1. README

ID: orgmode:gcr:vela:F651B86D-86C2-43A9-B0E6-CB94963BB502

Configure EMACS to for everything defined within this monolithic system.
Sysop is likely to use this constantly.
Start EMACS with this command:

```
open /Applications/Emacs.app
```

```
;; -*- lexical-binding: t -*-

(load-file "~/src/help/.org-mode-org2blog.emacs.el")
```

(a) HELP Enables Literate Programming

ID: orgmode:gcr:vela:README

Setup

i. Clone Org-Mode to ~/src/.
   A. Without Make: Generating autoloads and Compiling Org without make
   B. As of <2016-01-19 Tue>

```
cd ~/src/org-mode
git pull
emacs -batch -Q -L lisp -l ../mk/org-fixup -f org-make-autoloads
```

ii. Clone Org2Blog to ~/src/.
iii. Clone Use-Package to ~/src/.
iv. Install supporting software adding their exectuable location to the PATH.
   A. Install Oracle Java.
   B. Install LanguageTool renaming it's folder to LanguageTool.
   C. Install PlantUML.
   D. Install Ditaa.
   E. Install MacTeX.
v. 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: orgmode:gcr:vela:STYLEGUIDE

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

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

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

Literate Programming (LP) As Knuth intended.

Out of the Box (OOTB) The default configuration.

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

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

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

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

- Encoding
  - Prefer Unicode characters over ASCII equivalents.
    - Note eventual switch form PDFLaTeX to LuaTeX.
  - Consider Org-Mode automatic handling of ASCII to UTF-8 symbols.
- File/Package Loading
  - Load every one with `use-package` whether it came with EMACS OOTB or ELPA.
    - `ensure t` tells the reader which one it came from.
  - Binding definitions often live in Piano Lessons.
  - Configurations aren't meant to be fully transplant-able because this system monolithic. In the interest of collaboration as much of the package configuration lives in the `use-package` block as possible.
- Formatting
  - Code Snippet.
    - Programming language expressions.
    - Shell commands that are executable
    - Use `code style`
  - Non-Code Snippet.
    - File types
    - Program names that aren't referring to the artifact itself
      - Shell scripts and commands
      - `Scheme` and `Java`
    - Use `verbatim` style
  - 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.

2. Special Operating Procedure

   `ID: orgmode:gcr:vela:97A95862-3213-4035-9FF6-E041796DAB5C`

   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) Display

   `ID: orgmode:gcr:vela:1290DB2D-D05E-4DDD-B42F-6B11AE91F480`

   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))
```

(b) Hydra

```
ID: orgmode:gcr:vela:9B78FBB7-6C6A-4BD6-A9CC-FB192D37F6C2
```

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

(c) Keyboard

```
ID: orgmode:gcr:vela:8A0E58DF-7C90-4781-AC12-94D2D76F47C7
```

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)
```

(d) Libraries

```
ID: orgmode:gcr:vela:21919848-B720-4D30-880E-485C41250279
```

Dash is nice to use.

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

F is nice to use.

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

S is nice to use.

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

Caching.

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

(e) Modeline

```
ID: orgmode:gcr:vela:798F14D1-EDC6-4306-8E82-0854980AEFBA
```

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)
```

(f) OS X

```
ID: orgmode:gcr:vela:6556EACF-2F83-4B84-8456-5BEB981D290E
```

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

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

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

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

Configure the meta keys.
Use the OS X modifiers as Emacs meta keys. Don't pass them through to OS X.
Easily allow option pass through for alternate input methods.

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

EMACS dialogues don't work OSX. They lock up EMACS.
This is a known issue. Here is the solution.

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

(g) Windows

```
ID: orgmode:gcr:vela:B21664CF-62AF-4ACC-A239-FE20672FF9E4
```

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

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

Provide the proper shell.

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

Enable the super key-space.

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

3. Standard Operating Procedure

```
ID: orgmode:gcr:vela:8302B38B-67EC-4C37-9B42-69E278FF1277
```

Configure EMACS to maximum utility.

(a) Helper Functions

```
ID: orgmode:gcr:vela:B2257535-9891-48F1-B7CD-1B385F527C59
```

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

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

(defun describe-thing-in-popup ()
  "Attribution: URL `http://blog.jenkster.com/2013/12/popup-help-in-emacs-lisp.html'."
  (interactive)
```

```
  (let* ((thing (symbol-at-point))
         (help-xref-following t)
         (description (with-temp-buffer
                        (help-mode)
                        (help-xref-interned thing)
                        (buffer-string))))
    (popup-tip description
               :point (point)
               :around t
               :height 30
               :scroll-bar t
               :margin t)))

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

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

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

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

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

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

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

(defun help/indent-curly-block (&rest _ignored)
  "Open a new brace or bracket expression, with relevant newlines and indent. URL: ‘https://github.co
  (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 "..."))

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

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

(progn

  (defvar my-read-expression-map
    (let ((map (make-sparse-keymap)))
      (set-keymap-parent map read-expression-map)
```

```
      (define-key map [(control ?g)] #'minibuffer-keyboard-quit)
      (define-key map [up]   nil)
      (define-key map [down] nil)
      map))

  (defun my-read--expression (prompt &optional initial-contents)
    (let ((minibuffer-completing-symbol t))
      (minibuffer-with-setup-hook
          (lambda ()
            (emacs-lisp-mode)
            (use-local-map my-read-expression-map)
            (setq font-lock-mode t)
            (funcall font-lock-function 1))
        (read-from-minibuffer prompt initial-contents
                              my-read-expression-map nil
                              'read-expression-history))))

  (defun my-eval-expression (expression &optional arg)
    "Attribution: URL `https://lists.gnu.org/archive/html/help-gnu-emacs/2014-07/msg00135.html'."
    (interactive (list (read (my-read--expression ""))
                       current-prefix-arg))
    (if arg
        (insert (pp-to-string (eval expression lexical-binding)))
      (pp-display-expression (eval expression lexical-binding)
                             "*Pp Eval Output*"))))

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

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

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

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

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

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

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

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

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

Attribution: URL ‘http://www.masteringemacs.org/articles/2010/11/29/evaluating-elisp-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/msg0008
  ‘(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-directo
  (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 sacha/unfill-paragraph (&optional region)
  "Takes a multi-line paragraph and makes it into a single line of text.

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

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

```
(defun help/org-weave-readme ()
  (interactive)
  (help/org-weave-subtree-gfm
   "README"
   "README.md"))

(defun help/org-weave-style-guide ()
  (interactive)
  (help/org-weave-subtree-gfm
   "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 help/reformat-file (file)
  "Reformat a file.

Handle whether a buffer is attached to the file or not.

Be sure that most revent version of file is loaded into buffer first.

