

Contents

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 --fo
```

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

EMACS values lexical-scoping in the future.

There are peculiar 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-autoload
```

C. With Make use make.

iv. Clone Org2Blog to ~/src/.

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

vi. Install supporting software adding their executable 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 from 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. This is the most technical name that you can and do use in scripts It is meant for computers.
 - org-mode
 - * Class and Object names
 - * Shell commands that are executable
 - * Executable for programming language compilers and interpreters. To clarify these are either the executable names or the human names for example gcc versus ~C.
 - gcc and python
 - bash and perl

- scheme and java
 - GCC and Python
 - Bash and Perl
 - Scheme and Java
 - * Anything that you would read program or text
 - * Use code style of tildas
- 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-invoke 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 Emacs 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 L^AT_EX 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 Emacs 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) Benchmark

ID: org_gcr_2018-09-21T18-57-01-05-00_cosmicality:32215650-08D5-448E-8

Report startup benchmarks from here forward.

```
(use-package benchmark-init
  :ensure t
  :config
  (defun help/benchmark-deactivate ()
    (benchmark-init/deactivate)
    (benchmark-init/show-durations-tabulated)
    (benchmark-init/show-durations-tree))
  (add-hook 'after-init-hook #'help/benchmark-deactivate))
```

(b) 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)
```

(c) 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))
```

(d) Hydra

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

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

(e) 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)
```

(f) 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)
```

CL-LIB.

```
(require 'cl-lib)
```

(g) 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)
```

Emacs mode for hiding the mode-line from Bastien.

```
(defvar-local hidden-mode-line-mode nil)
(define-minor-mode hidden-mode-line-mode
  "Minor mode to hide the mode-line in the current buffer."
  :init-value nil
  :global t
  :variable hidden-mode-line-mode
  :group 'editing-basics
  (if hidden-mode-line-mode
      (setq hide-mode-line mode-line-format
            mode-line-format nil)
      (setq mode-line-format hide-mode-line
            hide-mode-line nil))
  (force-mode-line-update)
  ;; Apparently force-mode-line-update is not always enough
  to
  ;; redisplay the mode-line
  (redraw-display)
  (when (and (called-interactively-p 'interactive)
             hidden-mode-line-mode)
    (run-with-idle-timer
     0 nil 'message
     (concat "Hidden Mode Line Mode enabled.  "
             "Use M-x hidden-mode-line-mode to make the
             mode-line appear."))))
```

Fancy progress indicator.

```
(use-package unicode-progress-reporter
  :ensure t
  :config
  (unicode-progress-reporter-setup))
```

(h) macOS

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

Make it easy to evaluate code only when running on macOS

```
(defmacro help/on-mac-os (statement &rest statements)
  "Evaluate the enclosed body only when run on macOS."
```

```
`(when (eq system-type 'darwin)
  ,statement
  ,@statements))
```

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

```
(help/on-mac-os
 (use-package exec-path-from-shell
  :ensure t
  :config
  (setq exec-path-from-shell-check-startup-files nil)
  (setq exec-path-from-shell-debug 1)
  (exec-path-from-shell-initialize)))
```

Configure the meta keys.

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

Easily allow option pass through for alternate input methods.

```
(help/on-mac-os (load-file "~/src/help/.mac-os-keys.el"))
```

Run a bare Emacs instance on macOS like this:

```
emacs --debug-init -no-init-file --load ~/src/help/.mac-os-keys.el&
```

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

This is a known issue. Here is the solution.

```
(help/on-mac-os
 (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))
```

(i) 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))
```

(j) Linux

ID: org_gcr_2017-07-23_mara:7D02C86B-CDBE-466B-9647-3EDA6DB016C6

Make it easy to evaluate code only when running on .

```
(defmacro help/on-gnu/linux (statement &rest statements)
  "Evaluate the enclosed body only when run on GNU/Linux"
  `(when (eq system-type 'gnu/linux)
    ,statement
    ,@statements))
```

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/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 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/indent-curly-block (&rest _ignored)
  "Open a new brace or bracket expression, with relevant
  newlines and indent. URL: `https://github.com/Fucol/
  smartparens/issues/80'"
  (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)
          (insert "()")
          (backward-char))
        (read-from-minibuffer prompt initial-contents
                               my-read-expression-map nil
                               'read-expression-history))))
  (defun my-eval-expression (expression &optional arg)
    "Evaluate EXPRESSION adding the result to the kill-ring
then
either display it in a buffer or with a prefix argument
ARG
insert it into this buffer.
Attribution: URL `https://lists.gnu.org/archive/html/
help-gnu-emacs/2014-07/msg00135.html'."
    (interactive (list (read (my-read--expression "[]: ")
)
                  current-prefix-arg))
    (let* ((it (eval expression lexical-binding))

```

```

        (itstr (pp-to-string it)))
      (kill-new itstr)
      (if arg
        (insert itstr)
        (pp-display-expression it
                               "*HELP 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-emacs/'"
```

```

(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?source=rss'"
  (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/2008-06/msg00087.html'"
  `(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-parent-directories-on-visiting-a-new-file-in-em
/'"
  (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-directory))))
      (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 sachu/unfill-paragraph (&optional region)
  "Takes a multi-line paragraph and makes it into a single
line of text.
Attribution: URL https://github.com/sachac/.emacs.d/blob/
gh-pages/Sacha.org#unfill-paragraph"
  (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)))
(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-at-point/'"
(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-at-point/'"
(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-at-point/'"
  (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-at-point/'"

```

```

    (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-at-point/'"
    (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-at-point/'"
    (interactive "p")
    (increment-integer-at-point (- (or dec 1))))
  (defun switch-to-previous-buffer ()
    "Switch to most recent buffer. Repeated calls toggle back
    and forth between the most recent two buffers.
  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-mac-os
 (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 (buffer-name))
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-ignore-case ()
  "Sort lines ignoring case.
Attribution: `https://stackoverflow.com/questions/20967818/
emacs-function-to-case-insensitive-sort-lines'"
  (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-buffer-in-emacs.html'"
  (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: Happiness, Health, philosophy, Yoga
#+TITLE: (Happiness) "))
(defun help/wio ()
  (interactive)
  (when (use-region-p) (call-interactively 'kill-region))
  (insert "#+CATEGORY: Article
#+TAGS: Babel, Emacs, Ide, Lisp, Literate Programming,
Programming Language, Reproducible research, elisp,
org-mode
#+TITLE: (Emacs+Org-Mode) "))

```

```

(defun help/wip ()
  (interactive)
  (when (use-region-p) (call-interactively 'kill-region))
  (let ((name (read-string "Programming language name? ")))
    (insert (format
              "#+CATEGORY: Article
#+TAGS: Programming Language, %s
#+TITLE: (%s) " name name))))
  (defconst help/buffalo "[]() Buffalo buffalo Buffalo buffalo
  buffalo buffalo Buffalo buffalo []()"
    "URL: `https://en.wikipedia.org/wiki/
  Buffalo_buffalo_Buffalo_buffalo_buffalo_buffalo_Buffalo_buffalo
  '")
  (defun help/safb/compile ()
    "Save all file buffers and call `compile'"
    (interactive)
    (help/save-all-file-buffers)
    (call-interactively 'compile))
  (defun help/insert-formfeed ()
    (interactive)
    (insert "\f"))
  (defun help/itwas ()
    "Attribution: W<"
    (interactive)
    (insert-file-contents-literally "~/src/help/ascii/
  it-was-a-dark-and-stormy-night.asc"))
  (defun help/pie-blurb ()
    "Attribution: FontAwesome"
    (interactive)
    (insert-file-contents-literally "~/src/help/ascii/pie.asc
  "))
  (defun help/remove-vowel ($string &optional $from $to)
    "Remove the following letters: {a e i o u}.
  When called interactively, work on current paragraph or
  text selection.
  When called in lisp code, if $string is non-nil, returns a
  changed string.
  If $string nil, change the text in the region between
  positions $from $to.
  Attribution: URL `http://ergoemacs.org/emacs/
  elisp_command_working_on_string_or_region.html'"
    (interactive)
    (if (use-region-p)
        (list nil (region-beginning) (region-end))
        (let ((bds (bounds-of-thing-at-point 'paragraph)) )

```

```

      (list nil (car bds) (cdr bds)) ) ) )
(let (workOnStringP inputStr outputStr)
  (setq workOnStringP (if $string t nil))
  (setq inputStr (if workOnStringP $string (
buffer-substring-no-properties $from $to)))
  (setq outputStr
    (let ((case-fold-search t))
      (replace-regexp-in-string "a\\|e\\|i\\|o\\|u\\|
" "" inputStr) ) )
  (if workOnStringP
    outputStr
    (save-excursion
      (delete-region $from $to)
      (goto-char $from)
      (insert outputStr) )) ) )
(defun help/indent-buffer ()
  "Indent the currently visited buffer.
URL: `http://emacsredux.com/blog/2013/03/27/
indent-region-or-buffer/'"
  (interactive)
  (indent-region (point-min) (point-max)))
(defun help/indent-region-or-buffer ()
  "Indent a region if selected, otherwise the whole buffer.
URL: `http://emacsredux.com/blog/2013/03/27/
indent-region-or-buffer/'"
  (interactive)
  (save-excursion
    (if (region-active-p)
      (progn
        (indent-region (region-beginning) (region-end))
        (message "Indented selected region."))
      (progn
        (help/indent-buffer)
        (message "Indented buffer."))))))
(defun help/alist-set (key val alist &optional symbol)
  "Set property KEY to VAL in ALIST. Return new alist.
This creates the association if it is missing, and
otherwise sets
the cdr of the first matching association in the list. It
does
not create duplicate associations. By default, key
comparison is
done with `equal'. However, if SYMBOL is non-nil, then `eq'
is
used instead."

```

This method may mutate the original alist, but you still need to use the return value of this method instead of the original alist, to ensure correct results.

Atribution: URL ``https://emacs.stackexchange.com/a`

`/33893/341`"`

```
(if-let ((pair (if symbol (assq key alist) (assoc key
alist))))
```

```
  (setcdr pair val)
```

```
  (push (cons key val) alist))
```

```
alist)
```

```
(defun help/org-time-stamp-with-seconds-now ())
```

```
  (interactive)
```

```
  (let ((current-prefix-arg '(16)))
```

```
    (call-interactively 'org-time-stamp)))
```

```
(defun help/insert-datestamp-us ())
```

```
  "Produces and inserts a US datestamp."
```

```
  (interactive)
```

```
  (insert (format-time-string "%m/%d/%y")))
```

```
(defun help/insert-datestamp-us-full-year-and-dashes ())
```

```
  "Produces and inserts a US datestamp with full year and
  dashes."
```

```
  (interactive)
```

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

```
(defun help/insert-datestamp-us-full-year ())
```

```
  "Produces and inserts a US datestamp with full year."
```

```
  (interactive)
```

```
  (insert (format-time-string "%m/%d/%Y")))
```

```
(defun help/insert-datestamp-us-words ())
```

```
  "Produces and inserts a US datestamp using words."
```

```
  (interactive)
```

```
  (insert (format-time-string "%A %B %d, %Y")))
```

```
(defun help/insert-org-datestamp ())
```

```
  "Produces and inserts an Org-Mode timestamp."
```

```
  (interactive)
```

```
  (insert (format "<%s>"(format-time-string "%F"))))
```

```
(defun help/org-count-words ())
```

```
  "If region is active, count words in it; otherwise count
  words in current subtree."
```

```
  (interactive)
```

```
  (if (use-region-p)
```

```
    (funcall-interactively #'count-words-region (
```

```
    region-beginning) (region-end))
```

```
    (org-with-wide-buffer
```

```
      (cl-loop for (lines words characters)
```

```

        in (org-map-entries
            (lambda ()
              (help/org-forward-to-entry-content '
unsafe)
              (let ((end (org-entry-end-position)))
                (list (count-lines (point) end)
                      (count-words (point) end)
                      (- end (point)))))
            nil 'tree)
        sum lines into total-lines
        sum words into total-words
        sum characters into total-characters
        finally do (message "Subtree \"%s\" has %s
lines, %s words, and %s characters."
                           (org-get-heading t t)
                           total-lines
                           total-words
                           total-characters))))
(defun help/org-forward-to-entry-content (&optional unsafe)
  "Skip headline, planning line, and all drawers in current
entry.
If UNSAFE is non-nil, assume point is on headline."
  (unless unsafe
    (org-back-to-heading))
  (cl-loop for element = (org-element-at-point)
           for pos = (pcase element
                      (`(headline . ,_) (
org-element-property :contents-begin element))
                      (`(,(or 'planning 'property-drawer '
drawer) . ,_) (org-element-property :end element)))
           while pos
           do (goto-char pos)))
(defvar help/most-used-words-count 10 "Default most used
word count to show")
(defun help/most-used-words-fn (&optional n)
  "Make a list of the N most used words in buffer, or use
default.
Attribution: Udyant Wig <udyantw@gmail.com>"
  (let ((n (or n help/most-used-words-count))
        (counts (make-hash-table :test #'equal))
        (words (split-string (buffer-string)
                              sorted-counts))
        (dolist (word words)
          (let ((count (gethash (downcase word) counts 0)))
            (puthash (downcase word) (1+ count) counts))))

```

```

      (cl-loop for word being the hash-keys of counts
                using (hash-values count)
                do
                  (push (list word count) sorted-counts)
                  finally (setf sorted-counts (cl-sort
sorted-counts #'>
                                                                    :key #'
cl-second))))
      (mapcar #'cl-first (cl-subseq sorted-counts 0 n))))
(defun help/most-used-words ()
  "Report top N most used words in a new buffer, defaults
  to 1"
  (interactive)
  (let* ((input-row (read-string "Report the top N most
used words <hit enter for default>: " nil nil nil))
           (input-num (floor (string-to-number (string-trim
input-row))))
           (n (if (<= input-num 0) help/most-used-words-count
input-num))
           (words (help/most-used-words-fn n))
           (name (format "TOP %s USED WORDS IN %s" n (
buffer-name)))
           (underbar (make-string (length name) ?=))
           (buf (get-buffer-create (format "%s*"name))))
    (switch-to-buffer buf)
    (insert (format "%s\n%s\n" name underbar))
    (dolist (word words) (insert (format "%s\n" word)))
    (help-mode)
    (setq buffer-read-only t)
    (goto-char (point-min))))
(defun help/insert-it-then-real-space (char)
  "Insert CHAR followed by a space using a key event so
  modes can respond appropriately.
  I use this to automatically insert a space after
  punctuation characters. It
  might sound like a micro-optimization but I think of it
  more like making the
  keyboard work like it ought to work ;).
  When I first tried to set this up I bound the .-key to a
  function that inserted
  itself and then a space like this `(insert "\". \")'. This
  inserts what you
  want but doesn't generate a key event for other modes to
  handle. For example
  `auto-fill-mode' won't break the line because it doesn't

```



```

    know that anything
happened. When you do hit another key `auto-fill-mode'
    kicks in and breaks the-
lineI found that disturbing to my flow so I set this up
    instead.
Does the \"right thing\" under `org-mode'."
    (cl-flet ((do-insert
                () (if (bound-and-true-p org-mode)
                        (org-self-insert-command 1)
                        (self-insert-command 1))))
      (setq last-command-event char)
      (do-insert)
      (setq last-command-event ?\s)
      (do-insert)))
(defun help/real-insert (char)
  (cl-flet ((do-insert
              () (if (bound-and-true-p org-mode)
                      (org-self-insert-command 1)
                      (self-insert-command 1))))
    (setq last-command-event char)
    (do-insert)))
(defun help/dot-space () (interactive) (help/
  insert-it-then-real-space ?\.))
(defun help/dot-real () (interactive) (help/real-insert
  ?\.))
(defun help/comma-space () (interactive) (help/
  insert-it-then-real-space ?\,))
(defun help/comma-real () (interactive) (help/real-insert
  ?\,))

```

(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
- 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)
```

Don't correct quotes in text until I make ispell handle it correctly.

```
(electric-quote-mode)
```

(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 5)
```

Provide expected "Undo" functionality.

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

Every 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)
```

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))
```

Show text indentation guide lines.

highlight-indentation is too difficult to see.

