs -batch -Q -L lisp -l ../mk/org-fixup -f org-make-autoloads
            ```

         C. With `Make` use `make`.

      iv. Clone Org2Blog to `~/src/`.

      v. Clone Use-Package to `~/src/`.

vi. Install supporting software adding their exectuable location to the `PATH`.

A. Install Oracle Java.

B. Install LanguageTool renaming it's folder to `LanguageTool`.

C. Install PlantUML.

D. Install Ditaa.

E. Install MacTeX.

vii. Link:

- The Eshell directory to HELP's.
  - `ln -s ~/src/help/eshell/ ~/.emacs.d/eshell`
- The Initialization file to HELP's.
  - `ln -s ~/src/help/.emacs.el ~/.emacs.el`

(b) Style Guide

`ID: org_gcr_2017-05-12_mara:280391BE-63FC-4E2E-B734-AF18A449FA96`

- Appearance.
  - Never override theme colors.
  - When the theme doesn't configure a face then submit a patch.
- Content
  - When importing update to conform with Style-Guide.
  - Keep tangled and weaved documents synchronized with their web.
  - Ask yourself:
    * Does it belong in this web?
    * Does it belong in this headline?
- Dictionary

Exemple Complet Minimal (ECM) The minimal complete example of expected versus actual behavior. Source.

Key-Bind A verb. The act of creating a Key Binding.

Literate Programming (LP) As Knuth intended.

Out of the Box (OOTB) The default configuration.

Sysop A proper noun. The System-Operator. The human operating this EMACS based Org-Mode enabled literate programming system. The reader.

Tangle A verb. Assemble a document for consumption by another program or machine.

Weave A verb. Prepare a document for consumption by a human.

Web A document contained Source-Block definitions that define a system.

- Encoding
  - Prefer Unicode characters over ASCII equivalents.
    - Note eventual switch form PDFLaTeX to LuaTeX.
  - Consider Org-Mode automatic handling of ASCII to UTF-8 symbols.
- File/Package Loading
  - Load every one with `use-package` whether it came with EMACS OOTB or ELPA.
    - `ensure t` tells the reader which one it came from.
  - Binding definitions often live in Piano Lessons.
  - Configurations aren't meant to be fully transplant-able because this system monolithic. In the interest of collaboration as much of the package configuration lives in the `use-package` block as possible.
- Formatting
  - Code Snippet.
    - Key bindings
    - Programming language expressions
      - Variable names
    - Emacs mode function name, package name or Github name
      - `org-mode`
    - Class and Object names
    - Shell commands that are executable
    - Executables for programming language compilers and interpereters
      - `gcc` and `python`
      - `bash` and `perl`
      - `scheme` and `java`
      - `GCC` and `Pyton`
      - `Bash` and `Perl`
      - `Scheme` and `Java`
    - Anything that you would read program or text
    - Use `code style` of tildaes
  - Non-Code Snippet.
    - ALL PROPER NOUNS

        * File types
        * File names
        * Emacs mode name: THE HUMAN VERSION
            · Other stuff like function-name and package-name and Github name are not how humans write about them. Humans write about them like any other proper noun for example “Coca Cola” not “coca cola”.
            · Org-Mode is the best example
        * Concepts like `Object Oriented Programming` or a `Immediately-invoked function expression`
        * Use `verbatim` style of equal sign
    – Package.
        * Same as Headline.
        * Dashes separate definition.
        * Acronyms are all upper case to distinguish from words for example “GNU” vs “Gnu”.
- Headline.
    – Be sure that every one has an `ID` property with a `UUID` value.
        * `org-id-get-create` does this. So does code in Hacking/Org Mode/Utility.
    – Capitalize: nouns, verbs, and adjectives.
    – Don’t capitalize conjunctions unless they are starting the definition.
    – Sell this “chapter” to the reader.
    – Some headlines will be empty and significant; keep them.
        * Some modes don’t require any configuration. The headline still needs to be present to remind the reader to keep it in her cognitive landscape. Configure other properties and modes taking it into account.
        * For every language under Hacking you should:
            · Only include it if it is valuable and you will invest adequate time to configure this well and use it well. When reading Org-Mode examples you will want to add Org-Mode language support because it is easy and fun and then you are left with an insufficiently configured environment. That is unacceptable.
            · Enable it in `org-babel-load-languages`.
            · Read the user manual for it.
        * The `Prog-Mode` system configuration can result in Headlines that don’t need any configuration. The Headline still communicates the mode’s value to the reader even if it doesn’t configure EMACS.

  - It is important to include headlines that are empty and that you may not even end up keeping. You need them to help you explore the cognitive landscape with them present. In this one case, premature optimization is *not* the root of all evil.
  - Sometimes headlines might better be:
    * List items.
    * Stand-alone bold text without punctuation.
- Hyperlink.
  - External.
    * Exclude those easily found with a search-engine unless you are willing to verify their existence frequently.
    * Include when they make the task at hand immensely easier.
  - Internal.
    * Minimize usage.
- Literate Programming.
  - Comments.
    * Exclude from tangle-blocks and rely on source-block for traceability.
  - Noweb-Ref.
    * Same as Headline.
    * Replace spaces with dashes.
    * Probably the Heading name.
    * Keep depth shallow
  - Weaver and mode configurations are tightly bound.
- Maintenance.
  - Frequently check spelling, grammar, and weasel-words.
  - Only keep working features in the system.
- Macros
  - Rendered as written-text.
  - Don't contain source code.
  - Create for ideas expressed more than 2 times.
  - Expanded during weaving, not during tangling.
- Plain List.
  - End single sentences with a period.

- Programming Language.
  - Emacs-Lisp.
    - Almost always use `defun` instead of `advice-add`.
      - Functions are more normal and predictable.
      - Advice can subtly break without you noticing.
    - Parameter.
      - `nil` for `FALSE`.
      - `t` for `TRUE`.
      - `n` for numerical values.
    - Never `custom-set-variables`.
    - Always use relative file paths.
    - Byte-compile frequently to minimize System warnings.
    - Prefer to byte-compile all references by using `function`.
      - Fail-fast: it is better to know immediately if there are resolution issues.
    - Prefer to declare anonymous functions with `function`.
    - Quoted via.
    - If a non-special variable appears outside of a let form, the byte-compiler will warn about reference or assignment to a "free variable". An unused non-special variable binding within a let form provokes the byte-compiler will warn about an "unused lexical variable". The byte-compiler will also issue a warning if you use a special variable as a function argument.
- Source Block
  - Be sure that every one has a `NAME` property with a `UUID` value.
    - YASnippet `sc` does this. So does code in Hacking/Org Mode/Utility.
  - Tell the story in speech, and then in code.
  - Communicate the intent in written language as one paragraph and realize that intent in the next paragraph as a source block. Separate the two paragraphs like you would any other paragraph.
    - The exporter will probably separate the two as you would expect whether you separate the two entities with a space or not
  - When contained within a list:
    - Indent begin/end blocks with list content; this makes it clear to Org-Mode to export it as a code block.
  - Tangle `:file` should have the same `NAME`.
  - Virtually never edit the contents within buffer-of-origin.

        * Out of buffer edits:
            · Fast when spoken language.
            · Risky when LISP.
    - While similar to Org-Macro, the `RESULT` formatting indicates to the reader that the value is the result of an evaluation.
- Spelling
    - Place `LocalWords` at the beginning of the document. That way it won't get stomped on during development.
- Synonyms.
    - Document and System and Web.
        * A Web defines a system.
        * This document is a Web.
    - Weave and Export.
    - Sysop and Reader.
- Tangling.
    - When ordering matters, rely on block-reuse to enforce correct order.
- Voice.
    - Provide answers; do not pose questions or observations.
    - Simple and detailed.
    - Pleasant conversation style.
    - Audience is Sysop; the author included.
    - Capture decisions that allow this system to move forward.
- Weaving.
    - Strive to keep the weaving in synchronization with the tangling.
- Word Choice.
    - Use Arabic numerals.
    - Instead of writing "tells EMACS", communicate the result.
    - "EMACS" refers to the EMACS software
    - "HELP" refers to the system configured by tangling this we.
    - Never describe something as "perfect" or "delightful". If it is part of this system then it is perfect and delightful.

(c) History

`ID: org_gcr_2017-06-25_mara:F917FFA6-077B-4A9A-B325-7E9A0CFF9720`

i. Version 1: Sufficient And Slipshod

`ID: org_gcr_2017-06-25_mara:2AE32991-F8DA-4205-822C-51A267EC81D6`

Emacs never grew on me the first few times I tried using it. The first was out of curiosity because a friend used `jdee`. Five or ten years later I tried using it for `clojure` and lost interest first in `clojure` and then in Emacs. The third time was for `ocaml` and `sml`. That worked pretty well but I didn't stick with `sml` so I didn't stick with Emacs either. At least five configuration files came out of this time all hand-coded Elisp and they are all either lost or hanging around in a SVN repository on a backup. I never had a sense of how Emacs worked, found any of it interesting or a problem solved by it. That is a reflection of what I did and what I brought to it. And I understand that.

ii. Version 2: How to design a file conservator (HTDFC) or The Care and Conservation of Computer Files (TC3F)

`ID: org_gcr_2017-06-25_mara:8BD93843-0D61-45FA-98F4-C3BBA0FFBB47`

The second time around, VIM got me hooked on Emacs. *Growing up* with VIM I had returned to `gvim` and I was happy with plain old text files. For keeping todo lists, meeting minutes and working in LaTeX both worked great. That process of re-falling in love with plain old text files is what struck me. Once I heard the Org-Mode tag line "your life in plain text" I knew I had to take a look. And I did and it was delightful.

And I was hooked. The line pulled me into pure exploration. Reading the manuals, blogs and other folks' configuration files was an boundless adventure. This was entirely about learning different libraries and configuration options. And it was pretty fun. It was still hand coded Elisp and probably went through ten iterations until I was happy with it.

It meant so much to me that I started naming the configuration finally ending up with TC3F.

iii. Version 3: ALEC'S A LANGUAGE FOR EXPRESSING CREATIVITY (ALEC)

`ID: org_gcr_2017-06-25_mara:1D2B647B-DAEC-42FE-97C8-777CF8869C52`

ALEC is the Cambrian Explosion of my cognitive model of Emacs. Creativity abounded like never before. The technology reflected it, enabled and grew into the finest external tool that I had ever known. The passages elaborate on the technology, the thought processes and justification and go on much further into the nature of creativity, cognition and their grand intersection.

That intersection was in the great plains, steaming jungles and snow capped mountain peaks. It was everywhere and I was everywhere. This was the point where I stopped pretending that it was written for anyone else other than me. Part of that admission was finally understanding what it means to learn about how we think and how we use that to craft, shape and hone our tools. Emacs is referred to in the singular but it is really a collection of hundreds of creative parts. The work here helped me start to narrow down what I was trying to do and to think. And I thought that I wanted to learn Literate Programming.

So I did and I tried and it was a slow and confusing document because that is how it lived in my mind. There were hundreds and hundreds of commits until I was both happy enough with it and comfortable enough to share it. This was the first time that I started to feel like I had some clue about what I was doing.

iv. Version 4: HELP Enables Literate Programming

`ID: org_gcr_2017-06-25_mara:64CA819C-71AD-4E6E-9091-4FFCA95AAF08`

HELP was more than an attempt to create a recursive acronym. It *really was helpful*. Every good idea from ALEC got simplified and moved in here. Exploration of the tool and of creative processes were still part of it. But the bigger part was learning, studying and practicing Literate Programming (LP).

Donald Knuth wrote a lot *about* LP and *wrote* a lot *in* LP. That is revealing. To get the most about of LP you need to love programming. And you need to love writing. Really you need to love writing in whatever form that takes for you. And finally you need to love doing both together. Literate programmers are few and far between simply because the combination the three desires are uncommon. Despite that it is still fun to practice LP. And Org-Mode does LP so well.

Org-Mode's LP tools are stunning and flawless. They shine so brightly that it can be difficult to take a look and see what is possible. HELP is my attempt to take the options that configure them building the perfect LP environment for me. Though HELP is still somewhat wild and unrefined the LP components are both precisely configured *and* understood. Probably only to me and I am fine with that. Teaching LP with Org-Mode would be an entirely different task!

Mark my words for they are sure to change here and probably be struck down by the idiom-zapper plugin that I've yet to write: this will be a stable version of my Emacs configuration for a long time. All of my original desires are satisfied. This fine tool, the infinite abacus, has sat in this hearth long enough. Now it is time to carry it out into the wilderness and far above and beyond the mountaintops to reveal the other side and the mysteries that they reveal.

2. Special Operating Procedure

`ID: org_gcr_2017-05-12_mara:A5B71AC0-1725-4416-AB50-86496180526A`

The following code and packages are special to this configuration. They provide critical functionality for configuring the rest of the system. They provide ideas that make the entire system usable, productive, expressive, and fast.

(a) Customize

`ID: org_gcr_2017-05-12_mara:C8E7E545-E716-4ACE-9536-B7278F843632`

Easy Customization is great. Though most of the time I move the settings into hand-coded Elisp, I still love it.

Store Customizations in another file. Avoids the package-selected-packages portability issue.

```
(setq custom-file "~/src/help/custom.el")
(load custom-file :noerror)
```

(b) Display

ID: org_gcr_2017-05-12_mara:AC7B6F6A-EB1C-49AA-89F3-0B5DAC8BD759

Make it easy to conditionally evaluate code when running with a graphical display.

```
(defmacro help/on-gui (statement &rest statements)
  "Evaluate the enclosed body only when run on GUI."
  `(when (display-graphic-p)
     ,statement
     ,@statements))

(defmacro help/not-on-gui (statement &rest statements)
  "Evaluate the enclosed body only when run on GUI."
  `(when (not (display-graphic-p))
     ,statement
     ,@statements))
```

(c) Hydra

ID: org_gcr_2017-05-12_mara:96FBC635-B614-479C-BFA4-E9AE3D70EB8B

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

(d) Keyboard

ID: org_gcr_2017-05-12_mara:63790E33-FB60-4C0E-A13E-907D5A175CEC

Key-Chord mode is amazing. Piano-Lessons shows you how.

```
(use-package key-chord
  :ensure t
  :config
  (key-chord-mode t))
```

Echo keystrokes immediately.

```
(setq echo-keystrokes 0.02)
```

(e) Libraries

ID: org_gcr_2017-05-12_mara:E1A8398D-E2F6-485B-83FB-CE52AE9A5D1B

Lists.

```
(use-package dash
  :ensure t
  :config
  (dash-enable-font-lock))
(use-package dash-functional
  :ensure t)
```

Files and directories.

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

Strings.

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

Hash-tables.

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

Caching.

```
(use-package persistent-soft
  :ensure t)
```

Namespacing without language support.

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

Child process handling specifically for Magit.

```
(use-package with-editor
  :ensure t)
```

(f) Modeline

```
ID: org_gcr_2017-05-12_mara:953129BC-045D-43B5-A194-904818E44FC7
```

Reduce information about modes in the Modeline.

```
(use-package diminish)
```

Show the file size.

```
(size-indication-mode)
```

Show the column number.

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

Configuration uses active theme.

```
(use-package smart-mode-line
  :ensure t)
```

(g) OS X

```
ID: org_gcr_2017-05-12_mara:BC848D6F-7915-4151-8368-D473CA39E7C3
```

Make it easy to evaluate code only when running on OSX.

```
(defmacro help/on-osx (statement &rest statements)
  "Evaluate the enclosed body only when run on OSX."
  `(when (eq system-type 'darwin)
     ,statement
     ,@statements))
```

Pull in the ENVIRONMENT variables because the GUI version of EMACS does not.

```
(help/on-osx
 (use-package exec-path-from-shell
   :ensure t
   :config
   (setq exec-path-from-shell-check-startup-files nil)
   (exec-path-from-shell-initialize)))
```

Configure the meta keys.

Use the OS X modifiers as Emacs meta keys. Don't pass them through to OS X.

Easily allow option pass through for alternate input methods.

```
(help/on-osx
 (setq mac-control-modifier 'control)
 (setq mac-right-control-modifier 'left)
 (setq mac-command-modifier 'meta)
 (setq mac-right-command-modifier 'left)
 (setq mac-option-modifier 'super)
 (setq mac-right-option-modifier 'left)
 (setq mac-function-modifier 'hyper)
 (defun help/toggle-mac-right-option-modifier ()
   "Toggle between passing option modifier either to Emacs or OS X."
   (interactive)
   (let ((old-ropt mac-right-option-modifier))
     (setq mac-right-option-modifier
           (if (eq mac-right-option-modifier 'left)
               'none
             'left))
     (message "Toggled `mac-right-option-modifier' from %s to %s."
              old-ropt
              mac-right-option-modifier)))
 (defun help/toggle-mac-function-modifier ()
   "Toggle between passing function modifier either to Emacs or OS X."
   (interactive)
   (let ((old-func mac-function-modifier))
     (setq mac-function-modifier
           (if (eq mac-function-modifier 'hyper)
               'none
             'hyper))
     (message "Toggled `mac-function-modifier' from %s to %s."
              old-func
              mac-function-modifier))))
```

EMACS dialogues don't work OSX. They lock up EMACS.

This is a known issue. Here is the solution.

```
(help/on-osx
 (defun help/yes-or-no-p (orig-fun &rest args)
```

```
  "Prevent yes-or-no-p from activating a dialog."
  (let ((use-dialog-box nil))
    (apply orig-fun args)))
(advice-add #'yes-or-no-p :around #'help/yes-or-no-p)
(advice-add #'y-or-n-p :around #'help/yes-or-no-p))
```

(h) Windows

ID: org_gcr_2017-05-12_mara:F59CD0F9-9B04-493E-AA9A-2471F2ED0A05

Make it easy to evaluate code only when running on Windows.

```
(defmacro help/on-windows (statement &rest statements)
  "Evaluate the enclosed body only when run on Microsoft Windows."
  `(when (eq system-type 'windows-nt)
     ,statement
     ,@statements))
```

Provide the proper shell.

```
(help/on-windows
 (setq shell-file-name "cmdproxy.exe"))
```

Enable the super key-space.

```
(help/on-windows
 (setq w32-pass-lwindow-to-system nil)
 (defvar w32-lwindow-modifier 'super)
 (setq w32-pass-rwindow-to-system nil)
 (defvar w32-rwindow-modifier 'super))
```

3. Standard Operating Procedure

ID: org_gcr_2017-05-12_mara:205571BA-2DC4-4DCB-912C-65B9EC843574

Configure EMACS to maximum utility.