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

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

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

Attribution: URL `https://www.emacswiki.org/emacs/SwitchingBuffers#toc5'"
```

```
  (interactive)
  (switch-to-buffer (other-buffer (current-buffer) 1)))

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

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

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

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

(b) Typography

```
ID: orgmode:gcr:vela:F355CA52-794D-474E-959B-D85C689B96AA
```

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

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

Two spaces after a period end a sentence.

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

Two spaces follow a colon
Two spaces after a semi-colon.
One space after comma.

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

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

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

- Display line in file

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

(add-hook 'text-mode-hook #'help/text-prog*-setup)
```

(c) Buffer

ID: orgmode:gcr:vela:F3C9BDE1-C0E0-4BDF-B121-3CE2F0D16464

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

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

Provide expected "Undo" functionality.

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

Ensure that buffers do not end with a new line. This is the decision of Sysop. This is important to YASnippets and Source-Blocks. Doing so would violate POLA.

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

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)
```

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)
```

Visualize the formfeed character.

```
(use-package page-break-lines
  :ensure t)
```

Configure Page-Break-Lines-Mode.

```
(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)
```

(d) Code Folding

```
ID: orgmode:gcr:vela:3F70676D-C141-4093-9E40-F42B6C7B7232
```

```
(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)))
  :diminish hs-minor-mode)
```

(e) Colors

```
ID: orgmode:gcr:vela:7FA1B7C2-3C4B-4119-B9B7-4C0CC0EAA180
```

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

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

(f) Debugging

```
ID: orgmode:gcr:vela:59CAB0A5-9F2E-498C-B005-F87BBE974A35
```

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

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

(g) Evaluation

```
ID: orgmode:gcr:vela:5E067457-9B99-459F-A660-323774C14BF5
```

```
(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)
```

Easily send expressions to a REPL line by line by hitting `C-RET`.

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

This version has the opposite behavior to the eir-jump-after-eval
configuration when invoked to evaluate a line."
    (interactive)
    (let ((eir-jump-after-eval (not eir-jump-after-eval)))
      (eir-eval-in-shell)))
  (add-hook 'sh-mode-hook
            '(lambda()
               (local-set-key (kbd "C-M-<return>") 'eir-eval-in-shell2)))
  ;; racket-mode support (for Racket; if not using Geiser)
  (require 'racket-mode) ; if not done elsewhere
  (require 'eval-in-repl-racket)
  (define-key racket-mode-map (kbd "<C-return>") 'eir-eval-in-racket))
```

(h) Encryption

`ID: orgmode:gcr:vela:9A41F9EE-36D5-452A-986B-70B567255D36`

Easy to use file-based AES encryption.

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

(i) Eshell

`ID: orgmode:gcr:vela:B371A592-1251-4D88-A055-43CA3E33BC6D`

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

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

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

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

Configure a `PS1` like prompt.

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

(j) File Based System

`ID: orgmode:gcr:vela:E6F121F0-AC8E-45C7-9F11-0E7AB93E4B71`

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

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

(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))
```

Selection:

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

Future candidates:

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

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

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

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

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

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

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

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

```
ID: orgmode:gcr:vela:067D598E-7FE6-4BC5-AEF7-872966390970
```

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

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

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

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

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

```
(defun help/dired-copy-filename ()
  "Push the path and filename of the file under the point to the kill ring.
  Attribution: URL `https://lists.gnu.org/archive/html/help-gnu-emacs/2002-10/msg00556.html'"
  (interactive)
  (message "Added %s to kill ring" (kill-new (dired-get-filename))))
(defun help/dired-copy-path ()
  "Push the path of the directory under the point to the kill ring."
  (interactive)
  (message "Added %s to kill ring" (kill-new default-directory)))
(setq dired-listing-switches "-alh")
(setq dired-recursive-deletes 'top)
(use-package dired-details+
  :ensure t)
(setq-default dired-details-hidden-string "")
(defun help/dired-mode-hook-fn ()
  "HELP dired customizations."
  (local-set-key "c" #'help/dired-copy-filename)
  (local-set-key "]" #'help/dired-copy-path)
  (diff-hl-dired-mode)
  (load "dired-x")
  (turn-on-stripe-buffer-mode)
  (stripe-listify-buffer))
(add-hook 'dired-mode-hook #'help/dired-mode-hook-fn)
```

Try to guess the target directory for operations.

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

Use EMACS `ls`.

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

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

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

Substitution:

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

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

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

- Not sure what this means

Synchronicity:

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

`<cmd> &` execute in parallel

`<cmd> ;` execute sequentially, asynchronously

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

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

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

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

Noting that:

`find-dired` is the general use case

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

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

`find-lisp-find-dired` for anything

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

Key notes on working with editable buffers...

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

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

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

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

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

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

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

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

Use Ido with Dired.

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

(l) IMenu

```
ID: orgmode:gcr:vela:F748CAFD-0235-4E34-8546-A9EC515759BB
```

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: SRC 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))
```

(m) Interactively DO Things

```
ID: orgmode:gcr:vela:2DB9FDA1-9950-4FEA-A33C-E8DDFDA9420E
```

IDO is used everywhere possible.
Access nearly every object available in this system from one place.
These configurations are performed in the correct order. Any attempt to refactor this Source-Block will break Ido in this system.

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

Make functions search-able.

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

Make URLs a first-class object.

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

(n) Font

`ID: orgmode:gcr:vela:8F7A007E-5CBA-4651-84D8-5874FF393EA6`

`unicode-fonts` configured the fallback as Symbola. On my box it isn't working though. Until then I'm using the manual configuration of a fallback and excluding the entire `unicode-fonts` source block.

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

Use Unicode-Font to provide as many Unicode fonts as possible.
Here are the Unicode fonts that provide nearly everything.

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

To test it run `view-hello-file` and `M-x list-charset-chars RET unicode-bmp RET`.
Perhaps educationally there is a character for bowel-movements: .

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

Activate font locking everywhere possible.

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

Visualize ASCII values as their most likely Unicode representation.

```
(use-package pretty-mode
  :ensure t
  :config
  (global-pretty-mode))
```

(o) Going to Objects

`ID: orgmode:gcr:vela:835D3E9D-3044-4385-9AB1-F0DF17045565`

Go to any object in the frame quickly.

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

(p) Grammar

`ID: orgmode:gcr:vela:95A4EF33-F83D-49ED-AC74-A29DA19524BC`

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)))
```

```
(use-package langtool
  :ensure t
  :init
  (setq langtool-language-tool-jar "/usr/local/Cellar/languagetool/3.2/libexec/languagetool-commandli
  (setq langtool-mother-tongue "en")
  (setq langtool-java-bin (concat (getenv "JAVA_HOME") "/bin/java")))
```

(q) Intellisense (Auto Completion)

```
ID: orgmode:gcr:vela:A7225C28-B8AE-4960-9E2A-64E6E8B58400
```

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

Auto-completion for .-separated words.

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

(r) Macros

```
ID: orgmode:gcr:vela:E32B41C2-C761-42F0-A9AE-F89A2A18439F
```

The macro recorder and Multiple-Cursors provide two ways to do the right thing in different situations. Be very thoughtful and allow every function.