```
(use-package indent-guide
  :ensure t
  :diminish 'indent-guide-mode)
```

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->
- 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
  (key-chord-define ibuffer-mode-map "hh" #'
    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)
```

Move region.

```
(use-package smart-shift
  :ensure t
  :config
  (global-smart-shift-mode 1)
  (defhydra help/hydra/smart-shift (:color blue
                                     :hint nil)
    "
    Smart-Shift: (q to quit)
      ↑
    _i_ ←
  _j_ _l_ →
    _k_
      ↓
    "
    ("j" smart-shift-left :exit nil)
    ("k" smart-shift-down :exit nil)
    ("i" smart-shift-up :exit nil)
    ("l" smart-shift-right :exit nil)
    ("q" nil)))
```

Unfilling

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

(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)

```

Fold on indentation level.

```

(defun aj-toggle-fold ()
  "Toggle fold all lines larger than indentation on current
line
URL: `https://stackoverflow.com/questions/1587972/
how-to-display-indentation-guides-in-emacs
/4459159#4459159'"
  (interactive)
  (let ((col 1))
    (save-excursion
      (back-to-indentation)
      (setq col (+ 1 (current-column)))
      (set-selective-display
        (if selective-display nil (or col 1))))))
(global-set-key (kbd "s-q") #'aj-toggle-fold)

```


(e) 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)
```

(f) 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)
```

(g) 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)
```

(h) 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 `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-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)
      " ")))
```

Mode hook.

```
(defun help/eshell-mode-hook-fn ()
  (message "Welcome to Eshell.")
  (setq pcomplete-cycle-completions nil))
(add-hook 'eshell-mode-hook #'help/eshell-mode-hook-fn)
```

(i) 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)
  (vc-revert))
(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 began at: "
                     (help/get-timestamp)))
    (org-babel-tangle)
    (let* ((dur (float-time (time-since start)))
           (msg (format "Tangled for %s " (
seconds-to-string dur)))))
      (message (concat "org-babel-tangle completed on: "
                      (help/get-timestamp)))
      (message msg)
      (help/on-gui (alert msg :title "Org-Mode")))))
(defun help/safb-org-babel-detangle ()
  (interactive)
  (help/save-all-file-buffers)
  (let ((start (current-time)))
    (message (concat "org-babel-detangle began at: "
                     (help/get-timestamp)
                     ">"))
    (org-babel-detangle)
    (let* ((dur (float-time (time-since start)))
           (msg (format "DE-Tangled for %s" (
seconds-to-string dur)))))
      (message (concat "org-babel-detangle on: "
                      (help/get-timestamp)))
      (message msg)
      (help/on-gui (alert msg :title "Org-Mode")))))
(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
- eval-defun
- eval-region
- help/newline
- goto-line
- 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))
```

Show real end of lines.

```
(setq inhibit-eol-conversion t)
```

(j) 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))
```

(k) Interactive User Interface Completion

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

```
(use-package counsel
  :ensure t
  :config
  (use-package amx
    :ensure t)
  (use-package flx
    :ensure t)
  (ivy-mode 1)
  (diminish 'ivy-mode)
  (setq ivy-use-virtual-buffers t)
  ;; intentional space before end of string
  (setq ivy-count-format "(%d/%d) ")
  (setq ivy-initial-inputs-alist nil)
  (setq ivy-re-builders-alist
    '((t . ivy--regex-fuzzy)))
  (diminish 'ivy-mode))
```

(l) Font

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

Activate font locking everywhere possible.

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

Adds FontAwesome.

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

(m) 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)
```

(n) 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"))
  (let ((lang "en-US"))
    (setq langtool-default-language lang)
    (setq langtool-mother-tongue lang))
  (setq langtool-java-bin (concat (getenv "JAVA_HOME") "/"
    bin/java"))
  (setq langtool-disabled-rules nil)
  (setq langtool-java-user-arguments
    '("-Dfile.encoding=UTF-8" ))
  (setq langtool-user-arguments nil)
  (setq langtool-disabled-rules
    '(
      "DASH_RULE"
      "WHITESPACE_RULE"
      "PUNCTUATION_PARAGRAPH_END"
      "EN_QUOTES"
    ))
  (defhydra help/hydra/both/langtool (:color blue :hint nil
    )
    "
    Langtool: ^          ^|^          ^|^
              ^^+^      ^^+^
    -----
    _h_: check buffer   | _j_: next error   | _i_: brief
      message
    _y_: correct buffer | _k_: previous error | _o_: detailed
      message
    _n_: finished      | _q_: quit          |
    "
    ("h" langtool-check :exit nil))
```

```

("y" langtool-correct-buffer :exit nil)
("n" langtool-check-done)
("j" langtool-goto-next-error :exit nil)
("k" langtool-goto-previous-error :exit nil)
("i" langtool-show-brief-message-at-point :exit nil)
("o" langtool-show-message-at-point :exit nil)
("q" nil))
(key-chord-define-global "qk" #'help/hydra/both/langtool/
body))

```

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))

```

(o) 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))

```

(p) Word Completion

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/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)
```

(q) 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
    "
  ")
  )
(global-set-key (kbd "M-y") #'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)
```

(r) 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)
```

Temporarily give it unlimited memory to speed it up.

```
(add-hook 'minibuffer-setup-hook #'help/
  double-gc-cons-threshold)
(add-hook 'minibuffer-exit-hook #'help/
  set-gc-cons-threshold)
```

(s) 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)
```

(t) 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 t))
(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)
```

(u) 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-mac-os
      (setq alert-default-style 'notifier)))
  (setq alert-reveal-idle-time 120))
```

```
(require 'popup)
```

(v) 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 "C-S-o") #'projectile-find-file)
  (help/on-windows
    (setq projectile-indexing-method 'alien))
  (setq projectile-completion-system 'ivy)
  :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 (concat project "/.git/")))
        (projectile-relevant-known-projects)))
    magit-repository-directories-depth 1)))
```

(w) Printing

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

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

(x) Register

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

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

(y) 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 " ⇒ "))
```

(z) 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

-
- 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 for 'foo', then 'Foo' and 'foo' also match. Regexp's, 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'. 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 match exactly, including case. This is a per-buffer variable; altering the value normally affects only the current buffer, unless you change its default. See Locals. This variable applies to nonincremental searches also, 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 the 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.

Make `isearch` automatically search for Unicode homoglyphs of the search characters.

```
(setq search-default-mode #'char-fold-to-regexp)
```

Ditto for replace.

```
(setq replace-char-fold t)
```

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)
  (add-to-list 'ag-arguments "--hidden")
  (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)
  (defhydra help/hydra/ag (:color blue
                           :hint nil)
    "
    `ag', The Silver Searcher:
```

```

Present in window:
  Search in folder:
    _j_ limit search by file type _k_ search in
    everything
  Search in project:
    _l_ limit search by file type _;_ search in
    everything
Present in dired:
  Search in folder:
    _u_ limit search by file type _i_ search in
    everything
  Search in project:
    _o_ limit search by file type _p_ search in
    everything
Other:
  _n_ close every other buffer _m_ close every buffer
  _q_ quit
"
  ("u" ag-dired-regexp)
  ("i" ag-dired)
  ("o" ag-project-dired-regexp)
  ("p" ag-project-dired)
  ("j" ag-files)
  ("k" ag)
  ("l" ag-project-files)
  (";" ag-project)
  ("n" ag-kill-other-buffers)
  ("m" ag-kill-buffers)
  ("q" nil)))

```

Writeable Ag.

```

(use-package wget-ag
  :ensure t
  :after ag)

```

Google.

```

(use-package google-this
  :ensure t
  :config
  (global-set-key (kbd "M-s-g") #'google-this))

```

() Spell Checking

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)))))))
```

() 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)
```

Functions to remove TAB

It is at least unpleasant removing tabs that somebody wanted in the file, and for that alone it is worth removing untabification from the on save buffer file hook. Worst case it ruins files for example Makefiles or tab-delimited data files.

```
(defun help/untabify-buffer-if-not-indent-tabs-mode ()
  "Untabify buffer 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))))
(defun help/
  untabify-buffer-or-region-if-not-indent-tabs-mode ()
  "Untabify a region if selected, otherwise the whole
  buffer.
URL: `http://emacsredux.com/blog/2013/03/27/
  indent-region-or-buffer/'"
  (interactive)
  (save-excursion
    (if (region-active-p)
        (progn
          (untabify (region-beginning) (region-end))
          (message "Untabified selected region.))
        (progn
          (help/untabify-buffer-if-not-indent-tabs-mode)
          (message "Untabified buffer.)))))
```

JavaScript programmers set tabs to 2. C programmers set it to 8. Makefiles looks great at 4. 4 is the average and it looks good in exported L^AT_EX listings.

```
(setq-default tab-width 4)
```

() TODOs

ID: org_gcr_2018-11-14T01-09-50-06-00_cosmicality:FBB5DEF5-F814-4F81-8

During coding *sometimes* there is stuff that I want to do later. Highlight them so I can see them and also navigate through them and occur on them too.

Org provides a TODO mechanism for itself so don't use this there.

Normally I would say that this violates the law of least distraction but the don't stay in there very long: either I fix it in the next few days or it add it to an Org todo list in my work list file.

```
(use-package hl-todo
  :ensure t
  :config
  (define-key hl-todo-mode-map (kbd "C-c p") '
    hl-todo-previous)
  (define-key hl-todo-mode-map (kbd "C-c n") 'hl-todo-next)
  (define-key hl-todo-mode-map (kbd "C-c o") 'hl-todo-occur
  )
  (global-hl-todo-mode))
```

() Version Control

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

Use VC for single files and Magit for multiple files.

```
(use-package magit
  :ensure t
  :config
  (with-eval-after-load 'info
    (info-initialize)
    (add-to-list 'Info-directory-list ;; TODO remove this
      "~/src/magit/Documentation/"))
  (setq magit-completing-read-function 'ivy-completing-read
  ))
```

Close Magit-process related buffers when they become obsolete.

```
(defun help/magit-kill-buffers ()
  "Restore window configuration and kill all Magit buffers.
Attribution: URL `https://manuel-uberti.github.io/emacs
/2018/02/17/magit-bury-buffer/'"
  (interactive)
  (let ((buffers (magit-mode-get-buffers)))
    (magit-restore-window-configuration)
    (mapc #'kill-buffer buffers)))
(bind-key "q" #'help/magit-kill-buffers
  magit-status-mode-map)
```

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 "C-M-w") #'help/hydra/words/body)
(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)
```

Quick note about intent of the change.

```
(defhydra help/hydra/commit-message (:color blue
                                      :hint nil)
```

```

"
Commit Message Descriptor (q to quit)
_E_nhances: \"an increase or improvement in quality, value,
  or extent.\"
_R_efines: \"the improvement or clarification of something
  by the making of small changes.\"
_C_orrects: \"a change that rectifies an error or
  inaccuracy.\"
C_l_oses | _F_fixes | Resol_v_es
_A_rchives | Rest_o_res | Re_f_iles
"

("E" (lambda () (interactive) (insert "(nhncs) ")))
("R" (lambda () (interactive) (insert "(rfns) ")))
("C" (lambda () (interactive) (insert "(crrcts) ")))
("l" (lambda () (interactive) (insert "Closes #")))
("F" (lambda () (interactive) (insert "Fixes #")))
("v" (lambda () (interactive) (insert "Resolves #")))
("A" (lambda () (interactive) (insert "(rchvs) ")))
("o" (lambda () (interactive) (insert "(rstrs) ")))
("f" (lambda () (interactive) (insert "(rfls) ")))
("q" nil))

```

Commit message mode hook.

```

(defun help/commit-message-mode-hook-fn ()
  "HELP Commit Message Mode Hook Function"
  (key-chord-define-local "hh" #'help/hydra/commit-message/
    body))
(add-hook 'vc-git-log-edit-mode-hook #'help/
  commit-message-mode-hook-fn)
(add-hook 'git-commit-setup-hook #'help/
  commit-message-mode-hook-fn)

```

Smerge.

```
(diminish "smerge-mode")
```

() 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 𐀀[? ?\t] 𐀁[? ?\t] [?\t]))
    (setq whitespace-line-column help/column-width)
    (global-whitespace-mode nil)
    (setq whitespace-global-modes '(not go-mode))
    :diminish whitespace-mode global-whitespace-mode)

```

() whoami

ID: org_gcr_2018-06-17T11-45-33-05-00_mara:F4562648-EFF8-4C9A-8F2E-08F

```
(setq user-mail-address "gcr@wisdomandwonder.com")
```

() Word Wrap

ID: org_gcr_2017-05-12_mara:202157D8-58BB-4250-BDDD-98ADB97F801

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

How VLM works here:

- At the end of visible line, which is where the wrap occurred, show a right curly arrow to draw your eye to the next line and back left to the beginning of it.
- At the start of this new visible line, where the wrap continued, show a right straight arrow to draw your eye forward to continue the wrapped line where you came from.

With these VLM settings, along with delete trailing whitespace on save, you don't need end of line indicators because every line is either:

- An end of line with no trailing spaces between the last character and the newline
- The line exceeds the maximum visible length desired so it has wrapped and you see arrows in the left and right fringe so you know that the isn't a newline in that visible line. You'll see a row that starts with a right arrow, and with nothing, and the line above that will have a right curly arrow.

```
(setq visual-line-fringe-indicators '(right-arrow
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

Auto-Capitalize text modes.

```
(add-to-list 'load-path "~/src/auto-capitalize")
(eval-after-load "auto-capitalize" '(diminish "
  auto-capitalize-mode"))
(require 'auto-capitalize)
(diminish 'auto-capitalize-mode)
(dolist (it '(
  "24B"
  "Fr"
  "God"
  "HTTP"
  "HTTPS"
  "Mantra"
  "SI"
  "Sri"
  "StoneIsle"
  "Tantra"
  "Vidya"
  "Yantra"
  "Yoga"
))
  (add-to-list 'auto-capitalize-words it))
(add-to-list 'desktop-minor-mode-handlers
  '(auto-capitalize . (lambda (
    desktop-buffer-locals))))
```

Wrap Selection.