(a) Helper Functions

ID: org_gcr_2017-05-12_mara:1E88DD48-B992-46E1-B49F-EB8071E8EC37

```
(defun help/comment-or-uncomment ()
  "Comment or uncomment the current line or selection."
  (interactive)
  (cond ((not mark-active) (comment-or-uncomment-region (line-beginning-position)
                                                        (line-end-position)))
        ((< (point) (mark)) (comment-or-uncomment-region (point) (mark)))
        (t (comment-or-uncomment-region (mark) (point)))))

(defun help/save-all-file-buffers ()
  "Saves every buffer associated with a file."
  (interactive)
  (dolist (buf (buffer-list))
    (with-current-buffer buf
      (when (and (buffer-file-name) (buffer-modified-p))
```

```
        (save-buffer)))))

(defun describe-thing-in-popup ()
  "Attribution: URL ‘http://blog.jenkster.com/2013/12/popup-help-in-emacs-lisp.html’."
  (interactive)
  (let* ((thing (symbol-at-point))
         (help-xref-following t)
         (description (with-temp-buffer
                        (help-mode)
                        (help-xref-interned thing)
                        (buffer-string))))
    (popup-tip description
               :point (point)
               :around t
               :height 30
               :scroll-bar t
               :margin t)))

(defun help/kill-other-buffers ()
  "Kill all other buffers."
  (interactive)
  (mapc #'kill-buffer (delq (current-buffer) (buffer-list))))

(defvar help/delete-trailing-whitespace-p t
  "Should trailing whitespace be removed?")

(defun help/delete-trailing-whitespace ()
  "Delete trailing whitespace for everything but the current line.

If ‘help/delete-trailing-whitespace-p’ is non-nil, then delete the whitespace.
This is useful for fringe cases where trailing whitespace is important."
  (interactive)
  (when help/delete-trailing-whitespace-p
    (let ((first-part-start (point-min))
          (first-part-end (point-at-bol))
          (second-part-start (point-at-eol))
          (second-part-end (point-max)))
      (delete-trailing-whitespace first-part-start first-part-end)
      (delete-trailing-whitespace second-part-start second-part-end))))

(defun help/insert-timestamp ()
  "Produces and inserts a full ISO 8601 format timestamp."
  (interactive)
  (insert (format-time-string "%Y-%m-%dT%T%z")))

(defun help/insert-timestamp* ()
  "Produces and inserts a near-full ISO 8601 format timestamp."
  (interactive)
```

```
  (insert (format-time-string "%Y-%m-%dT%T")))

(defun help/insert-timestamp*-no-colons ()
  "Produces and inserts a near-full ISO 8601-like format timestamp."
  (interactive)
  (let* ((str (format-time-string "%Y-%m-%dT%T"))
         (fnl (s-replace ":" "-" str)))
    (insert fnl)))

(defun help/insert-datestamp ()
  "Produces and inserts a partial ISO 8601 format timestamp."
  (interactive)
  (insert (format-time-string "%Y-%m-%d")))

(defun help/indent-curly-block (&rest _ignored)
  "Open a new brace or bracket expression, with relevant newlines and indent. URL: ‘htt
  (interactive)
  (newline)
  (indent-according-to-mode)
  (forward-line -1)
  (indent-according-to-mode))

(defun beginning-of-line-dwim ()
  "Toggles between moving point to the first non-whitespace character, and
    the start of the line. Src: http://www.wilfred.me.uk/"
  (interactive)
  (let ((start-position (point)))
    ;; see if going to the beginning of the line changes our position
    (move-beginning-of-line nil)

    (when (= (point) start-position)
      ;; we're already at the beginning of the line, so go to the
      ;; first non-whitespace character
      (back-to-indentation))))

(defun help/lazy-new-open-line ()
  "Insert a new line without breaking the current line."
  (interactive)
  (beginning-of-line)
  (forward-line 1)
  (newline)
  (forward-line -1))

(defun help/smart-open-line ()
  "Insert a new line, indent it, and move the cursor there.

This behavior is different then the typical function bound to return
which may be ‘open-line’ or ‘newline-and-indent’. When you call with
```

```
the cursor between ^ and $, the contents of the line to the right of
it will be moved to the newly inserted line. This function will not
do that. The current line is left alone, a new line is inserted, indented,
and the cursor is moved there.

Attribution: URL ‘http://emacsredux.com/blog/2013/03/26/smarter-open-line/’"
  (interactive)
  (move-end-of-line nil)
  (newline-and-indent))

(defun help/insert-ellipsis ()
  "Insert an ellipsis into the current buffer."
  (interactive)
  (insert "\dots{}"))

(defun help/insert-checkmark ()
  "Insert a checkmark into the current buffer."
  (interactive)
  (insert "\checkmark{}"))

(defun help/insert-noticeable-snip-comment-line ()
  "Insert a noticeable snip comment line (NSCL)."
  (interactive)
  (if (not (bolp))
      (message "I may only insert a NSCL at the beginning of a line.")
    (let ((ncl (make-string 70 ?)))
      (newline)
      (forward-line -1)
      (insert ncl)
      (comment-or-uncomment-region (line-beginning-position) (line-end-position)))))

(progn

  (defvar my-read-expression-map
    (let ((map (make-sparse-keymap)))
      (set-keymap-parent map read-expression-map)
      (define-key map [(control ?g)] #'minibuffer-keyboard-quit)
      (define-key map [up]   nil)
      (define-key map [down] nil)
      map))

  (defun my-read--expression (prompt &optional initial-contents)
    (let ((minibuffer-completing-symbol t))
      (minibuffer-with-setup-hook
          (lambda ()
            (emacs-lisp-mode)
            (use-local-map my-read-expression-map)
            (setq font-lock-mode t)
```

```
            (funcall font-lock-function 1))
          (read-from-minibuffer prompt initial-contents
                                my-read-expression-map nil
                                'read-expression-history))))

  (defun my-eval-expression (expression &optional arg)
    "Attribution: URL `https://lists.gnu.org/archive/html/help-gnu-emacs/2014-07/msg001
    (interactive (list (read (my-read--expression ""))
                       current-prefix-arg))
    (if arg
        (insert (pp-to-string (eval expression lexical-binding)))
      (pp-display-expression (eval expression lexical-binding)
                             "*Pp Eval Output*"))))

(defun help/util-ielm ()
  "HELP buffer setup for ielm.

Creates enough space for one other permanent buffer beneath it."
  (interactive)
  (split-window-below -20)
  (help/safb-other-window)
  (ielm)
  (set-window-dedicated-p (selected-window) t))

(defun help/util-eshell ()
  "HELP buffer setup for eshell.

Depends upon `help/util-ielm' being run first."
  (interactive)
  (split-window-below -10)
  (help/safb-other-window)
  (eshell)
  (set-window-dedicated-p (selected-window) t))

(defvar help/util-state nil "Track whether the util buffers are displayed or not.")

(defun help/util-state-toggle ()
  "Toggle the util state."
  (interactive)
  (setq help/util-state (not help/util-state)))

(defun help/util-start ()
  "Perhaps utility buffers."
  (interactive)
  (help/util-ielm)
  (help/util-eshell)
  (help/util-state-toggle))
```

```
(defun help/util-stop ()
  "Remove personal utility buffers."
  (interactive)
  (if (get-buffer "*ielm*") (kill-buffer "*ielm*"))
  (if (get-buffer "*eshell*") (kill-buffer "*eshell*"))
  (help/util-state-toggle))

(defun help/ielm-auto-complete ()
  "Enables ‘auto-complete’ support in \\[ielm].

Attribution: URL ‘http://www.masteringemacs.org/articles/2010/11/29/evaluating-elisp-em
  (setq ac-sources ’(ac-source-functions
                     ac-source-variables
                     ac-source-features
                     ac-source-symbols
                     ac-source-words-in-same-mode-buffers))
  (add-to-list ’ac-modes #’inferior-emacs-lisp-mode)
  (auto-complete-mode 1))

(defun help/uuid ()
  "Insert a UUID."
  (interactive)
  (let ((org-id-prefix nil))
    (insert (org-id-new))))

(defun endless/sharp ()
  "Insert #’ unless in a string or comment.

SRC: URL ‘http://endlessparentheses.com/get-in-the-habit-of-using-sharp-quote.html?sour
  (interactive)
  (call-interactively #’self-insert-command)
  (let ((ppss (syntax-ppss)))
    (unless (or (elt ppss 3)
                (elt ppss 4))
      (insert "’"))))

(defun help/chs ()
  "Insert opening \"cut here start\" snippet."
  (interactive)
  (insert "--8<---------------cut here---------------start------------->8---"))

(defun help/che ()
  "Insert closing \"cut here end\" snippet."
  (interactive)
  (insert "--8<---------------cut here---------------end--------------->8---"))

(defmacro help/measure-time (&rest body)
  "Measure the time it takes to evaluate BODY.
```

```
Attribution Nikolaj Schumacher: URL ‘https://lists.gnu.org/archive/html/help-gnu-emacs/
  ‘(let ((time (current-time)))
     ,@body
     (message "%.06f" (float-time (time-since time)))))

(defun help/create-non-existent-directory ()
  "Attribution URL: ‘https://iqbalansari.github.io/blog/2014/12/07/automatically-create
  (let ((parent-directory (file-name-directory buffer-file-name)))
    (when (and (not (file-exists-p parent-directory))
               (y-or-n-p (format "Directory ‘%s’ does not exist. Create it?" parent-dir
      (make-directory parent-directory t))))

(defun help/occur-dwim ()
  "Call ‘occur’ with a mostly sane default.

Attribution Oleh Krehel (abo-abo): URL ‘http://oremacs.com/2015/01/26/occur-dwim/’"
  (interactive)
  (push (if (region-active-p)
            (buffer-substring-no-properties
             (region-beginning)
             (region-end))
          (let ((sym (thing-at-point 'symbol)))
            (when (stringp sym)
              (regexp-quote sym))))
        regexp-history)
  (call-interactively 'occur)
  (other-window 1))

(defun help/util-cycle ()
  "Display or hide the utility buffers."
  (interactive)
  (if help/util-state
      (help/util-stop)
    (help/util-start)))

(defun sacha/unfill-paragraph (&optional region)
  "Takes a multi-line paragraph and makes it into a single line of text.

Attribuation: URL https://github.com/sachac/.emacs.d/blob/gh-pages/Sacha.org#unfill-par
  (interactive (progn
                 (barf-if-buffer-read-only)
                 (list t)))
  (let ((fill-column (point-max)))
    (fill-paragraph nil region)))
(defun help/text-scale-increase ()
  "Increase font size"
  (interactive)
```

```
  (help/on-gui
   (setq help/font-size-current (+ help/font-size-current 1))
   (help/update-font))
  (help/not-on-gui
   (message "Please resize the terminal emulator font.")))
(defun help/text-scale-decrease ()
  "Reduce font size."
  (interactive)
  (help/on-gui
   (when (> help/font-size-current 1)
     (setq help/font-size-current (- help/font-size-current 1))
     (help/update-font)))
  (help/not-on-gui
   (message "Please resize the terminal emulator font.")))

(defun help/org-weave-subtree-gfm (id file)
  "Export the subtree with ID to FILE in gfm."
  (interactive)
  (help/save-all-file-buffers)
  (save-excursion
    (let ((hidx (org-find-property "ID" id)))
      (when hidx
        (goto-char hidx)
        (org-export-to-file 'gfm file nil t nil)))))

(defun help/org-weave-gfm (id)
  "Select an ID to export to the same name as Github Flavored Markdown.."
  (interactive "sEnter the ID to export: ")
  (help/org-weave-subtree-gfm id (s-prepend id ".md")))

(defun help/org-weave-readme ()
  (interactive)
  (help/org-weave-subtree-gfm
   "orgmode:gcr:vela:README"
   "README.md"))

(defun help/org-weave-style-guide ()
  (interactive)
  (help/org-weave-subtree-gfm
   "orgmode:gcr:vela:STYLEGUIDE"
   "STYLEGUIDE.md"))

(defun help/weave-everything-everywhere ()
  "Export this entire document in configured weavers."
  (interactive)
  (save-excursion
    (org-ascii-export-to-ascii)
    (org-html-export-to-html)
```

```
    (org-gfm-export-to-markdown)
    (org-latex-export-to-pdf))
  (help/org-weave-readme)
  (help/org-weave-style-guide))

(require 'thingatpt)

(defun thing-at-point-goto-end-of-integer ()
  "Go to end of integer at point.

Attribution: URL `http://emacsredux.com/blog/2013/07/25/increment-and-decrement-integer
  (let ((inhibit-changing-match-data t))
    ;; Skip over optional sign
    (when (looking-at "[+-]")
      (forward-char 1))
    ;; Skip over digits
    (skip-chars-forward "[[:digit:]]")
    ;; Check for at least one digit
    (unless (looking-back "[[:digit:]]")
      (error "No integer here"))))
(put 'integer 'beginning-op 'thing-at-point-goto-end-of-integer)

(defun thing-at-point-goto-beginning-of-integer ()
  "Go to end of integer at point.

Attribution: URL `http://emacsredux.com/blog/2013/07/25/increment-and-decrement-integer
  (let ((inhibit-changing-match-data t))
    ;; Skip backward over digits
    (skip-chars-backward "[[:digit:]]")
    ;; Check for digits and optional sign
    (unless (looking-at "[+-]?[[:digit:]]")
      (error "No integer here"))
    ;; Skip backward over optional sign
    (when (looking-back "[+-]")
      (backward-char 1))))
(put 'integer 'beginning-op 'thing-at-point-goto-beginning-of-integer)

(defun thing-at-point-bounds-of-integer-at-point ()
  "Get boundaries of integer at point.

Attribution: URL `http://emacsredux.com/blog/2013/07/25/increment-and-decrement-integer
  (save-excursion
    (let (beg end)
      (thing-at-point-goto-beginning-of-integer)
      (setq beg (point))
      (thing-at-point-goto-end-of-integer)
      (setq end (point))
      (cons beg end))))
```

```
(put 'integer 'bounds-of-thing-at-point 'thing-at-point-bounds-of-integer-at-point)

(defun thing-at-point-integer-at-point ()
  "Get integer at point.

Attribution: URL `http://emacsredux.com/blog/2013/07/25/increment-and-decrement-integer
  (let ((bounds (bounds-of-thing-at-point 'integer)))
    (string-to-number (buffer-substring (car bounds) (cdr bounds)))))
(put 'integer 'thing-at-point 'thing-at-point-integer-at-point)

(defun increment-integer-at-point (&optional inc)
  "Increment integer at point by one.

With numeric prefix arg INC, increment the integer by INC amount.

Attribution: URL `http://emacsredux.com/blog/2013/07/25/increment-and-decrement-integer
  (interactive "p")
  (let ((inc (or inc 1))
        (n (thing-at-point 'integer))
        (bounds (bounds-of-thing-at-point 'integer)))
    (delete-region (car bounds) (cdr bounds))
    (insert (int-to-string (+ n inc)))))

(defun decrement-integer-at-point (&optional dec)
  "Decrement integer at point by one.

With numeric prefix arg DEC, decrement the integer by DEC amount.

Attribution: URL `http://emacsredux.com/blog/2013/07/25/increment-and-decrement-integer
  (interactive "p")
  (increment-integer-at-point (- (or dec 1))))

(defun help/reformat-file (file)
  "Reformat a file. If it is attached to an existing buffer then use it and
  revert it.

Attribution: URL `https://www.emacswiki.org/emacs/ElispCookbook#toc46'."
  (interactive)
  (with-current-buffer (find-file-noselect file)
    (revert-buffer t t)
    (with-temp-message "Formatting file..."
      (indent-region (point-min) (point-max) nil))
    (message "Formatting file done")))

(defun switch-to-previous-buffer ()
  "Switch to most recent buffer. Repeated calls toggle back and forth between the most

Attribution: URL `http://pragmaticemacs.com/emacs/toggle-between-most-recent-buffers/'
```

```
Attribution: URL ‘https://www.emacswiki.org/emacs/SwitchingBuffers#toc5’"
  (interactive)
  (switch-to-buffer (other-buffer (current-buffer) 1)))

(defun help/dos2unix ()
  "Not exactly but it’s easier to remember.

Attribution: URL ‘https://www.emacswiki.org/emacs/DosToUnix’"
  (interactive)
  (set-buffer-file-coding-system ’unix ’t) )

(defun help/preview-buffer-file-in-marked-2 ()
  "View buffer file in Marked 2.

Attribution: URL
‘https://github.com/kotfu/marked-bonus-pack/blob/master/Emacs/dot.emacs.txt’"
  (interactive)
  (help/on-osx
   (shell-command
    (format "open -a ’Marked 2.app’ %s"
            (shell-quote-argument (buffer-file-name))))))

(defun help/safb-flycheck-list-errors ()
  "Save all file buffers and switch to flycheck error list"
  (interactive)
  (help/save-all-file-buffers)
  (flycheck-list-errors)
  (other-window 1))

(defun help/safb-kill-this-buffer ()
  "Save all file buffers and maybe kill this buffer."
  (interactive)
  (help/save-all-file-buffers)
  (kill-this-buffer))

(defmacro help/profile-org (times &rest body)
  "Makes profiling Org-Mode easy by automatically instrumenting the desired
  functions, running the code you want to test, removing the instrumentation,
  and presenting the results.

Attribution: Adam Porter <adam@alphapapa.net>"
  ‘(let (output)
     (dolist (p ’("org-"))  ; symbol prefixes to instrument
       (elp-instrument-package p))
     (dotimes (x ,times)
       ,@body)
     (elp-results)
```

```
    (elp-restore-all)
    (point-min)
    (forward-line 20)
    (delete-region (point) (point-max))
    (setq output (buffer-substring-no-properties (point-min) (point-max)))
    (kill-buffer)
    (delete-window)
    output))

(defun help/open-help ()
  "Switch to the buffer backed by ‘help/help.org’."
  (interactive)
  (if (get-buffer "help/help.org")
      (switch-to-buffer "help/help.org")))

(defun help/open-projects ()
  "Switch to the buffer backed by ‘bitbucket/projects.org’."
  (interactive)
  (if (get-buffer "bitbucket/projects.org")
      (switch-to-buffer "bitbucket/projects.org")))

(defun help/open-si-projects ()
  "Switch to the buffer backed by ‘bitbucket-gcrstoneisle/projects.org’."
  (interactive)
  (if (get-buffer "bitbucket-gcrstoneisle/projects.org")
      (switch-to-buffer "bitbucket-gcrstoneisle/projects.org")))

(defun help/move-file (new-location)
  "Write this file to NEW-LOCATION, and delete the old one.

Attribution: URL ‘http://zck.me/emacs-move-file’"
  (interactive (list (if buffer-file-name
                         (read-file-name "Move file to: ")
                       (read-file-name "Move file to: "
                                       default-directory
                                       (expand-file-name (file-name-nondirectory (buffe
                                                         default-directory)))))
  (when (file-exists-p new-location)
    (delete-file new-location))
  (let ((old-location (buffer-file-name)))
    (write-file new-location t)
    (when (and old-location
               (file-exists-p new-location)
               (not (string-equal old-location new-location)))
      (delete-file old-location))))

(defun help/rename-current-buffer-file ()
  "Renames current buffer and file it is visiting.
```

```
Attribution: ‘http://stackoverflow.com/a/25212377’"
  (interactive)
  (let ((name (buffer-name))
        (filename (buffer-file-name)))
    (if (not (and filename (file-exists-p filename)))
        (error "Buffer ’%s’ is not visiting a file!" name)
      (let ((new-name (read-file-name "New name: " filename)))
        (if (get-buffer new-name)
            (error "A buffer named ’%s’ already exists!" new-name)
          (rename-file filename new-name 1)
          (rename-buffer new-name)
          (set-visited-file-name new-name)
          (set-buffer-modified-p nil)
          (message "File ’%s’ successfully renamed to ’%s’"
                   name (file-name-nondirectory new-name)))))))


(defun help/sort-lines-nocase ()
  "Sort lines ignoring case.

Attribution: ‘https://stackoverflow.com/questions/20967818/emacs-function-to-case-insen
  (interactive)
  (let ((sort-fold-case t))
    (call-interactively ’sort-lines)))


(defun help/delete-this-buffer-and-file ()
  "Deletes file connected to this buffer and kills this buffer.

Attribution: URL ‘https://rejeep.github.io/emacs/elisp/2010/11/16/delete-file-and-buffe
  (interactive)
  (let ((filename (buffer-file-name))
        (buffer (current-buffer))
        (name (buffer-name)))
    (if (not (and filename (file-exists-p filename)))
        (error "Nothing to delete: ’%s’ is not visiting a file." name)
      (when (yes-or-no-p "Are you sure you want to delete this file? ")
        (delete-file filename)
        (kill-buffer buffer)
        (message "File ’%s’ successfully deleted." filename)))))

(defun help/wih ()
  (interactive)
  (when (use-region-p) (call-interactively ’kill-region))
  (insert "#+CATEGORY: Article
#+TAGS: Yoga, philosophy, Health, Happiness,
#+TITLE:"))
```

```
(defun help/wio ()
  (interactive)
  (when (use-region-p) (call-interactively 'kill-region))
  (insert "#+CATEGORY: Article
#+TAGS: Babel, Emacs, Ide, Lisp, Literate Programming, Programming Language, Reproducib
#+TITLE:"))
```

(b) Typography

```
ID: org_gcr_2017-05-12_mara:A5224E7E-EBAA-4970-951F-7405F04D4A26
```

- Use 78 characters for a text document
  - Column 0 is the first possible character
  - Column 77 is the last possible character
  - Column 78 will always be empty
    - This is the fill column
    - This gives some spacing between the text body and the 80 column indicator
  - Column 79 will always be the fill column indicator
    - It **isn't** the fill column though
    - I want it to indicate 80 chars, typically the maximum number of columns for a line, to know how to size the window itself
    - Fci-Mode supports this
  - Store this as the fill column because all supporting functions will do the right thing here

```
(defconst help/column-width 78)
(setq-default fill-column help/column-width)
```

Two spaces follow a colon

Two spaces after a semi-colon.

One space after comma.

```
(setq colon-double-space nil)
```

(c) Buffer

```
ID: org_gcr_2017-05-12_mara:D4FC68F9-CEA3-455A-B718-EBDB113CFA71
```

Maintain buffers across sessions. Desktop-Save-Mode persists every part of the buffer. If you upgrade a package that uses buffer-variables that have changed you may get unexpected behavior. Close all buffers and open them again after making such breaking changes.

```
(desktop-save-mode t)
(setq desktop-restore-eager 10)
```

Provide expected “Undo” functionality.

```
(use-package undo-tree
  :ensure t
  :config
  (global-undo-tree-mode 1)
  (global-set-key (kbd "C-M-u") #'undo-tree-visualize)
  :diminish undo-tree-mode)
```

Ever file must end with a newline.

```
(setq require-final-newline t)
```

If you are on the end of a line, and go up or down, then go to the end of line on that new line. Do not account for anything special about the character there.

```
(setq track-eol t)
(setq line-move-visual nil)
```

Take the cursor with scroll activities.

```
(setq scroll-preserve-screen-position t)
```

Scroll text line-by-line as the cursor scrolls off of the screen. Nice to keep the document from jumping around. Not nice for scrolling when you want to quickly see the next ten or twenty lines. But `recenter-top-bottom` give you that so you can have both features easily.

```
(setq scroll-conservatively 101)
```

More easily visualize tabular data. Considered to non-subjective.

```
(use-package stripe-buffer
  :ensure t)
```

End sentences with a single space.

```
(setq sentence-end-double-space nil)
```

Ban white-space at end of lines, globally.

```
(add-hook 'before-save-hook #'help/delete-trailing-whitespace)
```

Intelligently select the current char, then word, then object, then block, then document.

```
(use-package expand-region
  :ensure t)
```

Configure Page-Break-Lines-Mode to visualize the formfeed character: `C-q C-l`.

```
(use-package page-break-lines
  :diminish page-break-lines-mode
  :config
  (defun help/insert-formfeed ()
    (interactive)
    (insert "\f"))
  (global-set-key (kbd "C-M-<return>") #'help/insert-formfeed)
```

```
  (global-set-key (kbd "C-M-j") #'forward-page)
  (global-set-key (kbd "C-M-k") #'backward-page)
  (global-set-key (kbd "C-M-n") #'narrow-to-page)
  (global-set-key (kbd "C-M-h") #'mark-page))
```

Center the buffer after navigating pages.

```
(advice-add #'backward-page :after #'recenter)
(advice-add #'forward-page :after #'recenter)
```

Do the *right thing* for getting to the start of the line.

```
(use-package mwim
  :ensure t
  :config
  (global-set-key (kbd "C-a") 'mwim-beginning-of-code-or-line))
```

Fling buffers left and right.

```
(use-package buffer-move
  :ensure t
  :config
  (define-key org-mode-map (kbd "<C-S-up>") nil)
  (global-set-key (kbd "<C-S-up>") #'buf-move-up)
  (define-key org-mode-map (kbd "<C-S-down>") nil)
  (global-set-key (kbd "<C-S-down>") #'buf-move-down)
  (define-key org-mode-map (kbd "<C-S-left>") nil)
  (global-set-key (kbd "<C-S-left>") #'buf-move-left)
  (define-key org-mode-map (kbd "<C-S-right>") nil)
  (global-set-key (kbd "<C-S-right>") #'buf-move-right))
```

- Easily see the fill-column (or close too it)
  - Sometimes I set the fci rule at 81 because a char at 79 pushes the fci rule out one extra space. Sometimes it is not an issue.

```
(use-package fill-column-indicator
  :ensure t
  :config
  (setq fci-rule-column 79))
```

Show text indentation guide lines.

```
(use-package indent-guide
  :ensure t)
```

Management:

- `http://martinowen.net/blog/2010/02/03/tips-for-emacs-ibuffer.html`
- `https://mytechrants.wordpress.com/2010/03/25/emacs-tip-of-the-day-start-using-ibuf`
- Group approach
  - See the world organized by?

        * File names
        * Buffer names
        * Mode types
        * Groups of files
        * Projectile project idea
            · Top up to .git dir
        * Git working tree
            · This is how I see the world

- Hydra Helper

```
(defhydra hydra-ibuffer-main (:color pink :hint nil)
  "
   ^Navigation^ | ^Mark^        | ^Actions^        | ^View^
  -^-----------^-+-^----^--------+-^-------^--------+-^----^-------
    _k_:       | _m_: mark     | _D_: delete      | _g_: refresh
   _RET_: visit | _u_: unmark   | _S_: save        | _s_: sort
    _j_:    v   | _*_: specific | _a_: all actions | _/_: filter
  -^-----------^-+-^----^--------+-^-------^--------+-^----^-------
  "
  ("j" ibuffer-forward-line)
  ("RET" ibuffer-visit-buffer :color blue)
  ("k" ibuffer-backward-line)

  ("m" ibuffer-mark-forward)
  ("u" ibuffer-unmark-forward)
  ("*" hydra-ibuffer-mark/body :color blue)

  ("D" ibuffer-do-delete)
  ("S" ibuffer-do-save)
  ("a" hydra-ibuffer-action/body :color blue)

  ("g" ibuffer-update)
  ("s" hydra-ibuffer-sort/body :color blue)
  ("/" hydra-ibuffer-filter/body :color blue)

  ("o" ibuffer-visit-buffer-other-window "other window" :color blue)
  ("q" ibuffer-quit "quit ibuffer" :color blue)
  ("." nil "toggle hydra" :color blue))

(defhydra hydra-ibuffer-mark (:color teal :columns 5
                                     :after-exit (hydra-ibuffer-main/body))
  "Mark"
  ("*" ibuffer-unmark-all "unmark all")
  ("M" ibuffer-mark-by-mode "mode")
```

```
  ("m" ibuffer-mark-modified-buffers "modified")
  ("u" ibuffer-mark-unsaved-buffers "unsaved")
  ("s" ibuffer-mark-special-buffers "special")
  ("r" ibuffer-mark-read-only-buffers "read-only")
  ("/" ibuffer-mark-dired-buffers "dired")
  ("e" ibuffer-mark-dissociated-buffers "dissociated")
  ("h" ibuffer-mark-help-buffers "help")
  ("z" ibuffer-mark-compressed-file-buffers "compressed")
  ("b" hydra-ibuffer-main/body "back" :color blue))

(defhydra hydra-ibuffer-action (:color teal :columns 4
                                       :after-exit
                                       (if (eq major-mode 'ibuffer-mode)
                                           (hydra-ibuffer-main/body)))
  "Action"
  ("A" ibuffer-do-view "view")
  ("E" ibuffer-do-eval "eval")
  ("F" ibuffer-do-shell-command-file "shell-command-file")
  ("I" ibuffer-do-query-replace-regexp "query-replace-regexp")
  ("H" ibuffer-do-view-other-frame "view-other-frame")
  ("N" ibuffer-do-shell-command-pipe-replace "shell-cmd-pipe-replace")
  ("M" ibuffer-do-toggle-modified "toggle-modified")
  ("O" ibuffer-do-occur "occur")
  ("P" ibuffer-do-print "print")
  ("Q" ibuffer-do-query-replace "query-replace")
  ("R" ibuffer-do-rename-uniquely "rename-uniquely")
  ("T" ibuffer-do-toggle-read-only "toggle-read-only")
  ("U" ibuffer-do-replace-regexp "replace-regexp")
  ("V" ibuffer-do-revert "revert")
  ("W" ibuffer-do-view-and-eval "view-and-eval")
  ("X" ibuffer-do-shell-command-pipe "shell-command-pipe")
  ("b" nil "back"))

(defhydra hydra-ibuffer-sort (:color amaranth :columns 3)
  "Sort"
  ("i" ibuffer-invert-sorting "invert")
  ("a" ibuffer-do-sort-by-alphabetic "alphabetic")
  ("v" ibuffer-do-sort-by-recency "recently used")
  ("s" ibuffer-do-sort-by-size "size")
  ("f" ibuffer-do-sort-by-filename/process "filename")
  ("m" ibuffer-do-sort-by-major-mode "mode")
  ("b" hydra-ibuffer-main/body "back" :color blue))

(defhydra hydra-ibuffer-filter (:color amaranth :columns 4)
  "Filter"
  ("m" ibuffer-filter-by-used-mode "mode")
  ("M" ibuffer-filter-by-derived-mode "derived mode")
  ("n" ibuffer-filter-by-name "name")
```

```
  ("c" ibuffer-filter-by-content "content")
  ("e" ibuffer-filter-by-predicate "predicate")
  ("f" ibuffer-filter-by-filename "filename")
  (">" ibuffer-filter-by-size-gt "size")
  ("<" ibuffer-filter-by-size-lt "size")
  ("/" ibuffer-filter-disable "disable")
  ("b" hydra-ibuffer-main/body "back" :color blue))
(use-package ibuffer
  :config
  (define-key ibuffer-mode-map "." #'hydra-ibuffer-main/body))
(use-package ibuffer-vc
  :ensure t)
(defun help/ibuffer-hook-fn ()
  "HELP customizations."
  (interactive)
  (setq ibuffer-expert t)
  (setq ibuffer-show-empty-filter-groups nil)
  (ibuffer-auto-mode t)
  (stripe-buffer-mode)
  (ibuffer-vc-set-filter-groups-by-vc-root)
  (unless (eq ibuffer-sorting-mode 'alphabetic)
    (ibuffer-do-sort-by-alphabetic)))
(add-hook 'ibuffer-mode-hooks #'help/ibuffer-hook-fn)
```

This package for GNU APL.

```
(use-package face-remap
  :diminish 'buffer-face-mode)
```

(d) Code Folding

```
ID: org_gcr_2017-05-12_mara:94245A47-6A25-442A-B458-AD00A3DDA1A2
```

```
(use-package hideshow
  :config
  (setq hs-hide-comments-when-hiding-all t)
  (setq hs-isearch-open t)
  (defun display-code-line-counts (ov)
    "Displaying overlay content in echo area or tooltip"
    (when (eq 'code (overlay-get ov 'hs))
      (overlay-put ov 'help-echo
                   (buffer-substring (overlay-start ov)
                                     (overlay-end ov)))))
  (setq hs-set-up-overlay #'display-code-line-counts)
  (defun help/goto-line ()
    "How do I get it to expand upon a goto-line? hideshow-expand affected block
    when using goto-line in a collapsed buffer."
    (call-interactively #'goto-line)
    (save-excursion
      (hs-show-block)))
```

```
(defvar help/my-hs-hide nil "Current state of hideshow for toggling all.")
(defun help/my-toggle-hideshow-all () "Toggle hideshow all."
       (interactive)
       (setq help/my-hs-hide (not help/my-hs-hide))
       (if help/my-hs-hide
           (hs-hide-all)
         (hs-show-all)))
:diminish hs-minor-mode)
```

(e) Colors

ID: org_gcr_2017-05-12_mara:DA9E0CC6-F3D4-4B9B-944C-0ECB69D8899C

Colorize color names.

Rainbow-Mode handles most major modes color definitions as expected.

```
(use-package rainbow-mode
  :ensure t
  :config
  :diminish rainbow-mode)
```

(f) Debugging

ID: org_gcr_2017-05-12_mara:C54D8CF3-E2C4-4A7A-A92C-5252826EFD01

Sometimes the judicious use of Git and git bisect can obviate the need for manual bisections. Othertimes not. For the latter, use `elisp-bug-hunter`.

```
(use-package bug-hunter
  :ensure t)
```

(g) Emacs Lisp Macros

ID: org_gcr_2017-05-12_mara:6D4A22CE-EC84-4245-AD7E-2CCC0B35CBCA

Macro expander:

- `e`, `=`, `RET`
  - expand the macro form following point one step
- `c`, `u`, `DEL`
  - collapse the form following point
- `q`, `C-c C-c`
  - collapse all expanded forms and exit macrostep-mode
- `n`, `TAB`
  - jump to the next macro form in the expansion
- `p`, `M-TAB`
  - jump to the previous macro form in the expansion

```
(use-package macrostep
  :ensure t
  :config
  (global-set-key (kbd "C-M-8") #'macrostep-expand))
```

(h) Evaluation

```
ID: org_gcr_2017-05-12_mara:C88E8819-35D2-40E0-BBAD-3AC799DA0A33
```

```
(setq-default eval-expression-print-level nil)
```

Allow most commands.

```
(put #'upcase-region 'disabled nil)
(put #'downcase-region 'disabled nil)
(put #'narrow-to-region 'disabled nil)
```

Send the current line to the REPL, evaluate it and move to the next line. Works for lots of languages and does the *right thing* navigating to the next line.