```
(use-package multiple-cursors
  :ensure t)
```

(s) Mark and Region

```
ID: orgmode:gcr:vela:A3C2AF94-B834-4FD2-9B23-F64F618B31C3
```

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

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

(t) Minibuffer

```
ID: orgmode:gcr:vela:7A3C5EF1-BEF7-4007-86B1-78590CB62EB2
```

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)
```

(u) Mouse

```
ID: orgmode:gcr:vela:F3E75BDE-F853-488C-AF46-03B54C0A0919
```

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)
```

(v) Occur

```
ID: orgmode:gcr:vela:FA8195C5-30B7-44CF-8D0F-8FE2CE1CB3DA
```

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

(w) Popups

```
ID: orgmode:gcr:vela:E1E4E20E-F789-422B-B0B3-706BD8A842DF
```

Provide popup notifications.

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

(x) Projects

```
ID: orgmode:gcr:vela:B35103E2-0FE9-466C-9AB9-39EA28FADEDB
```

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

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

Notify Magit about every working copy that Projectile knows about.

```
(eval-after-load "projectile"
  '(progn (setq magit-repository-directories (mapcar (lambda (dir)
                                                       (substring dir 0 -1))
                                                     (remove-if-not (lambda (project)
                                                                      (file-directory-p (concat proje
                                                                    (projectile-relevant-known-projec

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

(y) Printing

```
ID: orgmode:gcr:vela:C6230D9E-8331-4092-8846-DB244455C922
```

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

(z) Register

ID: orgmode:gcr:vela:34801113-5002-4502-821E-248C6406395C

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

() Replacing

ID: orgmode:gcr:vela:B10A2279-4F34-4DA2-BB1A-491B82F2F6EA

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

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

() Save History of All Things

ID: orgmode:gcr:vela:31961F28-1913-4247-986A-273391C4A85D

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.

```
(setq savehist-save-minibuffer-history 1)
(setq savehist-additional-variables
      '(kill-ring
        search-ring
        regexp-search-ring))
(savehist-mode t)
```

() Searching

ID: orgmode:gcr:vela:960E2DE0-3F5A-40AB-A9BF-FF08A410EAB7

When searching don't use lax whitespace matching; and make it easy to toggle.

```
(setq isearch-lax-whitespace nil)
(setq isearch-regexp-lax-whitespace nil)
```

Make searches case-insensitive.

```
(setq-default case-fold-search t)
```

Other than ignoring case, search precisely for what was input.

```
(setq search-default-regexp-mode nil)
```

Easily search the filesystem using ag.

```
(use-package ag
  :ensure t
  :config
  (setq ag-highlight-search t)
  (setq ag-reuse-window nil)
  (setq ag-reuse-buffers t)
  (setq ag-arguments (-insert-at (- (length ag-arguments) 1) '"-i" ag-arguments)))
```

() Spell Checking

ID: orgmode:gcr:vela:902EAA81-4FC0-40A0-AE6D-D31C474B87E0

Ispell is simple and powerful.

i. Org-Mode

ID: orgmode:gcr:vela:72540881-8F99-4ED6-9FE4-7292A66B3089

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 '("- \\*.+" . ".*\\*: "))
```

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)
        ("SELECT_TAGS" nil)
        ("STARTUP" nil)
        ("TITLE" nil))
    (add-to-list
     'ispell-skip-region-alist
     (let ((special (concat "#[+]" (car it) ":")))
       (if (cadr it)
```

```
              (cons special "$")
            (list special))))))
```

() Sudo

ID: orgmode:gcr:vela:562F54F8-6E35-4DE1-9E9C-436B55CE83CE

Configure Sudo with Ido.

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

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

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

() Syntax Checking

ID: orgmode:gcr:vela:B4A8362E-B218-4353-AC4B-7059A686EA89

Perform syntactic analysis all the time.

```
(use-package flycheck
  :ensure t
  :config
  (add-hook ’after-init-hook #’global-flycheck-mode)
  :diminish flycheck-mode)
```

() TAB

ID: orgmode:gcr:vela:78E2BA2B-8289-422F-99DC-5E40DE928E68

Most modes in this system will never use TAB.

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

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

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

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

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

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

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

() Version Control

ID: orgmode:gcr:vela:F5E2718B-F54F-41C5-9CED-6E470CAC238D

Use VC for single files and Magit for multiple files.
The commit log editor uses With-Editor and Server modes. They are not diminished because they are infrequently used.

```
(use-package magit
  :ensure t
  :config
  (global-set-key (kbd "s-e") #’help/safb-help/magit-status))
```

Leave the VC message template empty.

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

Git ignore files are text files.

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

() Video

```
ID: orgmode:gcr:vela:4AFA5B76-4C9E-4728-8556-9163DA3D3CE7
```

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: orgmode:gcr:vela:5BBD948F-7239-457E-8BD9-710558C0E241
```

Make control characters easily visible.

```
(use-package whitespace
  :ensure t
  :config
  (setq whitespace-style '(trailing lines tab-mark))
  (setq whitespace-line-column help/column-width)
  (global-whitespace-mode t)
  :diminish whitespace-mode global-whitespace-mode)
```

() Word Wrap

```
ID: orgmode:gcr:vela:2156A7CE-297E-478F-AFF2-13CE64B3C5C3
```

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

4. Hacking

```
ID: orgmode:gcr:vela:B7CE60F5-5510-4358-8DD5-D42D9A2F4D9B
```

(a) Common Configurations

```
ID: orgmode:gcr:vela:BE02A401-AFF6-4B64-B7F3-589C69CA7099
```

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.

i. Text-Mode

`ID: orgmode:gcr:vela:7CFD11FB-F3D4-4272-9DBC-2A420884097C`

Set a default commenting prefix.