```
(use-package wrap-region
  :ensure t
  :config
  :diminish wrap-region-mode
  :config
  (wrap-region-add-wrapper "" "" nil 'text-mode)
  (wrap-region-add-wrapper "\"\" \"\" nil 'text-mode)
  (wrap-region-add-wrapper "*" "*" nil 'text-mode)
  (wrap-region-add-wrapper "/" "/" nil 'text-mode)
  (wrap-region-add-wrapper "=" "=" nil 'text-mode)
```

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

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)
  (help/try-to-add-imenu)
  (turn-on-page-break-lines-mode)
  (turn-on-auto-capitalize-mode)
  (wrap-region-mode)
  (local-set-key (kbd ".") #'help/dot-space)
  (local-set-key (kbd "C-.") #'help/dot-real)
  (local-set-key (kbd ",") #'help/comma-space)
  (local-set-key (kbd "C-,") #'help/comma-real)
  (local-set-key (kbd "C-n") #'next-logical-line)
  (local-set-key (kbd "C-p") #'previous-logical-line)
  (unless (eq major-mode 'org-mode)
    (orgtbl-mode nil)
    (diminish 'orgtbl-mode)))
  (add-hook 'text-mode-hook #'help/text-mode-fn))
```

Can use in source blocks.

```
(add-to-list 'org-babel-load-languages '(text . t))
```

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
yaml-mode-hook
crontab-mode-hook
apache-mode-hook
python-mode-hook
gnu-apl-mode-hook
geiser-mode-hook
geiser-repl-mode-hook
go-mode-hook
TeX-mode-hook
apples-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
    inferior-scheme-mode-hook
    geiser-mode-hook
    geiser-repl-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-A605-0D31C07F677

- Indent at every opportunity and automatically. Verify that it makes sense for the mode then enable or disable it. Originally enabled for individual modes. That worked fine until I wanted to disable it for a mode where the

mode hook ran before the prog-mode-hook ran. Well that didn't work because the child disabled this mode and then the parent enabled it again. You aren't supposed to rely on order of execution in them so I am switching to global mode and updating the excluded list in the same granular manner. In affect, nothing has changed.

```
(use-package aggressive-indent
  :ensure t
  :diminish aggressive-indent-mode
  :config
  (global-aggressive-indent-mode)
  (add-to-list 'aggressive-indent-excluded-modes
    'minibuffer-inactive-mode))
```

- 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-E9F0C86C23A

```
(defun help/prog-mode-hook-fn ()
  (interactive)
  (help/text-mode-fn)
  (smartparens-strict-mode)
  (unless (equal major-mode 'fundamental-mode)
    (hs-minor-mode))
  (help/on-gui (local-set-key (kbd "<return>") #'
    newline-and-indent))
  (turn-off-auto-capitalize-mode))
(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)
```

Symbol visualization.

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

Quasi-Quote visualization.

```
(use-package highlight-stages
  :ensure t
  :config
  (diminish 'highlight-stages-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 "s-0") #'pp-macroexpand-last-sexp)
(local-set-key (kbd "s-8") #'lispy-multiline)
(local-set-key (kbd "s-*) #'lispy-oneline)
(local-set-key (kbd "s-9") #'macrostep-expand))
(defhydra help/hydra-elisp-mode (:color blue
                                :hint nil)
  "
  elisp-mode: _q_uit
  Evaluation: buffe_r_, byte _c_ompile file
  ERT: _i_nteractively, _b_atc_h, _d_elete all
  Reformat s-expr: _m_ultiline or _o_ne line
  Other: Generate _R_eadme _C_heckdoc
  "
  ("q" nil)
  ("r" eval-buffer)
  ("c" (lambda () (interactive) (byte-compile-file (
    buffer-file-name))))
  ("i" (lambda () (interactive) (eval-buffer) (
    call-interactively 'ert)))
  ("b" (lambda () (interactive) (eval-buffer) (
    ert-run-tests-batch)))
  ("d" ert-delete-all-tests)
  ("m" lispy-multiline)
  ("o" lispy-oneline)
  ("R" (lambda () (interactive) (el2markdown-write-file
    "README.md" t)))
  ("C" checkdoc))
(key-chord-define emacs-lisp-mode-map "hh" #'help/
  hydra-elisp-mode/body)
(defun help/emacs-lisp-mode-hook-fn ()
  (interactive)
  (help/elisp-mode-local-bindings)
  (lexbind-mode)
  (eldoc-mode)
  (diminish 'eldoc-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)
(local-unset-key (kbd "C-j"))
(local-set-key (kbd "<C-return>") #'ielm-send-input))
(add-hook 'ielm-mode-hook #'help/ielm-mode-hook-fn)
(defun help/general-lisp-prettify ()
  (mapc (lambda (pair) (push pair prettify-symbols-alist
    ))
    '(("nil" . ø?)
      ("*" . x?)
      ("/" . ÷?)
      ("<=" . ≤?)
      (">=" . ≥?)
      ("lambda" . λ?)
      ("not" . ¬?)
      ("and" . ∧?)
      ("or" . ∨?)
      ("sqrt" . √?))))
(defun help/general-lisp-mode-hook-fn ()
  (highlight-quoted-mode)
  (highlight-stages-mode)
  (help/general-lisp-prettify)
  (local-set-key (kbd "#") #'endless/sharp))
(let ()
  (--each help/lisp-modes
    (add-hook it #'help/general-lisp-mode-hook-fn)))
(add-hook 'emacs-lisp-mode-hook #'help/
  emacs-lisp-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))

```

Lispy super editing functions.

```

(use-package lispy
  :ensure t)

```

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-AB12-525580DC8E

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/msg00534.html'"
  (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 / TO-DO Objects

ID: org_gcr_2017-05-12_mara:0CFAE46B-8D06-48FB-8D7E-7B636C91D44

- 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@)"
      "DUPLICATE(p@)"
    )))
```

- 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)
```

Inline Tasks.

```
(require 'org-inlinetask)
```

C. Refile

ID: org_gcr_2017-05-12_mara:C09D993C-6E48-4C84-8460-585F0A10DA6

D. Context

ID: org_gcr_2017-05-12_mara:DFCC7C00-F3B3-4C35-BBE6-A5ADBB60326

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. Sun05 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-BFB085E547D

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-DD7542E3D53

G. Core

ID: org_gcr_2017-05-12_mara:59F948C7-2606-4637-8EF6-4922BD231A9

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+/this/1, 1/choose-file/2
  "
  (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-EBD425569FC

- 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 navigation 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-CADEC71D2B6

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-4FA1-8666-39E2FFA2BB4

Essential for using tables with split windows.

```

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

```

Exports to CSV by default.

```

(setq org-table-export-default-format "orgtbl-to-csv"
  "")

```

K. Publishing With T_EX

ID: org_gcr_2018-03-07_mara:5C1154DE-B06C-444D-BB7B-32F0294BF64

L. Engine

ID: org_gcr_2018-03-07_mara:BF9E72A9-58C7-4BD8-B0E3-60185434EAD

Use XeTeX.

```

(setq org-latex-compiler "xelatex")
(setq org-latex-pdf-process '("latexmk -xelatex -
  quiet -shell-escape -f %f"))

```

M. Packages

ID: org_gcr_2018-03-07_mara:AE89B25B-486B-41BD-B3C5-C9499D93A4F

N. Installation

ID: org_gcr_2018-04-28T19-58-14-05-00_mara:B51D7F56-0F8F-4206-8

Copied and customized from here.

```

(setq org-latex-default-packages-alist
  (delq (rassoc '("hyperref" nil)
    org-latex-default-packages-alist)

```

```

        org-latex-default-packages-alist))
(defconst help/org-latex-packages-alist-pre-hyperref
  '(("letterpaper,margin=1.0in" "geometry")
    ("" "float")))
(defconst help/
  org-latex-packages-alist-post-hyperref
  '(("" "fontspec")
    ("section" "placeins")
    ("" "tikz")
    ("" "caption")
    ("" "latexsym")
    ("" "lettrine")
    ("" "listings")
    ("" "color")
    ("" "fancyvrb")
    ("" "fancyhdr")
    ("" "scrlayer-scrpage")
    ("" "lastpage")
    ("" "polyglossia")
    ("" "tabularx")
    ("" "tabu")
    ("" "grantcrettke")))
(setq org-latex-packages-alist
  (append help/
    org-latex-packages-alist-pre-hyperref
    '(("" "hyperref" nil))
    help/
    org-latex-packages-alist-post-hyperref))

```

O. Configuration

ID: org_gcr_2018-04-28T19-58-14-05-00_mara:8CC6E1C3-5300-4D9D-A

```

(setq org-latex-listings-options
  '(("basicstyle" "\\ttfamily")
    ("breakatwhitespace" "false")
    ("breakautoindent" "true")
    ("breaklines" "true")
    ("columns" "[c]fullflexible")
    ("commentstyle" "")
    ("emptylines" "*")
    ("extendedchars" "false")
    ("fancyvrb" "true")
    ("firstnumber" "auto")
    ("flexiblecolumns" "false")
    ("frame" "single"))

```

```
( "frameround" "tttt")
( "identifierstyle" "" )
( "keepspaces" "true" )
( "keywordstyle" "" )
( "mathescape" "false" )
( "numbers" "left" )
( "numbers" "none" )
( "numbersep" "5pt" )
( "numberstyle" "\\tiny" )
( "resetmargins" "false" )
( "showlines" "true" )
( "showspaces" "false" )
( "showstringspaces" "false" )
( "showtabs" "true" )
( "stepnumber" "2" )
( "stringstyle" "" )
( "tab" " ")
( "tabsize" "4" )
( "texcl" "false" )
( "upquote" "false" )))
```

P. Classes

ID: org_gcr_2018-03-07_mara:D397AC3E-DFED-4228-B8CD-7D62318414E
Standard article class.

```
(eval-after-load "ox-latex"
  '(add-to-list 'org-latex-classes
    ('("help-article"
      "\\documentclass[12pt]{article}"))
  )
(setq org-latex-default-class "help-article")
```

Q. Printing Source Code

ID: org_gcr_2018-04-28T19-58-14-05-00_mara:5CF00598-18B1-473C-8

```
(setq org-latex-listings t)
```

R. Unclassified

ID: org_gcr_2018-03-07_mara:593C2BE0-5AD2-4654-B44B-C7F80F33F99
Don't float images.

```
(setq org-latex-default-figure-position "H")
```

S. Unclassified

ID: org_gcr_2017-05-12_mara:FD7E89EA-72DC-4395-AE07-4CAA66B8BD2

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))
```

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 delimiter 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 performing 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)
```

Let pretify minimize macro text.

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

Follow links without using the mouse or more.

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

Org templates

```
(require 'org-tempo)
```

T. Keybindings

ID: org_gcr_2017-05-12_mara:F5256DB3-BD52-409B-80CF-6E601A29AA4

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

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/on-gui
  (define-key org-mode-map (kbd "<return>") #'
    org-return-indent)
  (define-key org-mode-map (kbd "C-M-<return>") #'
    newline))
```

U. Row 5

ID: org_gcr_2017-05-12_mara:B493B118-37D7-4C49-B300-EFBBEF91F0D

```
(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)
```

V. Row 4

ID: org_gcr_2017-05-12_mara:ADB2113A-6D9F-4145-9D78-1DDFE5A2C91

Release these bindings.

```
(global-unset-key (kbd "C-M-t"))
```

```
(define-key org-mode-map (kbd "s-]") (lambda () (
  interactive)
  (message "
    Removing all source block results")
  (help/
    org-2every-src-block
    ,
    org-babel-remove-result)
  (message "
    Done removing all source block results"))))
(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)
(defun help/unset-C-M-t ()
  (interactive)
  (local-unset-key (kbd "C-M-t")))
(add-hook 'org-mode-hook #'help/unset-C-M-t)
```

W. Row 3

ID: org_gcr_2017-05-12_mara:AA97F835-E969-43E4-AC84-B3CF472B872

```
(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 "}" ")))
(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 "}" ")))
```

Because I only use this for Org-Mode.

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

X. Row 2

ID: org_gcr_2017-05-12_mara:680824E4-2C0B-4E0F-BA6E-62AA84D484E

```
(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)
```

Because I only use this for Org-Mode.

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

Y. Hydra

ID: org_gcr_2017-05-12_mara:08A11E45-D41A-44D6-945D-45DF85F8D03

```
(defhydra help/hydra/right-side/org-mode (:color
  blue
  :
  hint nil)
  "
  Org-Mode: (_q_uit)
  Row 4: _1_ SHA-1-hash _2_ display images _3_ hide
    images _4_ id-create _8_ ltx/noindent _9_ o2b/
    more _0_ o2b/LaTeX
  Row 3: _w_ ob-clip-formatted _e_ edit field _E_
    list entities _r_ help/org-refile _R_ insert \"
    refile\" _t_oggle macro marker ATT_R_HTML 4
    table export format _u_tf2txt _p_
    copy-reg-2-property
  Row 2: _a_ archive-subtree _s_ sort _S_ lobigst _d_
    g2s-b/hd _g_ goto _k_ ob-check-src-blk _l_ lint
    _L_ insert inline task
  Row 1: _z_ insert-superscript _Z_ insert-subscript
    _x_ tangled file permissions _c_ org-fill-para
    _C_ount words _b_ swtch2sessn _n_ n2sbtre _m_
    mark-subtree"
  ("q" org-babel-switch-to-session-with-code)
  ;; Row 4
  ("1" org-babel-sha1-hash)
  ("2" org-display-inline-images)
  ("3" org-remove-inline-images)
  ("4" org-id-get-create)
  ("8" (lambda () (interactive) (insert "@@latex:\\
    noindent{ }@@"))))
```

```

("9" (lambda () (interactive) (insert "#+HTML:
<!--more-->"))))
("0" (lambda () (interactive) (insert "$\\LaTeX$")
)) ;; Row 4
;; Row 3
("w" ox-clip-formatted-copy)
("e" org-table-edit-field)
("W" help/org-mark-subtree/kill-ring-save)
("E" org-entities-help)
("r" help/org-refile)
("R" (lambda () (interactive) (insert "(refile)"))
)
("t" help/org-toggle-macro-markers)
("R" (lambda () (interactive) (insert "#+ATTR_HTML
: :border 2 :rules all :frame border")))
("u" org-utf-to-xetex-insert-or-wrap-with-macro)
("p" my-org-region-to-property)
;; TODO
;; Row 2
("a" org-archive-subtree-default)
("s" org-sort)
("S" org-babel-lob-ingest)
("d" org-babel-goto-src-block-head)
("g" org-goto)
("i" org-babel-do-key-sequence-in-edit-buffer)
("I" org-babel-insert-header-arg)
("k" org-babel-check-src-block)
("l" org-lint)
("L" org-inlinetask-insert-task)
;; Row 1
("z" help/org-insert-superscript)
("Z" help/org-insert-subscript)
("x" help/org-insert-tangle-mode-permission)
("c" org-fill-paragraph)
("C" help/org-count-words)
("b" org-babel-switch-to-session)
("n" org-narrow-to-subtree)
("m" org-mark-subtree)
("M" org-mark-element))
(defun help/org-mark-subtree/kill-ring-save ()
""
(interactive)
(call-interactively 'org-mark-subtree)
(call-interactively 'kill-ring-save))
(key-chord-define org-mode-map "hh" #'help/hydra/

```

```

    right-side/org-mode/body)
  (defun help/org-insert-tangle-mode-permission ()
    (interactive)
    (insert ":tangle-mode (identity #o755)"))

```

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{}"))))

```

Z. Prettification

ID: org_gcr_2018-04-20T15-46-50-05-00_mara:356CA076-3E60-4908-B

Macro markers.

```

(setq org-hide-macro-markers nil)

```

Prettifications.

```

(defun help/prettify-org-mode ()
  (interactive)
  (setq prettify-symbols-alist (help/alist-set "#+
    NAME\:" r? prettify-symbols-alist))
  (setq prettify-symbols-alist (help/alist-set "#+
    name\:" r? prettify-symbols-alist))
  (setq prettify-symbols-alist (help/alist-set "#+
    BEGIN_SRC" l? prettify-symbols-alist))
  (setq prettify-symbols-alist (help/alist-set "#+
    begin_src" l? prettify-symbols-alist))
  (setq prettify-symbols-alist (help/alist-set "#+
    END_SRC" l? prettify-symbols-alist))
  (setq prettify-symbols-alist (help/alist-set "#+
    end_src" l?

    prettify-symbols-alist))
  (setq prettify-symbols-alist (help/alist-set "#+
    BEGIN_EXAMPLE" l?