If you learned this in ESS then you already love it. If you didn't then you probably will now

```
(use-package eval-in-repl
  :ensure t
  :config
  (setq eir-jump-after-eval nil)
  (setq eir-always-split-script-window t)
  (setq eir-delete-other-windows t)
  (setq eir-repl-placement 'right)
  ;; ielm support (for emacs lisp)
  (require 'eval-in-repl-ielm)
  ;; for .el files
  (define-key emacs-lisp-mode-map (kbd "<C-return>") 'eir-eval-in-ielm)
  ;; for *scratch*
  (define-key lisp-interaction-mode-map (kbd "<C-return>") 'eir-eval-in-ielm)
  ;; for M-x info
  (eval-after-load "info"
    '(define-key Info-mode-map (kbd "<C-return>") 'eir-eval-in-ielm))
  ;; Shell support
  (require 'eval-in-repl-shell)
  (add-hook 'sh-mode-hook
            '(lambda()
               (local-set-key (kbd "C-<return>") 'eir-eval-in-shell)))
  ;; Version with opposite behavior to eir-jump-after-eval configuration
  (defun eir-eval-in-shell2 ()
    "eval-in-repl for shell script (opposite behavior)

This version has the opposite behavior to the eir-jump-after-eval
configuration when invoked to evaluate a line."
    (interactive)
    (let ((eir-jump-after-eval (not eir-jump-after-eval)))
```

```
    (eir-eval-in-shell)))
(add-hook 'sh-mode-hook
          '(lambda()
             (local-set-key (kbd "C-M-<return>") 'eir-eval-in-shell2)))
;; racket-mode support (for Racket; if not using Geiser)
(require 'racket-mode) ; if not done elsewhere
(require 'eval-in-repl-racket)
(define-key racket-mode-map (kbd "<C-return>") 'eir-eval-in-racket))
```

(i) Encryption

`ID: org_gcr_2017-05-12_mara:DF56C626-A2CD-4011-9B2E-9863E72F0563`

Easy to use file-based AES encryption.

```
(add-to-list 'load-path (getenv "CCRYPT"))
(use-package ps-ccrypt)
```

(j) Eshell

`ID: org_gcr_2017-05-12_mara:5FED47A4-C1F3-4866-AF87-86996BC999CA`

Provide a cross-platform command line shell that is a first-class EMACS citizen.

Commands input in eshell are delegated in order to an alias, a built in command, an Elisp function with the same name, and finally to a system call. Semicolons separate commands. `which` tells you what implementation will satisfy the call that you are going to make. The flag `eshell-prefer-lisp-functions` does what it says. `$$` is the result of the last command. Aliases live in `eshell-aliases-file`. History is maintained and expandable. `eshell-source-file` will run scripts. Since Eshell is not a terminal emulator, you need to configure it for any commands that need to run using a terminal emulator by adding it to to `eshell-visual-commands`.

```
(setq eshell-prefer-lisp-functions nil
      eshell-cmpl-cycle-completions nil
      eshell-save-history-on-exit t
      eshell-cmpl-dir-ignore "\\`\\(\\.\\.?\\|CVS\\|\\.svn\\|\\.git\\)/\\'")

(eval-after-load "esh-opt"
  '(progn
     (use-package em-cmpl)
     (use-package em-prompt)
     (use-package em-term)
     (setenv "PAGER" "cat")
     (add-hook 'eshell-mode-hook
               (lambda ()
                 (message "Welcome to Eshell.")
                 (setq pcomplete-cycle-completions nil)))
     (add-to-list 'eshell-visual-commands "ssh")
     (add-to-list 'eshell-visual-commands "tail")
     (add-to-list 'eshell-command-completions-alist
                  '("tar" "\\(\\.tar|\\.tgz\\|\\.tar\\.gz\\)\\'"))))
```

Configure a PS1 like prompt.

```
(setq eshell-prompt-regexp "^.+@.+:.+> ")
(setq eshell-prompt-function
      (lambda ()
        (concat
         (user-login-name)
         "@"
         (system-name)
         ":"
         (eshell/pwd)
         "> ")))
```

(k) File Based System

ID: org_gcr_2017-05-12_mara:E57D0D2A-0BF5-4733-AE1B-7CEA95171316

This system uses artifacts stored in files. It tries to persist file-stores every chance it gets without interrupting the user's flow. Flow is important.

Don't create backup files. Instead Git for versioning

Automatically back file-stores if no activity has occurred.

```
(setq auto-save-default t)
(setq make-backup-files nil)
(setq auto-save-visited-file-name t)
(setq auto-save-interval 0)
(setq auto-save-timeout (* 60 5))
```

Backup file-stores when the frame loses focus.

```
(add-hook 'focus-out-hook #'help/save-all-file-buffers)
```

Always keep buffers in-sync with changes in-file.

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

Save all files before common activities. Functions are easier to use than advice.

```
(defun help/safb-help/vc-next-action ()
  (interactive)
  (help/save-all-file-buffers)
  (help/vc-next-action))

(defun help/safb-vc-ediff ()
  (interactive)
  (help/save-all-file-buffers)
  (vc-ediff nil))

(defun help/safb-vc-diff ()
  (interactive)
  (help/save-all-file-buffers)
```

```
  (vc-diff nil))

(defun help/safb-vc-revert ()
  (interactive)
  (help/save-all-file-buffers)
  (unless (bound-and-true-p diff-hl-mode) (diff-hl-mode))
  (vc-revert)
  (when (bound-and-true-p diff-hl-mode) (call-interactively 'diff-hl-mode nil)))

(defun help/safb-help/magit-status ()
  (interactive)
  (help/save-all-file-buffers)
  (help/magit-status))

(defun help/safb-org-babel-tangle ()
  (interactive)
  (help/save-all-file-buffers)
  (let ((start (current-time)))
    (message (concat "org-babel-tangle BEFORE: <"
                     (format-time-string "%Y-%m-%dT%T%z")
                     ">"))
    (org-babel-tangle)
    (let* ((dur (float-time (time-since start)))
           (msg (format "Tangling complete after: %.06f seconds" dur)))
      (message (concat "org-babel-tangle AFTER: <"
                       (format-time-string "%Y-%m-%dT%T%z")
                       ">"))
      (message msg)
      (help/on-gui (alert msg :title "org-mode")))))

(defun help/safb-org-babel-detangle ()
  (interactive)
  (help/save-all-file-buffers)
  (org-babel-detangle))

(defun help/safb-other-window ()
  (interactive)
  (help/save-all-file-buffers)
  (other-window 1))

(defun help/ace-window ()
  (interactive)
  (help/save-all-file-buffers)
  (call-interactively #'ace-window))

(defun help/safb-org-edit-src-code ()
  (interactive)
  (help/save-all-file-buffers)
```

```
  (org-edit-src-code))

(defun help/safb-org-export-dispatch ()
  (interactive)
  (help/save-all-file-buffers)
  (org-export-dispatch))

(defun help/safb-TeX-command-master (&optional arg)
  (interactive)
  (help/save-all-file-buffers)
  (TeX-command-master arg))

(defun help/safb-org-babel-execute-buffer ()
  "Immediately save results."
  (interactive)
  (help/save-all-file-buffers)
  (org-babel-execute-buffer)
  (help/save-all-file-buffers))

(defun help/safb-org-babel-execute-subtree ()
  "Immediately save results."
  (interactive)
  (help/save-all-file-buffers)
  (org-babel-execute-subtree)
  (help/save-all-file-buffers))

(defun help/safb-help/org-babel-demarcate-block ()
  (interactive)
  (help/org-babel-demarcate-block)
  (help/save-all-file-buffers))

(defun help/safb-save-buffers-kill-terminal ()
  "Partially redundant; kept for consistency among ‘SAFB’ functions."
  (interactive)
  (help/save-all-file-buffers)
  (save-buffers-kill-terminal))

(defun help/safb-help/goto-line ()
  (interactive)
  (help/save-all-file-buffers)
  (help/goto-line))

(defun help/safb-switch-to-previous-buffer ()
  (interactive)
  (help/save-all-file-buffers)
  (switch-to-previous-buffer))

(defun help/safb-normal-mode ()
```

```
  (interactive)
  (help/save-all-file-buffers)
  (call-interactively #'normal-mode)
  (help/save-all-file-buffers))
```

Selection:

- Don't perform on frequent keys like enter and line navigation.

Future candidates:

- avy jump
- dired
- eshell
- ess-rdired
- eval-defun
- eval-region
- help/newline
- goto-line
- ido-switch-buffer
- ielm
- ispell
- ispell-word
- langtool-check-buffer
- newline-and-indent
- next-line
- org-edit-src-exit
- org-return
- pop-to-mark-command
- previous-line
- sp-newline
- with-current-buffer
- writegood-mode

Try to visit a non-existent file and get prompted to create its parent directories.

```
(add-to-list 'find-file-not-found-functions #'help/create-non-existent-directory)
```

Be aware of files larger than 2MiB. Turn off Aggressive-Indent and other expensive features in those buffers. NXML also seems to have a difficult time with large files.

```
(setq large-file-warning-threshold (* 1024 1024 2))
```

- Always use /tmp for temporary files
  - Via the thread "[O] org-file using tramp + babel?"

```
(setq temporary-file-directory "/tmp")
```

Probably never modify some files.

```
(use-package hardhat
  :ensure t
  :diminish global-hardhat-mode
  :config
  (global-hardhat-mode 1))
```

(l) File-system/directory management (Console)

`ID: org_gcr_2017-05-12_mara:AACEA837-9B4D-484A-B6C9-6C04DD8DE8A7`

You can use the usual machinery to work with the files. Highlight a region and operation selections occur for all files in that region. Commands are scheduled, and then executed, upon your command. Files can be viewed in modify or read-only mode, too. There is an idea of `mark` in files, which is to select them and perform operations on the marked files. There are helper methods for most things you can think if like directories or modified-files or whatever, meaning you can use regexen to mark whatever you like however you like. If that suits you, then don't be afraid of using the regular expression builder that is built into EMACS. Bulk marked file operations include additionally copying, deleting, creating hard links to, renaming, modifying the mode, owner, and group information, changing the time-stamp, listing the marked files, compressing them, decrypting, verifying and signing, loading or byte compiling them (Lisp files).

`g` updates the current buffer; `s` orders the listing by alpha or date-time.

`find-name-dired` beings the results back into Dired, which is nifty.

Wdired lets you modify files directly via the UI, which is interesting. Image-Dired lets you do just that.

`+` creates a new directory. `dired-copy-filename-as-kill` stores the list of files you have selected in the kill ring. `dired-compare-directories` lets you perform all sorts of directory comparisons, a handy tool that you need once in a while but definitely do need.

```
(defun help/dired-copy-filename ()
  "Push the path and filename of the file under the point to the kill ring.
Attribution: URL `https://lists.gnu.org/archive/html/help-gnu-emacs/2002-10/msg00556.ht
  (interactive)
  (message "Added %s to kill ring" (kill-new (dired-get-filename))))
(defun help/dired-copy-path ()
  "Push the path of the directory under the point to the kill ring."
  (interactive)
  (message "Added %s to kill ring" (kill-new default-directory)))
```

```
(setq dired-listing-switches "-alh")
(setq dired-recursive-deletes 'top)
(use-package dired-details+
  :ensure t)
(setq-default dired-details-hidden-string "")
(defun help/dired-mode-hook-fn ()
  "HELP dired customizations."
  (local-set-key "c" #'help/dired-copy-filename)
  (local-set-key "]" #'help/dired-copy-path)
  (diff-hl-dired-mode)
  (load "dired-x")
  (turn-on-stripe-buffer-mode)
  (dired-collapse-mode))
(add-hook 'dired-mode-hook #'help/dired-mode-hook-fn)
```

Try to guess the target directory for operations.

```
(setq dired-dwim-target t)
```

Use EMACS `ls`.

```
(help/on-osx
 (setq ls-lisp-use-insert-directory-program nil)
 (use-package ls-lisp))
```

After dabbling, something happened that really changed my mind. These three articles changed everything: Dired Shell Command, Working with multiple files in dired, and WDired: Editable Dired Buffers.. They just made the power of Dired so obvious, and so easy to use, that it instantly because delightful to use. That was very, very cool. Even though I was really, really happy with Finder and Explorer... suddenly it just became so obvious and pleasant to use Dired. That is so wild.

Key notes when executing shell commands on file selection...

Substitution:

`<cmd> ?` 1* calls to cmd, each file a single argument

`<cmd> *` 1 call to `cmd`, selected list as argument

`<cmd> *""` have the shell expand the * as a globbing wild-card

- Not sure what this means

Synchronicity:

`<cmd> ...` by default commands are called synchronously

`<cmd> &` execute in parallel

`<cmd> ;` execute sequentially, asynchronously

`<cmd> ;&` execute in parallel, asynchronously

Key notes on working with files in multiple directories... use the following:

Use `find` just like you would at the command line and all of the results show up in a single Dired buffer that you may work with just like you would any other file appearing in a Dired buffer. The abstraction here becomes so obvious, you may ask yourself why you never considered such a thing *before* now (as I did):

```
(use-package find-dired
  :ensure t
  :config
  (setq find-ls-option '("-print0 | xargs -0 ls -ld" . "-ld")))
```

Noting that:

`find-dired` is the general use case

`find-name-dired` is for simple, single string cases

And if you want to use the faster Elisp version, that uses lisp regex, use:

`find-lisp-find-dired` for anything

`find-lisp-find-dired-subdirectories` for only directories

Key notes on working with editable buffers...

As the author notes, you probably already instinctually knew what is possible. After reading his brief and concise exposition, it would be hard *not* to intuit what is possible! The options are big if you make a writable file buffer. Think about using multiple cursors. Done? Well, that is a no-brainer. Once you grok multiple cursors just `find-dired` what you need and then do what you need to do to it. Very cool.

`dired-toggle-read-only, C-x C-q` cycle between dired-mode and wdired-mode

`wdired-finish-edit, C-c C-c` commit your changes

`wdired-abort-changes, C-c ESC` revert your changes

```
(use-package wdired
  :ensure t
  :config
  (setq wdired-allow-to-change-permissions t)
  (setq wdired-allow-to-redirect-links t)
  (setq wdired-use-interactive-rename t)
  (setq wdired-confirm-overwrite t)
  (setq wdired-use-dired-vertical-movement 'sometimes))
```

When you selected a bunch of files or directories, you *may* want to communicate somewhere your selection somehow. The simplest way to do this is to utilize `dired-copy-filename-as-kill`. What a nice idea, and its default binding is `w`.

Since I started using a menu bar again, and wanting to get Imenu really exercised, Dired in Imenu seems like an obvious choice.

```
(use-package dired-imenu
  :ensure t)
```

Use Ido with Dired.

```
(setq ido-show-dot-for-dired t)
```

Collapse long empty paths.

```
(use-package dired-collapse
  :ensure t)
```

(m) IMenu

```
ID: org_gcr_2017-05-12_mara:12791F68-A7B0-4F93-9648-386C65C09355
```

Major productivity hack

```
(use-package imenu
  :config
  (setq imenu-sort-function #'imenu--sort-by-name))
(defun help/try-to-add-imenu ()
  "Add Imenu to modes that have ‘font-lock-mode’ activated.

Attribution: URL http://www.emacswiki.org/emacs/ImenuMode"
  (condition-case nil (imenu-add-to-menubar "Imenu") (error nil)))
(add-hook 'font-lock-mode-hook #'help/try-to-add-imenu)
```

Provide it in a buffer.

```
(use-package imenu-list
  :ensure t
  :config
  (setq imenu-list-focus-after-activation t)
  (setq imenu-list-auto-resize t)
  (setq imenu-list-position 'left)
  (setq imenu-list-size 40))
```

(n) Interactively DO Things

```
ID: org_gcr_2017-05-12_mara:29AAB820-5824-43F8-9E7F-7FD1CB6F4624
```

IDO is used everywhere possible.

Access nearly every object available in this system from one place.

These configurations are performed in the correct order. Any attempt to refactor this Source-Block will break Ido in this system.

```
(use-package ido)
(use-package flx-ido
  :ensure t
  :config
  (ido-mode t))
(use-package ido-hacks
  :ensure t)
(use-package ido-completing-read+
  :config
  (ido-ubiquitous-mode t)
  (setq ido-create-new-buffer 'always)
```

```
  (flx-ido-mode t)
  (setq ido-use-faces nil))
(use-package ido-vertical-mode
  :ensure t
  :config
  (ido-vertical-mode t)
  (setq ido-vertical-define-keys 'C-n-C-p-up-down-left-right))
```

Make functions search-able.

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

Make URLs a first-class object.

```
(setq ido-use-url-at-point t)
(setq ido-use-filename-at-point 'guess)
```

(o) Font

`ID: org_gcr_2017-05-12_mara:D12919AC-9D9B-4C40-9872-1E01C92FA493`

Configure the manual configuration of a fallback Unicode font.

```
(set-fontset-font "fontset-default" nil
                  (font-spec :size 20 :name "Symbola"))
```

Here are the Unicode fonts that provide nearly everything.

Name	Version	URL	Comments
DejaVu	2.43	.	Modern classic
Symbola	7.17	.	Neat
Quivira	4.0	.	Amazing
Noto	?	1 2	Has morse code, and more

To test it run `view-hello-file` and `M-x list-charset-chars RET unicode-bmp RET`.

Activate font locking everywhere possible.

```
(global-font-lock-mode t)
```

(p) Going to Objects

`ID: org_gcr_2017-05-12_mara:8A73E509-A878-4E89-86DC-9BDA8C898F8C`

Go to any object in the frame quickly.

```
(use-package avy
  :ensure t
  :config)
```

(q) Grammar

`ID: org_gcr_2017-05-12_mara:2E0F4D9D-5AB6-407C-BBC6-BD0277ECA015`

Warn of poor grammar immediately interrupting flow with a visual indicator.

```
(use-package writegood-mode
  :ensure t
  :config
  (eval-after-load "writegood-mode"
    '(diminish 'writegood-mode)))
```

Integrate Langtool.

```
(use-package langtool
  :ensure t
  :init
  (setq langtool-language-tool-jar (getenv "LANGTOOL"))
  (setq langtool-mother-tongue "en")
  (setq langtool-java-bin (concat (getenv "JAVA_HOME") "/bin/java")))
```

Integrate Proselint. I wanted to load the executable location from `PROSELINT_HOME` but I haven't figured out how yet. `flycheck-define-checker` is a macro but I don't know how to expand the string for use in the macro. I tried quasiquote and that was wrong.

```
(with-eval-after-load "flycheck"
  (flycheck-define-checker proselint
    "A linter for prose."
    :command ("/Users/gcr/proselint/env/bin/proselint" source-inplace)
    :error-patterns
    ((warning line-start (file-name) ":" line ":" column ": "
              (id (one-or-more (not (any " "))))
              (message (one-or-more not-newline)
                       (zero-or-more "\n" (any " ") (one-or-more not-newline)))
              line-end))
    :modes (text-mode org-mode markdown-mode gfm-mode))
  (add-to-list 'flycheck-checkers 'proselint))
```

(r) Keyboard Macros

`ID: org_gcr_2017-05-12_mara:F2701D64-8484-4EB0-9FBD-692C98CA3C58`

Keep many keyboard macros.

```
(setq kmacro-ring-max 32)
```

Persist keyboard macros in Emacs Lisp alternative to `insert-kbd-macro`.

```
(use-package elmacro
  :ensure t
  :diminish (elmacro-mode)
  :config
  (elmacro-mode))
```

(s) Intellisense

`ID: org_gcr_2017-05-12_mara:DE05FF62-53EA-491A-881B-650030210591`

```
(use-package fuzzy
  :ensure t)
(use-package auto-complete
  :ensure t
  :config
  (use-package auto-complete-config)
  (setq ac-quick-help-prefer-pos-tip nil)
  (ac-config-default)
  (setq ac-auto-start nil)
  (help/not-on-gui (ac-set-trigger-key "TAB"))
  (help/on-gui (ac-set-trigger-key "<tab>"))
  :diminish auto-complete-mode)
(use-package auto-complete-chunk
  :ensure t)
```

Auto-completion for .-separated words.

```
(use-package auto-complete-chunk
  :ensure t)
```

IntelliJ-Like `jump to definition`.

```
(use-package dumb-jump
  :ensure t
  :config
  (define-prefix-command 'help/dj-map)
  (key-chord-define-global "jj" #'help/dj-map)
  (define-key help/dj-map "j" #'dumb-jump-go)
  (define-key help/dj-map "i" #'dumb-jump-go-other-window)
  (define-key help/dj-map "k" #'dumb-jump-back)
  (define-key help/dj-map "l" #'dumb-jump-go-prompt)
  (define-key help/dj-map "m" #'dumb-jump-quick-look))
```

(t) Mark, Kill and Region

`ID: org_gcr_2017-05-12_mara:432D3FDF-38B3-499D-94D3-338CFEF2957C`

Easy kill ring access.

```
(use-package browse-kill-ring
  :ensure t
  :config
  (browse-kill-ring-default-keybindings)
  (setq browse-kill-ring-highlight-current-entry t)
  (setq browse-kill-ring-highlight-inserted-item t)
  (setq browse-kill-ring-show-preview nil)
  (setq browse-kill-ring-separator
        ".-~-.-~-.-~-.-~-.-~-.-~-.-~-.-~-.-~-.-~-.-~-.-~-.-~-.-~-.-~-.-~-.-~-.-~-.-~-
")
  (defun help/browse-kill-ring ()
    "Disable 'fci-mode' before entering 'browse-kill-ring'."
    (interactive)
```

```
    (fci-mode)
    (browse-kill-ring)
    (fci-mode))
  (global-set-key (kbd "M-y") #'help/browse-kill-ring))
```

When you start typing and text is selected, replace it with what you are typing, or pasting

```
(delete-selection-mode t)
```

Automatically save data copied from the system clipboard into the kill ring before killing Emacs data.

Via Ben Maughan:

> ;; Save whatever's in the current (system) clipboard before ;; replacing it with the Emacs' text. ;; `https://github.com/dakrone/eos/blob/master/eos.org`

```
(setq save-interprogram-paste-before-kill t)
```

Via: copy selected text to the clip board.

```
(setq mouse-drag-copy-region t)
```

(u) Minibuffer

`ID: org_gcr_2017-05-12_mara:FABF1013-0F88-4207-AB8E-982E7F4C250D`

Make it easier to answer questions.

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

Comfortably display information.

```
(setq resize-mini-windows t)
(setq max-mini-window-height 0.33)
```

Allow recursive commands-in-commands and highlight the levels of recursion.

```
(setq enable-recursive-minibuffers t)
(minibuffer-depth-indicate-mode t)
```

(v) Mouse

`ID: org_gcr_2017-05-12_mara:6171A841-DFD0-44A7-AAA6-BB099B1922FE`

Scroll pleasantly with the mouse wheel. A slow turn moves the buffer up and down one line at a time. So does a fast turn. Anything further than 5-10 lines deserves a fast navigation vehicle.

```
(setq mouse-wheel-scroll-amount '(1 ((shift) . 1)))
(setq mouse-wheel-progressive-speed nil)
(setq mouse-wheel-follow-mouse t)
```

(w) Occur

`ID: org_gcr_2017-05-12_mara:54249478-0107-4838-8394-4A6A1850ED99`

- Alternate search result background color

- Multiple Occur buffers
- n and p navigate search results
- Jump near buffer top

```
(defun help/occur-mode-hook-fn ()
  "HELP customizations."
  (interactive)
  (turn-on-stripe-buffer-mode)
  (occur-rename-buffer))
(add-hook 'occur-mode-hook #'help/occur-mode-hook-fn)
(define-key occur-mode-map (kbd "n") #'next-logical-line)
(define-key occur-mode-map (kbd "p") #'previous-logical-line)
(add-hook 'occur-mode-find-occurrence-hook #'help/recenter-line-near-top-fn)
```

(x) Popups

ID: org_gcr_2017-05-12_mara:5EFCDB34-DC9E-4502-93AD-8E31363F8A5C

Provide popup notifications.

```
(use-package alert
  :ensure t
  :config
  (setq alert-fade-time 10)
  (setq alert-default-style 'message)
  (help/on-gui
   (help/on-osx
    (setq alert-default-style 'notifier)))
  (setq alert-reveal-idle-time 120))
```

(y) Projects

ID: org_gcr_2017-05-12_mara:BEFE841E-58A8-43D6-84F0-5BA2EBCFE6B8

Directories that have Git working copies are logically projects. Manage them with Projectile.

```
(use-package projectile
  :ensure t
  :config
  (projectile-global-mode t)
  (global-set-key (kbd "s-c") #'projectile-find-file)
  (help/on-windows
   (setq projectile-indexing-method 'alien))
  :diminish projectile-mode)
```

Notify Magit about every working copy that Projectile knows about.

```
(eval-after-load "projectile"
  '(progn (setq magit-repository-directories (mapcar (lambda (dir)
                                                       (substring dir 0 -1))
                                                     (remove-if-not (lambda (project)
```

```
                                                                    (file-directory-p
                                                                  (projectile-relevan

                    magit-repository-directories-depth 1)))
```

(z) Printing

```
ID: org_gcr_2017-05-12_mara:93730D86-E142-49E7-9325-0A48D23A8BE5
```

```
(use-package pp
  :commands (pp-display-expression))
```

() Register

```
ID: org_gcr_2017-05-12_mara:132C5084-FD72-45C5-BD32-DCE78A00ED34
```

```
(setq register-preview-delay 2)
(setq register-separator "\n\n")
```

() Replacing

```
ID: org_gcr_2017-05-12_mara:8DDCD1EC-C603-41D8-85A4-EA7C11C86207
```

Display information about search-and-or-replace operation.

```
(use-package anzu
  :ensure t
  :config
  (global-anzu-mode t)
  (setq anzu-mode-lighter "")
  (setq anzu-deactivate-region t)
  (setq anzu-search-threshold 1000)
  (setq anzu-replace-to-string-separator "  "))
```

() Save and Restore Operating State

```
ID: org_gcr_2017-05-12_mara:217B8635-CA69-4F58-A12D-124ED384EE22
```

i. Mark

```
ID: org_gcr_2017-05-12_mara:2CBAFE00-5B18-4798-AEF3-488D622374BE
```

When you re-open a file return the mark to its last position

```
https://www.emacswiki.org/emacs/SavePlace
```

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

ii. Minibuffer

```
ID: org_gcr_2017-05-12_mara:88B140F9-567E-47DA-8137-42EF4CE0904E
```

It is nice to have commands and their history saved so that every time you get back to work, you can just re-run stuff as you need it.

```
https://www.emacswiki.org/emacs/SaveHist
```

```
(setq savehist-save-minibuffer-history t)
(setq savehist-additional-variables
```

```
      '(kill-ring
        search-ring
        regexp-search-ring
        last-kbd-macro
        kmacro-ring
        shell-command-history))
(savehist-mode t)
```

Use the history as a lookup instead of a log.

```
(setq history-delete-duplicates t)
```

() Searching

ID: org_gcr_2017-05-12_mara:AA7F12B0-13E2-4B5A-A8BB-B42E1FCC6E55

i. Internal

ID: org_gcr_2017-05-12_mara:11DF2712-61B1-4E9F-9950-8BDB0E57E580

It is **important** to understand how Emacs performs searching.