```
(setq-default comment-start "> ")
```

ii. Prog*-Mode Modes

`ID: orgmode:gcr:vela:6F71F8AF-4227-46D3-9BD8-2F86B5815B72`

- 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/hack-modes '(makefile-mode-hook ruby-mode-hook sh-mode-hook plantuml-mode-hook tex
    ```

  - LISP mode hooks.
    * Are hacking modes.

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

    * IELM mode hook.
      · Does one or two more things.

iii. Prog*-Mode Hook

`ID: orgmode:gcr:vela:FF132B34-B61B-4DAE-A0B9-E37E39B9BFCE`

A. Goal

`ID: orgmode:gcr:vela:E2C7121E-2E56-4A77-8347-2E7DFB73E9B3`

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

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

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

  ```
  (use-package smartparens-config
    :ensure smartparens
    :config
    (setq sp-show-pair-from-inside nil)
    :diminish smartparens-mode)
  ```

B. Implementation.

ID: orgmode:gcr:vela:61A981CB-1311-4F51-A264-D748FA34F1D3

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

iv. Wiring

ID: orgmode:gcr:vela:50304E30-682C-4C9A-9615-D6E61DAE533B

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

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

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

(b) Literate Programming

ID: orgmode:gcr:vela:6B83373B-8898-4AC0-B7F6-C42418CCE5E4

i. Emacs Lisp

ID: orgmode:gcr:vela:3AD91697-42DE-4555-9F49-B7D9F5E502D3

```
(setq initial-scratch-message nil)
(use-package lexbind-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-:") #'my-eval-expression)
  (local-set-key (kbd "#") #'endless/sharp))

(defun help/emacs-lisp-mode-hook-fn ()
  (interactive)
  (help/elisp-mode-local-bindings)
  (lexbind-mode)
  (eldoc-mode)
  (diminish 'eldoc-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))
```

A. Keybinding

ID: orgmode:gcr:vela:3A6B16EC-870A-4EFE-935A-C03F8DFB67BF

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

ii. Org-Mode

ID: orgmode:gcr:vela:EBDA3D1C-536F-4252-AE26-32A3FDF5326C

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)
```

Personal workflow:

TODO Zero progress.
IN-PROGRESS Forward movement.
HELD-BLOCKED Can't move forward until external agent completes.
HELD-FROZEN Won't move forward until I reinitialize it.
HELD-UNTIL Reinitialize when conditional.
REVIEW Work is complete and needs to be evaluated.
DONE Completed.

- Retire using `org-archive-subtree`.

```
(setq org-todo-keywords
      '((sequence
         "TODO"
         "IN-PROGRESS"
         "HELD-BLOCKED"
         "HELD-FROZEN"
         "HELD-UNTIL"
         "REVIEW"
         "DONE"
         )))
```

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

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

Use Ido completion in Org-Mode.

```
(setq org-completion-use-ido t)
(setq org-outline-path-complete-in-steps nil)
(setq org-completion-use-iswitchb nil)
```

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

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

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

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

Ask before execution of Emacs-Lisp.

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

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

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

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

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

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

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

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

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

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

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

Display emphasized text as you would in a WYSIWYG editor.

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

Use Unicode characters to visualize things like right arrow eg → . 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.

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

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

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

Realign tables automatically.

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

Always use Unicode checkboxes.

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

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

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

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

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

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

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

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

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

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

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

Never use the original version.

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

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

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

Bold.

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

Italic.

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

`Verbatim`.

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

`Code`.

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

~~Strike-Through~~.

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

Minimize Macro text.

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

Follow links without using the mouse or more.

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

A. Keybindings

ID: orgmode:gcr:vela:0AA3F69B-F5F1-48DA-B8F7-B7C92CD30DB1

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

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

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

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

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

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

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

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

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

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

B. Row 5

ID: orgmode:gcr:vela:B09BE660-B5D5-40CA-8952-D2DEAE20E7BD

```
(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)
```

C. Row 4

ID: orgmode:gcr:vela:1E49694B-6350-45E2-BE58-21EAAF09D4A2

```
(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)
```

D. Row 3

ID: orgmode:gcr:vela:D11109EE-AD92-4E26-988F-AF3CC70A2F69

```
(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)
```

E. Row 2

ID: orgmode:gcr:vela:CA925DDA-8EA8-47B0-AE9A-D73073CF51B7

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

F. Hydra

ID: orgmode:gcr:vela:46BCE65B-B8C9-49A0-A687-30D1330DB07D

```
(defhydra help/hydra/right-side/org-mode (:color blue
                                                 :hint nil)
  "
_1_ SHA-1-hash _2_ +imgs _3_ -imgs _4_ detangle _5_ id-create _6_ toggle-macro
_q_ ←/w-code _w_ tbletfld _e_ g2nmrst _r_ g2nms-b _t_ g2s-b/hd _y_ org-archive-subtree _u_ got
_a_ inshdrgs _s_ oblobigst            _h_ dksieb _k_ ob-check-src-blk
_c_ org-fill-para _b_ swtch2sessn _n_ n2sbtre"
  ;; Row 5
  ("1" org-babel-sha1-hash)
  ("2" org-display-inline-images)
  ("3" org-remove-inline-images)
  ("4" org-babel-detangle)
  ("5" org-id-get-create)
  ("6" help/org-toggle-macro-markers)
  ;; Row 4
  ("q" org-babel-switch-to-session-with-code)
  ("w" org-table-edit-field)
  ("e" org-babel-goto-named-result)
  ("r" org-babel-goto-named-src-block)
  ("t" org-babel-goto-src-block-head)
  ("y" org-archive-subtree)
  ("u" org-goto)
  ;; Row 3
  ("a" org-babel-insert-header-arg)
  ("s" org-babel-lob-ingest)
  ("h" org-babel-do-key-sequence-in-edit-buffer)
  ("k" org-babel-check-src-block)
  ;; Row 2
  ("c" org-fill-paragraph)
  ("b" org-babel-switch-to-session)
  ("n" org-narrow-to-subtree))
(key-chord-define-global "hh" #'help/hydra/right-side/org-mode/body)
```

Save all buffers before working with Exports.

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

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

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

Easily enter guillemots.

```
(key-chord-define org-mode-map "<<" (lambda () (interactive) (insert "«")))
(key-chord-define org-mode-map ">>" (lambda () (interactive) (insert "»")))
```

iii. Unicode

ID: orgmode:gcr:vela:BF8F326E-1B81-49AD-AD90-74E42E4611DB

Easily understand the current character.

```
(defun help/describe-char ()
  "Evaluate ‘describe-char’ and then ‘other-window’."
  (interactive)
  (call-interactively #'describe-char)
  (call-interactively #'other-window))
(global-set-key (kbd "H-d") #'help/describe-char)
```

Easily identify Unicode homoglyphs most often utilized by pranksters.
Good reference for Unicode character study and exploration.
Used intermittently so don't diminish it.

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

Easily escape and un-escape Unicode hex notation.

```
(use-package unicode-escape
  :ensure t
  :config
  (global-set-key (kbd "H-e") #'unicode-escape-region )
  (global-set-key (kbd "H-u") #'unicode-unescape-region))
```

Warn of UTF BOM bytes via skeeto.

> The UTF-8 representation of the BOM is the byte sequence oxEF,oxBB,oxBF.

Test using Hexl mode.