```

```

prettify-symbols-alist))
(setq prettify-symbols-alist (help/alist-set "#+
begin_example" []?

prettify-symbols-alist))
(setq prettify-symbols-alist (help/alist-set "#+
END_EXAMPLE" []?

prettify-symbols-alist))
(setq prettify-symbols-alist (help/alist-set "#+
end_example" []?

prettify-symbols-alist))
(setq prettify-symbols-alist (help/alist-set "#+
BEGIN_QUOTE" []? prettify-symbols-alist))
(setq prettify-symbols-alist (help/alist-set "#+
begin_quote" []? prettify-symbols-alist))
(setq prettify-symbols-alist (help/alist-set "#+
END_QUOTE" []? prettify-symbols-alist))
(setq prettify-symbols-alist (help/alist-set "#+
end_quote" []? prettify-symbols-alist))
(help/prettify-org-mode-names))
(add-hook 'org-mode-hook #'help/prettify-org-mode)
(defun help/prettify-org-mode-names (&optional
  replacement)
  (interactive)
  (or replacement (setq replacement _?)))
  (save-excursion
    (goto-char (point-min))
    (let ((case-fold-search t))
      (while (re-search-forward "#\\++name: " nil t)
        (copy-region-as-kill (point) (
line-end-position))
        (let ((name (substring-no-properties(pop
kill-ring))))
          (when (string-prefix-p "org_" name)
            (setq prettify-symbols-alist (help/
alist-set name replacement prettify-symbols-alist
))))))))))

```

Custom font lock

- Hyphen appears as em-dash

```
(font-lock-add-keywords
```



```
'org-mode
'(("^[:space:]]*\\(-\\) "
  0 (progl () (compose-region (match-beginning 1)
    (match-end 1) "-")))))
```

. HTML Exporter

ID: org_gcr_2018-07-02T07-07-57-05-00_mara:7236EDFF-7176-4C5F-8

Doctype—use the default.

org-html-table-default-attributes (which are ignored in the HTML5).

```
(setq org-html-table-default-attributes
  (plist-put org-html-table-default-attributes :
    rules "all"))
(setq org-html-table-default-attributes
  (plist-put org-html-table-default-attributes :
    frame "border"))
```

iii. Intermittent Literate Programming

ID: org_gcr_2018-01-04_mara:43462ECD-9C40-43D2-872F-A2E9DB134199

See this post.

Edit region in separate buffer. "Final" version works without carrying over buffer properties.

```
(use-package edit-indirect
  :ensure t)
```

iv. 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))
```

v. Words

ID: org_gcr_2017-08-04_mara:0A02BFBE-91BE-4629-B4B3-F72F2AAEAD3E

```
(use-package lorem-ipsu
:ensure t
:config
(defun help/lorem-ipsu-insert-word ()
  (interactive)
  (let* ((paragraph
          (nth (random (length lorem-ipsu-text))
               lorem-ipsu-text))
         (sentence
          (concat (nth (random (length paragraph))
                       paragraph)
                  lorem-ipsu-sentence-separator))
         (words (split-string sentence))
         (word (nth (random (length words)) words)))
    (insert word))))
(use-package academic-phrases
:ensure t)
(defhydra help/hydra/words (:color blue :hint nil)
  "
Words: _C_ount, Count In _O_rg , _T_op#, _G_rade Level,
      _R_eading Ease, _A_ffect vs Effect
Lorem Ipsum: _w_ord _s_entence _p_aragraphs _l_list
Academic Phrases: _a_ll or by se_c_tion
      _E_macs Name (insert _e_macs name)
      _F_lame (insert _f_lame)
      _H_oroscope (insert _h_oroscope)
      _K_ibologize (insert _k_ibologize)
      _S_hop _m_iddle name
      _Y_ouwill (insert _y_ouwill)
      Corporate _B_.S.: (_I_nsert B.S.) (A_d_d B.S.
To Kill Ring)
      _q_uit"
  ("q" nil)
  ("C" count-words)
  ("O" help/org-count-words)
  ("T" help/most-used-words)
  ("G" writegood-grade-level)
  ("R" writegood-reading-ease))
```

```

("A" affect-vs-effect-explanation)
("w" help/lorem-ipsum-insert-word)
("s" lorem-ipsum-insert-sentences :exit nil)
("p" lorem-ipsum-insert-paragraphs :exit nil)
("l" lorem-ipsum-insert-lists :exit nil)
("a" academic-phrases)
("c" academic-phrases-by-section)
("E" emacs-name)
("e" (lambda () (interactive) (insert (emacs-name))))
("F" flame)
("f" insert-flame)
("H" horoscope)
("h" (lambda () (interactive) (horoscope t)))
("K" kibologize)
("k" help/kibologize-insert)
("S" shop)
("m" shop-middle-name)
("Y" youwill)
("y" help/youwill-insert)
("B" corporate-bs-generator-create)
("I" (lambda () (interactive) (let ((
  current-prefix-arg '(4))) (
  corporate-bs-generator-create))))
("d" (lambda () (interactive) (let ((
  current-prefix-arg '(16))) (
  corporate-bs-generator-create))))
(use-package affect-vs-effect
  :ensure t)

```

vi. Dash

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

```

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

```

(c) Unicode

ID: org_gcr_2018-06-06T22-01-54-05-00_mara:DD18C2DA-10EE-43BD-AED0-5DD0

i. System

ID: org_gcr_2018-06-16T16-57-47-05-00_mara:189DB36F-0E6F-451F-B879

Prefer UTF-8 by default.

```
(let ((lang 'utf-8))
  (message "In the environment, 'LANG' and 'LC_ALL' are
    set to '%s' and '%s'."
    (getenv "LANG") (getenv "LC_ALL"))
  (message "However, the init file is overriding them
    with the profile for '%s', manually configuring the
    language and coding system."
    (buffer-file-name) lang)
  (set-language-environment lang)
  (prefer-coding-system lang))
```

Make copying and pasting easier:

On the GUI.

```
(help/on-gui
  (setq x-select-request-type '(UTF8_STRING
    COMPOUND_TEXT TEXT STRING)))
```

On Windows.

```
(help/on-gui
  (help/on-windows
    (set-clipboard-coding-system 'utf-16le-dos)))
```

ii. Font

ID: org_gcr_2018-06-06T22-01-54-05-00_mara:99E80492-40C4-4879-B2D6

Configure fonts for Unicode blocks.

```
(use-package unicode-fonts
  :ensure t
  :config
  (unicode-fonts-setup))
```

- Test with
 - view-hello-file
 - list-charset-chars RET unicode-bmp RET
 - list-charset-chars RET unicode-smp RET
 - unicode-fonts-debug-insert-block RET MathematicalOperators RET

iii. Utility

ID: org_gcr_2018-06-06T22-01-54-05-00_mara:702B80D4-22BB-4A0B-B33C

Show all code points in a block

A. Char

ID: org_gcr_2018-06-06T22-01-54-05-00_mara:D507FC09-B53D-4206-

Escape and unescape characters.

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

Homoglyph warning system.

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

Roland's utilities.

```
(use-package ucs-utils
  :ensure t
  :config
  (ucs-utils-install-aliases))
```

Convert to ASCII.

```
(use-package unidecode
  :ensure t
  :config
  (defun help/unidecode-region (beginning end)
    "Unidecode region.
For example try it on: éàDj vu"
    (interactive "r")
    (if (use-region-p)
        (unidecode-region beginning end)
        (message "Sorry, there was no active region
to unidecode.")))
  (defun help/unidecode-sanitize-region (beginning
end)
  "Unidecode sanitize region.
For example try it on: []"
  (interactive "r")
  (if (use-region-p)
```

```
(unicode-sanitize-region beginning end)
(message "Sorry, there was no active region
to unicode sanitize.")))
```

B. File

ID: org_gcr_2017-05-12_mara:ADAE0A1B-3FD8-4605-8611-388E5ADBAF

Warn of UTF BOM bytes via skeeto.

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

```
(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_warning_if_i_open_a_file_with_utf8/
d5kszh'"
  (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)
```

C. Point

ID: org_gcr_2018-06-06T22-01-54-05-00_mara:5686D072-60ED-4F68-

Search code points by name and present the result(s).

- code point
- character
- name

```
(use-package list-unicode-display
:ensure t)
```

D. Org-Mode

ID: org_gcr_2018-06-18T11-01-53-05-00_mara:51B3EC8F-B3FB-46E4-

```
(add-to-list 'load-path "~/src/org-utf-to-xetex")
(require 'org-utf-to-xetex)
(add-hook 'org-mode-hook #'
  org-utf-to-xetex-prettify)
```

iv. Hydra

ID: org_gcr_2018-06-06T22-01-54-05-00_mara:F0E4B9C4-4873-4519-9FE5

```
(defhydra help/hydra/unicode (:color blue :hint nil)
  "
Unicode: (_q_uit)
Insert: ^By _N_ame, ^By _C_odepoint (name string | hex
      string | code point w/radix octal #o21430, hex #
      x2318, or decimal #10r8984)
Search: []RegE_x_
'Ify Region: _A_sci'ify, _R_oman'ify
_E_escape Region: [] → \\\u00D83D\\\u00DE04
_U_unescape Region: \\\u00D83D\\\u00DE04 → []
Blocks: _B_lock Name At Point, _I_nsert Block
Char: _U_CS Name At Point, _H_omoglyph Warning
System
"
("q" nil)
("N" ucs-utils-ucs-insert)
("C" insert-char)
("x" list-unicode-display)
("A" help/unidecode-region)
("R" help/unidecode-sanitize-region)
("E" unicode-escape)
("U" unicode-unescape)
("B"
  org-utf-to-xetex-get-unicode-block-for-string-char-after
)
("I" (lambda () (interactive) (
  unicode-fonts-debug-insert-block)))
("U" ucs-utils-eval)
("H" unicode-troll-stopper-mode))
(global-set-key (kbd "C-M-u") #'help/hydra/unicode/body
)
```

(d) 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

- Prefix key options
 - C-M: No structural navigation
 - C-M-s: No that is for the operating system
 - M-s: Existing bindings that I don't want to mess with
 - M-s: Pretty much never use this, so use it

```
(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 "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))))
(defun help/gnu-apl-prefix-string ()
  (interactive)
  gnu-apl-mode-map-prefix)
(defhydra help/hydra/gnu-apl (:color blue
                             :hint nil)
  "
GNU APL: [% (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
_c_ start APL _v_ stop APL _b_ switch to APL _n_
switch back
Prefix key is: % (help/gnu-apl-prefix-string)
"
  ("i" help/gnu-apl-eval-line)
  ("o" gnu-apl-interactive-send-current-function)
  ("j" (lambda () (interactive) (call-interactively '
next-logical-line)) :exit nil)
  ("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)
(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. Scheme

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

File extensions:

- Traditional.

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

- R6RS.

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

- Racket.

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

Scheme.

```
(use-package scheme
:config
(setq scheme-program-name "chez")
(defun help/scheme-mode-hook-fn ()
  "HELP Scheme Mode Hook Function")
(add-hook 'scheme-mode-hook #'help/scheme-mode-hook-fn)
(defun help/inferior-scheme-mode-hook-fn ()
  "HELP Inferior Scheme Mode Hook Function")
(add-hook 'inferior-scheme-mode-hook #'help/
inferior-scheme-mode-hook-fn))
```

Geiser.

```
(use-package geiser
:ensure t
:config
(setq geiser-default-implementation 'chez)
(setq geiser-active-implementations (list
geiser-default-implementation))
(setq geiser-repl-history-no-dups-p t)
(defun help/geiser-exit-chez ()
  (interactive)
  (call-interactively 'geiser-mode-switch-to-repl)
  (insert "(exit)"))
  (geiser-repl--maybe-send))
(defun help/geiser-on-p ()
  (if (geiser-repl--live-p) 'ON
    'OFF))
(defun help/geiser-eval-line ()
  "Evaluate this line and move to next."
  (interactive)
  (end-of-line)
  (set-mark (line-beginning-position))
  (call-interactively 'geiser-eval-region)
  (deactivate-mark)
  (call-interactively 'next-logical-line))
(defhydra help/hydra-geiser-mode (:color blue
:hint nil)
"

```

```
Geiser REPL: ␣%(help/geiser-on-p)
_y_ eval-buffer _u_ eval-region _U_ (and go)
_z_ run (another) geiser _x_ stop latest chez _c_
exit geiser _b_ to latest geiser _n_ switch back
_q_ quit
"
```

```

("y" geiser-eval-buffer)
("u" geiser-eval-region)
("U" geiser-eval-region-and-go)
("z" run-geiser)
("x" help/geiser-exit-chez)
("b" geiser-mode-switch-to-repl)
("c" geiser-repl-exit)
("n" (lambda () (interactive) (other-window -1)))
("q" nil))
(defun help/geiser-mode-hook-fn ()
  (local-unset-key (kbd "C-."))
  (key-chord-define geiser-mode-map "hh" #'help/
hydra-geiser-mode/body)
  (local-unset-key (kbd "C-j"))
  (local-set-key (kbd "<C-return>") #'help/
geiser-eval-line))
(add-hook 'geiser-mode-hook #'help/
geiser-mode-hook-fn)
(defun help/geiser-repl-mode-hook-fn ()
  (local-unset-key (kbd "C-."))
  (local-unset-key (kbd "C-j"))
  (local-set-key (kbd "<C-return>") #'
geiser-repl--maybe-send)
  (key-chord-define-local "hh" #'help/
hydra-geiser-mode/body))
(add-hook 'geiser-repl-mode-hook #'help/
geiser-repl-mode-hook-fn))