Copied and pasted Emacs literature from

- `(info "(info) Searching Case")`
- Searching and Case

follows

```
Searches in Emacs normally ignore the case of the text they are searching
through, if you specify the text in lower case. Thus, if you specify searching
for 'foo', then 'Foo' and 'foo' also match. Regexps, and in particular
character sets, behave likewise: '[ab]' matches 'a' or 'A' or 'b' or 'B'.

An upper-case letter anywhere in the incremental search string makes the
search case-sensitive. Thus, searching for 'Foo' does not find 'foo' or 'FOO'.
This applies to regular expression search as well as to string search. The
effect ceases if you delete the upper-case letter from the search string.

If you set the variable case-fold-search to nil, then all letters must match
exactly, including case. This is a per-buffer variable; altering the variable
normally affects only the current buffer, unless you change its default value.
See Locals. This variable applies to nonincremental searches also, including
those performed by the replace commands (see Replace) and the minibuffer
history matching commands (see Minibuffer History).

Typing M-c within an incremental search toggles the case sensitivity of that
search. The effect does not extend beyond the current incremental search to
the next one, but it does override the effect of adding or removing an
upper-case letter in the current search.
```

So let Emacs search how it was built to search instead of attempting to make it case-sensitive everywhere. Emacs already makes it easy to perform case-sensitive

searches when you want. It is simple and predictable.

When you don't want to do that interactively type M-c in the search input area to toggle case sensitivity.

When you don't want to do that non-interactively then search for that command or activities toggling variable `M-x apropos-variable RET case-fold-search RET` and bind it during the search.

ii. External

`ID: org_gcr_2017-05-12_mara:D55FB85E-A70E-4A56-AA5F-089B4FE52E7F`

Writeable Grep integration.

```
(use-package wgrep
  :ensure t
  :config
  (setq wgrep-auto-save-buffer t))
```

Ag integration.

```
(use-package ag
  :ensure t
  :config
  (setq ag-highlight-search t)
  (setq ag-reuse-window nil)
  (setq ag-reuse-buffers nil)
  (setq ag-arguments (-insert-at (- (length ag-arguments) 1) '"--ignore-case"
                                 ag-arguments))
  (setq ag-arguments (-insert-at (- (length ag-arguments) 1) '"--file-search-regex"
                                 ag-arguments))
  (defun help/ag-mode-hook-fn ()
    "HELP ag customizations."
    (interactive)
    (turn-on-stripe-buffer-mode))
  (defun help/ag-mode-finished-hook-fn ()
    "HELP ag finished hook function."
    (interactive)
    (let ((compilation-scroll-output 'first-error))
      (pop-to-buffer next-error-last-buffer)))
  (add-hook 'ag-mode-hook #'help/ag-mode-hook-fn)
  (add-hook 'ag-search-finished-hook #'help/ag-mode-finished-hook-fn))
```

Writeable Ag.

```
(use-package wgrep-ag
  :ensure t
  :after ag)
```

() Spell Checking

`ID: org_gcr_2017-05-12_mara:44909368-2D9A-40B9-A3F4-DF297A0EE570`

Ispell is simple and powerful.

i. Org-Mode

ID: org_gcr_2017-05-12_mara:4892BEB5-C0B5-4506-BBB3-40E41F9E25FF

Never ispell the following objects.

Block regex helper.

```
(defun help/block-regex (special)
  "Make an ispell skip-region alist for a SPECIAL block."
  (interactive)
  `(,(concat help/org-special-pre "BEGIN_" special)
    .
    ,(concat help/org-special-pre "END_" special)))
```

Source-Blocks.

```
(add-to-list 'ispell-skip-region-alist (help/block-regex "SRC"))
```

Example-Blocks. This system often uses Source-Blocks to edit content and Example-Blocks to make it easily renderable when it is not for running.

```
(add-to-list 'ispell-skip-region-alist (help/block-regex "EXAMPLE"))
```

Properties.

```
(add-to-list 'ispell-skip-region-alist '("^\s*:PROPERTIES\:$" . "^\s*:END\:$"))
```

Footnotes.

```
(add-to-list 'ispell-skip-region-alist '("\\[fn:.+:" . "\\]"))
```

Footnotes with URLs that contain line-breaks.

```
(add-to-list 'ispell-skip-region-alist '("^http" . "\\]"))
```

Bold text list items.

```
(add-to-list 'ispell-skip-region-alist '("- \\*.+" . ".*\\*: "))
```

Right arrows.

```
(add-to-list 'ispell-skip-region-alist '("\\rarr"))
```

Check `SPECIAL LINE` definitions, ignoring their type.

```
(let ()
  (--each
      '(("ATTR_LATEX" nil)
        ("AUTHOR" nil)
        ("BLOG" nil)
        ("CREATOR" nil)
        ("DATE" nil)
        ("DESCRIPTION" nil)
        ("EMAIL" nil)
        ("EXCLUDE_TAGS" nil)
```

```
                ("HTML_CONTAINER" nil)
                ("HTML_DOCTYPE" nil)
                ("HTML_HEAD" nil)
                ("HTML_HEAD_EXTRA" nil)
                ("HTML_LINK_HOME" nil)
                ("HTML_LINK_UP" nil)
                ("HTML_MATHJAX" nil)
                ("INFOJS_OPT" nil)
                ("KEYWORDS" nil)
                ("LANGUAGE" nil)
                ("LATEX_CLASS" nil)
                ("LATEX_CLASS_OPTIONS" nil)
                ("LATEX_HEADER" nil)
                ("LATEX_HEADER_EXTRA" nil)
                ("NAME" t)
                ("OPTIONS" t)
                ("POSTID" nil)
                ("RESULTS" t)
                ("SELECT_TAGS" nil)
                ("STARTUP" nil)
                ("TITLE" nil))
              (add-to-list
               'ispell-skip-region-alist
               (let ((special (concat "#[+]" (car it) ":")))
                 (if (cadr it)
                     (cons special "$")
                   (list special))))))
```

() Sudo

ID: org_gcr_2017-05-12_mara:7853E266-9462-4CEB-A6AB-873EAC06666F

Configure Sudo with Ido.

```
(help/on-osx
 (defun help/ido-find-file ()
   "Find file as root if necessary.

Attribution: URL `http://emacsredux.com/blog/2013/04/21/edit-files-as-root/'"
   (unless (and buffer-file-name
                (file-writable-p buffer-file-name))
     (find-alternate-file (concat "/sudo:root@localhost:" buffer-file-name))))

 (advice-add #'ido-find-file :after #'help/ido-find-file))
```

() Syntax Checking

ID: org_gcr_2017-05-12_mara:DD506D5A-5F57-4939-B446-536437FDDC97

Do not perform syntactic analysis on-demand. Something about my config makes global-mode puke and kill Emacs every so often.

```
(use-package flycheck
  :ensure t
  :diminish flycheck-mode)
```

() TAB

ID: org_gcr_2017-05-12_mara:C9ACAB8C-1D19-4CC2-ABE9-ACEBCDBCA9E6

Most modes in this system will never use TAB stops.

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

Remove TAB from all buffers before persisting to the backing file unless it is configured to retain TAB. The use case is a Makefile.

```
(defun help/untabify-if-not-indent-tabs-mode ()
  "Untabify if ‘indent-tabs-mode’ is false.

Attribution: URL ‘http://www.emacswiki.org/emacs/UntabifyUponSave’"
  (interactive)
  (when (not indent-tabs-mode)
    (untabify (point-min) (point-max))))

(add-hook 'before-save-hook #'help/untabify-if-not-indent-tabs-mode)
```

Most programing modes indent to 2 spaces. TABs should be the same width.

```
(setq-default tab-width 2)
```

() Version Control

ID: org_gcr_2017-05-12_mara:BDC3035D-4F09-4A7B-8F14-1EE4604ACC0D

Use VC for single files and Magit for multiple files.

The commit log editor uses With-Editor and Server modes. They are not diminished because they are infrequently used.

```
(add-to-list 'load-path "~/src/magit/lisp")
(require 'magit)

(with-eval-after-load 'info
  (info-initialize)
  (add-to-list 'Info-directory-list
               "~/src/magit/Documentation/"))
```

Leave the VC message template empty.

```
(eval-after-load "log-edit"
  '(remove-hook 'log-edit-hook 'log-edit-insert-message-template))
```

Change browsing.

```
(use-package git-timemachine
  :ensure t)
```

VC activity keybindings.

```
(global-set-key (kbd "s-w") #'git-timemachine)
(global-set-key (kbd "s-e") #'help/safb-help/magit-status)
(global-set-key (kbd "s-r") #'help/safb-vc-revert)
(global-set-key (kbd "s-f") #'help/safb-help/vc-next-action)
```

Git ignore files are text files.

```
(add-to-list 'auto-mode-alist '(".gitignore$" . text-mode))
```

Git config files.

```
(use-package gitignore-mode
  :ensure t)

(use-package gitconfig-mode
  :ensure t)

(use-package gitattributes-mode
  :ensure t)
```

() Video

```
ID: org_gcr_2017-05-12_mara:BB0F7B20-AC2C-49DB-B6A2-07D3C5D7A197
```

Embedding Youtube videos with org-mode links.

```
(defvar yt-iframe-format
  ;; You may want to change your width and height.
  (concat "<iframe width=\"440\""
          " height=\"335\""
          " src=\"https://www.youtube.com/embed/%s\""
          " frameborder=\"0\""
          " allowfullscreen>%s</iframe>"))

(org-add-link-type
 "yt"
 (lambda (handle)
   (browse-url
    (concat "https://www.youtube.com/embed/"
            handle)))
 (lambda (path desc backend)
   (cl-case backend
     (html (format yt-iframe-format
                   path (or desc "")))
     (latex (format "\href{%s}{%s}"
                    path (or desc "video"))))))
```

() Whitespace Management

```
ID: org_gcr_2017-05-12_mara:9FBD1929-29C7-45CE-B9EC-DF0711778439
```

- Visible Whitespace
  - Tabs

```
(use-package whitespace
  :ensure t
  :config
  (setq whitespace-style '(tab-mark))
  (setq whitespace-display-mappings
        '((tab-mark ?\t [? ?\t] [? ?\t] [?\t])))
  (setq whitespace-line-column help/column-width)
  (global-whitespace-mode t)
  :diminish whitespace-mode global-whitespace-mode)
```

() Word Wrap

ID: org_gcr_2017-05-12_mara:202157D8-58BB-4250-BDDD-98ADBF97F801

```
(diminish 'visual-line-mode)
```

Remind me that VLM is turned on so I pay attention to the wrap. There is a Hydra that makes it easy to remember to turn it off.

```
(setq visual-line-fringe-indicators '(nil right-curly-arrow))
```

4. Hacking

ID: org_gcr_2017-05-12_mara:E900FBB2-2D32-4253-BFE6-36BC951384A2

Emacs has three primal major modes that you must understand:

- `text-mode`
- `prog-mode`
- `special-mode`

(a) Common Configurations

ID: org_gcr_2017-05-12_mara:1864484F-237D-4443-BDD6-D429AAB7BD1A

This system configures `text-mode` and `prog-mode` very similarly:

- EMACS **exists** to help you work with text.
- EMACS' entire configuration helps you work with text whether it is in a specific mode or not.
- Org-Mode's motto is "**Organize Your Life In Plain Text!**".
- From an EMACS and a LP perspective `text-mode` **is a** programming mode.
- In this system: **Text is the User-Interface**.

This system does not rely on `prog-mode` inheritance to configure it's hacking modes:

- The EMACS literature advises that modes extend `text-mode` or `prog-mode`
- That *would* make it easier to configure nearly everything using `prog-mode-hook`.
- In practice `prog-mode` is too new.
- Not all programming modes inherit from it. Not even `IELM` is ready.

With that in mind this system:

- Defines common configuration here for reuse in every desired mode starting with `text-mode` and then all logical programming modes.
- Explicitly utilizes it directly instead of using inheritance.
- This system refers to this configuration of programming modes as `prog*-mode`.
- The line between "configuring EMACS", "configuring `text-mode`", and "configuring `prog*-mode`" is often blurred and sometimes confusing. The lines become wavy and intertwined with mastery of EMACS and LP.

Make sense of the current mode.

```
(use-package parent-mode
  :ensure t
  :config
  (defun help/parent-mode-display ()
    "Display this buffer's mode hierarchy."
    (interactive)
    (let ((ls (parent-mode-list major-mode)))
      (princ ls))))
```

i. Special Mode

   `ID: org_gcr_2017-06-25_mara:7C9667D1-C881-40AC-9312-EB11A9F4F61E`

   Every special mode needs this. Which is nothing yet.

ii. Text-Mode

   `ID: org_gcr_2017-05-12_mara:C3270DE0-55FD-4612-B3E3-226CAC5A8FDA`

   Every text editing buffer needs this.

```
(defun help/text-mode-fn ()
  "HELP's standard configuration for buffer's working with text, often for
   programming."
  (interactive)
  (auto-fill-mode)
  (diminish 'auto-fill-function)
  (visual-line-mode)
  (nlinum-mode)
  (fci-mode)
  (rainbow-mode)
  (help/try-to-add-imenu)
  (turn-on-page-break-lines-mode))

(add-hook 'text-mode-hook #'help/text-mode-fn)
```

iii. Prog-Mode Modes

   `ID: org_gcr_2017-05-12_mara:E20FE37A-E10D-406C-B95D-545607F7A93C`

   - Mode inheritance is represented by list definition & indentation.

- Some modes are so simple that inheritance isn't defined.
- Hacking mode hooks.
  - Configurations common to every hacking vehicle.

```
(setq help/prog-modes '(makefile-mode-hook
                        ruby-mode-hook
                        sh-mode-hook
                        plantuml-mode-hook
                        tex-mode-hook
                        R-mode-hook
                        SAS-mode-hook
                        graphviz-dot-mode-hook
                        c-mode-common-hook
                        php-mode-hook
                        scad-mode-hook
                        web-mode-hook
                        js2-mode-hook
                        json-mode-hook
                        crontab-mode-hook
                        apache-mode-hook
                        python-mode-hook
                        gnu-apl-mode-hook))
```

  - LISP mode hooks.
    - Are hacking modes.

```
(setq help/lisp-modes
      '(emacs-lisp-mode-hook
        ielm-mode-hook
        lisp-interaction-mode-hook
        scheme-mode-hook))
(setq help/prog-modes (append help/prog-modes help/lisp-modes))
```

    - IELM mode hook.
      - Does one or two more things.

iv. Prog-Mode Hook Function

ID: org_gcr_2017-05-12_mara:750D5ACE-FB06-4A71-A0C1-634C71198A5F

A. Goal

ID: org_gcr_2017-05-12_mara:34B0116F-B50F-487A-A605-0D31C07F6773

- Indent at every opportunity and automatically. Verify that it makes sense for the mode. Explicitly define instead of relying on `prog-mode` inheritance; use this documents logical `prog*-mode` approach instead.

```
(use-package aggressive-indent
  :ensure t
  :config)
```

- Always maintain balanced brackets. Easily wrap the selected region. Auto-escape strings pasted into other strings. Smartparens provides built-in correct behavior for most modes.

```
(use-package smartparens
  :ensure t
  :config
  (setq sp-show-pair-from-inside nil)
  (require 'smartparens-config)
  :diminish smartparens-mode)
```

B. Implementation

ID: org_gcr_2017-05-12_mara:7917A66D-8F20-43A1-888A-E9F0C86C23A7

```
(defun help/prog-mode-hook-fn ()
  (interactive)
  (help/text-mode-fn)
  (smartparens-strict-mode)
  (aggressive-indent-mode)
  (hs-minor-mode)
  (help/not-on-gui (local-set-key (kbd "RET") #'newline-and-indent))
  (help/on-gui (local-set-key (kbd "<return>") #'newline-and-indent)))

(let ()
  (--each help/prog-modes
    (add-hook it #'help/prog-mode-hook-fn)))
```

(b) Literate Programming

ID: org_gcr_2017-05-12_mara:18EBD4EF-AFAD-4421-A09F-A7DB14AD1398

i. Emacs Lisp

ID: org_gcr_2017-05-12_mara:3A31F89D-2534-4E5F-8C3B-3CEBAFAC9E2D

Have fun with scratch.

```
(setq initial-scratch-message ";; Happy Hacking \n\n")
```

Use `emacs-lisp` in scratch.

```
(setq initial-major-mode 'emacs-lisp-mode)
```

Immortal scratch via.

```
(with-current-buffer "*scratch*"
  (emacs-lock-mode 'kill))
```

Persistent scratch.

```
(use-package persistent-scratch
  :ensure t
  :config
  (persistent-scratch-setup-default))
```

Broadcast scoping mode.

```
(use-package lexbind-mode)
```

Slime navigation:

- “navigation to the symbol at point”: `M-.`
- “pop back to previous marks”: `M-,`
- “describe the symbol at point”: `C-c C-d C-d`

Symbol visualization.

```
(use-package highlight-quoted
  :ensure t)
```

Quasi-Quote visualization.

```
(use-package highlight-stages
  :ensure t)

(use-package elisp-slime-nav
  :ensure t
  :diminish elisp-slime-nav-mode)

(defun help/elisp-eval-buffer ()
  "Intelligently evaluate an Elisp buffer."
  (interactive)
  (help/save-all-file-buffers)
  (eval-buffer))

(defun help/elisp-mode-local-bindings ()
  "Helpful behavior for Elisp buffers."
  (local-set-key (kbd "s-l eb") #'help/elisp-eval-buffer)
  (local-set-key (kbd "s-l ep") #'eval-print-last-sexp)
  (local-set-key (kbd "s-l td") #'toggle-debug-on-error)
  (local-set-key (kbd "s-l mef") #'macroexpand)
  (local-set-key (kbd "s-l mea") #'macroexpand-all)
  (local-set-key (kbd "#") #'endless/sharp)
  (local-set-key (kbd "C-c e") #'macrostep-expand))

(defun help/emacs-lisp-mode-hook-fn ()
  (interactive)
  (help/elisp-mode-local-bindings)
  (lexbind-mode)
  (eldoc-mode)
  (diminish 'eldoc-mode)
  (turn-on-elisp-slime-nav-mode)
  (highlight-quoted-mode)
  (highlight-stages-mode))

(setq ielm-noisy nil)
```

```
(setq ielm-prompt "LISP> ")

(setq ielm-dynamic-return nil)

(setq ielm-dynamic-multiline-inputs nil)

(defun help/ielm-mode-hook-fn ()
  "HELP customizations."
  (interactive)
  (help/ielm-auto-complete)
  (turn-on-elisp-slime-nav-mode))

(let ()
  (--each help/lisp-modes
    (add-hook it #'help/emacs-lisp-mode-hook-fn)))

(add-hook 'ielm-mode-hook #'help/ielm-mode-hook-fn)
```

"find callers of elisp functions or macros".

```
(use-package elisp-refs
  :ensure t)
```

Package lint.

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

(use-package flycheck-package
  :ensure t
  :after (package-lint))
```

A. Keybinding

ID: org_gcr_2017-05-12_mara:8F05616B-84BF-4E34-8964-2E200807E5E9

```
(define-key emacs-lisp-mode-map (kbd "s-p") #'describe-thing-in-popup)
```

ii. Org-Mode

ID: org_gcr_2017-05-12_mara:D017621F-5169-442A-A52C-7D91EEB7D796

A. Literate Programming

ID: org_gcr_2017-05-12_mara:CD4943EE-DE3D-4B3A-AB12-525580DC8EB7

When source blocks are evaluated, their results get stored in a result area, typically for display. If the results are small, they are displayed with colons instead of an `example` block. Instead, **always** place them in an example block. This makes exports more consistent and other Org-Mode features seem to behave more predictably.

```
(setq org-babel-min-lines-for-block-output 0)
```

Configure Org-Mode to manage it's Source-Block backed buffers the same as the rest of this system.

```
(setq org-edit-src-auto-save-idle-delay 0)
(setq org-edit-src-turn-on-auto-save nil)
```

Update in-buffer images after Source-Block execution. This is a programming task. That is why it is under this heading and Evaluation. This is a setting configuring how the results of evaluation are refreshed in EMACS.

```
(defun help/org-babel-after-execute-hook ()
  "HELP settings for the ‘org-babel-after-execute-hook’.

This does not interfere with exports.

Attribution: URL ‘https://lists.gnu.org/archive/html/emacs-orgmode/2015-01/msg0
  (interactive)
  (org-redisplay-inline-images))

(add-hook 'org-babel-after-execute-hook #'help/org-babel-after-execute-hook)
```

Never "automatically" evaluate a source block.

```
(setq org-confirm-babel-evaluate nil)
```

Make it unpleasant for Sysop to modify source-block outside of a source-block backed buffer. The next step is to write some code to prevent modifying source-blocks outside of that place.

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

B. Workflow

ID: org_gcr_2017-05-12_mara:0CFAE46B-8D06-48FB-8D7E-7B636C91D443

- Custom TODO workflow states requirements, each answering the following questions:
  - Version 01
    - What needs to be done? TODO
    - What is being done right now? IN-PROGRESS
    - What is waiting because an external agent? HELD-BLOCKED
    - What is waiting because I have put it on hold? HELD-FROZEN
    - What is waiting until another date? (although I'm not using the Agenda) HELD-UNTIL
    - What is complete and needs to be reviewed? REVIEW
    - What is complete? DONE
  - Version 02
    - Questions about every state

- When was it created?
- The purpose of its creation is contained within its Headline and its contents “when?” and “why?” aren’t required
- When was it changed and why?
- The reason for the change is probably in the body of the Headline so why bother keeping a note here? Bother because it makes it easier to review the state-change of a task because
- It probably isn’t worth keeping in the task long-term (it is transient)
- But it still matters and I want to know about it
- If it is worth keeping in the task long-term then it will be there but
- It makes it easier to review the modification of a task without
- Having to re-read the entire task to understand the context
- Track down the why it modified
- When did it complete (done or not) and why?
- The goals for completion are probably in the body of the Headline itself so why keeping a note here? Bother because it makes it easier to review the completion of a task without
- Having to re-read the entire task to understand the context
- Track down the why it completed

* Migration from Version 01
  - TODO → no change (timestamp)
  - IN-PROGRESS → GO (timestamp) (not DO because of DONE fastkey)
  - HELD-BLOCKED → WAIT with note (timestamp, explanation)
  - HELD-FROZEN → WAIT with note (timestamp, explanation)
  - HELD-UNTIL → WAIT with note (timestamp, explanation)
  - REVIEW → no change (timestamp)
  - DONE → no change (timestamp, explanation)
* Workflow cheatsheet for every state change
  - Entry related
  - `!` store a timestamp when **entered**
  - `@` store a timestamp with a note when **entered**
  - `C-c C-c` provides an empty note

· Exit related

· `/!` **in addition** to the state's entry configuration

· record a timestamp when **leaving**

· *if and only if*

· the target state not already configure logging when entering it

· Access related

· `char` defines a fast-access key for the state name

```
(setq org-todo-keywords
      '((sequence
         "TODO(t!)"
         "GO(g@)"
         "WAIT(w@)"
         "BLOCKED(b@)"
         "REVIEW(r!)"
         "|"
         "DONE(d@)"
         "CANCELED(c@)"
         )))
```

- Reference
  - StackOverflow: Add CREATED date property to TODOs in org-mode
  - Org: 5.2.1 TODO keywords as workflow states
  - 5.3.2 Tracking TODO state changes

It is easier to understand the history of a task by reading it in chronological order.

```
(setq org-log-states-order-reversed nil)
```

Make it easy to hide the state changes until I want to see them with a Drawer.

```
(setq org-log-into-drawer t)
```

Maintain state when archiving a heading.