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

Attribution: URL 'https://www.reddit.com/r/emacs/comments/4tw0iz/can_i_have_a_warning_if_i_open_a
  (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)
```

iv. Dash

```
ID: orgmode:gcr:vela:20393F2C-F619-4E12-B862-1A4CCB97B742
```

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

(c) Applied Mathematics

```
ID: orgmode:gcr:vela:BE2550C9-231A-4824-BE6C-14231A971FE9
```

i. APL

```
ID: orgmode:gcr:vela:D150E5D5-BAFB-4C96-AD1E-C687C0216790
```

```
(setq gnu-apl-mode-map-prefix "H-")
(setq gnu-apl-interactive-mode-map-prefix "H-")
(use-package gnu-apl-mode
  :ensure t)
```

ii. Emacs Speaks Statistics (ESS)

```
ID: orgmode:gcr:vela:CB6305D8-DDBB-4865-8CAD-3648B31B76DB
```

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

Display object documentation.

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

Data viewing:

- **Never** rely upon on the REPL for data viewing

    - Will mix up exploratory code with data
        * Can't easily distinguish between code and data
        * Distracting you
        * Breaking your flow
- Sometimes
    - You end up somewhere
        * And the ess buffer cursor is at the top!
        * No problem, call ess-switch-to-end-of-ESS
- Make it easier to know what object values are.
    - ess-describe-object-at-point

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

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

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

Help buffers all belong in the same frame.

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

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

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

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

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

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

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

Use ido completion whenever possible.

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

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

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

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

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

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

Philosophy

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

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

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

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

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

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

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

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

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

iii. SAS (ESS)

```
ID: orgmode:gcr:vela:2442E555-0F82-48E6-96EA-2ABB5C9CC666
```

iv. R (ESS)

```
ID: orgmode:gcr:vela:1183D35B-77FC-4CFD-9BAA-4F7656AD8943
```

Enable a debugger.

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

Configure debugger search path per-project.

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

Easily nevigate errors.

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

Diminish watched variable font-size.

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

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

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

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

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

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

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

Rdired is another way to work with object:

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

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

Visualize data frames better:

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

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

inlineR

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

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

Documentation:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

R mode hook.

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

```
  (key-chord-define-local "<>" #'(lambda () (interactive) (insert " %<>% ")))
  (local-set-key (kbd "s-.") #'(lambda () (interactive) (insert " %>% ")))
  (r-autoyas-ess-activate)
  (help/turn-on-r-hide-show)
  (lambda () (add-hook 'ess-presend-filter-functions
                 (lambda ()
                   (warn
                    "ESS now supports a standard pre-send filter hook. Please update your config
  (ess-set-style 'RRR))

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

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

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

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

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

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

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

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

v. Scheme (LISP)

ID: orgmode:gcr:vela:D68EC042-5CBA-4547-B28A-CF878FB080C1

Handle all file extensions:

- Traditional.

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

- Racket.

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

- R6RS.

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

Use Geiser for Racket and Guile

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

Use Racket.

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

Enable Auto-Complete via Geiser.

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

vi. C

```
ID: orgmode:gcr:vela:6F4A6F8E-277C-45E2-B26E-51455C2CBC16
```

vii. Python

```
ID: orgmode:gcr:vela:F4577EEA-FB0F-4E7B-AF02-A4DEA74BB763
```

viii. YASnippet & Abbrev

```
ID: orgmode:gcr:vela:5C48A01F-D522-4AC9-A523-F8EE2E9EB384
```

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

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

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

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

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

ix. Structured Query Language (SQL)

```
ID: orgmode:gcr:vela:987C1C05-F880-4312-B902-5060208A3506
```

x. Structured Document Development

ID: orgmode:gcr:vela:206F4363-F4D6-4DDE-BB46-174D0F68FBB4

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
  (setf (cdr (rassoc 'php-mode auto-mode-alist)) 'web-mode)
  (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))
  (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-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))

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

(d) Publishing

ID: orgmode:gcr:vela:7D07C2F6-38B9-49BF-A584-B029CEED6141

i. TEX

```
ID: orgmode:gcr:vela:E2A1BFA2-0246-4376-9A33-E35A8DE2E5A3

(use-package tex-site
  :ensure auctex
  :config
  (eval-after-load "tex"
    '(define-key TeX-mode-map (kbd "C-c C-c") #'help/safb-TeX-command-master)))
```

Save style info. This doesn't control the buffer save.

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

Parse on load.

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

Parse on save.

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

Use PDFTeX to generate both DVI and PDF files.

```
(setq TeX-PDF-mode t)
(setq TeX-DVI-via-PDFTeX t)
```

Don't prompt every time you run `C-c C-c` about saving the file, instead, just save it.

```
(setq TeX-save-query nil)
```

Load LCO files with AucTeX.

```
(add-to-list 'auto-mode-alist '("\\.lco?\\'" . TeX-latex-mode))
```

Culture-dependent typographical results.

```
(add-to-list 'org-latex-packages-alist '("english" "babel" t))
```

Palatino friendly mathematics fonts.

```
(add-to-list 'org-latex-packages-alist '("osf" "mathpazo" t))
```

Small margins.

```
(add-to-list 'org-latex-packages-alist '("margin=0.5in" "geometry" nil))
```

Standard LaTeX class options.

```
(defvar help/ltx-cls-opt "paper=letter, fontsize=10pt, parskip")
```

Standard article class.

```
(eval-after-load "ox-latex"
  '(add-to-list 'org-latex-classes
                `("help-article"
                  ,(concat "\\documentclass["
                           help/ltx-cls-opt
                           "]{article}"))))

(setq org-latex-default-class "help-article")
```

Use “Smartquotes”.

```
(setq org-export-with-smart-quotes t)
```

ii. KOMA-Script

```
ID: orgmode:gcr:vela:BFF1402E-98D4-4F36-ACCB-B1E88A3EB2D9
```

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"
                    ,(concat "\\documentclass["
                             help/ltx-cls-opt
                             "]{scrlttr2}")))

     (setq org-koma-letter-default-class "help-letter")))
```

    There are two formats for the letters: heading-based and property-based.

    Set up my default LCO files.

```
(setq org-koma-letter-class-option-file "UScommercial9 KomaDefault")
```

iii. Texinfo

```
ID: orgmode:gcr:vela:F2C5CEB7-1252-4E6F-B192-C5D2D41A6D47
```

    Perhaps the first document typeset with Org-Texinfo.

iv. Pandoc

```
ID: orgmode:gcr:vela:1E4AB0D4-F92E-48E6-9313-278C27DED142
```

v. Markdown

```
ID: orgmode:gcr:vela:748319ED-9F02-4A4D-BEE8-E71C462663FC
```

Provide Github Flavored Markdown (GFM).

```
(use-package ox-gfm)
```

Program GFM.