```

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)))

```

iii. 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))

```

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

iv. Go

ID: org_gcr_2017-07-30_mara:2F891F22-5C0C-43BA-8891-62DA7E9540FD

```
(use-package go-mode
  :ensure t
  :config
  (use-package go-eldoc
    :ensure t)
  (use-package go-autocomplete
    :ensure t
    :config
    (ac-config-default))
  (use-package golint
    :ensure t)
  (use-package go-errcheck
    :ensure t)
  (use-package ob-go
    :ensure t
    :config
    (add-to-list 'org-babel-load-languages '(go . t)))
  (use-package go-snippets
    :ensure t)
  (use-package flycheck-gometalinter
    :ensure t
    :config
    (add-hook 'flycheck-mode-hook #'
      flycheck-gometalinter-setup)
    (setq flycheck-gometalinter-concurrency 2)
    (setq flycheck-gometalinter-deadline "7s")
    (setq flycheck-gometalinter-disable-all nil)
    (setq flycheck-gometalinter-disable-linters nil)
    (setq flycheck-gometalinter-enable-linters nil)
    (setq flycheck-gometalinter-errors-only nil)
    (setq flycheck-gometalinter-fast nil)
    (setq flycheck-gometalinter-tests t)
    (setq flycheck-gometalinter-vendor t))
  (use-package go-guru
    :ensure t)
  (use-package go-hydra)
  (use-package gotest
```

```

:ensure t)
(use-package go-rename
:ensure t)
(use-package go-direx
:ensure t)
(add-to-list 'load-path "~/src/go-snippets")
(require 'go-snippets)
(defun help/go-mode/safb/go-run ()
  ""
  (interactive)
  (let* ((compile-command (file-name-nondirectory
                                (directory-file-name
                                default-directory))))
    (help/safb/compile)))
(defun help/go-mode/safb/go-rename ()
  ""
  (interactive)
  (help/save-all-file-buffers)
  (call-interactively 'go-rename))
(add-to-list 'load-path "~/git/github-anonymous/
gorepl-mode")
(require 'gorepl-mode)
(defhydra help/hydra/right/go-mode (:color blue :hint
nil)
  "
  Go: (_q_uit)          | ^          ^ | ^
        ^ |
  -^-----^+-----^+-----^+-----
  _e_: explorer          | _r_: run          | _t_:
    tools          | g_o_re
  _g_: guru              | _j_: previous error | _k_:
    next error |
  _c_: build              | _n_: rename          |
        ^^ |
  "
  ("e" go-direx-switch-to-buffer)
  ("r" help/go-mode/safb/go-run)
  ("t" go-hydra-go-mode/body)
  ("o" gorepl-hydra/body)
  ("g" go-hydra-guru-go-mode/body)
  ("k" godef-jump-other-window)
  ("l" pop-tag-mark)
  ("c" help/safb/compile)
  ("n" help/go-mode/safb/go-rename)

```

```

("j" previous-error :exit nil)
("k" next-error :exit nil)
("q" nil))
(key-chord-define go-mode-map "hh" #'help/hydra/right
/go-mode/body)
(defun help/go-mode-hook-fn ()
  (go-eldoc-setup)
  (setq gofmt-command "gofmt")
  (add-hook 'before-save-hook #'gofmt-before-save)
  (setq compilation-read-command nil)
  (setq compile-command "go install && go test -v &&
gometalinter ./...")
  (local-set-key (kbd "s-i") #'godoc-at-point)
  (local-set-key (kbd "s-I") #'godoc)
  (local-set-key (kbd "s-j") #'help/safb/compile)
  (local-set-key (kbd "s-k") #'help/go-mode/safb/
go-run)
  (local-set-key (kbd "=") #'(lambda () (interactive)
(insert " = ")))
  (local-set-key (kbd "s-=") #'(lambda () (
interactive) (insert " := ")))
  (local-set-key (kbd "s-<") #'(lambda () (
interactive) (insert " <- ")))
  (local-set-key (kbd "s-o") #'gorepl-hydra/body)
  (go-guru-hl-identifier-mode)
  (gorepl-mode))
(add-hook 'go-mode-hook #'help/go-mode-hook-fn))

```

v. 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)

```

vi. 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-package yasnippet
  :ensure t
  :config
  (use-package yasnippet-snippets
    :ensure t)
  (yas-global-mode t)
  (help/on-gui (define-key yas-minor-mode-map (kbd "<tab>") nil))
  (define-key yas-minor-mode-map (kbd "C-'" ) #'
    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)
  (defun help/yas-after-exit-snippet-hook-fn ()
    (help/prettify-org-mode-names)
    (prettify-symbols-mode)
    (prettify-symbols-mode))
  (add-hook 'yas-after-exit-snippet-hook #'help/
    yas-after-exit-snippet-hook-fn)
  :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))
```

vii. Structured Query Language (SQL)

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

viii. 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)
(setq ispell-skip-html t)
(add-hook 'web-mode-hook #'help/web-mode-hook-fn)

```

ix. JavaScript

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

js2-mode is derived from js-mode and most of the indentation settings are done there.

Add JavaScript language to Org-Mode.

```

(use-package js2-mode
  :ensure t
  :mode (("\\.js$" . js2-mode)
        (\\.jsx$" . js2-mode))
  :interpreter ("node" . js2-mode)
  :config
  (add-hook 'js2-mode-hook #'js2-imenu-extras-mode)
  (add-to-list 'org-src-lang-modes ("javascript" . js2
  )))
(use-package js2-refactor
  :after (js2-mode)
  :ensure t

```

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

x. AppleScript

ID: org_gcr_2018-04-22T00-19-01-05-00_mara:BB59BC9A-5CBE-4A0A-9276

```
(use-package apples-mode
  :ensure t
  :config
  (setq apples-compile-create-file-flag t)
  (setq apples-continuation-offset 2)
  (setq apples-decompile-query ?o)
  (setq apples-follow-error-position t)
  (setq apples-indent-offset 2)
  (setq apples-prefer-coding-system
    coding-category-utf-8)
  (setq apples-tmp-dir "~/tmp")
  (defhydra help/hydra/right/apples-mode (:color blue :
    hint nil)
    "
    AppleScript: (_q_uit)      | ^                ^|^
                    ^|
  -^-----^+-----^+-----^+-----
    _j_: eval region           | _o_: continuation      |
    _i_: show last result     |
    _J_: eval buffer          | _O_: continue+newline  |
    _9_: open dict
    _i_: compile              | _u_: close block       |
    _0_: send-to-editor
    _I_: decompile            | _U_: toggle-indentation|
    "
    ("j" apples-run-region)
    ("J" apples-run-buffer)
    ("i" apples-compile)
    ("I" apples-decompile)
    ("o" apples-insert-continuation-char)
    ("O" apples-insert-continuation-char-and-newline)
    ("u" apples-end-completion)
    ("U" apples-toggle-indent)
    ("i" apples-show-last-result))
```

```

("9" apples-open-dict-index)
("0" apples-send-to-applescript-editor)
("q" nil))
(eval-after-load "apples-mode"
  `(key-chord-define apples-mode-map "hh" #'help/
    hydra/right/apples-mode/body))
(use-package ob-applescript
  :ensure t
  :config
  (add-to-list 'org-babel-load-languages '(apples . t
    )))
(defun help/apples-mode-hook-fun ())
(add-hook 'apples-mode-hook #'help/
  apples-mode-hook-fun)
(autoload 'apples-mode "apples-mode" "Happy
  AppleScripting!" t)
(autoload 'apples-open-scratch "apples-mode" "Open
  scratch buffer for AppleScript." t)
(add-to-list 'auto-mode-alist '("\\\\.\\(applescri\\|sc
  \\)pt\\'" . apples-mode)))

```

xi. Emacs Speaks Statistics (ESS)

ID: org_gcr_2018-11-21T18-07-10-06-00_cosmicality:23926BE2-3E84-4B

This space used to contain a pretty awesome ESS configuration for The R Project. I removed it after switching go RStudio but you can find all of the necessary changes and support files here:

- ESS+R Config
- .Renviron
- .Rinstall.r
- .Rprofile
- .Rsetup.sh

(e) Data

ID: org_gcr_2018-04-20T15-46-50-05-00_mara:3259C745-2E58-4F02-B384-A2B

i. JSON

ID: org_gcr_2018-01-16_mara:59857D34-FF7E-49DC-B639-DCE18813974E

javascript-mode is an alias for json-mode.

json-mode is derived from javascript-mode.

```
(use-package json-mode
  :ensure t
  :diminish json-mode
  :mode (("\\.jshintrc$" . json-mode))
  :config
  (defhydra help/hydra/json (:color blue
                             :hint nil)
    "
JSON: (q to quit)
_b_eautify
_s_how-path _k_ill-path
_t_oggle boolean _n_ullify sexp
_i_ncrement number _d_ecrement number
"
    ("b" json-mode-beautify)
    ("s" json-mode-show-path)
    ("k" json-mode-kill-path)
    ("t" json-mode-toggle-boolean)
    ("n" json-mode-nullify-sexp)
    ("i" json-mode-increment-number-at-point :exit nil)
    ("d" json-mode-decrement-number-at-point :exit nil)
    ("q" nil))
  (define-key json-mode-map (kbd "RET") #'newline)
  (defun help/json-mode-hook-fn ()
    (indent-guide-mode)
    (turn-off-auto-capitalize-mode)
    (turn-off-auto-fill)
    (key-chord-define-local "hh" #'help/hydra/json/body)
  )
  (add-hook 'json-mode-hook #'help/json-mode-hook-fn))
```

ii. YAML

ID: org_gcr_2018-01-16_mara:D8AD667E-3B40-44A1-A4D3-CC3383DAA4A0

```
(use-package yaml-mode
  :ensure t
  :config
  (defun yaml-next-field ()
    "Jump to next yaml field."
    URL: `https://stackoverflow.com/questions/12648388/
    emacs-yaml-editing')
```

```

    (interactive)
    (search-forward-regexp ": *"))
  (defun yaml-prev-field ()
    "Jump to next yaml field.
URL: `https://stackoverflow.com/questions/12648388/
emacs-yaml-editing'"
    (interactive)
    (search-backward-regexp ": *"))
  (defhydra help/hydra/yaml (:color blue
                             :hint nil)
    "
YAML: (q to quit)
  _i_ indent line
  _n_ narrow to block literal
  _c_ fill paragraph
  "
    ("i" yaml-indent-line)
    ("n" yaml-narrow-to-block-literal)
    ("c" yaml-fill-paragraph)
    ("q" nil))
  (defun help/yaml-mode-hook-fn ()
    (indent-guide-mode)
    (turn-off-auto-capitalize-mode)
    (turn-off-auto-fill)
    (setq flycheck-checker 'yaml-yamllint))
  (add-hook 'yaml-mode-hook #'help/yaml-mode-hook-fn)
  (add-to-list 'auto-mode-alist '("\\.yaml\\'" .
    yaml-mode))
  (define-key yaml-mode-map (kbd "<backspace>") nil)
  (define-key yaml-mode-map (kbd "<backspace>") #'
    sp-backward-delete-char)
  (define-key yaml-mode-map (kbd "s-j") #'
    yaml-next-field)
  (define-key yaml-mode-map (kbd "s-k") #'
    yaml-prev-field)
  (key-chord-define yaml-mode-map "hh" #'help/hydra/
    yaml/body)
  (use-package flycheck-yamllint
    :ensure t
    :config
    (with-eval-after-load "flycheck"
      (add-hook 'flycheck-mode-hook '
        flycheck-yamllint-setup))
    (setq flycheck-yaml-yamllint-executable "/Users/gcr
    /yamllint/env/bin/yamllint"))))

```

iii. Comma-Separated Values (CSV)

ID: org_gcr_2018-05-11T16-33-31-05-00_mara:98E96CAF-25E3-420B-914C

FYI: Strip-buffer-mode doesn't help here.

```
(use-package csv-mode
  :ensure t
  :config
  (use-package pcsv
    :ensure t)
  (use-package nlinum
    :ensure t
    :config
    (setq nlinum-format "%04d"))
  (defun help/csv-mode/csv-descending ()
    (interactive)
    csv-descending)
  (defun help/csv-mode/csv-invisibility ()
    (interactive)
    buffer-invisibility-spec)
  (defhydra help/hydra/csv-mode (:color blue :hint nil)
    "
    CSV: (_q_uit)
    Row: Sort: _l_exographically _n_umerically
           Descending is %(help/csv-mode/csv-descending):
    _t_oggle it
           Re_v_erse
           Invisibility is %(help/csv-mode/csv-invisibility)
    : toggle _i_t
    Row or Columns: t_r_anspose
    Fields or Columns: _k_ill _y_ank
    Fields: _a_lign _u_nalign
    Table: yank a_s_ new
    "
    ("q" nil)
    ("l" csv-sort-fields)
    ("n" csv-sort-numeric-fields)
    ("t" csv-toggle-descending :exit nil)
    ("v" csv-reverse-region)
    ("i" csv-toggle-invisibility)
    ("r" csv-transpose)
    ("k" csv-kill-fields)
```

```

("y" csv-kill-fields)
("a" csv-align-fields :exit nil)
("u" csv-unalign-fields :exit nil)
("s" csv-yank-as-new-table))
(key-chord-define csv-mode-map "hh" #'help/hydra/
  csv-mode/body)
(defun help/csv-mode-hook-fn ()
  (nlinum-mode)
  (visual-line-mode))
;; todo: can I toggle this: csv-descending
(add-hook 'csv-mode-hook #'help/csv-mode-hook-fn))

```

iv. Sparkline

ID: org_gcr_2018-11-07T14-59-31-06-00_cosmicality:1B5AE823-4CE9-45

```

(defun help/sparkline-theory-and-practice ()
  "Sparkline theory and practice"
  (interactive)
  (switch-to-buffer "*Sparkline*")
  (let ((inhibit-read-only t))
    (erase-buffer))
  (remove-overlays)
  (widget-insert "Sparkline theory and practice Edward
  Tufte\n")
  (widget-insert "
  -----\n")
  (widget-insert "\n")
  (widget-insert "\"A sparkline is a small intense,
  simple, word-sized graphic with typographic\n")
  (widget-insert " resolution. Sparklines mean that
  graphics are no longer cartoonish special\n")
  (widget-insert " occasions with captions and boxes,
  but rather sparkline graphics can be\n")
  (widget-insert " everywhere a word or number can be:
  embedded in a sentence, table, headline,\n")
  (widget-insert " map, spreadsheet, graphic. Data
  graphics should have the resolution of\n")
  (widget-insert " typography.\n\n")
  (widget-insert "\n")
  (widget-insert "See Edward Tufte, ")
  (widget-create
    'url-link
    :tag "Beautiful Evidence"

```



```

"https://www.edwardtufte.com/bboard/
q-and-a-fetch-msg?msg_id=000262&topic_id=1")
(widget-insert ", 46-63.\n")
(widget-insert "\n")
(widget-create
 'url-link
 :tag "Source"
 "https://www.edwardtufte.com/bboard/
q-and-a-fetch-msg?msg_id=00010R")
(let ((map (make-sparse-keymap)))
  (set-keymap-parent map widget-keymap)
  (define-key map "q" 'kill-this-buffer)
  (use-local-map map))
(widget-setup)
(beginning-of-line))

```

```

(use-package spark
 :ensure t)

```

(f) Publishing

ID: org_gcr_2017-05-12_mara:0A4A1510-41F7-4469-87D1-156D44A06216

i. AUCTeX

ID: org_gcr_2017-05-12_mara:2DF7BDAD-45C9-4D8D-AC34-FE77E119A093

A. Introduction

ID: org_gcr_2018-03-05_mara:EF3E6F3E-B457-477F-95B3-B08E83B797

Load AUCTeX.