```
(setq org-archive-mark-done nil)
```

C. Refile

```
ID: org_gcr_2017-05-12_mara:C09D993C-6E48-4C84-8460-585F0A10DA63
```

D. Context

```
ID: org_gcr_2017-05-12_mara:DFCC7C00-F3B3-4C35-BBE6-A5ADBB60326D
```

I wrote my first to-do list on a funeral home notepad. It was either that or it was from the furniture store. My dad worked at both. The National Selected Morticians logo sat somewhere on its pages. There wasn't much for me to

note back then and the notepad worked fine. When I got a little older I graduated to a Mead spiral notebook. I wasn't organized and wasted space. When you tore out the pages they made a mess. At home we had two computers over the years an Apple 2E and then a IBM PC compatible. Mom used PFS Write on the 2E and I used Notepad once in a while but my usage on both games. They were at home but they were not my home. The mad was my home until college where I made two new friends: Composition Book (CB) and Bill Joy's `vi`. CB seemed like an elegant upgrade from the Mead spiral but it didn't last. CB's lines were too tall and the pages were too short, the worst of both worlds. The speckled black and white covers were its most useful part which was it's beauty as no one would ever have guessed. Compared to the Mead, the CB's were downright elegant. That leaves `vi`. Mrs. Marian Manyo prepared a handout to help us learn `vi`. It was a single page. `SunOS` was at a whopping version of 5.7 yet the handout left ink on your finger suggesting that it a mimeograph instead of a Xerox that deposited it into our awaiting writing fingers. It has seven commands that I learned: open, close, save, close and save, insert and command mode. That worked to store my todo list for years. This as before CVS usurped RCS so backups were basic. My workflow had two states: existence and non-existence. Simple. This well for another nineteen years. Until I followed the GTD approach but in a simpler ad-hock form. Emacs was never part of my life even when it lived next door to vi. Growing up on the cusp of beefy IDEs living on puny desktops most schools weren't rooted in the tradition of LISP and Emacs and I never joined it. Scheme got my attention though and it sucked me into Emacs, but it didn't keep me. Twice I tried. Nope. Three times I tried and at the end I liked Emacs a little bit worse. Then something funny happened: I wanted to learn OCaml. Emacs was the only good editor for it, so I dug up Emacs. Then I wanted to learn SML and go figure: Emacs was the only good editor for it. The same story was retold ten or fifteen times. The proverbial lotus kept expanding every time you touched because not only did satisfy your ideas but it gave you new ones. Totally hooked, I am still keeping my todo lists in ASCII text files. Then three years I heard about Org-Mode. My first Org file had you guessed it: lists converted into headings. Oops, that wasn't right so I converted them all to lists. That worked well for years. Then I got interested in publishing and spent another year with headings and lists. A year or so ago I got tired of my ad-hock workflow I checked if Org had a way to deal with workflow. Of course it did! Then I got interested in tracking state changes, and of course it had that. Beautiful. This whole story lives here to introduce refile. When I moved headings I moved them by selecting the whole subtree, killing it, and pasting it somewhere else. Fifteen or twenty times I lost the heading. Sure VCS saved it but it was a hassle and a mess, a total frustration. My error was not marking the entire subtree. Oops. When I searched for a solution I ended up on the refile page. Of course that is how you do it. Now *what* was the *it* that I wanted to do?

E. Desire

`ID: org_gcr_2017-05-12_mara:E2BDFB86-4DCC-41BE-9F00-BFB085E547D6`

Three or four Org-Mode files sit open in buffers for the entire year. Once in a while there are new ones added but they usually replace one of the original four. Exceptions are one-offs like provisioning scripts for my systems at work or at home. Rarely a heading grows so large that it graduates to its own file. This is by design: I refuse to pre-optimize and grow a menagerie files to house every ever taxonomy conceivable only to end up retiring the file having closed only a single heading.

Here is how my usage looks to me:

- 99.00% of the time I am working in the same three files
  - Refiling across subtrees at level two
- <00.90% of the time I am moving headings to level zero, or the top level of the document
- <00.09% of the time I am moving headings into a different file

My predictable and small usage makes for an easy system configuration goal below.

F. Implementation

`ID: org_gcr_2017-05-12_mara:BB186822-AFD2-4609-8CFB-DD7542E3D532`

G. Core

`ID: org_gcr_2017-05-12_mara:59F948C7-2606-4637-8EF6-4922BD231A9B`

Because 99% of the time I will refile headings as sub-headings within the same file I never need to configure `Refile` file targets in `org-refile-targets`. Refiling at any depth makes

```
(setq org-refile-targets '((nil . (:maxlevel . 10))))
(setq org-refile-use-outline-path t)
```

doing so natural, flexible and efficient

Because <90% I'm moving headings to level zero I want to

- Make it easy when I need it which is not often
- Make it difficult the rest of the time because I don't want to accidentally move headings there. It isn't the end of the world I just don't want to do it accidentally because I will be *rarely* doing so in the first place.
- Single previx argument (4)

Because <09% of the time I am moving headings into a new file

- Make it as easy and a difficult as moving to the top-level
- Prompt for the file name
  - If it does not exist `Org` will transparently create it
- Perform the move

- Single previx argument (16)

```
(defun help/org-refile (arg)
  "Refile to /level/ in /file/ by using use /prefix args/: 2+/this/[none], 1+/t
  (interactive "P")
  (cond
   ((not (null arg))
    (let ((val (car current-prefix-arg))
          (current-prefix-arg nil)
          (org-refile-use-outline-path 'file)
          (org-reverse-note-order nil))
      (cond ((= val 4)
             (call-interactively 'org-refile))
            ((= val 16)
             (let* ((fil (read-file-name "Enter destination file: "))
                    (xfil (expand-file-name fil))
                    (_ (when (not (file-exists-p xfil))
                         (with-temp-file xfil (insert))))
                    (org-refile-targets
                     `((,xfil :maxlevel . 10))))
               (progn (call-interactively 'org-refile)
                      (find-file xfil)))))))
   (t
    (call-interactively 'org-refile))))
```

H. General Operation

ID: org_gcr_2017-05-12_mara:16842F15-40C2-4E68-83F2-EBD425569FCB

- Refiles are always coming from another headline to this headline. They are new and need to be prioritized. When they are in front it is easy to notice because your eyes go there first (even know you *are* the one refiling)

  ```
  (setq org-reverse-note-order t)
  ```

- Select headings with the full path immediately instead of with clunky depth-first navigation

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

- Create a new parent heading destination first instead of using this GUI to do it because it is error prone to do it in the naviation GUI despite the available confirmation message

  ```
  (setq org-refile-allow-creating-parent-nodes nil)
  ```

- Track refile operations to make sense of why the `Refile` occurred in the same spirit of how the workflow state changes are recorded

  ```
  (setq org-log-refile 'note)
  ```

I. Properties

ID: org_gcr_2017-05-12_mara:D73BDF5F-AA12-4832-9B28-CADEC71D2B6C

Select a region, yank it, maybe choose a property under the current headline and set it's value to the killed text.

```
(defun org-read-entry-property-name ()
  "Read a property name from the current entry."
  (let ((completion-ignore-case t)
        (default-prop (or (and (org-at-property-p)
                               (org-match-string-no-properties 2))
                          org-last-set-property)))
    (org-completing-read
     (format "Property [%s]: " (if default-prop default-prop ""))
     (org-entry-properties nil nil)
     nil nil nil nil default-prop)))

(defun my-org-region-to-property (&optional property)
  "Copies the region as value to an Org-mode property"
  (interactive)
  ;; if no region is defined, do nothing
  (if (use-region-p)
      ;; if a region string is found, ask for a property and set property to
      ;; the string in the region
      (let ((val (replace-regexp-in-string
                  "\\`[ \t\n]*" ""
                  (replace-regexp-in-string "[ \t\n]*\\'" ""
                                            (substring (buffer-string)
                                                       (- (region-beginning) 1)
                                                       (region-end))))
                 )
            ;; if none was stated by user, read property from user
            (prop (or property
                      (org-read-entry-property-name))))
        ;; set property
        (org-set-property prop val))))
```

J. Tables

ID: org_gcr_2017-05-12_mara:862B1BEA-4FAD-4FA1-8666-39E2FFA2BB4F

Essential for using tables with split windows.

```
(use-package org-table-sticky-header
  :ensure t
  :diminish org-table-sticky-header-mode
  :config
  (add-hook 'org-mode-hook #'org-table-sticky-header-mode))
```

K. Unclassified

ID: org_gcr_2017-05-12_mara:FD7E89EA-72DC-4395-AE07-4CAA66B8BD28

When running in a GUI, I would like linked images to be displayed inside of Emacs.

```
(setq org-startup-with-inline-images (display-graphic-p))
```

Use Ido completion in Org-Mode.

```
(setq org-completion-use-ido t)
(setq org-completion-use-iswitchb nil)
```

Org-Mode lets you use single letter commands to do stuff on headers. I like to use `c` for cycling the header expansion.

```
(setq org-use-speed-commands t)
```

Ask before execution of shell links. This might seem like an Evaluation activity. It is. It is interactive.

```
(setq org-confirm-shell-link-function 'y-or-n-p)
```

Ask before execution of Emacs-Lisp.

```
(setq org-confirm-elisp-link-function 'y-or-n-p)
```

Make sure that incomplete TODO entries prevent the enclosing parent from ever turning to DONE.

```
(setq org-enforce-todo-dependencies t)
```

Allow the mouse to do Org-Mode things like expand and collapse headings.

```
(when (display-graphic-p)
  (use-package org-mouse))
```

Use a real ellipsis to render an ellipsis for Org-Mode stuff like showing that a header is collapsed. Artur Artur go me thinking that an arrow would be more expressive; in particular revealing that there is more content to be "unrolled" below the current line.

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

It is easy to see indentation of headlines without having to count asterisks, so don't show them, only show the significant and last one.

```
(setq org-hide-leading-stars t)
```

Maximize character space for writing. Do not indent according to the outline node level because it would waste a lot of space. Ident the next body just like any other text document.

```
(setq org-adapt-indentation nil)
```

Display emphasized text as you would in a WYSIWYG editor.

```
(setq org-fontify-emphasized-text t)
```

Use Unicode characters to visualize things like right arrow eg $\rightarrow$. Most of those symbols are correctly exported to the destination format. The most obvious is this example in LaTeX versus Text.

```
(setq org-pretty-entities t)
```

Enable sub and super scripts **only** when wrapped in squiggly brackets.

```
(setq org-use-sub-superscripts '{})
```

Highlight LaTeX and related markup.

Normally, I don't do any syntax highlighting, as I believe that should be delegated to source buffers, thinking that to do otherwise is distracting. However, I already do configure subscripts and Greek letters to be displayed with syntax highlighting, because I want to indicate to the human reader that they are special, and specifically *not*-Unicode. Do the same thing for LaTeX and related markup.

```
(setq org-highlight-latex-and-related '(latex script entities))
```

Allow "refactoring" of Footnotes between documents.

```
(setq org-footnote-define-inline t)
(setq org-footnote-auto-label 'random)
(setq org-footnote-auto-adjust nil)
(setq org-footnote-section nil)
```

This is an amazingly easy way to screw up your document. The more you edit org docs, the more you realize how you must truly protect it.

```
(setq org-catch-invisible-edits 'error)
```

Though I am not delving deep, it is hard not to want to customize some stuff and perhaps this is the start. Even though I enabled this, I don't think that I ever used it.

```
(setq org-loop-over-headlines-in-active-region t)
```

It is *almost always* faster to work with org documents when they are fully expanded. Anyway, the structure cycling makes it really, really easy to get an *outline view* again.

```
(setq org-startup-folded "nofold")
```

When images are displayed in the buffer, display them in their actual size. As the operator, I want to know their true form. Any modifications required for export will be stated explicitly. Override this by setting `#+ATTR_ORG: :width N` in the file.

```
(setq org-image-actual-width t)
```

Hide the delimeter for emphasized text. Unicode characters break table alignment.

```
(setq org-hide-emphasis-markers t)
```

Realign tables automatically.

```
(setq org-startup-align-all-tables t)
```

Always use Unicode checkboxes.

```
(setq org-html-checkbox-type 'unicode)
```

You may display syntax highlighting for code in source blocks. I don't.

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

When edit mode is exited, the option exists to automatically remove empty opening and closed lines for the source block. Never do this. The thing is that I forgot why. When I was working on a recent analysis with R there was a space appearing in the opening and closing line of the source block that didn't appear in the source editing buffer. That surprised me. I am sure that I've forgotten why this is the case. I don't like it because you add a bunch of empty lines in the source buffer for every source block. With that in mind I will enable this feature and try it out again.

```
(setq org-src-strip-leading-and-trailing-blank-lines t)
```

The source block buffer may be configured to appear in a few different places. For a while I really liked `reorganize-frame` because sometimes you want to be able to see the code you are editing in edition to the rest of the document. At least that is what I am telling myself. Once I learned you could change it I realized that 1 I should have asked if it could be changed and 2 I should have changed it. The flow that I've got configured here is that you are either in the source document where code blocks are not highlighted or you are in the source block so you are editing in a buffer that is full-fledged HELP. That is the best way so you can focus completely on each task at hand in the ideal mode for that task. Anything else results in distractions and errors.

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

Org-Mode has a really nice feature that hitting `C-c C-c` will generally just do the *right thing*. It is really nice. That feature extends to source blocks of course. Ironically I had a typo here, typing *of curse* instead of *of course*. The thing is that you really, really need to develop a personal workflow, and then configure the tool to enable it. The more I learn about Org-Mode, the more leery I am about making it really easy to evaluate code. I want it to be a really, really specific and decided action to evaluate a code block, so don't make it so easy as `C-c C-c`.

```
(setq org-babel-no-eval-on-ctrl-c-ctrl-c t)
```

Configure the system to successfully use `vc-next-action` while editing a Source-Block. Before perfoming the edit, check if it is Org-Mode and exit the Source-Block Buffer (SBB). If this system stays in the SBB when calling `vc-next-action` the entire contents of the buffer are escaped as Org-Mode source code upon returning to the source buffer (this). Do the same thing before any version control modes that would result in the same condition.

```
(defun help/vc-next-action ()
  "If in org source block, exit it before `vc-next-action'."
  (interactive)
  (when (condition-case nil
            (org-src-edit-buffer-p)
```

```
          (error nil))
    (org-edit-src-exit))
  (vc-next-action nil))
(defun help/magit-status ()
  "If in org source block, exit it before ‘magit-status’."
  (interactive)
  (when (condition-case nil
            (org-src-edit-buffer-p)
          (error nil))
    (org-edit-src-exit))
  (magit-status))
```

Never use the original version.

```
(setq org-edit-src-code nil)
```

Easily wrap text in Org-Mode. This is not used by the rest of HELP because Smartparens provides that functionality for programming modes.

```
(use-package wrap-region
  :ensure t
  :config
  :diminish wrap-region-mode
  :config
  (add-hook 'org-mode-hook 'wrap-region-mode))
```

Bold.

```
(wrap-region-add-wrapper "*" "*" nil 'org-mode)
```

Italic.

```
(wrap-region-add-wrapper "/" "/" nil 'org-mode)
```

`Verbatim`.

```
(wrap-region-add-wrapper "=" "=" nil 'org-mode)
```

`Code`.

```
(wrap-region-add-wrapper "~" "~" nil 'org-mode)
```

~~Strike-Through~~.

```
(wrap-region-add-wrapper "+" "+" nil 'org-mode)
```

Minimize Macro text.

```
(setq org-hide-macro-markers t)
```

Follow links without using the mouse or more.

```
(setq org-return-follows-link t)
```

L. Keybindings

`ID: org_gcr_2017-05-12_mara:F5256DB3-BD52-409B-80CF-6E601A29AA4A`

Started questioning why after hitting RETURN while in lists I have to hit TAB to get indented properly. Kind of a dead giveaway that I should be return-and-indenting! Looked at `org-return` to find that it has an argument about indenting and then saw that `org-return-indent` passes it for you. With that in mind, RETURN is bound to that now. Now HELP has four different kinds of "returns" in Org in order of likelihood of usage:

org-return-indent Make it really easy to work in existing list items, headings, and tables

- This is listed first because I often go back to modify entries
- <return> because it is used the most

org-meta-return Make it really easy to add new list items, headings, and table contents

- M-<return> because the binding comes with Org

electric-indent-just-newline For when I want to break out of the default Org indentation to start working at the beginning of the line for example when I'm done working in a list or have just created a new heading

- C-M-<return> because it is next step "lower" in the binding

help/smart-open-line When I want to insert a new line between the current and next line then position the cursor correctly indented at the start of it.

- s-<return> because it is that is the last place in the modifier key chain

```
(help/not-on-gui
 (define-key org-mode-map (kbd "RET") #'org-return-indent)
 (define-key org-mode-map (kbd "C-M-RET") #'electric-indent-just-newline))
(help/on-gui
 (define-key org-mode-map (kbd "<return>") #'org-return-indent)
 (define-key org-mode-map (kbd "C-M-<return>") #'electric-indent-just-newline))
```

M. Row 5

```
ID: org_gcr_2017-05-12_mara:B493B118-37D7-4C49-B300-EFBBEF91F0D6
```

```
(global-set-key [(shift f6)] #'kmacro-name-last-macro)
(global-set-key [(f6)] #'insert-kbd-macro)
(global-set-key [(f5)] #'elmacro-show-last-macro)
(define-key org-mode-map (kbd "s-6") #'org-babel-load-in-session)
(define-key org-mode-map (kbd "s-7") #'org-babel-switch-to-session)
(define-key org-mode-map (kbd "s-8") #'org-babel-switch-to-session-with-code)
(define-key org-mode-map (kbd "s-9") #'org-todo)
```

Easily manipulate lists and headlines staying close to home.

```
(key-chord-define org-mode-map "U*" #'org-metaup)
(key-chord-define org-mode-map "I(" #'org-metadown)
(key-chord-define org-mode-map "u8" #'org-metaleft)
```

```
(key-chord-define org-mode-map "i9" #'org-metaright)
```

N. Row 4

```
ID: org_gcr_2017-05-12_mara:ADB2113A-6D9F-4145-9D78-1DDFE5A2C916
```

```
(define-key org-mode-map (kbd "s-]") (lambda () (interactive)
                                       (message "Removing all source block resu
                                       (help/org-2every-src-block
                                        'org-babel-remove-result)
                                       (message "Done removing all source block
(define-key org-mode-map (kbd "s-y") #'help/safb-org-babel-execute-buffer)
(define-key org-mode-map (kbd "s-u") #'help/safb-org-babel-execute-subtree)
(define-key org-mode-map (kbd "s-U") #'org-mark-ring-goto)
(define-key org-mode-map (kbd "s-i") #'org-babel-execute-src-block)
(define-key org-mode-map (kbd "s-o") #'org-babel-remove-result)
(define-key org-mode-map (kbd "s-p") #'org-babel-execute-maybe)
(define-key org-mode-map (kbd "s-[") #'org-babel-remove-inline-result)
(define-key org-mode-map (kbd "M-{") (lambda () (interactive) (insert "- [ ] "))
```

O. Row 3

```
ID: org_gcr_2017-05-12_mara:AA97F835-E969-43E4-AC84-B3CF472B8726
```

```
(define-key org-mode-map (kbd "C-c C-k") nil)
(define-key org-mode-map (kbd "s-h") #'help/safb-org-babel-tangle)
(define-key org-mode-map (kbd "s-j") #'org-babel-next-src-block)
(define-key org-mode-map (kbd "s-k") #'org-babel-previous-src-block)
(define-key org-mode-map (kbd "s-l") #'help/safb-org-edit-src-code)
(define-key org-mode-map (kbd "s-;") #'help/safb-help/org-babel-demarcate-block)
(define-key org-mode-map (kbd "C->") #'(lambda () (interactive) (insert "\\rarr
(defun help/org-insert-subscript (arg)
  "Maybe insert a subscript with the postfix space."
  (interactive "MSubscript: ")
  (if (s-blank? arg)
      (message "Nothing to insert")
    (insert "_{" arg "} ")))
(define-key org-mode-map (kbd "s-s") #'help/org-insert-subscript)
(defun help/org-insert-superscript (arg)
  "Maybe insert a super with the postfix space."
  (interactive "MSuperscript: ")
  (if (s-blank? arg)
      (message "Nothing to insert")
    (insert "^{" arg "} ")))
(define-key org-mode-map (kbd "s-S") #'help/org-insert-superscript)
```

Because I only use this for Org-Mode.

```
(define-key global-map (kbd "s-o") nil)
```

P. Row 2

```
ID: org_gcr_2017-05-12_mara:680824E4-2C0B-4E0F-BA6E-62AA84D484E4
```

```
(define-key org-mode-map (kbd "s-n") #'org-babel-view-src-block-info)
(define-key org-mode-map (kbd "s-m") #'org-babel-expand-src-block)
(define-key org-mode-map (kbd "s-,") #'org-babel-open-src-block-result)
(define-key org-mode-map (kbd "s-.") #'org-time-stamp)
```

Because I only use this for Org-Mode.

```
(define-key global-map (kbd "s-m") nil)
```

Q. Hydra

```
ID: org_gcr_2017-05-12_mara:08A11E45-D41A-44D6-945D-45DF85F8D037
```

```
(defhydra help/hydra/right-side/org-mode (:color blue
                                                 :hint nil)
  "
_1_ SHA-1-hash _2_ +imgs _3_ -imgs _4_ id-create _5_ toggle-macro
_q_ ←/w-code _w_ tbletfld _e_ g2nmrst _r_ help/org-refile _R_ g2nms-b _t_ g2s-b
_a_ archive-subtree _s_ oblobigst _u_ goto _h_ dksieb _k_ ob-check-src-blk _l_
_c_ org-fill-para _b_ swtch2sessn _n_ n2sbtre _m_ mark-subtree"
  ;; Row 5
  ("1" org-babel-sha1-hash)
  ("2" org-display-inline-images)
  ("3" org-remove-inline-images)
  ("4" org-id-get-create)
  ("5" help/org-toggle-macro-markers)
  ;; Row 4
  ("q" org-babel-switch-to-session-with-code)
  ("w" org-table-edit-field)
  ("e" org-babel-goto-named-result)
  ("r" help/org-refile)
  ("R" org-babel-goto-named-src-block)
  ("t" org-babel-goto-src-block-head)
  ("u" org-goto)
  ("p" my-org-region-to-property)
  ;; Row 3
  ("a" org-archive-subtree-default)
  ("s" org-babel-lob-ingest)
  ("g" org-goto)
  ("h" org-babel-do-key-sequence-in-edit-buffer)
  ("H" org-babel-insert-header-arg)
  ("k" org-babel-check-src-block)
  ("l" org-lint)
  ;; Row 2
  ("c" org-fill-paragraph)
  ("b" org-babel-switch-to-session)
  ("n" org-narrow-to-subtree)
  ("m" org-mark-subtree)
  ("M" org-mark-element))
(key-chord-define org-mode-map "hh" #'help/hydra/right-side/org-mode/body)
```

Save all buffers before working with Exports.

```
(define-key org-mode-map (kbd "C-c C-e") #'help/safb-org-export-dispatch)
```

Make `s-l` do the same thing to leave the Source-Block-Buffer.

```
(define-key org-src-mode-map (kbd "s-l") #'org-edit-src-exit)
```

Easily enter guillemots.

```
(key-chord-define org-mode-map "<<" (lambda () (interactive) (insert "\\laquo{}
(key-chord-define org-mode-map ">>" (lambda () (interactive) (insert "\\raquo{}
```

iii. Transliteration

```
ID: org_gcr_2017-05-12_mara:D15B4305-4503-4D8C-8CDA-258FA4AF7EB2
```

Black board bold characters.

```
(use-package blackboard-bold-mode
  :ensure t)
```

Fraktur characters.

```
(use-package fraktur-mode
  :ensure t)
```

```
(defhydra help/hydra/transliterate (:color blue :hint nil)
  "
_b_ black-board-bold _f_ fraktur
"
  ("b" blackboard-bold-mode)
  ("f" fraktur-mode))
```

iv. Unicode

```
ID: org_gcr_2017-05-12_mara:ADAE0A1B-3FD8-4605-8611-388E5ADBAFA8
```

Easily understand the current character.

```
(defun help/describe-char ()
  "Evaluate ‘describe-char’ and then ‘other-window’."
  (interactive)
  (call-interactively #'describe-char)
  (call-interactively #'other-window))
```

Easily identify Unicode homoglyphs most often utilized by pranksters.

Good reference for Unicode character study and exploration.

Used intermittently so don’t diminish it.

```
(use-package unicode-troll-stopper
  :ensure t)
```

Easily escape and un-escape Unicode hex notation.

```
(use-package unicode-escape
  :ensure t
```

```
  :config)
```

Warn of UTF BOM bytes via skeeto.

> The UTF-8 representation of the BOM is the byte sequence 0xEF,0xBB,0xBF.

Test using Hexl mode.

```
(defun warn-if-utf-8-bom ()
  "Warn if UTF-8 BOM bytes are present.

Attribution: URL ‘https://www.reddit.com/r/emacs/comments/4tw0iz/can_i_have_a_warni
  (let ((name (symbol-name buffer-file-coding-system)))
    (when (string-match-p "utf-8-with-signature" name)
      (message "Call the BOM squad! This UTF-8 file has a BOM!"))))

(add-hook 'find-file-hook #'warn-if-utf-8-bom)
```

v. Dash

```
ID: org_gcr_2017-05-12_mara:B541F7EE-BF7E-4055-9EE6-3B0AC34A1602
```

```
(use-package dash-at-point
 :ensure t)
```

(c) Applied Mathematics

```
ID: org_gcr_2017-05-12_mara:423BA6AA-5F7D-450D-B686-78A028A37FE2
```

i. APL

```
ID: org_gcr_2017-05-12_mara:5CB03225-60DE-4AC2-8A67-3D7D21BF7818
```

```
(use-package gnu-apl-mode
  :ensure t
  :init
  (setq gnu-apl-show-keymap-on-startup nil)
  (setq gnu-apl-show-apl-welcome nil)
  (setq gnu-apl-show-tips-on-start nil)
  (setq gnu-apl-mode-map-prefix "C-M-s-")
  (setq gnu-apl-interactive-mode-map-prefix gnu-apl-mode-map-prefix)
  :config
  (defun em-gnu-apl-init ()
    (setq buffer-face-mode-face 'gnu-apl-default)
    (buffer-face-mode))
  (add-hook 'gnu-apl-interactive-mode-hook 'em-gnu-apl-init)
  (add-hook 'gnu-apl-mode-hook 'em-gnu-apl-init)
  (defun help/gnu-apl-runningp ()
    (interactive)
    (let ((session (gnu-apl--get-interactive-session-with-nocheck)))
      (if session 'ON 'OFF)))
  (defun help/start-gnu-apl ()
    (interactive)
    (if (equal (help/gnu-apl-runningp) 'ON) (message "GNU APL is already ON.")
```

```
      (call-interactively 'gnu-apl)))
  (defun help/stop-gnu-apl ()
    (interactive)
    (if (equal (help/gnu-apl-runningp) 'OFF) (message "GNU APL is already OFF.")
      (progn
        (gnu-apl-switch-to-interactive)
        (switch-to-buffer "*gnu-apl*")
        (insert ")off")
        (comint-send-input))))
  (defhydra help/hydra/gnu-apl (:color blue
                                       :hint nil)
    "
GNU APL is: %(help/gnu-apl-runningp)
 _y_ eval-buffer _u_ eval-region _i_ eval-line _o_ eval-function
  _f_ apropos-symbol _g_ help-symbol _h_ keyboard _j_ next _k_ previous
   _q_ quit _c_ start APL _v_ stop APL _b_ switch to APL _n_ switch back
"
    ("i" help/gnu-apl-eval-line)
    ("o" gnu-apl-interactive-send-current-function)
    ("j" (lambda () (interactive) (call-interactively 'next-logical-line)) :exit ni
    ("k" (lambda () (interactive) (call-interactively 'previous-logical-line))
     :exit nil)
    ("b" gnu-apl-switch-to-interactive)
    ("u" gnu-apl-interactive-send-region)
    ("h" gnu-apl-show-keyboard)
    ("y" gnu-apl-interactive-send-buffer)
    ("t" gnu-apl-trace)
    ("f" gnu-apl-apropos-symbol)
    ("g" gnu-apl-show-help-for-symbol)
    ("c" help/start-gnu-apl)
    ("v" help/stop-gnu-apl)
    ("n" (lambda () (interactive) (other-window -1)))
    ("q" nil))
  (key-chord-define gnu-apl-mode-map "hh" #'help/hydra/gnu-apl/body)
  (key-chord-define gnu-apl-interactive-mode-map "hh" #'help/hydra/gnu-apl/body)
  (defun help/gnu-apl-eval-line ()
    "Evaluate this line and move to next."
    (interactive)
    (end-of-line)
    (set-mark (line-beginning-position))
    (call-interactively 'gnu-apl-interactive-send-region)
    (deactivate-mark)
    (call-interactively 'next-logical-line))
  (define-key gnu-apl-mode-map (kbd "C-<return>") #'help/gnu-apl-eval-line)
  (defun help/gnu-apl-interactive-mode-hook-fn ()
    (nlinum-mode)
    (rainbow-mode))
  (add-hook 'gnu-apl-interactive-mode-hook
```

```
            #'help/gnu-apl-interactive-mode-hook-fn)
  (add-to-list 'org-babel-load-languages '(gnu-apl . t)))
```

ii. Emacs Speaks Statistics (ESS)

```
ID: org_gcr_2017-05-12_mara:92D8EC6D-9E6B-482A-B100-66CD75870EF5
```

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

Display object documentation.