```
(use-package markdown-mode
  :ensure t
  :config
  (add-to-list 'auto-mode-alist '("\\.md\\'" . gfm-mode))
  (setq markdown-coding-system "utf-8"))
```

vi. HTML

```
ID: orgmode:gcr:vela:84F673DF-2E6F-4BAA-8095-4A7586BB73FC

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

Load Htmlize for HTML export. Use in-line CSS.

```
(use-package htmlize
  :config
  (setq org-html-htmlize-output-type htmlize-output-type)
  (setq htmlize-output-type 'inline-css)
  (defvar help/htmlize-initial-fci-state nil
    "Variable to store the state of ‘fci-mode’ upon calling ‘htmlize-buffer’.

Attribution: URL ‘http://permalink.gmane.org/gmane.emacs.orgmode/98153’.")
  (defvar help/htmlize-initial-flyspell-state nil
    "Variable to store the state of ‘flyspell-mode’ upon calling ‘htmlize-buffer’.

Attribution: URL ‘http://permalink.gmane.org/gmane.emacs.orgmode/98153’.")

  (defun help/htmlize-before-hook-fn ()
    (when (fboundp 'fci-mode)
      (setq help/htmlize-initial-fci-state fci-mode)
      (when fci-mode
        (fci-mode -1))))
  (add-hook 'htmlize-before-hook #'help/htmlize-before-hook-fn)

  (defun help/htmlize-after-hook-fn ()
    (when (fboundp 'fci-mode)
      (when help/htmlize-initial-fci-state
        (fci-mode t))))
  (add-hook 'htmlize-after-hook #'help/htmlize-after-hook-fn))
```

vii. ASCII

```
ID: orgmode:gcr:vela:1F20F4EC-1D8B-402A-A0A9-504E733AEFDC

(setq org-ascii-text-width 80)
(setq org-ascii-global-margin 0)
```

viii. Beamer

ID: orgmode:gcr:vela:6B86302E-D3EC-413A-A844-9ACCAA23A056

Load Beamer for creating presentations.

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

ix. Screenwriting

ID: orgmode:gcr:vela:504560F0-00BD-4E63-8617-A20EF74C6906

```
(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)))
```

(e) DevOps

ID: orgmode:gcr:vela:8B78A8C3-E6B6-4722-9822-EF428E5DB823

i. Shell Script

ID: orgmode:gcr:vela:06D022B2-53ED-4042-8152-F383F5FE309E

ii. Make

ID: orgmode:gcr:vela:5F2B3343-631D-41C3-BE6B-D98548F77E07

iii. Vagrant

ID: orgmode:gcr:vela:A1335FE5-6DA7-4FE2-A77C-60032C76B40D

`ruby-mode` supports `Vagrantfile` OOTB.

iv. Apache

ID: orgmode:gcr:vela:885C43F5-1716-4547-8F62-03348E553E5E

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

v. SSH

ID: orgmode:gcr:vela:8F551D6F-5F17-4D82-BBA1-D71CABA92742

```
(use-package ssh-config-mode
  :ensure t
  :init
  (add-to-list 'auto-mode-alist '("/\\.ssh/config\\'"     . ssh-config-mode))
  (add-to-list 'auto-mode-alist '("/sshd?_config\\'"      . ssh-config-mode))
  (add-to-list 'auto-mode-alist '("/known_hosts\\'"       . ssh-known-hosts-mode))
  (add-to-list 'auto-mode-alist '("/authorized_keys2?\\'" . ssh-authorized-keys-mode))
  (add-hook 'ssh-config-mode-hook 'turn-on-font-lock))
```

(f) Diagram

ID: orgmode:gcr:vela:FA47D423-05B3-4911-9CEC-28A534E49428

i. Artist

ID: orgmode:gcr:vela:F920A833-60D7-41C2-9363-EA2A8BD19009

```
(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: orgmode:gcr:vela:FCC8A8F8-A967-4981-9260-CFF60CC56494

```
(defconst help/ditaa-jar "/usr/local/Cellar/ditaa/0.9/libexec/ditaa0_9.jar")
(setq org-ditaa-jar-path help/ditaa-jar)
```

iii. Graphviz

ID: orgmode:gcr:vela:A84665A3-4A2D-4040-926A-17159A6D4647

```
(use-package graphviz-dot-mode
  :ensure t
  :config
  (setf (cdr (assoc "dot" org-src-lang-modes)) 'graphviz-dot))
```

iv. PlantUML

ID: orgmode:gcr:vela:FCC259F8-0858-4778-B97F-07D2B21012F5

```
(use-package plantuml-mode
  :ensure t
  :init
  (defconst help/plantuml-jar "/usr/local/Cellar/plantuml/8031/plantuml.8031.jar")
  (setq plantuml-jar-path help/plantuml-jar)
  :config
  (eval-after-load "ob-plantuml"
    (setq org-plantuml-jar-path help/plantuml-jar)))
```

(g) Computer-aided design

ID: orgmode:gcr:vela:36EB1867-0EDB-4C22-871D-7D730CF4DF53

i. OpenSCAD

ID: orgmode:gcr:vela:090201D4-6799-4DC2-B353-C574F3C8B918

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

5. Quiet and Pleasant Appearance

ID: orgmode:gcr:vela:197B7B84-5090-47AE-9180-F8F606D0012F

Configure EMACS to personal-taste for "noise" and "form".

(a) Key Press

ID: orgmode:gcr:vela:A2E555B7-678D-4F38-A42D-D2E72A056B8D

Make key-presses sound like the ground-breaking Selectric typewriter.

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

(b) Line Number

ID: orgmode:gcr:vela:4D367462-1C7B-4110-B7D1-E973D386B4E1

The nlinum gutter should not "shift" as it transitions between line numbers of different magnitudes. For example going from line 99 to 100 will shift the buffer contents by one character. That is distracting and interrupts the flow.

- Switched to `nlinum` from `linum` because of slowness with this file when headings are collapsed

Most files will be less than 100,000 lines.

```
(use-package nlinum
  :ensure t
  :config
  (setq nlinum-format "%05d"))
```

(c) Buffer

```
ID: orgmode:gcr:vela:61586A23-B774-4436-B916-348453EEA3DD
```

Never automatically convert the end of the line character. For most of us this is between UNIX and DOS.

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

Give buffers backed by identically named files distinguishable names.

```
(use-package uniquify)
(setq uniquify-buffer-name-style 'forward)
```

Don't use audible bells, use visual bells.

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

Highlight s-expressions.

```
(setq blink-matching-paren nil)
(show-paren-mode)
(setq show-paren-delay 0)
(setq show-paren-style 'expression)
```

The cursor should not blink and distract you. On a graphic display make the cursor a box and stretch it as wide as the character below it.

```
(blink-cursor-mode 0)
(help/on-gui
 (setq-default cursor-type 'box)
 (setq x-stretch-cursor 1))
```

EMACS used UTF-8 by default. Make copying and pasting easier.

```
(prefer-coding-system 'utf-8)
(help/on-gui
 (setq x-select-request-type '(UTF8_STRING COMPOUND_TEXT TEXT STRING))
 (help/on-windows
  (set-clipboard-coding-system 'utf-16le-dos)))
```

Make it very easy to see the line with the cursor.