```

(use-package tex
 :ensure auctex)

```

Load LatexMk.

```

(use-package auctex-latexmk
 :ensure t
 :config
 (setq auctex-latexmk-inherit-TeX-PDF-mode t)
 (auctex-latexmk-setup))

```

B. Editing the Document Source

ID: org_gcr_2018-03-06_mara:15BDC594-E44E-41A0-B5E3-A5490E2DD3

Insert and toggle inline math.

```
(setq TeX-electric-math t)
```

Balance opening and closing brackets because I don't think that SmartParens will always know *the right thing* to do here.

```
(setq LaTeX-electric-left-right-brace t)
```

Section hooks.

```
(setq LaTeX-section-hook
  '(LaTeX-section-heading
    LaTeX-section-title
    LaTeX-section-toc
    LaTeX-section-section
    LaTeX-section-label))
```

Always prompt for item labels.

```
(setq TeX-arg-item-label-p t)
```

Show Unicode symbols in the menu.

```
(setq LaTeX-math-menu-unicode t)
```

Automatically bracket sub and superscripts.

```
(setq TeX-electric-sub-and-superscript t)
```

On macro insertion prompt for both required and optional arguments every time.

```
(setq TeX-insert-macro-default-style '
  show-all-optional-args)
```

Make \ insert a macro.

```
(setq TeX-electric-escape t)
```

Always bracket macros.

```
(setq TeX-insert-braces t)
```

Complete expert commands.

```
(setq TeX-complete-expert-commands t)
```

Always check environment indentation.

```
(setq LaTeX-indent-environment-check t)
```

Indent comment syntactically.

```
(setq LaTeX-syntactic-comments 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))
```

C. Controlling Screen Display

ID: org_gcr_2018-03-06_mara:AD0AF49A-F5EE-4931-867A-63E0C7F87F

Fontify super and subscripts.

```
(setq font-latex-fontify-script t)
```

Allow n super and subscript depth.

```
(setq font-latex-fontify-script-max-level 3)
```

D. Starting Processors, Viewers and Other Programs

ID: org_gcr_2018-03-06_mara:8D6395B6-CFE8-4A87-96E2-C4D1A24117

Tell ispell to ignore things.

```
(setq TeX-ispell-extend-skip-list t)
```

Create DVI from PDF.

```
(setq TeX-PDF-mode t)
```

Use XeTeX.

```
(setq-default TeX-engine 'xetex)
(setq-default TeX-PDF-mode t)
```

Show compilation is another window.

```
(setq TeX-show-compilation t)
```

Error message format.

```
(setq TeX-file-line-error t)
```

Prompt before cleaning.

```
(setq TeX-clean-confirm t)
```

E. Customization and Extension

ID: org_gcr_2018-03-07_mara:B440BBB2-DC5D-421D-85BD-988BF56B7E

Never use multi-file document structure.

```
(setq-default TeX-master t)
```

Automatically parse on load and save.

```
(setq TeX-parse-self t)
(setq TeX-auto-save t)
```

Recursively search for all customizations.

```
(setq TeX-file-recurse t)
```

F. Key Bindings

ID: org_gcr_2018-03-05_mara:D945AAE3-61B6-4EE8-8693-D67685344A

```
(define-key TeX-mode-map (kbd "C-c C-c") #'help/
  safb-TeX-command-master)
(define-key TeX-mode-map (kbd "C-j") nil)
(define-key TeX-mode-map (kbd "C-M-f") nil)
(define-key TeX-mode-map (kbd "C-M-f") #'
  LaTeX-find-matching-end)
(define-key TeX-mode-map (kbd "C-M-b") nil)
(define-key TeX-mode-map (kbd "C-M-b") #'
  LaTeX-find-matching-begin)
(define-key TeX-mode-map (kbd "s-l") nil)
(define-key TeX-mode-map (kbd "s-l") #'
  TeX-complete-symbol)
```

G. Hydras

ID: org_gcr_2018-03-05_mara:56F54131-AD99-411E-89D0-1AD104B3E2

```

(defhydra help/hydra/auctex (:color blue
                             :hint nil)
  "
  AUCTeX: _H_elp _q_uit
  Do: Run_a_ll →R_e_gion →B_u_ffer →Section(_1_)
      _K_ill-Job
      Center-Buffer(_2_) Go-Home(_3_) Clean(_4_)
      Save-Belonging(_6_)
  Insert: _S_ection _E_nvironment _M_acro _F_ont;()
      _C_close-Env _I_tem _N_o Indent
  Mark: E_n_vironment (_j_ust one) Se_c_tion (jus_t_
      one)
  Fold: Fol_d_* Bu_f_fe_r_ Re_g_i_o_n _P_aragraph
      Macr_o_ En_v_ Mat_h_ Co_m_ment
  Narrow: _G_roup Env_i_ironment
  Errors: _R_eport □_B_oxes □_W_arnings
  "

  ;; Do
  ("a" TeX-command-run-all)
  ("e" TeX-command-region)
  ("u" TeX-command-buffer)
  ("1" LaTeX-command-section)
  ("K" TeX-kill-job)
  ("2" TeX-recenter-output-buffer)
  ("3" TeX-home-buffer)
  ("4" TeX-clean)
  ("H" TeX-documentation-texdoc)
  ("6" TeX-save-document)
  ;; Insert
  ("S" LaTeX-section)
  ("E" LaTeX-environment)
  ("M" TeX-insert-macro)
  ("F" TeX-font)
  ("C" LaTeX-close-environment)
  ("I" LaTeX-insert-item)
  ("N" (lambda () (interactive) (insert "\\noindent
  ")))
  ;; Mark
  ("n" LaTeX-mark-environment)
  ("j" (lambda (interactive)

```

```

      (let ((current-prefix-arg '(4))) (
call-interactively 'LaTeX-mark-environment)) ))
("c" LaTeX-mark-section)
("t" (lambda (interactive)
      (let ((current-prefix-arg '(4))) (
call-interactively
      ,
      LaTeX-mark-section)) ))
;; Folding
("d" TeX-fold-dwim)
("f" TeX-fold-buffer)
("r" TeX-fold-clearout-buffer)
("g" TeX-fold-region)
("i" TeX-fold-clearout-region)
("P" TeX-fold-paragraph)
("o" TeX-fold-macro)
("v" TeX-fold-env)
("h" TeX-fold-math)
("m" TeX-fold-comment)
;; Narrow:
("G" TeX-narrow-to-group)
("i" LaTeX-narrow-to-environment)
;; Errors
("R" TeX-error-overview)
("B" TeX-toggle-debug-bad-boxes)
("W" TeX-toggle-debug-warnings)
("q" nil))

```

H. Hooks

ID: org_gcr_2018-03-05_mara:5C39E41D-C6EC-4A6F-ACB5-A59B472D02

- Hook customization
 - If for all text modes then add it to text-mode-hook
 - If for all AUCTeX modes then add it to TeX-mode-hook
 - Otherwise add it to a mode specific hook

```

(defun help/TeX-mode-hook ()
  (key-chord-define-local "hh" #'help/hydra/auctex/
    body)
  (LaTeX-math-mode)
  (TeX-fold-mode 1)

```

```
(prettify-symbols-mode)
(TeX-install-toolbar)
(setq LaTeX-enable-toolbar t)
(TeX-interactive-mode)
(local-unset-key (kbd "C-j"))
(add-hook 'TeX-mode-hook 'help/TeX-mode-hook)
```

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-class`

- 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"
        "\documentclass[12pt, pagesize,
UScommercial9]{scrlettr2}"))
    (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. BibTeX

ID: org_gcr_2018-09-19T12-32-27-05-00_cosmicality:5B76D1FD-35D2-43

Load BibTeX mode.

At some point I want to cut over to BibLaTeX.

```
(require 'bibtex)
```

Helper functions.

```
(defun help/bibtex-calculate-new-reference-key ()
  (interactive)
  (bibtex-clean-entry))
(defun help/bibtex-maybe-troublesome-mode-disable
  (interactive)
  (aggressive-indent-mode 'toggle)
  (visual-line-mode 't))
(defun help/bibtex-maybe-troublesome-mode-enable
  (interactive)
  (aggressive-indent-mode nil)
  (visual-line-mode nil))
```

Helper Hydra

```
(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" help/bibtex-calculate-new-reference-key)
  ("f" bibtex-fill-entry)
  ("F" bibtex-reformat)
  ("m" help/bibtex-maybe-troublesome-mode-disable)
  ("M" help/bibtex-maybe-troublesome-mode-enable))
```

Mode hook and key bindings.

```
(defun help/bibtex-mode-hook-fn ()
  (key-chord-define-local "hh" #'help/bibtex/body))
(add-hook 'bibtex-mode-hook #'help/bibtex-mode-hook-fn)
```

A. Entry Format Definition

ID: org_gcr_2018-11-13T18-00-23-06-00_cosmicality:715AC227-166

Start from scratch.

```
(setq bibtex-entry-format nil)
```

"Delete empty optional and alternative fields and remove OPT and ALT prefixes from used fields."

```
(add-to-list 'bibtex-entry-format 'opts-or-alts)
```

"Signal an error if a required field is missing."

```
(add-to-list 'bibtex-entry-format 'required-fields)
```

"Delete whitespace at the beginning and end of fields."

```
(add-to-list 'bibtex-entry-format 'whitespace)
```

"Realign entries, so that field texts and perhaps equal signs begin in the same column. Also fill fields."

```
(add-to-list 'bibtex-entry-format 'realign)
```

"Change delimiters according to variables 'bibtex-field-delimiters' and 'bibtex-entry-delimiters'."

```
(add-to-list 'bibtex-entry-format 'delimiters)
```

BibTeX is a structured data format. Like most other ones after you stare at it a while you get the format stuck in your head pretty well. So well in fact that you only look at the field names as a reference point and fast confirmation that they are present in order to get what you really want—the field value. In other words, eventually you quit reading the field definitions. Therefore they ought to be as inconspicuous as possible. In my setup that means they are in lower case.

"Change case of entry types and field names." by lower-casing them.

```
(add-to-list 'bibtex-entry-format 'unify-case)
```

"Sort fields to match the field order in 'bibtex-BibTeX-entry-alist'."

```
(add-to-list 'bibtex-entry-format 'sort-fields)
```

B. Autokey

ID: org_gcr_2018-11-12T12-28-14-06-00_cosmicality:9E369A4B-01E

The goal is to make key generation 100% automated and 100% future proof. A combination of rules and best-practices should make this "just work".

C. Key

ID: org_gcr_2018-11-12T12-28-14-06-00_cosmicality:99541FA5-70D

Separate them using a Hyphen-Minus sign. Theye easy to read, acceptable to operating systems, ASCII-safe, and don't take up much space.

```
(setq bibtex-autokey-name-year-separator "-")
(setq bibtex-autokey-year-title-separator "-")
(setq bibtex-autokey-titleword-separator "-")
```

D. Name

ID: org_gcr_2018-11-12T12-28-14-06-00_cosmicality:0BC72687-242

When there are only one or two authors then include each of them, using their full names. Separate them using a Hyphen-Minus sign.

When there are more than two authors then only use the first author, using his full name, followed by "et al".

```
(setq bibtex-autokey-names 1)
(setq bibtex-autokey-names-stretch 1)
(setq bibtex-autokey-name-separator "-")
(setq bibtex-autokey-additional-names "-et-al")
```

E. Year

ID: org_gcr_2018-11-12T12-28-14-06-00_cosmicality:4E9BF2AD-F29

Use four digits for the year: it is easier to read.

```
(setq bibtex-autokey-year-length 4)
```

F. Title

ID: org_gcr_2018-11-12T12-28-14-06-00_cosmicality:12B4D728-DF

Include two title words.

I don't want them to take up too much space to trim them to 5 characters. WolframAlpha says that the average word length in English 5.1 characters. What kind of words? All kinds. I looked at examples and it seems good enough to jog your memory of the publication.

```
(setq bibtex-autokey-titlewords 2)
(setq bibtex-autokey-titlewords-stretch 1)
(setq bibtex-autokey-titleword-length 5)
```

G. Suggestions

ID: org_gcr_2018-11-12T12-28-14-06-00_cosmicity:A10046EA-810

To reinforce this automated approach: disallow the user from editing the generated key before it is used. The idea here is that when key naming issues appear that it is better to correct it in the algorithm instead of individual key name.

```
(setq bibtex-autokey-edit-before-use nil)
```

iv. Texinfo

ID: org_gcr_2017-05-12_mara:2A19DF86-96B0-4371-8DE9-1F79861F5E89

Perhaps the first document typeset with Org-TeXinfo.

v. Pandoc

ID: org_gcr_2017-05-12_mara:BF923ADF-EB81-4695-9F98-F68D557D667B

```
(use-package ox-pandoc
  :ensure t
  :config
  (setq org-pandoc-options-for-beamer-pdf '((pdf-engine
    . "xelatex"))))
(setq org-pandoc-options-for-latex-pdf '((pdf-engine
  . "xelatex")))
(setq org-pandoc-format-extensions '(markdown_github+
  pipe_tables+raw_html))
(with-eval-after-load 'ox
  (require 'ox-pandoc)))
```

vi. Markdown

ID: org_gcr_2017-05-12_mara:005B395D-AF3F-471D-AD11-45FDBAA85D7A

Markdown.

```
(use-package markdown-mode
  :ensure t
  :config
  (add-to-list 'auto-mode-alist '("\\.markdown\\'" .
    markdown-mode))
  (add-to-list 'auto-mode-alist '("\\.md\\'" .
    markdown-mode))
  (use-package ox-gfm
    :ensure t
    :config
    (add-to-list 'auto-mode-alist '("README\\.md\\'" .
      gfm-mode)))
  (help/on-mac-os
    (setq markdown-open-command "/Users/gcr/util/mark")))
)
```

Generate Markdown from Emacs Lisp comments.