```
(setq ess-eldoc-show-on-symbol t)
```

Data viewing:

- **Never** rely upon on the REPL for data viewing
  - Will mix up exploratory code with data
    - Can't easily distinguish between code and data
    - Distracting you
    - Breaking your flow
- Sometimes
  - You end up somewhere
    - And the `ess` buffer cursor is at the top!
    - No problem, call `ess-switch-to-end-of-ESS`
- Make it easier to know what object values are.
  - `ess-describe-object-at-point`

```
(setq ess-describe-at-point-method 'tooltip)
```

Always start `ess` within the curent `emacs` frame, it doesn't need to be separate.

```
(setq inferior-ess-same-window nil)
(setq inferior-ess-own-frame nil)
```

Help buffers all belong in the same frame.

```
(setq ess-help-own-frame nil)
```

When commands are executed, display their output within the current buffer, rather than to a new dedicated buffer for them.

```
(setq ess-execute-in-process-buffer t)
```

When you cycle between a the REPL buffer and the script, you get to the process buffer, you will go to the end of the buffer. This setting is specifically to handle a buffer that is scrolling when you want to see the last result and will scroll back after the fact to see the history.

```
(setq ess-switch-to-end-of-proc-buffer t)
```

Use typical auto completion in buffers here, but don't do it when the next char is a symbol or closed paren.

```
(setq ess-tab-complete-in-script t)
(setq ess-first-tab-never-complete 'symbol-or-paren-or-punct)
```

Use `ido` completion whenever possible.

```
(setq ess-use-ido t)
```

Use `eldoc` for this mode. Always show it when the point is on a symbol. Try to keep help strings at 10 chars or less.

```
(setq ess-use-eldoc t)
(setq ess-eldoc-show-on-symbol t)
(setq ess-eldoc-abbreviation-style 'normal)
```

These functions are mentioned, and I am not sure where or how to use them yet, but Vitalie Spinu mentioned them as being useful:

- `comint-previous-matching-input-from-input`
- `comint-history-isearch-backward-regexp`

For a while I used `ess-eval-buffer-and-go`, but now I know that it is insanely faster to use `ess-eval-buffer` instead. Previously I've read people saying that, and it is true.

Philosophy

The current `ESS` maintainers philosophies about how to maintain an R code-base make sense to me and are virtually the same as my own. Quite simply, the rule is that the code artifacts are the single source of system definition. Consequently, the system should be configured in this manner:

We want to keep dump files after loading them; never delete them. The idea is that if we use them, then they are a valid part of the system definition and need to be kept.

```
(setq ess-keep-dump-files +1)
```

`ESS` allows us to quite easily modify live R objects and functions. It provides this functionality via `ess-dump-object-into-edit-buffer`. These changes are considered to be experimental, and not part of the master record according to our philosophy. As such, we don't care to know that these new versions ever existed and their record will be forgotten from history. In other words, that new, modified version of the object or function, is never saved to a file for later reuse.

```
(setq ess-delete-dump-files nil)
```

Since our systems are entirely file-based, the entirety of the system most likely lives in different files. Before loading any file for sourcing, save any ESS source buffers. This approach is in addition to two other things: (1) Emacs is auto-saving every file buffer quite frequently and (2) there is advice before every manual `eval` call so that the buffers and their files stay in sync. Yes, it is really that important.

```
(setq ess-mode-silently-save +1)
```

ESS executes code in another process. That is no secret. When displaying output from that code running in another process though, it can look like Emacs is locking up. That is not the case: `https://stackoverflow.com/questions/2770523/how-can-i-background-the-r-process-in-ess-emacs`. What is happening that Emacs is waiting for the output. Configure this mode to continue to accept user input, which is obviously critical, but don't wait for the process to provide its output. Instead, all output is printed after the last input lines. What we gain is perceived speed, and what we lose is the nice sequential this/that/this/that we get from a typical REPL interaction. As I write this, I'm not totally sure how this will work, but the documentation and post are consistent and describe what I had wanted here so I will give it a try and see how it goes.

```
(setq ess-eval-visibly 'nowait)
```

iii. SAS (ESS)

```
ID: org_gcr_2017-05-12_mara:BACA8D4C-7DAD-4457-B089-345BAD1BF3A2
```

iv. R (ESS)

```
ID: org_gcr_2017-05-12_mara:6AE60368-3D9F-400D-AAEC-A9B99F5CE480
```

Enable a debugger.

```
(setq ess-use-tracebug t)
```

Configure debugger search path per-project.

```
(setq ess-tracebug-search-path '())
```

Easily nevigate errors.

```
(define-key compilation-minor-mode-map [(?n)] #'next-error-no-select)
(define-key compilation-minor-mode-map [(?p)] #'previous-error-no-select)
```

Diminish watched variable font-size.

```
(setq ess-watch-scale-amount -1)
```

When `ess` starts, or when R starts, it takes the current directory as its working directory. This is totally fine; so don't ask what the working directory should be.

```
(setq ess-ask-for-ess-directory nil)
```

My preference is for ESS to quit and not ask me whether or not I am sure. There is an intentional line-break after the closing round bracket because that is the approach of the original value here.

```
(setq inferior-ess-exit-command "q('no')
")
```

Visualize just about anything with `ess-R-object-popup`.

```
(use-package ess-R-object-popup
  :ensure t)
```

Rdired is another way to work with object:

- `ess-rdired`
- View, delete, plot, and update buffer (ala *revert*) are single key commands

```
(autoload 'ess-rdired "ess-rdired")
```

Visualize data frames better:

- `ess-R-dv-ctable`
- `ess-R-dv-pprint`

```
(use-package ess-R-data-view
  :ensure t)
```

inlineR

- *Not* a competitor to `org-mode`
- Ultra lightweight LP, really

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

Documentation:

- Whole section on native documentation; I'll re-visit as needed.
- Roxygen, too.

`ess-developer` helps you to easily work within specific name-spaces.

Store history files and dump files in a single known location. If that location doesn't exist, then make it.

```
(setq help/r-dir "~/.R/")
(defun help/make-warn-R-dir ()
  "Handle of R directory misconfiguration."
  (interactive)
  (unless (f-directory? help/r-dir)
    (progn
      (message "Couldn't find %S... creating it." help/r-dir)
      (f-mkdir help/r-dir))))
(help/make-warn-R-dir)
(setq ess-history-directory help/r-dir)
(setq ess-source-directory help/r-dir)
```

Since I'm using R for everything, configure *everything* to be using R.

```
(setq inferior-ess-program "R")
(setq inferior-R-program-name "R")
(setq ess-local-process-name "R")
```

Handle rdoc and rmd files, though I have never used them... yet.

```
(add-to-list 'auto-mode-alist '("\\.rd\\'" . Rd-mode))
(add-to-list 'auto-mode-alist '("\\.Rmd$" . r-mode))
```

Make it really easy to search the R archives for anything.

```
(local-set-key (kbd "C-c C-. S") #'ess-rutils-rsitesearch)
```

Make it really easy to do common stuff for R with good keybindings.

```
(use-package ess-rutils
  :config
  (setq ess-rutils-keys t))
```

`r-autoyas` does argument completion. I had it working nice, and didn't use it for a while, and now it doesn't work. This needs some TLC.

```
(use-package r-autoyas
  :ensure t
  :config
  (setq r-autoyas-debug t)
  (setq r-autoyas-expand-package-functions-only nil)
  (setq r-autoyas-remove-explicit-assignments nil))
```

Save two spaces showing function information in the mini-buffer.

```
(setq ess-R-argument-suffix "=")
```

Don't use the default assignment binding and allow underscores in names.

```
(setq ess-S-assign-key (kbd "C-<"))
(ess-toggle-S-assign-key t)
(ess-toggle-underscore nil)
```

Don't save the workspace when you quit R and don't restore **ANYTHING** when you start it, either. This adheres to the philosopy that the system is file based.

```
(setq inferior-R-args "--no-save --no-restore")
```

R mode hook.

```
(defun help/R-mode-hook-fn ()
  (local-set-key (kbd "s-6") #'ess-switch-to-end-of-ESS)
  (local-set-key (kbd "s-7") #'ess-rdired)
  (local-set-key (kbd "s-8") #'ess-R-dv-ctable)
  (local-set-key (kbd "s-9") #'ess-R-dv-pprint)
  (local-set-key (kbd "s-y") #'r-autoyas-expand)
  (local-set-key (kbd "s-o") #'ess-describe-object-at-point)
  (local-set-key (kbd "s-p") #'ess-R-object-popup)
  (local-set-key (kbd "C->") #'(lambda () (interactive) (insert " -> ")))
  (key-chord-define-local "<<" #'(lambda () (interactive) (insert " <<- ")))
  (key-chord-define-local ">>" #'(lambda () (interactive) (insert " ->> ")))
  (key-chord-define-local "<>" #'(lambda () (interactive) (insert " %<>% ")))
  (local-set-key (kbd "s-.") #'(lambda () (interactive) (insert " %>% ")))
  (r-autoyas-ess-activate)
  (help/turn-on-r-hide-show)
```

```
  (lambda () (add-hook 'ess-presend-filter-functions
                 (lambda ()
                   (warn
                    "ESS now supports a standard pre-send filter hook. Please upd
  (ess-set-style 'RRR))

(add-hook 'R-mode-hook #'help/R-mode-hook-fn)

(defun help/turn-on-r-hide-show ()
"Attribution: URL https://github.com/mlf176f2/EmacsMate/blob/master/EmacsMate-ess.
  (when (string= "S" ess-language)
    (set (make-local-variable 'hs-special-modes-alist) #'((ess-mode "{" "}" "#" ni
    (hs-minor-mode 1)
    (when (fboundp 'foldit-mode)
      (foldit-mode 1))
    (when (fboundp 'fold-dwim-org/minor-mode)
      (fold-dwim-org/minor-mode))))

(defun help/Rd-mode-hook-fn ()
  (help/R-mode-hook-fn))

(add-hook 'Rd-mode-hook #'help/Rd-mode-hook-fn)

(defun help/inferior-ess-mode-hook-fn ()
  (help/R-mode-hook-fn))

(add-hook 'inferior-ess-mode-hook #'help/inferior-ess-mode-hook-fn)

(defun help/ess-rdired-mode-hook-fn ()
  "Personal customizations."
  (interactive)
  (turn-on-stripe-buffer-mode)
  (stripe-listify-buffer))

(add-hook 'ess-rdired-mode-hook #'help/ess-rdired-mode-hook-fn)
```

v. Scheme (LISP)

`ID: org_gcr_2017-05-12_mara:80DF94ED-4868-47CE-BAC8-978863BEF5AF`

Handle all file extensions:

- Traditional.

```
(add-to-list 'auto-mode-alist '("\\.scm\\'" . scheme-mode))
(add-to-list 'auto-mode-alist '("\\.ss\\'" . scheme-mode))
```

- Racket.

```
(add-to-list 'auto-mode-alist '("\\.rkt\\'" . scheme-mode))
```

- R6RS.

```
      (add-to-list 'auto-mode-alist '("\\.sls\\'" . scheme-mode))
      (add-to-list 'auto-mode-alist '("\\.sps\\'" . scheme-mode))
```

Use Geiser for Racket and Guile

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

Use Racket.

```
(use-package racket-mode
  :ensure t)
```

Enable Auto-Complete via Geiser.

```
(use-package ac-geiser
  :ensure t
  :config
  (add-hook 'geiser-mode-hook 'ac-geiser-setup)
  (add-hook 'geiser-repl-mode-hook 'ac-geiser-setup)
  (eval-after-load "auto-complete"
    '(add-to-list 'ac-modes 'geiser-repl-mode))
  (setq geiser-active-implementations '(racket guile))
  (setq geiser-repl-history-no-dups-p t))
```

vi. C

```
ID: org_gcr_2017-05-12_mara:F72D838E-4540-4EC1-AECB-B521DF78D8D8
```

```
(defun help/c-mode-common-hook-fn ()
  "HELP c-mode-common customizations."
  (interactive)
  (rainbow-mode))
(add-hook 'c-mode-common-hook #'help/c-mode-common-hook-fn)
```

vii. Python

```
ID: org_gcr_2017-05-12_mara:BA88E1CC-386E-4CFF-89B3-7E003EC92504
```

```
(defun help/python-mode-hook-fn ()
  "HELP python mode customizatin."
  (interactive)
  (indent-guide-mode))
(add-hook 'python-mode-hook #'help/python-mode-hook-fn)
```

viii. YASnippet & Abbrev

```
ID: org_gcr_2017-05-12_mara:8B8B0626-3A67-4F10-9106-ACFD4E9731E5
```

- Enable everywhere.
- Never expand with `TAB` **anywhere**.
  - Allow expansion to occur within fields.
- Load HELP snippets.

- Use Ido to handle user decisions.

```
(use-package yasnippet
  :ensure t
  :config
  (yas-global-mode t)
  (help/not-on-gui (define-key yas-minor-mode-map (kbd "TAB") nil))
  (help/on-gui (define-key yas-minor-mode-map (kbd "<tab>") nil))
  (define-key yas-minor-mode-map (kbd "s-t") #'yas-expand)
  (help/not-on-gui (define-key yas-keymap (kbd "TAB") #'yas-next-field))
  (help/on-gui (define-key yas-keymap (kbd "<tab>") #'yas-next-field))
  (add-to-list #'yas-snippet-dirs "~/src/help/yasnippet")
  (yas-reload-all)
  (setq yas-prompt-functions '(yas-ido-prompt))
  :diminish yas-minor-mode)
```

Some modes turn on `abbrev-mode`. Diminish it.

Fails with `(eval-after-load 'abbrev (diminish 'abbrev-mode))`.

```
(eval-after-load "abbrev"
  '(diminish 'abbrev-mode))
```

ix. Structured Query Language (SQL)

`ID: org_gcr_2017-05-12_mara:74BF0520-1F06-4FAF-921C-64E8109CEF03`

x. Web Development

`ID: org_gcr_2017-05-12_mara:58CFE4EF-8A8A-4C41-9ECF-C9F6D814218E`

`web-mode` works great for `AngularJS` and now I see a whole lot more.

```
(use-package web-mode
  :ensure t
  :init
  (setq web-mode-enable-current-element-highlight t)
  :config
  (add-to-list 'auto-mode-alist '("\\.tpl'" . web-mode))
  (add-to-list 'auto-mode-alist '("\\.[agj]sp\\'" . web-mode))
  (add-to-list 'auto-mode-alist '("\\.as[cp]x\\'" . web-mode))
  (add-to-list 'auto-mode-alist '("\\.erb\\'" . web-mode))
  (add-to-list 'auto-mode-alist '("\\.mustache\\'" . web-mode))
  (add-to-list 'auto-mode-alist '("\\.djhtml\\'" . web-mode))
  (add-to-list 'auto-mode-alist '("\\.html?\\'" . web-mode))
  (add-to-list 'auto-mode-alist '("\\.js?\\'" . web-mode))
  (add-to-list 'auto-mode-alist '("\\.jsx?\\'" . web-mode))
  (add-to-list 'auto-mode-alist '("\\.css?\\'" . web-mode))
  (add-to-list 'auto-mode-alist '("\\.scss?\\'" . web-mode))
  (add-to-list 'auto-mode-alist '("\\.xml?\\'" . web-mode))
  (add-to-list 'auto-mode-alist '("\\.php?\\'" . web-mode))
  (setq web-mode-enable-engine-detection t)
  (define-key web-mode-map (kbd "s-n") 'web-mode-tag-match))
```

```
(defun help/web-mode-hook-fn ()
  "HELP web-mode customizations."
  (interactive)
  (setq web-mode-enable-auto-indentation nil)
  (setq web-mode-markup-indent-offset 2)
  (setq web-mode-css-indent-offset 2)
  (setq web-mode-code-indent-offset 2)
  (setq web-mode-style-padding 1)
  (setq web-mode-script-padding 1)
  (setq web-mode-block-padding 0)
  (setq web-mode-comment-style 2)
  (setq web-mode-extra-snippets
        '( ("php" . (("dowhile" . ("<?php do { ?>\n\n<?php } while (|); ?>"))
                     ("debug" . ("<?php error_log(__LINE__); ?>"))))))
  (setq web-mode-enable-auto-pairing nil)
  (defun sp-web-mode-is-code-context (id action context)
    (and (eq action 'insert)
       (not (or (get-text-property (point) 'part-side)
             (get-text-property (point) 'block-side)))))

  (sp-local-pair 'web-mode "<" nil :when '(sp-web-mode-is-code-context))
  (setq web-mode-enable-css-colorization t)
  (setq web-mode-enable-block-face t)
  (setq web-mode-enable-part-face t)
  (setq web-mode-enable-comment-keywords t)
  (setq web-mode-enable-heredoc-fontification t)
  (turn-off-fci-mode))

(add-hook 'web-mode-hook #'help/web-mode-hook-fn)
```

xi. JavaScript

```
ID: org_gcr_2017-05-12_mara:92C9A271-BC16-431B-8457-2901DE330164
```

```
(use-package js2-mode
  :ensure t
  :mode (("\\.js$" . js2-mode)
         ("\\.jsx$" . js2-mode))
  :interpreter ("node" . js2-mode)
  :config
  (progn
    (add-hook 'js2-mode-hook #'js2-imenu-extras-mode)))

(use-package js2-refactor
  :after (js2-mode)
  :ensure t
  :diminish js2-refactor-mode
  :config
```

```
  (progn
    (add-hook 'js2-mode-hook #'js2-refactor-mode)
    (js2r-add-keybindings-with-prefix "C-c C-m")))
```

`json-mode` is derived from `javascript-mode` so configure the latter's indent to two.

```
(use-package json-mode
  :ensure t
  :diminish json-mode
  :mode (("\\.jshintrc$" . json-mode))
  :config
  (progn
    (setq js-indent-level 2)
    (setq json-reformat:indent-width 2)
    (setq json-reformat:pretty-string\? t)))
```

(d) Publishing

`ID: org_gcr_2017-05-12_mara:0A4A1510-41F7-4469-87D1-156D44A06216`

i. TEX

`ID: org_gcr_2017-05-12_mara:2DF7BDAD-45C9-4D8D-AC34-FE77E119A093`

```
(use-package tex
  :ensure auctex
  :config
  (define-key TeX-mode-map (kbd "C-c C-c") #'help/safb-TeX-command-master))
```

Save style info. This doesn't control the buffer save.

```
(setq TeX-auto-save t)
```

Parse on load.

```
(setq TeX-parse-self t)
```

Parse on save.

```
(setq TeX-auto-save t)
```

Use PDFTeX to generate both DVI and PDF files.

```
(setq TeX-PDF-mode t)
(setq TeX-DVI-via-PDFTeX t)
```

Don't prompt every time you run `C-c C-c` about saving the file, instead, just save it.

```
(setq TeX-save-query nil)
```

Load LCO files with AucTeX.

```
(add-to-list 'auto-mode-alist '("\\.lco?\\'" . TeX-latex-mode))
```

Culture-dependent typographical results.

```
(add-to-list 'org-latex-packages-alist '("english" "babel" t))
```

Use the closest thing to Times New Roman.

```
(add-to-list 'org-latex-packages-alist '("" "mathptmx" t))
```

Small margins.

```
(add-to-list 'org-latex-packages-alist '("margin=1.5in" "geometry" nil))
```

One empty line between paragraphs.

```
(add-to-list 'org-latex-packages-alist '("" "parskip" nil))
```

Dropped capitals.

```
(add-to-list 'org-latex-packages-alist '("" "lettrine" nil))
```

Standard LaTeX class options.

```
(defvar help/ltx-cls-opt "paper=letter, fontsize=12pt")
```

Standard article class.

```
(eval-after-load "ox-latex"
  '(add-to-list 'org-latex-classes
                `("help-article"
                  ,(concat "\\documentclass["
                           help/ltx-cls-opt
                           "]{article}"))))

(setq org-latex-default-class "help-article")
```

Use “Smartquotes”.

```
(setq org-export-with-smart-quotes t)
```

ii. KOMA-Script

```
ID: org_gcr_2017-05-12_mara:623E9A43-A969-48FD-9045-356F83B135E2
```

I enjoy writing letters. I enjoy reading letters. LaTeX produces letters that are easy to print and read. Org provides a KOMA Script exporter for KOMA-script. The Org documentation mentions that the user should read the ox-koma-letter.el header documentation.

The babel packages is mentioned in the Org documentation. The package documentation explains that it should be used with LaTeX, but not XeTeX. Some time ago I decided to stick with LaTeX. This decision needs documentation. This system leans towards LuaTeX because of its Unicode support and sticks with PDFLaTeX because of its broad acceptance. Those two goals are at odds with each other.

Load the KOMA exporter.

```
(eval-after-load "ox" '(require 'ox-koma-letter))
```

- Understanding KOMA and how to use it

  - There are 4 ways to set letter metadata, listed "from the most specific to the most general" (not sure exactly what this statement means, and the conclusion of my notes tries to make sense of what is really going on here and what is the best way to do things)
    * Org option lines (ORG)
    * Separate Org latex classes (LTX)
    * Emacs Lisp variables (LISP)
    * Letter Class Option files (LCO)
- Notes and thoughts on the ways to use it
  - LTX
    * By following the setup directions, you do this, creating "help-letter" class
    * Familiar and easy if you already know LaTeX
      · At some point in your workflow, you **must** define a class to use, anyway
    * Very easy to do, just define the class template and set `org-koma-letter-default-clas`
  - ORG
    * Simple way that makes it very easy to just focus on the document content
    * This metadata takes highest priority in the workflow
      · So you should set your typical defaults in LISP or LCO and customize it here. This is exactly what I wanted to know.
      · This lets you do your tweaking in each unique file while relying on the most common defaults defined elsewhere
  - LISP
    * Very familiar style of configuring things
  - LCO
    * LCO == Letter Class Option files
    * LCO files are TeX
    * They are included in the generated TeX source code from the letter
    * Gives **full** access to KOMA-Script
      · Big deal, because not everything is exposed through ORG or LISP
      · Also gives full access to any and all TeX and LaTeX code

        * LCO files are a KOMA-Script thing, so they are a LaTeX thing
        * Letter metadata set in LCO files overwrites letter metadata set in Emacs variables but not letter metadata set in the Org file.
        * When you include multiple LCO files, they are evaluated LIFO. Properties are set as they first appear, and are not set again. Say you include "MyGeneralStuffForAnyLetter.lco" and then include "MyStuffSpecificToThisLetter.lco". The specific stuff will get set first. Then general stuff will get set last.
            · Surely there is a better way to phrase this. I will work on that.

- Recommendations
    - What is the easiest way to start using KOMA-Script based on what you know today?
    - If you don't know any of the approaches
        * Then choose between learning LaTeX and Org
    - If you only know LaTeX
        * Then you will use the LTX/LCO metadata approach
    - If you only know ORG
        * Then you will use the ORG metadata approach
    - If you only know ORG and LISP
        * Then you will use the LISP approach for general metadata and the ORG approach specific metadata
    - If you know LTX/LCO, ORG, and LISP
        * Then you have total flexibility
        * The fact is that
            · ORG settings always trump LTX/LCO and LISP
            · LISP settings are a subset of all of the settings available in KOMA-Script, so you will always have to fall back to LTX/LCO if you want to use unexposed features
            · LCO files are just plain old LaTeX, which you already know
        * So the best thing to do
            · Is to use ORG for letter-specific settings
            · And LTX for general settings
            · Everything is a lot simpler this way because

- One less metadata approach to keep track of
- All KOMA-Script features are present
- Need to learn details of KOMA-Script package anyway

Configure the default class.

This post explains how to default the US letter size. That is the likely default for my printed correspondence.

```
(eval-after-load "ox-koma-letter"
  '(progn
     (add-to-list 'org-latex-classes
                  `("help-letter"
                    ,(concat "\\documentclass["
                             help/ltx-cls-opt
                             ", pagesize, UScommercial9]{scrlttr2}")))

     (setq org-koma-letter-default-class "help-letter")))
```

There are two formats for the letters: heading-based and property-based.

Set up my default LCO files.

```
(setq org-koma-letter-class-option-file "KomaDefault")
```

iii. Texinfo

```
ID: org_gcr_2017-05-12_mara:2A19DF86-96B0-4371-8DE9-1F79861F5E89
```

Perhaps the first document typeset with Org-Texinfo.

iv. Pandoc

```
ID: org_gcr_2017-05-12_mara:BF923ADF-EB81-4695-9F98-F68D557D667B
```

v. Markdown

```
ID: org_gcr_2017-05-12_mara:005B395D-AF3F-471D-AD11-45FDBAA85D7A
```

Provide Github Flavored Markdown (GFM).

```
(use-package ox-gfm
  :ensure t)
```

Program GFM.

```
(use-package markdown-mode
  :ensure t
  :config
  (add-to-list 'auto-mode-alist '("\\.md\\'" . gfm-mode))
  (setq markdown-coding-system "utf-8"))
```

vi. HTML

```
ID: org_gcr_2017-05-12_mara:5AE61C99-1862-456E-B6FA-44225675B208
```

Load Htmlize for HTML export.

```
(use-package htmlize
  :config
  (setq org-html-htmlize-output-type htmlize-output-type)
  (setq htmlize-output-type 'inline-css)
  (defvar help/htmlize-initial-fci-state nil
    "Variable to store the state of ‘fci-mode’ upon calling ‘htmlize-buffer’.