```
(global-hl-line-mode t)
```

Make it very easy to input special-characters using TEX coding.

```
(setq default-input-method 'TeX)
```

(d) Color Theme

```
ID: orgmode:gcr:vela:057BBA77-4662-4F7B-B47A-CB1E79A1618B
```

Solarized Theme

- ~1,000 Faces Defined
- 47,869 Downloads
- A distinct fringe provides a definition of space.
- The modeline is always at the bottom and doesn't need differentiation.
- Minimize bold and italic faces.
- Minimize fringe indicators.

```
(use-package solarized-theme
  :ensure t
  :config
  (setq solarized-distinct-fringe-background t)
  (setq solarized-high-contrast-mode-line nil)
  (setq solarized-use-less-bold t)
  (setq solarized-use-more-italic nil)
  (setq solarized-emphasize-indicators nil)
  (load-theme 'solarized-dark t))
```

(e) Comint

```
ID: orgmode:gcr:vela:B4E17CF5-5542-4526-ADEE-D5EC3DB9131F
```

`comint-mode` is only maybe the second most important thing for making Emacs really, really special.

```
(setq comint-scroll-to-bottom-on-input 'this)
(setq comint-scroll-to-bottom-on-output 'others)
(setq comint-move-point-for-output 'others)
(setq comint-show-maximum-output t)
(setq comint-scroll-show-maximum-output t)
(setq comint-move-point-for-output t)
```

This configuration had been working fine for a long time. The intent was for it to be crystal clear that the prompt line in comint buffers would be read only. This turned out to be a mistake; though I am not sure why, when, or how it became a mistake. Nonetheless, this should be left alone. The way the issue here manifested was that all R buffers opened by `ess` were 100% read only which obviously is a **big issue** if you actually want to use! ROFL

```
(setq comint-prompt-read-only nil)
```

(f) Font

```
ID: orgmode:gcr:vela:EC675F88-89C0-4A5A-B910-843F28C0F90F
```

The best programming font is Deja Vu Sans Mono because it sans-serif and support a lot of Unicode characters. Set it to a good default for an 80 character wide buffer and make it easy to adjust it.

```
(help/on-gui
 (defvar help/font-size-current 10 "The preferred font size.")
 (help/on-osx (setq help/font-size-current 17))
 (help/on-windows (setq help/font-size-current 13))
 (defconst help/font-size-ideal help/font-size-current "The ideal font for this system.")
 (defconst help/font-base "DejaVu Sans Mono" "The preferred font name.")
 (defun help/font-ok-p ()
   "Is the configured font valid?"
   (interactive)
   (member help/font-base (font-family-list)))
 (defun help/font-name ()
   "Compute the font name and size string."
   (interactive)
   (let* ((size (number-to-string help/font-size-current))
          (name (concat help/font-base "-" size)))
     name))
 (defun help/update-font ()
   "Updates the current font given configuration values."
   (interactive)
   (if (help/font-ok-p)
       (progn
         (message "%s : Font Set" (help/font-name))
         (set-frame-font (help/font-name)))
     (message (concat "Your preferred font is not available: " help/font-base))))
 (defun help/font-size-reset ()
   "Restore the ideal font size."
   (interactive)
   (setq help/font-size-current help/font-size-ideal)
   (help/update-font))
 (help/update-font))
```

(g) Frame

```
ID: orgmode:gcr:vela:B3F90439-D007-42EE-95FC-E93BBA827325
```

The scroll-bars are helpful for new users.

```
(scroll-bar-mode 0)
```

The tool-bar is helpful for new users. Isn't the argument funny?

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

(h) Pointer

```
ID: orgmode:gcr:vela:2A437D32-2944-41B3-AD8F-438ABBD4E0CF
```

Hide the pointer when typing.

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

(i) Version Control

```
ID: orgmode:gcr:vela:99337D9E-DBC9-4673-B814-EBC94C044E3E
```

Provide VC file status indicators.

```
(use-package diff-hl
  :ensure t)
```

Ediff split frame horizontally.

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

(j) Window

```
ID: orgmode:gcr:vela:9A848D65-DE56-4F95-A84D-CAE74781CD25
```

Menu bars make EMACS more accessible to non-EMACS users.

```
(menu-bar-mode t)
```

Easily return to previous configurations. 2-4 windows are easily managed by hand.

```
(winner-mode t)
```

Frequently use between 1 and 3 windows.

```
(defun help/1-window ()
  "Work with this buffer in 1 window."
  (interactive)
  (delete-other-windows))

(defun help/2-window ()
  "Work with this buffer in 2 windows."
  (interactive)
  (delete-other-windows)
  (split-window-below)
  (balance-windows))

(defun help/3-window ()
  "Work with this buffer in 3 windows."
  (interactive)
  (delete-other-windows)
  (split-window-below)
  (split-window-below)
  (balance-windows))
```

Most of the time when opening other buffers, go to them. This configuration appears for different modes in this system. Modes distributed with Emacs are configured here.

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

6. Piano Lessons

```
ID: orgmode:gcr:vela:31274432-4BA2-4B03-8DDB-E590C245244D
```

(a) A Fine Cup of EMACS