```
(use-package el2markdown
  :ensure t
  :config
  (defun help/el2markdown-post-convert-hook-fn (buf)
    (interactive)
    (message (format "el2markdowned: %s" buf)))
  (add-hook 'el2markdown-post-convert-hook #'help/
    el2markdown-post-convert-hook-fn))
```

vii. 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))
```

Export HTML to the clipboard.

```
(use-package ox-clip
  :ensure t)
```

viii. 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:]]"))
```

ix. Beamer

ID: org_gcr_2017-05-12_mara:1D459BA8-B1B0-40B9-B733-6B59216FD1B2

Load Beamer for creating presentations.

```
(use-package ox-beamer)
```

x. 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)))
```

(g) DevOps

ID: org_gcr_2017-05-12_mara:F79C9679-7887-40B8-8882-51EE6EC1BDE2

i. Shell Scripty

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-C9D987A83E

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

```
(use-package sh-script
  :config
  (defun help/sh-mode-hook-fn ()
    (interactive)
    (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)
```

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

```
(use-package make-mode
  :config
  (defun help/make-mode-hook ())
  (add-hook 'makefile-mode-hook #'help/make-mode-hook)
  (add-to-list 'auto-mode-alist '("Makefile" .
    makefile-mode))
```

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

```
(require 'crontab-mode)
(add-to-list 'auto-mode-alist '("\\.cron\\(tab\\)?\\\\"
  . crontab-mode))
(add-hook 'crontab-mode-hook #'
  turn-on-stripe-buffer-mode)
```

vii. Ansible

ID: org_gcr_2018-01-17_mara:3CB6A9AB-EB08-4762-A017-249764566C1E

Remember it is a minor mode that will only ever run under
YAML mode.

:ID: org_gcr_2018-01-03_mara:626CA760-B710-4A90-A585-66C5EEC2DAD9

```
(use-package ansible
  :ensure t
  :config
  (defhydra help/hydra/ansible (:color blue
                                :hint nil)
    "
  Ansible: (q to quit)
  _D_ecrypt buffer _E_ncrypt buffer
  "
    ("D" #'ansible::decrypt-buffer)
    ("E" #'ansible::encrypt-buffer)
    ("q" nil))
  (key-chord-define ansible::key-map "hh" #'help/hydra/
    ansible/body))
```

(h) 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)
```

(i) Computer-aided design

ID: org_gcr_2017-05-12_mara:50E7A50A-4259-4B6C-8612-99671C6723CC

(j) 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) 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))
```

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)))
```

Visualize ASCII strings with Unicode.

```
(global-prettify-symbols-mode)
(setq prettify-symbols-unprettify-at-point 'right-edge)
```

(b) Color Theme

ID: org_gcr_2017-05-12_mara:3C38AF9D-1905-4B33-A4F5-065D17E9B647

Solarized Theme

- ~1,000 Faces Defined
- 110,003
- 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.

```
(add-to-list 'load-path "~/src/solarized-emacs")
(add-to-list 'custom-theme-load-path "~/src/solarized-emacs")
(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-light t)
(eval-after-load "smart-mode-line"
  '(sml/setup))
```

(c) 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)
```

(d) 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-mac-os (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))

```

(e) 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 - PIE Interactive
Editor "))

```

A. Bitmaps

ID: org_gcr_2017-07-11_mara:02FCF4D1-5613-49B3-A247-6BFA846D7E

Use local bitmaps.

```

(add-to-list 'image-load-path "~/src/help/xpm")
(add-to-list 'image-load-path "~/src/help/tiff")

```

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 --ke
```
 - Convert PNG to XPM


```
convert new.png new.xpm
```

Emacs added macOS support for TIFF. Emacs now prefers TIFF. XPM should work everywhere so I want those to be the masters. They are the masters because when I make corrections they go there. For example I had a 27x24 bitmap that I had to trim to 24x24. So I converted XPMs to TIFF and use those. Then I switched to stock Emacs and deleted the TIFF files.

B. Menu bar

ID: org_gcr_2017-06-23_mara:47EF669D-E999-4573-AA5C-5CF038ADA8

Enable the menu bar.

```
(menu-bar-mode nil)
```

C. Tool bar

ID: org_gcr_2017-06-23_mara:EC50EB46-7878-4799-9805-EF0CB3D3E5

Disable the tool bar.

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

Style the tool bar.

```
(setq tool-bar-style 'both)
```

ibuffer.

```
(tool-bar-add-item
 "list-alt"
 'ibuffer
 'Buffers
 :help "List buffers")
```

swap.

```
(tool-bar-add-item
 "exchange"
 'help/safb-switch-to-previous-buffer
 'Swap
 :help "Swap buffers")
```

Page up and down.

```
(tool-bar-add-item
 "arrow-up"
 'scroll-down-command
 'ScrollUp
 :help "Scroll Down")
(tool-bar-add-item
 "arrow-down"
 'scroll-up-command
 'ScrollDown
 :help "Scroll Up")
```

OS Banner.

```
(help/on-mac-os
 (tool-bar-add-item
  "apple"
  'do-nothing
  'Powered-by-macOS
  :help "Powered by macOS"
  :enable nil))
(help/on-gnu/linux
 (tool-bar-add-item
  "linux"
  'do-nothing
  'Powered-by-GNU-Linux
  :help "Powered by GNU/Linux")
```

```

:enable nil))
(help/on-windows
(tool-bar-add-item
 "windows"
 'do-nothing
 'Powered-by-Microsoft-Windows
 :help "Powered by Microsoft Windows"
 :enable nil))

```

D. Scroll bar

ID: org_gcr_2017-06-23_mara:590AD87D-0567-454F-98AE-3AE64A1710

Enable both the vertical and horizontal scroll bar.

```

(scroll-bar-mode nil)
(help/on-gnu/linux
(horizontal-scroll-bar-mode nil))

```

(f) Pointer

ID: org_gcr_2017-05-12_mara:17DB4E08-F6C4-44AF-935B-27E8984F13DB

Hide the pointer when typing.

```
(setq make-pointer-invisible t)
```

(g) Version Control

ID: org_gcr_2017-05-12_mara:09918DB9-3AE3-44DF-9E1F-A4A3918F3315

Ediff split frame horizontally.

```
(setq ediff-split-window-function '
  split-window-horizontally)
```

Smerge.

```
(diminish 'smerge-mode)
```

(h) Window

ID: org_gcr_2017-05-12_mara:FBD41A4B-9764-49CB-8439-74581B2211A0


```

("C" #'eyebrowse-create-window-config :exit nil)
("R" #'eyebrowse-rename-window-config :exit nil)
("T" #'eyebrowse-switch-to-window-config :exit nil)
("I" #'eyebrowse-prev-window-config :exit nil)
("O" #'eyebrowse-next-window-config :exit nil)
("P" #'eyebrowse-last-window-config :exit nil)
;;;;
("e" windmove-up :exit nil)
("s" windmove-left :exit nil)
("d" windmove-down :exit nil)
("f" windmove-right :exit nil)
;;
("a" ace-window :exit nil)
;;
("c" (lambda () (interactive) (
  resize-window--cycle-window-negative)) :exit nil)
("n" (lambda () (interactive) (
  resize-window--cycle-window-positive)) :exit nil)
;;
("i" (lambda () (interactive) (
  resize-window--enlarge-down)) :exit nil)
("j" (lambda () (interactive) (
  resize-window--shrink-horizontally)) :exit nil)
("k" (lambda () (interactive) (resize-window--enlarge-up)
) :exit nil)
("l" (lambda () (interactive) (
  resize-window--enlarge-horizontally)) :exit nil)
;;
("v" split-window-vertically :exit nil)
("b" split-window-horizontally :exit nil)
("g" (lambda () (interactive) (
  resize-window--reset-windows)) :exit nil)
;;
("x" golden-ratio :exit nil)
("X" golden-ratio-toggle-widescreen :exit nil)
;;
("z" kill-this-buffer)
("/" delete-other-windows :exit nil)
;;
("w" winner-undo :exit nil)
("r" winner-redo :exit nil)
;;
("q" nil))

```

Cycle forward and backward in window flow.

```
(setq winner-dont-bind-my-keys t)
(winner-mode t)
(defhydra help/hydra/winner-mode (:color blue
                                   :hint nil)
  "
winner-mode:  _q_uit  _p_revious  _n_ext
"
  ("p" winner-undo :exit nil)
  ("n" winner-redo :exit nil)
  ("q" nil))
```

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-right)
  (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 dis-

tributed with Emacs are configured here.

```
(setq help-window-select t)
```

Fling buffers left and right.

```
(use-package buffer-move
  :ensure t
  :config
  (defhydra help/hydra/buf-move (:color blue
                                :hint nil)
    "
  Buf-Move: (q to quit)
    ↑
  _e__i_←
_s__j__f__l_ →
  _d__k_
    ↓
  "
    ("e" buf-move-up :exit nil)
    ("d" buf-move-down :exit nil)
    ("s" buf-move-left :exit nil)
    ("f" buf-move-right :exit nil)
    ("i" buf-move-up :exit nil)
    ("k" buf-move-down :exit nil)
    ("j" buf-move-left :exit nil)
    ("l" buf-move-right :exit nil)
    ("q" nil)))
```

Resize window.

```
(use-package resize-window
  :ensure t
  :config
  (key-chord-define-global "ww" #'resize-window))
```

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 ideas 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 will 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 or C-M-s 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.
 - C-M-s is for the operating system
 - * Discovered this in macOS
- 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 perform. 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 assumptions 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-B26F335410

- 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.
- 2017-07-24
 - This worked out well and close to expected. The s(uper) name-space is mostly free. Any package using it well I stomped on its bindings without even looking at them. They are supposed to be free but the C-c and C-x name-space is running out and people get mad about it (I did, and my solution is below). Despite my approach I still ran out of namesake for my super bindings. That surprised me. So I tried using H(yper) but isn't easily on every keyboard so I converted all of those bindings to C-M- and that worked out great but now I can't use that for more specific bindings. The whole left and right zones of the keyboard has worked out well when I remember and it is pretty easy to remember. The only problem with this approach is that I still ran out of name-space despite my approach below. So between key-chords and using the super key and moving a bunch of stuff into Hydras I still ran out of space. That leaves C-s for more name-space for the left and right side. I'm glad to try this out.

For example, in Org-Mode:

- I traverse the entire document very quickly with `org-babel-previ` 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:D7EDA057-1289-4EB9-8A72-3B00DC3C87

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-51B7860535

: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-4A1E50C746

: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-00B46A3B39

- 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` name-space 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

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

```

+-----+
| +-----+                               +-----+ |
| |s      |                               |s/ret| |
| +-----+                               +-----+ |
| +-----+                               +-----+ |
| |SHIFT  |                               | SHIFT| |
| +-----+                               +-----+ |
|                                     +-+ +-+ +-----+ +-+ +-+ |
|                                     |M| |C| |spc | |C| |M| |
|                                     +-+ +-+ +-----+ +-+ +-+ |
|                                     |                                     |
+-----+

```

On macOS Dyalog uses an input source for entering APL symbols. Enable its input source, press option-i, and you get the symbol \square (APL FUNCTIONAL SYMBOL IOTA). It should work in every macOS 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 macOS. 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 macOS. 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 macOS) 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.

```

+-----+
| +-----+                               +-----+ |
| |C      |                               |C/ret| |
| +-----+                               +-----+ |
| +-----+                               +-----+ |
| |SHIFT  |                               | SHIFT| |
| +-----+                               +-----+ |
| +-+      +-+ +-+ +-----+ +-+ +-+    |
| |H|      |s| |M| |spc  | |M| |s|      |
| +-+      +-+ +-+ +-----+ +-+ +-+    |
|                                         |
+-----+

```

- Cheat-Sheet

- C, M, C-M, C-M, C-S, C-s are for Emacs
- C-M-s are for the operating system

(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

Keybindings for every mode. Starts and stop:

- 1-5
- Q-T
- A-G
- Z-B

A. Row 6

ID: org_gcr_2018-04-04T19-35-03-05-00_mara:5674E80B-C4B1-4AFA-

Try to reserve this for the OS.

:ID: org_gcr2018-01-08_mara:D12EA6D3-9746-4AA3-A622-71314840A6FA

```
(global-set-key [(shift f6)] #'
  kmacro-name-last-macro)
(global-set-key [(f6)] #'insert-kbd-macro)
(global-set-key [(f5)] #'elmacro-show-last-macro)
```

B. Row 5

ID: org_gcr_2018-04-03T15-26-04-05-00_mara:8EE3BC1F-945D-422C-

```
(global-set-key (kbd "C-4") #'hs-toggle-hiding)
(global-set-key (kbd "C-3") #'help/
  my-toggle-hideshow-all)
(defun help/kill-buffer ()
  (interactive)
  (kill-buffer (current-buffer)))
(global-set-key (kbd "C-5") #'help/kill-buffer)
(global-set-key (kbd "C-M-=") #'
  edit-indirect-region)
(global-set-key (kbd "C-M-4") #'help/1-window)
;; Leave C-8, C-9, and C-0 free for the OS
(defun help/insert-em-dash ()
  "Insert a EM-DASH.
- \"best limited to two appearances per sentence\"
- \"can be used in place of commas to enhance
  readability."
```

Note, however, that dashes are always more emphatic than commas\."

- \"can replace a pair of parentheses. Dashes are considered less formal than parentheses; they are also more intrusive.
If you want to draw attention to the parenthetical content, use dashes. If you want to include the parenthetical content more subtly, use parentheses.\"
- \"Note that when dashes are used in place of parentheses, surrounding punctuation should be omitted.\"
- \"can be used in place of a colon when you want to emphasize the conclusion of your sentence. The dash is less formal than the colon.\"
- \"Two em dashes can be used to indicate missing portions of a word, whether unknown or intentionally omitted.\"
- \"When an entire word is missing, either two or three em dashes can be used. Whichever length you choose, use it consistently throughout your document. Surrounding punctuation should be placed as usual.\"
- \"The em dash is typically used without spaces on either side, and that is the style used in this guide. Most newspapers, however, set the em dash off with a single space on each side.\"

Source: URL ``https://www.thepunctuationguide.com/em-dash.html'`
(interactive)
(help/real-insert -?))
(defun help/insert-en-dash ()
"Insert a EN-DASH.
- \"is used to represent a span or range of numbers , dates,

```

    or time. There should be no space between the en
    dash and
    the adjacent material. Depending on the context,
    the en
    dash is read as ""to or "through".\"
- \"If you introduce a span or range with words
  such as
    'from' or 'between', do not use the en dash.\"
- \"is used to report scores or results of contests
  .\"
- \"an also be used between words to represent
  conflict,
  connection, or direction.\"
- \"When a compound adjective is formed with an
  element that
  is itself an open compound or hyphenated
  compound, some
  writers replace the customary hyphen with an en
  dash. This
  is an aesthetic choice more than anything.
Source: URL `https://www.thepunctuationguide.com/
en-dash.html'\"
(interactive)
(help/real-insert -?))
(defun help/insert-hyphen ()
  \"Insert a HYPHEN
- \"For most writers, the 'hyphens primary function
  is the
  formation of certain compound terms. The hyphen
  is also
  used for word division [in typesetting].
- \"Compound terms are those that consist of more
  than one
  word but represent a single item or idea.\"
Source: URL `https://www.thepunctuationguide.com/
hyphen.html'\"
(interactive)
(help/real-insert ?-))
(global-set-key (kbd \"-\") #'help/insert-hyphen)
(global-set-key (kbd \"s-_\") #'help/insert-em-dash)
(global-set-key (kbd \"s--\") #'help/insert-en-dash)
(global-set-key (kbd \"s-1\") #'mc/edit-lines)
(global-set-key (kbd \"s-2\") #'mc/mark-all-like-this
)
(global-set-key (kbd \"s-3\") #'mc/

```



```

    mark-previous-like-this)
(global-set-key (kbd "s-4") #'mc/
  mark-next-like-this)
(global-set-key (kbd "s-$") #'
  mc-hide-unmatched-lines-mode)
(global-set-key (kbd "s-5") #'mc/
  skip-to-previous-like-this)
(global-set-key (kbd "s-6") #'mc/
  skip-to-next-like-this)

```

Checking things.