Attribution: URL ‘http://permalink.gmane.org/gmane.emacs.orgmode/98153’.")
  (defvar help/htmlize-initial-flyspell-state nil
    "Variable to store the state of ‘flyspell-mode’ upon calling ‘htmlize-buffer’.

Attribution: URL ‘http://permalink.gmane.org/gmane.emacs.orgmode/98153’.")

  (defun help/htmlize-before-hook-fn ()
    (when (fboundp 'fci-mode)
      (setq help/htmlize-initial-fci-state fci-mode)
      (when fci-mode
        (fci-mode -1))))
  (add-hook 'htmlize-before-hook #'help/htmlize-before-hook-fn)

  (defun help/htmlize-after-hook-fn ()
    (when (fboundp 'fci-mode)
      (when help/htmlize-initial-fci-state
        (fci-mode t))))
  (add-hook 'htmlize-after-hook #'help/htmlize-after-hook-fn))
```

Use in-line CSS.

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

vii. ASCII

```
ID: org_gcr_2017-05-12_mara:7BEBFA9B-E445-41F3-ABC6-08AFB3020D30
```

```
(setq org-ascii-text-width 80)
(setq org-ascii-global-margin 0)
```

Show non-ascii characters

```
(defun help/occur-non-ascii ()
  "Find any non-ascii characters in the current buffer.
Attribution: URL ‘https://www.emacswiki.org/emacs/FindingNonAsciiCharacters’"
  (interactive)
  (occur "[^[:ascii:]]"))
```

viii. Beamer

```
ID: org_gcr_2017-05-12_mara:1D459BA8-B1B0-40B9-B733-6B59216FD1B2
```

Load Beamer for creating presentations.

```
(use-package ox-beamer)
```

ix. Screenwriting

ID: org_gcr_2017-05-12_mara:AE18CBCE-F03C-4422-AAC8-E1E2C76A7C5E

```
(use-package fountain-mode
  :config
  (add-to-list 'auto-mode-alist '("\\.fountain$" . fountain-mode))
  (let ((fountain-stx '(
                        "CONT'D"
                        "CROSSFADE"
                        "FLASHCUTS"
                        "FLASHFORWARD"
                        "INTERCUT"
                        "PRE"
                        "PRELAP"
                        )))
    (mapc (lambda (stx) (add-to-list 'ispell-skip-region-alist (list stx)))
          fountain-stx)))
```

x. Reveal.js

ID: org_gcr_2017-05-19_mara:73719624-5BA9-4000-B70C-77C30CAB592B

```
(use-package ox-reveal
  :ensure t
  :config
  (setq org-reveal-root (expand-file-name "~/src/reveal.js")))
```

(e) DevOps

ID: org_gcr_2017-05-12_mara:F79C9679-7887-40B8-8882-51EE6EC1BDE2

i. Shell Script

ID: org_gcr_2017-05-12_mara:68AC26B4-B52C-4A1F-AF1E-78AFE5A3D91D

A. Editing

ID: org_gcr_2017-07-16_mara:D2B25F16-8384-4641-A2C0-C9D987A83E10

This Bash IDE appears less valuable because it automatically inherits its best features unseen here (you are looking at the rest below this big block):

- Standard programming configuration
- ShellCheck integration
- Default to Bash
- Smart Hashbang line generation

```
(defun help/sh-mode-hook-fn ()
  (interactive)
  (setq sh-shell "bash")
  (setq sh-basic-offset 2))
(add-hook 'sh-mode-hook #'help/sh-mode-hook-fn)
```

Toggle string quotes between single and double.

```
(use-package toggle-quotes
  :ensure t
  :config
  (define-key sh-mode-map (kbd "C-'") #'toggle-quotes))
```

Sh-Mode Hydra.

```
(defhydra help/hydra-sh-mode (:color blue
                                     :hint nil)
  "
sh-mode:
 _i_ insert #!-line
  _q_ quit
"
  ("i" sh-set-shell)
  ("q" :nil))
(key-chord-define sh-mode-map "hh" #'help/hydra-sh-mode/body)
```

ii. Make

```
ID: org_gcr_2017-05-12_mara:4EC46A0A-2D59-4C21-8013-6A86CD46BC5F
```

iii. Vagrant

```
ID: org_gcr_2017-05-12_mara:D8A25DB9-FFAD-4C06-95C8-948AB0AFC4DE
```

`ruby-mode` supports `Vagrantfile` OOTB.

iv. Apache

```
ID: org_gcr_2017-05-12_mara:719B95DC-1AF2-4581-BDEB-FC2430EAE076
```

```
(use-package apache-mode
  :ensure t)
```

v. SSH

```
ID: org_gcr_2017-05-12_mara:EF50829C-C6BA-4D22-ACD0-5386EF38155D
```

```
(use-package ssh-config-mode
  :ensure t
  :mode ((("/\\.ssh/config\\'"     . ssh-config-mode)
         ("/sshd?_config\\'"       . ssh-config-mode)
         ("/known_hosts\\'"        . ssh-known-hosts-mode)
         ("/authorized_keys2?\\'" . ssh-authorized-keys-mode))
  :config
  (add-hook 'ssh-config-mode-hook #'turn-on-font-lock)
  (add-hook 'ssh-config-mode-hook #'help/text-mode-fn))
```

vi. CRON

```
ID: org_gcr_2017-06-13_mara:9B3AB875-4459-4EC7-A7C5-4C32EC3A5E18
```

```
(use-package crontab-mode
  :ensure t
  :config
```

```
(add-to-list 'auto-mode-alist '("\\.cron\\(tab\\)?\\'" . crontab-mode))
(add-hook 'crontab-mode-hook #'turn-on-stripe-buffer-mode))
```

(f) Multimedia

ID: org_gcr_2017-05-12_mara:D3EE5DB5-EC7B-4D7C-9749-1BC351B8214D

i. Artist

ID: org_gcr_2017-05-12_mara:4C5A0835-B5EE-4248-A04A-5F1E90FDC308

```
(add-to-list 'auto-mode-alist '("\\.asc" . artist-mode))
(add-to-list 'auto-mode-alist '("\\.art" . artist-mode))
(add-to-list 'auto-mode-alist '("\\.asc" . artist-mode))
```

ii. DITAA

ID: org_gcr_2017-05-12_mara:6D7CE54F-D123-4C90-BFB6-374B2ACB1D1C

```
(defconst help/ditaa-jar (getenv "DITAA"))
(setq org-ditaa-jar-path help/ditaa-jar)
```

iii. Graphviz

ID: org_gcr_2017-05-12_mara:8523DEE7-31C4-445A-B300-4923BCA3F7D7

```
(use-package graphviz-dot-mode
  :ensure t
  :config
  (setf (cdr (assoc "dot" org-src-lang-modes)) 'graphviz-dot))
```

iv. PlantUML

ID: org_gcr_2017-05-12_mara:1AE2633B-E811-4117-ACE7-0D7D2F88EDEB

```
(use-package plantuml-mode
  :ensure t
  :init
  (defconst help/plantuml-jar (getenv "PLANTUML"))
  (setq plantuml-jar-path help/plantuml-jar)
  :config
  (eval-after-load "ob-plantuml"
    (setq org-plantuml-jar-path help/plantuml-jar)))
```

v. X PixMap (XPM)

ID: org_gcr_2017-06-25_mara:53D58BE5-1E7D-4E75-8359-AF7EF001CE2B

X PixMap (XPM)

Emacs' provide editing support.

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

(g) Computer-aided design

ID: org_gcr_2017-05-12_mara:50E7A50A-4259-4B6C-8612-99671C6723CC

i. OpenSCAD

ID: org_gcr_2017-05-12_mara:423A1845-3D84-45FB-8373-509033537F86

```
(use-package scad-mode
  :ensure t)
```

(h) Special

ID: org_gcr_2017-06-25_mara:BEC7C2AA-4602-4F1C-8AE5-2DAD6439C90F

`messages-mode` inherits `special-mode` hooks but it didn't get stripe buffered so I'm trying to set it up here with `message-mode-hook` but that didn't work either

5. Quiet and Pleasant Appearance

ID: org_gcr_2017-05-12_mara:298AA1C9-5745-464F-92A6-2FB04EFB079E

Configure EMACS to personal-taste for "noise" and "form".

(a) Key Press

ID: org_gcr_2017-05-12_mara:EBAA168D-C2E9-40E3-B448-9AA076802029

Make key-presses sound like the ground-breaking Selectric typewriter.

```
(use-package selectric-mode
  :ensure t)
```

(b) Line Number

ID: org_gcr_2017-05-12_mara:5FFDA284-6C92-40C8-B23E-E345940DC6C4

The nlinum gutter should not "shift" as it transitions between line numbers of different magnitudes. For example going from line 99 to 100 will shift the buffer contents by one character. That is distracting and interrupts the flow.

- Switched to `nlinum` from `linum` because of slowness with this file when headings are collapsed

Most files will be less than 100,000 lines.

```
(use-package nlinum
  :ensure t
  :config
  (setq nlinum-format "%05d"))
```

(c) Buffer

ID: org_gcr_2017-05-12_mara:4A3C9150-EC66-4462-B14A-92C8BE34FB94

Never automatically convert the end of the line character. For most of us this is between UNIX and DOS.

```
(setq inhibit-eol-conversion t)
```

Give buffers backed by identically named files distinguishable names.

```
(use-package uniquify)
(setq uniquify-buffer-name-style 'forward)
```

Don't use audible bells, use visual bells.

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

Highlight s-expressions.

```
(setq blink-matching-paren nil)
(show-paren-mode)
(setq show-paren-delay 0)
(setq show-paren-style 'expression)
```

The cursor should not blink and distract you. On a graphic display make the cursor a box and stretch it as wide as the character below it.

```
(blink-cursor-mode 0)
(help/on-gui
 (setq-default cursor-type 'box)
 (setq x-stretch-cursor 1))
```

EMACS used UTF-8 by default. Make copying and pasting easier.

```
(prefer-coding-system 'utf-8)
(help/on-gui
 (setq x-select-request-type '(UTF8_STRING COMPOUND_TEXT TEXT STRING))
 (help/on-windows
  (set-clipboard-coding-system 'utf-16le-dos)))
```

Make it very easy to see the line with the cursor.

```
(global-hl-line-mode t)
```

Make it very easy to input special-characters using TEX coding.

```
(setq default-input-method 'TeX)
```

Automatically Copy Text Selected With The Mouse via.

```
(setq mouse-drag-copy-region t)
```

The macro recorder and Multiple-Cursors provide two ways to do the right thing in different situations. Be very thoughtful and allow every function. My configuration *barely* utilizes MC because I use all that I can handle right now.

```
(use-package multiple-cursors
  :ensure t
  :config
  (setq mc/always-run-for-all t))
```

Recenter near top of screen.

```
(defun help/recenter-line-near-top-fn ()
  "Move current line near top"
  (interactive)
  (let ((recenter-positions '(5)))
    (recenter-top-bottom)))
```

(d) Color Theme

`ID: org_gcr_2017-05-12_mara:3C38AF9D-1905-4B33-A4F5-065D17E9B647`

Solarized Theme

- ~1,000 Faces Defined
- 95,677 Downloads
- A distinct fringe provides a definition of space.
- The modeline is always at the bottom and doesn't need differentiation.
- Minimize bold and italic faces.
- Minimize fringe indicators.

```
(use-package solarized-theme
  :ensure t
  :config
  (setq solarized-distinct-fringe-background t)
  (setq solarized-use-variable-pitch nil)
  (setq solarized-use-less-bold t)
  (setq solarized-use-more-italic nil)
  (setq solarized-emphasize-indicators nil)
  (setq solarized-scale-org-headlines t)
  (load-theme 'solarized-dark t)
  (eval-after-load "smart-mode-line"
    '(sml/setup)))
```

(e) Comint

`ID: org_gcr_2017-05-12_mara:EFB91C72-0036-49D8-B4B9-C7473C838E11`

`comint-mode` is only maybe the second most important thing for making Emacs really, really special.

```
(setq comint-scroll-to-bottom-on-input 'this)
(setq comint-scroll-to-bottom-on-output 'others)
(setq comint-move-point-for-output 'others)
(setq comint-show-maximum-output t)
(setq comint-scroll-show-maximum-output t)
(setq comint-move-point-for-output t)
```

This configuration had been working fine for a long time. The intent was for it to be crystal clear that the prompt line in comint buffers would be read only. This turned out to be a mistake; though I am not sure why, when, or how it became a mistake. Nonetheless, this should be left alone. The way the issue here manifested was that all R buffers opened by `ess` were 100% read only which obviously is a **big issue** if you actually want to use! ROFL

```
(setq comint-prompt-read-only nil)
```

(f) Font

ID: org_gcr_2017-05-12_mara:948F97ED-7FD3-4AC9-89D4-CB270CD0540C

The best programming font is Deja Vu Sans Mono because it sans-serif and support a lot of Unicode characters. Set it to a good default for an 80 character wide buffer and make it easy to adjust it.

```
(help/on-gui
 (defvar help/font-size-current 10 "The preferred font size.")
 (help/on-osx (setq help/font-size-current 17))
 (help/on-windows (setq help/font-size-current 13))
 (defconst help/font-size-ideal help/font-size-current "The ideal font for this system.
 (defconst help/font-base "DejaVu Sans Mono" "The preferred font name.")
 (defun help/font-ok-p ()
   "Is the configured font valid?"
   (interactive)
   (member help/font-base (font-family-list)))
 (defun help/font-name ()
   "Compute the font name and size string."
   (interactive)
   (let* ((size (number-to-string help/font-size-current))
          (name (concat help/font-base "-" size)))
     name))
 (defun help/update-font ()
   "Updates the current font given configuration values."
   (interactive)
   (if (help/font-ok-p)
       (progn
         (message "%s : Font Set" (help/font-name))
         (set-frame-font (help/font-name)))
     (message (concat "Your preferred font is not available: " help/font-base))))
 (defun help/font-size-reset ()
   "Restore the ideal font size."
   (interactive)
   (setq help/font-size-current help/font-size-ideal)
   (help/update-font))
 (help/update-font))
```

(g) Frame

ID: org_gcr_2017-05-12_mara:8D473EA1-DFEE-408E-A1F5-AF1C7D03317D

i. Inviting and familiar GUI features

ID: org_gcr_2017-06-23_mara:77B3D3E9-B7F7-44FB-8B82-A23CAC02C83B

Title.

```
(setq frame-title-format '("" "%b - Super Text Editor "))
```

A. Bitmaps

ID: org_gcr_2017-07-11_mara:02FCF4D1-5613-49B3-A247-6BFA846D7EEA

Use local bitmaps.

```
(add-to-list 'image-load-path "~/src/help/xpm")
```

Bitmap requirements

- 24x24 seems to be the most common dimension

Finding bitmaps

- Search for an existing bitmap by subject
- Prefer SVG for flexibility
- Font Awesome seems like the best Font

Generating bitmaps

- By hand
  - Emacs already supports editing and visuaation
  - Gimp
- Convert SVG to XPM
  - Convert SVG to PNG

    ```
    rsvg-convert --width=24 --height=24 --format=png --keep-aspect-ratio
    ```

  - Convert PNG to XPM

    ```
    convert new.png new.xpm
    ```

B. Icons

```
ID: org_gcr_2017-07-11_mara:5E885B6C-66CF-4F58-8888-F257F4AB8AFA
```

Replace major mode lighters with icons.

```
(use-package mode-icons
  :ensure t
  :config
  (mode-icons-mode))
```

C. Menu bar

```
ID: org_gcr_2017-06-23_mara:47EF669D-E999-4573-AA5C-5CF038ADA870
```

Enable the menu bar.

```
(menu-bar-mode nil)
```

D. Tool bar

```
ID: org_gcr_2017-06-23_mara:EC50EB46-7878-4799-9805-EF0CB3D3E526
```

Enable the tool bar.

```
(tool-bar-mode nil)
```

Style the tool bar.

```
(setq tool-bar-style 'both)
```

ibuffer (image source).

```
(tool-bar-add-item
 "fontawesome_list-alt"
 'ibuffer
 'iBuffer
 :help "List buffers")
```

buffer swap (image source).

```
(tool-bar-add-item
 "evan-shuster-ca_swap"
 'help/safb-switch-to-previous-buffer
 'Swap
 :help "Swap buffers")
```

E. Scroll bar

```
ID: org_gcr_2017-06-23_mara:590AD87D-0567-454F-98AE-3AE64A171012
```

Enable both the vertical and horizontal scroll bar.

```
(scroll-bar-mode nil)
(horizontal-scroll-bar-mode nil)
```

F. Transparency

```
ID: org_gcr_2017-06-23_mara:143A134C-02D9-412F-9D9C-FC6E7B43D505
```

```
(use-package seethru
  :ensure t
  :config
  (defhydra hydra-transparency (:color blue :hint nil)
    "
this frame's opacity: %(frame-parameter nil 'alpha)
        _i_ reset!
 _j_ less _k_ 50/50   _l_ more
 _m_ quit
"
    ("j" (lambda () (interactive) (seethru-relative -1)) :exit nil)
    ("i" (lambda () (interactive) (seethru 100)) :exit nil)
    ("k" (lambda () (interactive) (seethru 50)) :exit nil)
    ("l" (lambda () (interactive) (seethru-relative 1)) :exit nil)
    ("m" nil)))
```

(h) Pointer

```
ID: org_gcr_2017-05-12_mara:17DB4E08-F6C4-44AF-935B-27E8984F13DB
```

Hide the pointer when typing.

```
(setq make-pointer-invisible t)
```

(i) Version Control

ID: org_gcr_2017-05-12_mara:09918DB9-3AE3-44DF-9E1F-A4A3918F3315

Selectively provide VC file status indicators.

Sometimes it is worth interrupting the flow.

```
(use-package diff-hl
  :ensure t)
```

Ediff split frame horizontally.

```
(setq ediff-split-window-function 'split-window-horizontally)
```

(j) Window

ID: org_gcr_2017-05-12_mara:FBD41A4B-9764-49CB-8439-74581B2211A0

Easily return to previous configurations. 2-4 windows are easily managed by hand.

```
(winner-mode t)
```

Frequently use between 1 and 4 windows.

```
(defun help/1-window ()
  "Work with this buffer in 1 window."
  (interactive)
  (delete-other-windows))

(defun help/2-window ()
  "Work with this buffer in 2 windows."
  (interactive)
  (delete-other-windows)
  (split-window-below)
  (balance-windows))

(defun help/3-window ()
  "Work with this buffer in 3 windows."
  (interactive)
  (delete-other-windows)
  (split-window-below)
  (split-window-below)
  (balance-windows))

(defun help/4-window ()
  "Work with this buffer in 4 windows."
  (interactive)
  (delete-other-windows)
  (split-window-right)
  (split-window-below)
  (call-interactively #'other-window)
  (call-interactively #'other-window)
  (split-window-below)
```

```
  (call-interactively #'other-window)
  (call-interactively #'other-window)
  (balance-windows))
```

Most of the time when opening other buffers, go to them. This configuration appears for different modes in this system. Modes distributed with Emacs are configured here.

```
(setq help-window-select t)
```

Select windows.

```
(use-package ace-window
  :ensure t
  :config
  (setq aw-keys '(?a ?s ?d ?f ?g ?h ?j ?k ?l))
  (setq aw-scope 'frame)
  (setq aw-background nil))

(use-package eyebrowse
  :ensure t
  :config
  (setq eyebrowse-wrap-around t)
  (eyebrowse-mode t)
  (defhydra help/hydra-left-side/eyebrowse (:color blue :hint nil)
    "
current eyebrowse slot: %(eyebrowse--get 'current-slot)
 _j_ previous _k_ last _l_ next _u_ close _i_ choose _o_ rename _q_ quit
   _a_ 00 _s_ 01 _d_ 02 _f_ 03 _g_ 04 _z_ 05 _x_ 06 _c_ 07 _v_ 08 _b_ 09"
    ("j" #'eyebrowse-prev-window-config :exit nil)
    ("k" #'eyebrowse-last-window-config)
    ("l" #'eyebrowse-next-window-config :exit nil)
    ("u" #'eyebrowse-close-window-config :exit nil)
    ("i" #'eyebrowse-switch-to-window-config)
    ("o" #'eyebrowse-rename-window-config :exit nil)
    ("q" nil)
    ("a" #'eyebrowse-switch-to-window-config-0)
    ("s" #'eyebrowse-switch-to-window-config-1)
    ("d" #'eyebrowse-switch-to-window-config-2)
    ("f" #'eyebrowse-switch-to-window-config-3)
    ("g" #'eyebrowse-switch-to-window-config-4)
    ("z" #'eyebrowse-switch-to-window-config-5)
    ("x" #'eyebrowse-switch-to-window-config-6)
    ("c" #'eyebrowse-switch-to-window-config-7)
    ("v" #'eyebrowse-switch-to-window-config-8)
    ("b" #'eyebrowse-switch-to-window-config-9))
  (global-set-key (kbd "C-M-e") #'help/hydra-left-side/eyebrowse/body))
```

6. Piano Lessons

   ID: org_gcr_2017-05-12_mara:B9C71531-F4DD-4180-950D-AD3494C5D566

   (a) A Fine Cup of EMACS

ID: org_gcr_2017-05-12_mara:80EEF715-CC24-4B14-A12B-1CD4F16D13A9

Every EMACS user ought to have a Emacs Reference Mug at their desk. The mug invites other users to ask questions. Give the mug as a gift to every user you know who would benefit from learning EMACS. The mug reminds us all that EMACS is the perfect configuration of EMACS. It is available on every machine. When you break your system, you can always fall back to the good and reliable default EMACS configuration to get your system up and running again. The OOTB configuration of EMACS is one of the most important system configurations that you will every find. That is why it is important never to ruin it.

This system wants to maximize accessibility for new users. It wants anyone to be able to download and use it without surprises. It wants the mug to serve as a fine reference for anyone to use. It wants to keep things simple and familiar so that anyone who has learned EMACS OOTB can use it pleasantly and productively. These goals are essential to configuring the keyboard for this system. This system will always respect the POLA.

(b) A Keyboard on Every Desk

ID: org_gcr_2017-05-12_mara:7A27D97D-9D7D-43DC-B961-A4AFC9032609

The configuration of the keyboard on an EMACS system can completely change the experience. No keyboard makes it impossible. A Kinesis Ergo makes it feel really good. Soft keys make it feel soft; hard keys make it feel faster. The layout of letters is claimed to make you "more productive" using statistics. You may even study the statistics of your own writing and choose a layout optimized for you. The ways to configure your keyboard are limitless because everyone is unique. How to get the best configuration tips for your system? Do what works for everyone.

Choose a keyboard that will satisfy 80% of EMACS users using 80% of the keyboards out there. Make this system easy to use on any one of those keyboards. Make this system easy to use in English. Make this system easy to use with average hand strength using two hands. These goals are essential to configuring the keyboard for this system.

(c) A Display with Every Keyboard

ID: org_gcr_2017-05-12_mara:ED485AD6-A185-465C-9B8A-2E9E848E02DA

Every system requires an output. You have two options. The first is a terminal that only displays characters. The second is a display that provides detailed graphics. "Display" is the EMACS term for a GUI.

Some users prefer the former. Some users prefer the latter. Some users prefer to use a $4000USD machine to emulate the latter. Both are good options.

This system is configured to work pleasantly for either type of output.

(d) A Full Pot of EMACS on Every Desk

ID: org_gcr_2017-05-12_mara:3165655B-5F9B-4712-896B-32EE9EEEF946

i. Keyboard Layout & Operation

ID: org_gcr_2017-05-12_mara:10CA1648-3A69-47DA-B20C-495E24D7E54A

- Use QWERTY layout.

- Everyone knows it.
- Easy to learn.
- Available on every keyboard.
- Inexpensive.
- When graduation time comes, plenty of great alternatives available like DVORAK and Colemak.

- Keep hands in home position as much as possible.
  - Every finger is strong in the home position so RSI reduced.
  - Single key presses are easy there.
- Table-bang the shift, caps-lock and enter keys.
  - Table-bang is a position of your hand. Make it by:
    * Starting with your hands in the home position.
    * Make a "high-five" with both of them parallel to the keyboard.
    * Turn your left hand counter-clockwise and right hand clockwise to make them perpendicular to the keyboard.
    * Squeeze all of your fingers together.
    * Push the keys using the side of your Pinky.
    * In this position you are using the strength of all of your fingers.
  - Never use those key using your Pinky alone.
  - Practice depends 100% on user-discipline.
- Try to achieve balance with meta keys.
  - Provide same key of each side of the keyboard.
- Be conscious of key operations on different outputs.
  - Always provide both.
  - Note what is getting stomped on.
  - For return bind to:
    * `RET` in the terminal.
    * `<return>` in the GUI.
  - Also for tab `TAB` vs `C-i`.
  - Also for escape `ESC` vs `C-[`

ii. Understanding Your Cognitive Landscape.

`ID: org_gcr_2017-05-12_mara:7CAD36B9-85CC-4417-B1BF-34BF0E1FD704`

You operate within a cognitive landscape. Every moment you are in a single `place`. While residing in each `place` you perform logically related `activities`. Activities facilitate logical actions like modification within that `place`. Modifications are performed objects. Objects include things like the contents of a buffer, contents of memory, or the file that backs a buffer. While performing those `activities` there is a logical sense of "flow". That should never be interrupted. Usually an interruption occurs when you are going to go to a new `place`. The `distance` between `places` is measured in the similarity between the `actions` that you find there. As you develop these ides it will be obvious where key-bindings should go

iii. Key-Bindings Take You to Places to Perform Activities

`ID: org_gcr_2017-05-12_mara:B2C6C0E2-03F4-4C9E-828D-E972EF84F7AB`

OOTB you wil be visiting many `places` and performing many `activities`. EMACS comes with a good configuration that minimizes `distance`. This isn't worth changing. You can use EMACS for a lifetime without ever having to customize any of the key-bindings. This is what lets anyone use your system. This is what lets you use the system with `-Q` when you break it. You need to decide if you every want to alter the default configuration. This system does not want to. It wants to keep EMACS true to EMACS and your hands happy. To satisfy those goals the following practices were defined.

- 99.999% of the time never bind to the `C` or `M` name-space.
  - They are for system key-bindings. You can break them. Don't.
  - In theory `C-c` is the "user name-space" but packages stomp on this all of the time anyway so don't use it.
  - Some bindings are just too valuable to pass up:
    * `C-;`
      · Your hands are in the home position already.
    * Every modifier key with return.
- Never bind to `F` keys.
  - They are a painful stretch on most keyboards.
  - Some require a lone Pinky with is worse.
  - Most operating systems bind actions to them OOTB anyway.
  - EMACS comes with key-bindings OOTB.
- Don't try to set up a Hyper-key.
  - I tried it. It is great to open another namespace. But it ruins cross-platform portability.
  - However I'm leaving it available for anything possible.
- Use shift as a name-space expansion vehicle.

- Shift doubles every name-space in which you use it.
- Use cautiously, not every name-space vehicle supports it.

- About the `s` (super) name-space.
  - In theory it is the best place for user-defined key-bindings because EMACS OOTB uses `C` and `M` completely leaving `s` mostly open.
  - In practice `C` and `M` are running out of space because there are a lot of new packages being added to EMACS. Most new packages are binding key in the `s` name-space.
  - This system reserves `s` completely for Sysop.

These practices say nothing about the `places` or `activities` that you choose to peform. The practices only look at the key-binding configuration. There are a limited number of keys on a keyboard and there are physical limitations on your hands. Along with the previous assuptions it may look like there are less. Fortunately it just looks that way and it isn't true. There are a lot of powerful ways to "go places" with EMACS. The next heading contains my attempt.

iv. How to Get There Pleasantly and Quickly

`ID: org_gcr_2017-05-12_mara:28C3C38D-A81D-4E09-8AEA-38C8E7F05CC4`

You need to learn how to use EMACS. You need to develop a personal preference. You need to develop an idea of `places` and `activities` and `distance`. The following headings are delineated by breaks in flow.

The examples try to talk about doing those things and do it by exploring:

- "going places to do things".
- "how quickly I will get there and how long I will be there"
- "how quickly I want to go somewhere else".

They were initially described by the properties:

Actions The number of related actions in that place.

Expertise The level of skill and speed with which you are performing the activity.

Relationship How closely those activities are related in the current place.

Frequency How many times you perform these actions when you here.

The relationship between "doing those things" and those 4 properties is still unclear and being explored.

A. `s`

```
Actions: High
Expertise: High
Relationship: High
Frequency: High
```

ID: org_gcr_2017-05-12_mara:6477D74C-FC51-4722-A665-B26F33541078

- Actions here are for the place inside of the buffer itself. They are for immediate acting upon the contents of the buffer. They are logically related, used frequently, and likely to be memorized.
- When you come here, you are likely to stay for some time before getting out.
- Only use single key bindings; anything more may be a new logical name-space and may use a Hydra.
- Split the home sides of the keyboard in half.
- The left side of the keyboard should be use for operations common to every mode.
  - For example goto-line and ispell.
  - It has 15 bindings available; 20 if you use 1-5. 40 if you shift them.
- The right side of the keyboard should be used operations specific to the current major mode.
  - For example in Org-Mode navigating between source-blocks and evaluating them.
  - It has 19 bindings available; 26 if you use 6-=. 52 if you shift them.

For example, in Org-Mode:

- I traverse the entire document very quickly with org-babel-previous-src-block and and org-babel-next-src-block.
- I execute source-blocks.
- I edit source-blocks.

Every activity is related to reading, modifying, executing, and tangling code.

B. Key-Chord

ID: org_gcr_2017-05-12_mara:D7EDA057-1289-4EB9-8A72-3B00DC3C87BA

Key-Chord is intriguing because it works on every keyboard. It is powerful because it can you bring you to any place easily. It is good for taking you places in two differnt kinds of scenarios.

One example is grammar-checking. There are a few ways to do that. I don't remember them all. In a given mode I want to see a list of all the ways. I really just want to see all of the stuff that I value for a given mode and don't use frequently.

Another example are things that I value for a mode and use a lot but are not logically related to other activities in that place. For example moving the

mark around and going to lines are performed a lot so they need to be done quickly and left. This is a place where key-chords and the shift modifier are a fast and intuitive way to go places.

C. Single-Key Key-Chord Name-Space.

   `ID: org_gcr_2017-05-12_mara:13704474-DDBB-4D3E-995B-51B7860535DD`

   :Actions: High :Expertise: Low :Frequency: High :Relationship: Low

   - Nice if you don't mind hitting the same key twice.
   - You will use come here often, perform your single `action`, and be done and leave very frequently and quickly.
   - Using alphabetical characters always results in unpleasant surprises.
   - Harder for breakage but it still occurs.
     - `#FF` color code.
     - `cc` carbon copy.
     - `JJ` nick-name.
       * Rare! Using.
     - `dd` add
   - Symbols are more likely to be safer bets.
     - Only use the symbols.
       * 8 if you use rows 3-4; 16 if you shift.
       * Fifth row has 13; 26 with shift.
   - Good vehicle to reach a Hydra.

D. Two-Key Key-Chord Name-Space.

   `ID: org_gcr_2017-05-12_mara:033FD965-8C1D-46E4-87FC-4A1E50C746A9`

   :Actions: Low :Expertise: High :Frequency: High :Relationship: Low

   - Very attractive.
   - Nice if you don't like hitting the same key twice.
   - Easy to use all fingers.
     - Finger strength is not an issue here; use any of them.
   - Unexpected breakage very easy.
     - `cd` in `=eshell=`.
   - Use sparingly.
   - Not worth analyzing ideal combinations; just use it and see if it doesn't break.

- Bringing over existing bindings. They are all for every mode so I will keep it that way.

E. Hydra

```
Actions: High
Expertise: Low
Frequency: Low
Relationship: High
ID: org_gcr_2017-05-12_mara:631201BF-3F66-4C64-A7DC-00B46A3B39ED
```

- Sometimes you want to do something in a `place` but you aren't sure what and you aren't sure where you will go next from there. For example you might want to perform an Org-Mode `action` that is important but you don't really use much. For example exporting to HTML might not be common for you but you value.
- `Hydras` can be used for very related actions too. The difference between the `s` name-space is the distance between them and where you are now. In the `s` namespace you go there very quickly. For `Hydras` sometimes you can get the fast and sometimes more slowly. They are complementary to every name-space.
- `SHIFT` doubles your key-space.
- Use `C-g` to exit the Hydra.

For example, in Org-Mode I am still learning about functions and haven't used them much and forget their names. It is faster to put them in a Hydra. If they get used a lot, I will add them to `s`.

v. Building Your Own Keyboard

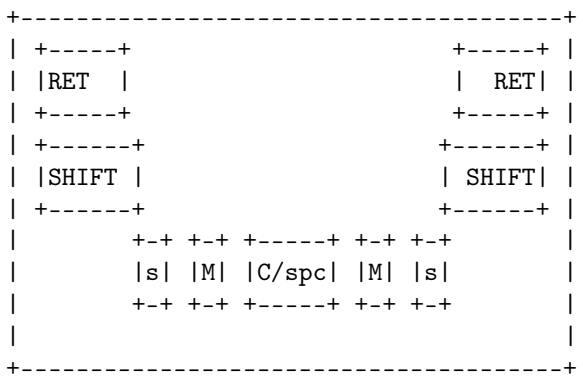
```
ID: org_gcr_2017-05-12_mara:BED4C08B-260A-4FC5-B5FF-AFEC420E9C66
```

As your mastery of EMACS grows so too will your desire to build your own keyboard. It is natural. As you explore various dimensions of expression you will have a lot of fun. You will act more quickly and skillfully. Even with the goals of this system in place the desire grows.

3D printing is one area worth exploring. A lot of EMACS users design and print their own custom keyboards. That looks very fun. Ukulele is softer way to explore your keyboard. Reading its user manual is important. It contains ideas about stack-able-environments for bindings. You may use Ukulele or Hydras to do the same thing. Karabiner is a nice way to re-map your keys. It's easiest adjustment is to make return act as return when pressed alone and as control when pressed with another key. That introduces a symmetry to your keyboard which can be helpful. All of those dimensions are worth exploring.

When I explored them I felt that they led me further away from the majority of users. Every time explored a different key-mapping (not key-binding) it reduced accessibility for new users. Each time I tried to work around that hiccup. The last pursuit was `;` and space.

It would be great to set up your keyboard with the meta keys on the bottom like this:

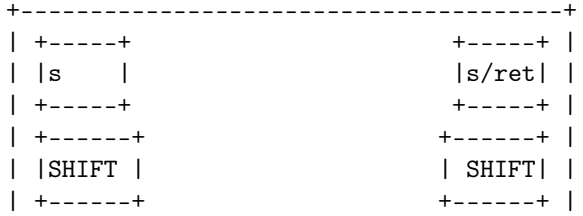


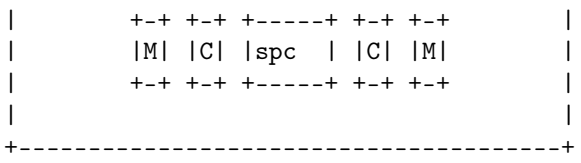
Karabiner was too slow for my typing speed though. It happens. It seemed like a minimal change to use Ukelele to:

- Make space send `C`
- Make `;` send space
- Make `'` a dead key
  - In it's dead key state make
    - `;` → `;`
    - `:` → `:`
    - `'` → `'`
    - `"` → `"`

The trouble is that it violates the POLA. Therefore, I left it alone and stuck with a simple "Get C on both sides".

That has worked out very well. It is very easy to do on every operating system. It holds true to the values of this system. When you develop an idea of `places` and how often you go there the key-mapping becomes more obvious. Make it easy to get to key-bindings that take you to familiar places. For this system it is the home keys, `s`, and `key-chord`. Make those keys more easily accessible. `C` and `M` often have symmetric-definitions. `s` and `SHIFT` also often have symmetric definitions in this system (mostly through Key-Chords). Therefore those key-mappings are kept close together





On OS X Dyalog uses an input source for entering APL symbols. Enable its input source, press option-i, and you get the symbol (APL FUNCTIONAL SYMBOL IOTA). It should work in every OS X application. When I tried to use it in Emacs though it didn't work. This configuration swallows the option key and translates it to the Emacs Super key. Any option chords are never sent to OS X. This had worked fine because I'd never used command. Most of the time I want Emacs to work this way, but some of the time I want to pass command through to OS X. How can I do this?

When I read the Emacs documentation they explained that you can choose whether or not you want ALT or GUI (option and command on OS X) to be swallowed by Emacs or passed through to the OS. Most of us already have this in our config. How can we switch it back and forth then?

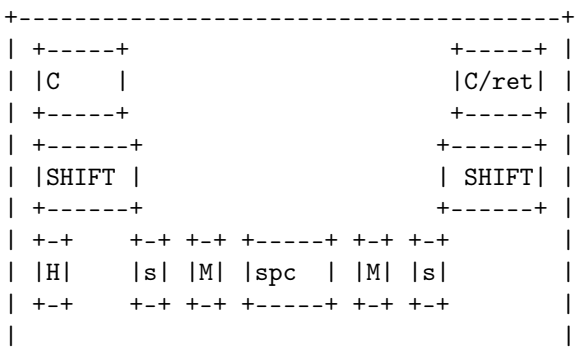
Elisp is of course the solution so I wrote some code to toggle this behavior. It didn't work! Didn't look like a bug in my code so I dug a little further into the documentation. Around the same time I was reading the USB HID specification. The spec explains that for every modifier there is both a left and a right code. When I got to the relevant document for Emacs it all started to make sense: the toggling modifier keys in Emacs only works for the right keys!

It is exciting to learn that Emacs makes this distinction. What I found next though was that it only works when the OS tells Emacs which modifier it is, but not from Karabiner. It makes sense, Emacs expects to be told at a lower level than Karabiner runs. Long story short I had to change the layout to get the toggling behavior that I wanted.

For simplicity I added a Hydra to toggle both the option and the function key between being sent to the OS or swallowed by Emacs.

Function started making a place in my config as Hyper after I got tired of searching for combinations of Hydras for common actions.

These changes helped guide the custom keyboard design.



```
+--------------------------------------+
```

(e) Take a Sip

ID: org_gcr_2017-05-12_mara:D33533BC-3E1F-4E7E-B9B3-2C0D0518400E

i. Left Side

ID: org_gcr_2017-05-12_mara:6F293EA2-D2D4-42F7-8684-D5B53CB849DA

A. Row 5

ID: org_gcr_2017-05-12_mara:5E250B06-C35C-4AEE-90C6-F5333A2D1BE3

```
(global-set-key (kbd "C-8") (lambda () (interactive) (switch-to-buffer
                                                      "*scratch*")))
(global-set-key (kbd "C-9") (lambda () (interactive) (switch-to-buffer
                                                      "projects.org")))
(global-set-key (kbd "C-0") (lambda () (interactive) (switch-to-buffer
                                                      "scratch.org")))
(global-set-key (kbd "M-9") (lambda () (interactive) (switch-to-buffer
                                                      "help.org")))
(global-set-key (kbd "C-5") #'ido-kill-buffer)
(global-set-key (kbd "C--") (lambda () (interactive) (insert "Vigneswari")))
(global-set-key (kbd "s-5") #'mc/mark-previous-like-this)
(global-set-key (kbd "s-4") #'mc/mark-next-like-this)
(global-set-key (kbd "s-3") #'mc/mark-previous-like-this)
(global-set-key (kbd "s-2") #'mc/mark-all-like-this)
(global-set-key (kbd "s-1") #'mc/edit-lines)
(global-set-key (kbd "s--") #'decrement-integer-at-point)
(global-set-key (kbd "s-+") #'increment-integer-at-point)
```

Checking things.

```
(defhydra help/hydra-checking (:color blue
                                      :hint nil)
  "
Flycheck On? %(bound-and-true-p flycheck-mode)
 WriteGood On? %(bound-and-true-p writegood-mode)
  _j_ checker/toggle _k_ checker/run _l_ checker/list
   _u_ writegood/toggle _i_ grade-level _o_ reading-ease
    _q_ quit
"
  ("j" flycheck-mode :exit nil)
  ("k" flycheck-buffer :exit nil)
  ("l" help/safb-flycheck-list-errors)
  ("u" writegood-mode)
  ("i" writegood-grade-level :exit nil)
  ("o" writegood-reading-ease :exit nil)
  ("q" nil))
(global-set-key (kbd "C-M-7") #'help/hydra-checking/body)
```

B. Row 4

ID: org_gcr_2017-05-12_mara:20E96ACD-1D91-480C-BA3B-D5730EA2173C

```
(global-set-key (kbd "C-M-1") #'help/1-window)
(global-set-key (kbd "C-M-2") #'help/2-window)
(global-set-key (kbd "C-M-3") #'help/3-window)
(global-set-key (kbd "C-M-4") #'help/4-window)
(global-set-key (kbd "s-w") (lambda ()
                              (interactive)
                              (save-excursion
                                (call-interactively 'mark-whole-buffer)
                                (call-interactively 'kill-ring-save))))
(global-set-key (kbd "s-q") #'kill-buffer)
(global-set-key (kbd "C-M-y") #'insert-char)
(global-set-key [(control meta ?p)] #'help/insert-datestamp)
(global-set-key [(control meta shift ?p)] #'help/insert-timestamp*-no-colons)
(global-set-key (kbd "C-M-o") #'help/occur-dwim)
(global-set-key (kbd "M-i") nil)
(global-set-key (kbd "M-i") #'describe-symbol)
```

Org handle `TAB` correctly and nothing is bound to it like `Auto-Complete` or `yasnippet` so this doesn't *need* ot use `C-i` freeing it up. Rebinding `C-g` seems to make the minibuffer, `eval-*` and `smex` become confused → quit working.

C. Row 3

ID: org_gcr_2017-05-12_mara:C0B604C5-F530-4770-9061-92E8468C8557

Eval expression.

```
(global-unset-key (kbd "C-M-j"))
(global-set-key (kbd "M-:") #'my-eval-expression)
(global-set-key (kbd "C-c C-k") #'help/delete-this-buffer-and-file)
```

```
(global-set-key (kbd "C-h") nil)
(global-set-key (kbd "C-h") #'ace-window)
```

In Org-Mode I constantly use `s-h` for tangling. When I forget that I am *out of* Org-Mode and I hit `s-h` it hides Emacs. Oops!

```
(global-set-key (kbd "s-h") nil)
```

```
(global-set-key (kbd "s-a") #'help/safb-switch-to-previous-buffer)
(global-set-key (kbd "s-d") #'er/expand-region)
```

```
(defhydra help/hydra/left-side/global (:color blue
                                                :hint nil)
  "
_0_ base64-encode-region _P_ base64-decode-region _|_ split-window-horizontall
_1_ reset-font _2_ -font  _3_ +font _4_ ellipsis _5_ UUID _6_ bfr-cdng-systm _
_Q_ exit-Emacs _q_uit T_ trademarks
_a_ ag  _A_ apropo'ish _s_ help/toggle-mac-right-option-modifier _S_ help/togg
_l_ visual-line-mode _L_ aggressive-indent-mode
```

```
_x_ delete-indentation _X_pm grok _c_ fill-paragraph _V_ view-mode _b_ erase-b
_<_ cmtIn _>_ cmtOut _?_ snp"
  ("Q" help/safb-save-buffers-kill-terminal)
  ("q" nil)
  ("|" split-window-horizontally)
  ("-" split-window-vertically)
  ("1" help/font-size-reset :exit nil)
  ("2" help/text-scale-decrease :exit nil)
  ("3" help/text-scale-increase :exit nil)
  ("4" help/insert-ellipsis)
  ("r" count-words)
  ("R" help/rename-current-buffer-file)
  ("T" help/trademark/body)
  ("=" reposition-window)
  ("5" help/uuid)
  ("6" set-buffer-file-coding-system)
  ("a" ag)
  ("A" hydra-apropos-ish/body)
  ("s" help/toggle-mac-right-option-modifier)
  ("g" grep)
  ("h" hack-local-variable)
  ("l" visual-line-mode)
  ("L" aggressive-indent-mode)
  ("S" help/toggle-mac-function-modifier)
  ("x" delete-indentation)
  ("X" xpm-grok)
  ("f" describe-key)
  ("V" view-mode)
  ("w" widen)
  ("t" rectangle-mark-mode)
  ("y" hydra-transparency/body)
  ("j" org-babel-tangle-jump-to-org)
  ("u" ucs-insert)
  ("i" scroll-down-command :exit nil)
  ("d" help/describe-char)
  ("D" help/safb-org-babel-detangle)
  ("k" scroll-up-command :exit nil)
  ("I" previous-logical-line :exit nil)
  ("K" next-logical-line :exit nil)
  ("n" help/safb-normal-mode)
  ("m" describe-mode)
  ("M" help/checks/body)
  ("<" help/chs)
  (">" help/che)
  ("." help/parent-mode-display)
  ("?" help/insert-noticeable-snip-comment-line)
  (";" isearch-toggle-lax-whitespace)
  ("o" toggle-debug-on-error)
```

```
  ("p" anzu-query-replace)
  ("O" base64-encode-region)
  ("P" base64-decode-region)
  ("}" help/hydra/transliterate/body)
  ("c" fill-paragraph )
  ("b" erase-buffer)
  ("B" help/bibtex/body)
  ("'" help/move-file)
  ("\"" imenu-list-minor-mode))

(key-chord-define-global "vv" #'help/hydra/left-side/global/body)
```

Attribution.

```
(defhydra hydra-apropos-ish (:color blue
                                    :hint nil)
  "
_a_propos        _c_ommand
_d_ocumentation  _l_ibrary
_v_ariable       _u_ser-option
valu_e_          _i_nfo
e_m_acs          elis_p_
"
  ("a" apropos)
  ("d" apropos-documentation)
  ("v" apropos-variable)
  ("c" apropos-command)
  ("l" apropos-library)
  ("u" apropos-user-option)
  ("i" info-apropos)
  ("e" apropos-value)
  ("m" emacs-index-search)
  ("p" elisp-index-search))
```

Trademark.

```
(defhydra help/trademark (:color blue :hint nil)
  "
American _R_egistered Trademark
American _U_nregistered Trademark
American _S_ervice Mark
_J_apanese Industrial Standard
_K_orean Standard
"
  ("R" (lambda () (interactive) (insert "®")))
  ("U" (lambda () (interactive) (insert "™")))
  ("S" (lambda () (interactive) (insert "℠")))
  ("J" (lambda () (interactive) (insert "")))
  ("K" (lambda () (interactive) (insert ""))))
```

Kinds of checks.

```
(defhydra help/checks (:color blue :hint nil)
  "
_q_  _w_  _e_  _r_  _t_ _
"
  ("q" (lambda () (interactive) (insert "")) :exit nil)
  ("w" (lambda () (interactive) (insert "")) :exit nil)
  ("e" (lambda () (interactive) (insert "")) :exit nil)
  ("r" (lambda () (interactive) (insert "")) :exit nil)
  ("t" (lambda () (interactive) (insert "")) :exit nil))
```

BibTeX stuff.

```
(defhydra help/bibtex (:color blue :hint nil)
  "
_c_ Clean Entry _C_ Clean Entry & Create New Reference
_f_ Reindent Field _F_ Reindent Buffer/Region
_m_ Disable Unhelpful Modes _M_ Enable Unhelpful Modes
"
  ("c" bibtex-clean-entry)
  ("C" (lambda () (interactive) (bibtex-clean-entry 't)))
  ("f" #'bibtex-fill-entry)
  ("F" #'bibtex-reformat)
  ;; Disable them
  ("m" (lambda () (interactive)
         (progn (aggressive-indent-mode 'toggle)
                (visual-line-mode 't))))
  ;; Enable them
  ("M" (lambda () (interactive)
         (progn (aggressive-indent-mode nil)
                (visual-line-mode nil)))))
```

D. Row 2

```
ID: org_gcr_2017-07-17_mara:B91C9779-B2AD-403B-A11F-5847EF8883C6
```

:ID: $\text{org}_{\text{gcr2017}}$-05-$12_{\text{mara}}$:8CA5EED7-5301-43E2-B6B9-C92AC66A2BE6

```
(global-set-key (kbd "s-v") #'ido-find-file)
(global-set-key (kbd "C-x C-c") #'help/safb-save-buffers-kill-terminal)
(define-key org-mode-map (kbd "C-,") nil)
(global-set-key (kbd "C-,") #'ido-switch-buffer)
(global-set-key (kbd "C-M-,") #'ibuffer)
(global-set-key (kbd "C-.") nil)
(global-set-key (kbd "C-.") #'smex)
(global-set-key (kbd "C-M-.") nil)
(global-set-key (kbd "C-M-.") #'dired)
(global-set-key (kbd "s-<") (lambda () (interactive) (insert "«")))
(global-set-key (kbd "s->") (lambda () (interactive) (insert "»")))
```

E. Row 1

```
ID: org_gcr_2017-05-12_mara:A791BE09-ED4B-46F4-8225-EE73C15C1DD0
```

Cycle through common options for removing space between characters. Thank you Pragmatic Emacs.

```
(global-set-key (kbd "s-SPC") #'cycle-spacing)
```

F. Unsorted

`ID: org_gcr_2017-05-12_mara:4FEBE5F3-3B9E-438D-9E7F-DA337FB43BB7`

Powerful buffer movement. Split between home and lower because

- It is important if you have multiple windows
  - So it is close to home
- It interrupts flow naturally
  - So it isn't on the home row but it is close

Can't use `nv` because of:

- Canvas
- Conve*
- Envision
- Inverse

There are a lot. Ways to find them and where

Toggle utility buffers ("logical F" key, so left side; "logical J" key on right).

```
(key-chord-define-global "f9" #'help/util-cycle)
```

Hide and show code blocks.

```
(global-set-key (kbd "s-b") #'hs-toggle-hiding)
(global-set-key (kbd "M-s-b") #'help/my-toggle-hideshow-all)
```

ii. Left & Right Side

`ID: org_gcr_2017-05-12_mara:AF27A71C-C30D-45C8-B3E7-BB044506B534`

Exploratory programming in EMACS.

Don't use "qi"; "unique".

```
(global-set-key (kbd "s-‘") #'help/comment-or-uncomment)
```

Make `ispell` accessible.

```
(key-chord-define-global "qp" #'ispell)
(key-chord-define-global "qo" #'ispell-word)
```

Use the default Langtool bindings.

```
(define-prefix-command 'help/langtool-map)
(key-chord-define-global "qk" #'help/langtool-map)
(define-key help/langtool-map "c" #'langtool-check-buffer)
(define-key help/langtool-map "C" #'langtool-correct-buffer)
```

```
(define-key help/langtool-map "j" #'langtool-goto-previous-error)
(define-key help/langtool-map "k" #'langtool-show-message-at-point)
(define-key help/langtool-map "l" #'langtool-goto-next-error)
(define-key help/langtool-map "q" #'langtool-check-done)
```

Easily kill buffers. Can't use "df" because of "PDF".

Can't use "fr" because of "fright" and "France".

Can't use "dc" because of `cd (change directory)`.

Can't use "gr" brecause of "Grant" and "Great.

Go to a word.

```
(key-chord-define-global "fj" #'avy-goto-word-1)
(key-chord-define-global "fm" #'avy-goto-char)
(key-chord-define-global "FJ" #'avy-pop-mark)
```

Go to a line.

```
(key-chord-define-global "fk" #'help/safb-help/goto-line)
```

Pop the mark back.

```
(key-chord-define-global "FK" #'pop-to-mark-command)
```

iii. Right Side

`ID: org_gcr_2017-05-12_mara:6F9375AB-C92C-4986-AECE-ABC945375CA2`

Try to reserve the right side for mode-specific activities.

iv. Exceptions

`ID: org_gcr_2017-05-12_mara:187B7649-D799-4E41-9DE7-448BC50E0250`

Return.

Do smart new line inside, indenting given the mode.

```
(help/not-on-gui (global-set-key (kbd "s-RET") #'help/smart-open-line))
(help/on-gui (global-set-key (kbd "s-<return>") #'help/smart-open-line))
```

Move the buffer contents and cursor up or down by one line

```
(global-set-key (kbd "M-p") (kbd "C-u 1 C-v"))
(global-set-key (kbd "M-n") (kbd "C-u 1 M-v"))
```

```
(global-set-key (kbd "C-n") #'next-logical-line)
(global-set-key (kbd "C-s-n") #'next-line)
(global-set-key (kbd "C-p") #'previous-logical-line)
(global-set-key (kbd "C-s-p") #'previous-line)
```

Ansu.

```
(global-set-key (kbd "M-%") #'anzu-query-replace)
(global-set-key (kbd "C-M-%") #'anzu-query-replace-regexp)
```