```

(defhydra help/hydra-checking (:color blue
                               :hint nil)
  "
Flycheck:  ☐%(bound-and-true-p flycheck-mode)
Controls: _u_: checker/toggle _i_ checker/list
Navigate: _j_ next _k_ previous _f_ first _l_ last
Error: _d_ describe _e_ explain
       _q_ quit
"
  ("u" flycheck-mode :exit nil)
  ("i" help/safb-flycheck-list-errors)
  ("j" flycheck-next-error :exit nil)
  ("k" flycheck-previous-error :exit nil)
  ("f" flycheck-first-error)
  ("l" flycheck-last-error)
  ("d" flycheck-display-error-at-point)
  ("e" flycheck-explain-error-at-point)
  ("q" nil))
(global-set-key (kbd "C-M-9") #'help/hydra-checking
  /body)

```

C. Row 4

ID: org_gcr_2018-06-06T22-01-54-05-00_mara:D34D40ED-F8E3-405D-

:ID: org_gcr2017-12-07_mara:752198AC-06ED-436E-B526-5A3C8ED32375

```

(global-set-key (kbd "s-w") #'help/hydra/megawin/
  body)

```

Date related things.

```

(defhydra help/hydra/timestamp (:color blue :hint
  nil)

```

```

"
Timestamps: (_q_uit)
  Date: _I_S0, _U_S, US With _Y_ear and _D_ashes,
        US In _W_ords
  Date/Time: _N_o Colons or _w_ith
  Org-Mode: _T_oday, _R_ight Now, or _c_hoose
"

("q" nil)
("I" help/insert-datestamp)
("U" help/insert-datestamp-us)
("Y" help/insert-datestamp-us-full-year)
("D" help/
  insert-datestamp-us-full-year-and-dashes)
("W" help/insert-datestamp-us-words)
("N" help/insert-timestamp-no-colons)
("w" help/insert-timestamp)
("T" help/insert-org-datestamp)
("R" help/org-time-stamp-with-seconds-now)
("c" org-time-stamp))
(global-set-key (kbd "C-t") #'help/hydra/timestamp/
  body)

```

```

(global-set-key (kbd "C-M-<tab>") #'help/
  untabify-buffer-or-region-if-not-indent-tabs-mode
  )
(global-set-key (kbd "C-M-\\") #'help/
  indent-region-or-buffer)
(defun help/copy-entire-buffer ()
  (interactive)
  (save-excursion
    (call-interactively 'mark-whole-buffer)
    (call-interactively 'kill-ring-save)
    (message "Buffer contents copied")))
(global-set-key (kbd "s-q") (lambda () (interactive
  ) (switch-to-buffer "*scratch*")))
(global-set-key (kbd "C-c C-r") 'ivy-resume)
(global-set-key (kbd "C-M-y") #'insert-char)
(define-key org-mode-map (kbd "C-o") nil)
(global-set-key (kbd "C-o") nil)
(global-set-key (kbd "C-o") #'counsel-find-file)
(global-set-key (kbd "C-M-o") #'help/occur-dwim)
(global-set-key (kbd "M-i") nil)
(global-set-key (kbd "M-i") #'help/hydra/left/
  describe/body)
(defhydra help/hydra/left/describe (:color blue

```

```

nil)
"
Describe Something: (q to quit)
_a_ all help for everything screen
_b_ bindings
_B_ personal bindings
_r_ character char name only
_c_ character briefly
_C_ character full
_T_ character set _1_ list characters in oneset
    _2_ list all set2
_S_ coding system
_f_ function
_F_ flycheck checker
_g_ categories
_i_ input method
_I_ info buffer
_k_ key briefly
_K_ key
_l_ language environment
_L_ mode lineage
_m_ major mode
_M_ minor mode
_n_ current coding system briefly
_N_ current coding system full
_o_ lighter indicator
_O_ lighter symbol
_p_ package
_P_ text properties
_s_ symbol
_t_ theme
_v_ variable
_w_ where is something defined
"

("1" list-charset-chars)
("2" list-character-sets)
("b" describe-bindings)
("B" describe-personal-keybindings)
("r" ucs-eval)
("c" (lambda () (interactive) (
  unicode-fonts-debug-info-at-point)))
("C" describe-char)
("f" counsel-describe-function)
("F" flycheck-describe-checker)

```

```
( "g" describe-categories)
( "i" describe-input-method)
( "I" info-buffer)
( "K" describe-key)
( "k" describe-key-briefly)
( "l" describe-language-environment)
( "L" help/parent-mode-display)
( "M" describe-minor-mode)
( "m" describe-mode)
( "N" describe-current-coding-system)
( "n" describe-current-coding-system-briefly)
( "o" describe-minor-mode-from-indicator)
( "O" describe-minor-mode-from-symbol)
( "p" describe-package)
( "P" describe-text-properties)
( "q" nil)
( "a" help)
( "s" describe-symbol)
( "S" describe-coding-system)
( "t" describe-theme)
( "T" describe-character-set)
( "v" counsel-describe-variable)
( "w" where-is))
```

:ID: org_{gcr}2017-05-12_{mara}:20E96ACD-1D91-480C-BA3B-D5730EA2173C

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 and eval-* become confused → quit working.

D. Row 3

ID: org_{gcr}_2017-05-12__{mara}:C0B604C5-F530-4770-9061-92E8468C85

CAPS-LOCK is disappeared with my Karabiner-Elements config so duplicate it.

```
(use-package caps-lock
  :ensure t
  :config
  (global-set-key (kbd "C-M-s-c") #'caps-lock-mode)
  )
```

```
(global-set-key (kbd "C-s") #'swiper)
```

```
(defun help/messages ()
  (interactive)
  (switch-to-buffer "*Messages*"))
(global-set-key (kbd "s-a") #'help/messages)
```

Eval expression.

```
(global-set-key (kbd "C-M-a") #'help/hydra/apropo/
  body)
(global-set-key (kbd "C-d") #'delete-forward-char)
(global-set-key [(control shift ?d)] #'
  backward-delete-char)
(global-set-key (kbd "C-M-k") #'my-eval-expression)
(global-set-key (kbd "C-c C-k") #'help/
  delete-this-buffer-and-file)
(global-set-key (kbd "C-M-'"') #'toggle-quotes)
(global-set-key (kbd "M-'"') #'yas-insert-snippet)
```

```
(global-set-key (kbd "C-h") nil)
(global-set-key (kbd "C-h") #'help/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 "C-M-j") #'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)
  "
  _1_ reset-font _2_ -font _3_ +font _4_ ellipsis _5_
      UUID _6_ bfr-cdng-systm _8_ □ _9_ □ _0_ □ _
      split-window-vertically _= reposition-window
  _q_ uit _Q_ exit-Emacs
  _w_ widen _W_ git time machine
  _e_ FontAwesome _E_ prettify-symbols-mode
  _r_ rename this buffer and file
  _R_ insert the default file header
  _t_ rectangle-mark _T_ trademarks
  _y_ yas tables _Y_ transparency
```

```

_u_ universal-emotions-emoticons _U_ git-grep
_o_ dbgOnErr _O_ base64-encode-region
_p_ query-replace _P_ base64-decode-region
}_ transliterate
|_ split-window-horizontally
[_ single bracket, white, square, left
_]_ single bracket, white, square, right
_d_ dired
_D_ help/safb-org-babel-detangle
_f_ ag _F_ ag*
_g_ grep
_h_ hidden-mode-line
_H_ hack-local-variables (see also normal-mode)
_J_ org-babel-tangle-jump-to-org
_l_ visual-line-mode _L_ aggressive-indent-mode
_K_ nextLogLine
;_ toggle-lax-whitespace
_x_ delete-indentation _X_ checkmarks
_c_ fill-paragraph
_V_ view-only mode
_b_ erase-buffer
_N_ normal-mode (see also hack-local-variables)
_m_ font lock _M_ Marked 2
<_ double bracket, angle, left
>_ double bracket, angle, right
,_ snip comment start
._ snip comment stop
?_ tons of scissors
"

("Q" help/safb-save-buffers-kill-terminal)
("q" nil)
("e" (lambda () (interactive) (insert (
  call-interactively 'fontawesome))))
("E" prettify-symbols-mode)
("|" 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)
("5" help/uuid)
("6" set-buffer-file-coding-system)
("8" (lambda () (interactive) (insert "□")))
("9" (lambda () (interactive) (insert "□")))
("0" (lambda () (interactive) (insert "□"))) :exit

```

```

    nil)
  ("[" (lambda () (interactive) (insert "["))))
  ("]" (lambda () (interactive) (insert "]"))))
  ("r" help/rename-current-buffer-file)
  ("R" auto-insert)
  ("T" help/trademark/body)
  ("=" reposition-window)
  ("g" grep)
  ("h" hidden-mode-line-mode)
  ("H" (lambda () (interactive) (
hack-local-variables) (message "Local variables
hacked"))))
  ("l" visual-line-mode)
  ("L" aggressive-indent-mode)
  ("x" delete-indentation)
  ("X" help/checks/body)
  ("f" ag)
  ("F" help/hydra/ag/body)
  ("v" help/hydra/vocabulary/body)
  ("V" view-mode)
  ("w" widen)
  ("W" git-timemachine)
  ("t" rectangle-mark-mode)
  ("y" yas-describe-tables)
  ("Y" hydra-transparency/body)
  ("J" org-babel-tangle-jump-to-org)
  ("u" universal-emotions-emoticons)
  ("U" counsel-git-grep)
  ("d" dired)
  ("D" help/safb-org-babel-detangle)
  ("K" next-logical-line :exit nil)
  ("N" (lambda () (interactive) (help/
safb-normal-mode) (message "Entering normal-mode
"))))
  ("m" font-lock-mode)
  ("M" help/preview-buffer-file-in-marked-2)
  ("<" (lambda () (interactive) (insert "<"))))
  (">" (lambda () (interactive) (insert ">"))))
  ("," help/chs)
  (". " help/che)
  ("?" 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)
( "' " help/move-file)
( "\" " imenu-list-minor-mode))
```

```
(key-chord-define-global "vv" #'help/hydra/
  left-side/global/body)
```

Attribution.

```
(defhydra help/hydra/apropo (:color blue
                             :hint nil)
  "
  =====+=====
  Apropos'      (_q_uit)
  =====+=====

  Everything:    ..\"all meaningful Lisp ..._s_ymbols\"
  Info:         ..\"all known _I_nfo ...files\"
  =====+=====

  Documentation ..\"symbols whose ..._d_ocumentation\"
  User Manual:  ..\"the Emacs _U_ser Manual.\"
  Lisp Manual:  ..\"the Emacs _L_isp Reference Manual
  .\"
  =====+=====

  Commands      ..\"_c_ommands (interactively
    callable functions)...\"
  Variables      ..\"..._v_ariables\"
  Printed Value: ..\"symbols whose 'values printed
    ..._r_epresentation\"
  =====+=====

  Library:       ..\"variables and functions defined
    by ..._l_library\" or _F_ind It
  User Options:  ..\"user ..._o_ptions\"
  =====+=====

  "
  ("s" apropos)
  ("I" info-apropos)
```



```
( "d" apropos-documentation)
( "U" emacs-index-search)
( "L" elisp-index-search)
( "F" counsel-find-library)
( "c" apropos-command)
( "v" apropos-variable)
( "r" apropos-value)
( "l" apropos-library)
( "o" apropos-user-option)
( "q" nil))
```

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)
  "
_a_ ✓ _s_ x _d_ ☐ _f_ ☑ _g_ ☒_
"
  ("a" (lambda () (interactive) (insert "✓")) :exit
    nil)
  ("s" (lambda () (interactive) (insert "x")) :exit
    nil)
  ("d" (lambda () (interactive) (insert "☐")) :exit
    nil)
  ("f" (lambda () (interactive) (insert "☑")) :exit
    nil)
  ("g" (lambda () (interactive) (insert "☒")) :exit
    nil))
```

```
(global-set-key (kbd "M-s M-s") #'shell)
```

E. Row 2

ID: org_gcr_2017-07-17_mara:B91C9779-B2AD-403B-A11F-5847EF8883

```
;; Leave C-z free for Tmux
(global-set-key (kbd "C-z") nil)
(global-set-key (kbd "C-x C-c") #'help/
  safb-save-buffers-kill-terminal)
(define-key org-mode-map (kbd "C-j") nil)
(global-set-key (kbd "C-j") nil)
(global-set-key (kbd "C-j") #'ivy-switch-buffer)
(global-set-key (kbd "M-j") nil)
(define-key org-mode-map (kbd "M-j") nil)
(global-set-key (kbd "M-j") #'counsel-M-x)
(global-set-key (kbd "C-M-,") #'ibuffer)
(global-set-key (kbd "C-M-.")) nil)
(global-set-key (kbd "M-s-m") #'ns-do-hide-emacs)
(global-set-key (kbd "C-M-n") #'info-buffer)
(global-set-key (kbd "s-s") #'shell)
(global-set-key (kbd "s-v") #'help/hydra/megawin/
  body)
(global-set-key (kbd "s-c") #'help/hydra/buf-move/
  body)
(global-set-key (kbd "s-x") #'help/hydra/
  smart-shift/body)
```

F. Row 1

ID: org_gcr_2017-05-12_mara:A791BE09-ED4B-46F4-8225-EE73C15C1D

Cycle through common options for removing space between characters. Thank you Pragmatic Emacs.

```
(global-set-key (kbd "s-SPC") #'cycle-spacing)
```

G. Unsorted

ID: org_gcr_2017-05-12_mara:4FEBE5F3-3B9E-438D-9E7F-DA337FB43E

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)
```

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".

Make ispell accessible.

```
(key-chord-define-global "qp" #'ispell)
(key-chord-define-global "qo" #'ispell-word)
```

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" because 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

Reserve the right side for mode-specific activities.

It starts with:

- 6
- Y
- H
- N

Try to use s-i for things like info-at-point.

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/on-gui (global-set-key (kbd "s-<return>") #'help/
  smart-open-line))
```

Ansu.

```
(global-set-key (kbd "M-%") #'anzu-query-replace)
(global-set-key (kbd "C-M-%") #'
  anzu-query-replace-regexp)
```