```
ID: orgmode:gcr:vela:01EEEC32-91D4-4DDC-A100-52CE571558DC
```

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: orgmode:gcr:vela:A50A19BB-1DE7-48C1-AEE4-03D1E88E887C

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: orgmode:gcr:vela:7E76A660-7828-4747-90DE-84BD293CD4E7

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: orgmode:gcr:vela:D8ADD840-9E9E-4A2E-B085-245C7BFA5F48

i. Keyboard Layout & Operation

ID: orgmode:gcr:vela:D8420B75-E4B9-4DB1-885E-D5290FE9A3EA

- 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: orgmode:gcr:vela:60A17CE8-C905-4443-90A2-10D2C12F23AF`

You operate within a cognitive landscape. Every moment you are in a single `place`. While residing in each `place` you perform logically related `activities`. Activities facilitate logical actions like modification within that `place`. Modifications are performed objects. Objects include things like the contents of a buffer, contents of memory, or the file that backs a buffer. While performing those `activities` there is a logical sense of "flow". That should never be interrupted. Usually an interruption occurs when you are going to go to a new `place`. The `distance` between `places` is measured in the similarity between the `actions` that you find there. As you develop these ides it will be obvious where key-bindings should go

iii. Key-Bindings Take You to Places to Perform Activities

`ID: orgmode:gcr:vela:E765C8BB-ECC3-4791-A287-83B6DED2F6C3`

OOTB you wil be visiting many `places` and performing many `activities`. EMACS comes with a good configuration that minimizes `distance`. This isn't worth changing. You can use EMACS for a lifetime without ever having to customize any of the key-bindings. This is what lets anyone use your system. This is what lets you use the system with `-Q` when you break it. You need to decide if you every want to alter the default configuration. This system does not want to. It wants to keep EMACS true to EMACS and your hands happy. To satisfy those goals the following practices were defined.

- 99.999% of the time never bind to the `C` or `M` name-space.
  - They are for system key-bindings. You can break them. Don't.
  - In theory `C-c` is the "user name-space" but packages stomp on this all of the time anyway so don't use it.
  - Some bindings are just too valuable to pass up:
    * `C-;`
      · Your hands are in the home position already.
    * Every modifier key with return.
- Never bind to `F` keys.
  - They are a painful stretch on most keyboards.
  - Some require a lone Pinky with is worse.
  - Most operating systems bind actions to them OOTB anyway.
  - EMACS comes with key-bindings OOTB.
- Don't try to set up a Hyper-key.
- Use shift as a name-space expansion vehicle.
  - Shift doubles every name-space in which you use it.
  - Use cautiously, not every name-space vehicle supports it.

- About the `s` (super) name-space.
  - In theory it is the best place for user-defined key-bindings because EMACS OOTB uses `C` and `M` completely leaving `s` mostly open.
  - In practice `C` and `M` are running out of space because there are a lot of new packages being added to EMACS. Most new packages are binding key in the `s` name-space.
  - This system reserves `s` completely for Sysop.

These practices say nothing about the `places` or `activities` that you choose to peform. The practices only look at the key-binding configuration. There are a limited number of keys on a keyboard and there are physical limitations on your hands. Along with the previous assuptions it may look like there are less. Fortunately it just looks that way and it isn't true. There are a lot of powerful ways to "go places" with EMACS. The next heading contains my attempt.

iv. How to Get There Pleasantly and Quickly

`ID: orgmode:gcr:vela:38026C1B-44D4-47EF-90D2-239876F7F31C`

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: orgmode:gcr:vela:0A491DA9-212E-4F01-8C08-EA09E9B6D82C
```

- `Actions` here are for the `place` inside of the buffer itself. They are for immediate acting upon the contents of the buffer. They are logically related, used frequently, and likely to be memorized.
- When you come here, you are likely to stay for some time before getting out.
- Only use single key bindings; anything more may be a new logical name-space and may use a Hydra.
- Split the home sides of the keyboard in half.
- The left side of the keyboard should be use for operations common to every mode.
  - For example `goto-line` and `ispell`.
  - It has 15 bindings available; 20 if you use 1-5. 40 if you shift them.

- The right side of the keyboard should be used operations specific to the current major mode.
  - For example in Org-Mode navigating between source-blocks and evaluating them.
  - It has 19 bindings available; 26 if you use 6-=. 52 if you shift them.

For example, in Org-Mode:

- I traverse the entire document very quickly with `org-babel-previous-src-block` and and `org-babel-next-src-block.`
- I execute source-blocks.
- I edit source-blocks.

Every activity is related to reading, modifying, executing, and tangling code.

B. `Key-Chord`

`ID: orgmode:gcr:vela:76C81A53-52BA-47C4-A8FF-651E10A4620F`

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: orgmode:gcr:vela:B198918B-F8C8-4036-A41B-237BDA793EC0`

: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.
  - `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: orgmode:gcr:vela:9CF95F30-5872-40F4-AF00-BDB82E3D7399`

: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: orgmode:gcr:vela:0410F66C-40F4-46A1-9E69-56658EA815A9
```

- Sometimes you want to do something in a `place` but you aren't sure what and you aren't sure where you will go next from there. For example you might want to perform an Org-Mode `action` that is important but you don't really use much. For example exporting to HTML might not be common for you but you value.
- `Hydras` can be used for very related actions too. The difference between the `s` name-space is the distance between them and where you are now. In the `s` namespace you go there very quickly. For `Hydras` sometimes you can get the fast and sometimes more slowly. They are complementary to every name-space.
- `SHIFT` doubles your key-space.
- Use `C-g` to exit the Hydra.

For example, in Org-Mode I am still learning about functions and haven't used them much and forget their names. It is faster to put them in a Hydra. If they get used a lot, I will add them to `s`.

v. Building Your Own Keyboard

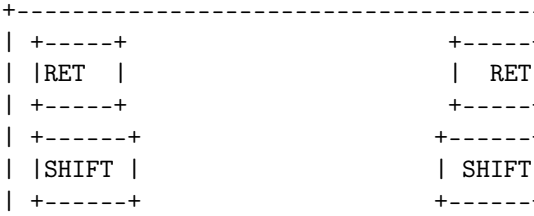
```
ID: orgmode:gcr:vela:A4257881-BD92-4826-8B0F-74B9557442F9
```

As your mastery of EMACS grows so too will your desire to build your own keyboard. It is natural. As you explore various dimensions of expression you will have a lot of fun. You will act more quickly and skillfully. Even with the goals of this system in place the desire grows.

3D printing is one area worth exploring. A lot of EMACS users design and print their own custom keyboards. That looks very fun. Ukulele is softer way to explore your keyboard. Reading its user manual is important. It contains ideas about stack-able-environments for bindings. You may use Ukulele or Hydras to do the same thing. Karabiner is a nice way to re-map your keys. It's easiest adjustment is to make return act as return when pressed alone and as control when pressed with another key. That introduces a symmetry to your keyboard which can be helpful. All of those dimensions are worth exploring.

When I explored them I felt that they led me further away from the majority of users. Every time explored a different key-mapping (not key-binding) it reduced accessibility for new users. Each time I tried to work around that hiccup. The last pursuit was `;` and space.

It would be great to set up your keyboard with the meta keys on the bottom like this:



```
|        +-+ +-+ +-----+ +-+ +-+        |
|        |s| |M| |C/spc| |M| |s|        |
|        +-+ +-+ +-----+ +-+ +-+        |
|                                       |
+---------------------------------------+
```

Karabiner was too slow for my typing speed though. It happens. It seemed like a minimal change to use Ukelele to:

- Make space send `C`
- Make `;` send space
- Make `'` a dead key
  - In it's dead key state make

    `* ; → ;`
    `* : → :`
    `* ' → '`
    `* " → "`

The trouble is that it violates the POLA. Therefore, I left it alone and stuck with a simple "Get C on both sides".

That has worked out very well. It is very easy to do on every operating system. It holds true to the values of this system. When you develop an idea of `places` and how often you go there the key-mapping becomes more obvious. Make it easy to get to key-bindings that take you to familiar places. For this system it is the home keys, `s`, and `key-chord`. Make those keys more easily accessible. `C` and `M` often have symmetric-definitions. `s` and `SHIFT` also often have symmetric definitions in this system (mostly through Key-Chords). Therefore those key-mappings are kept close together

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

(e) Take a Sip

`ID: orgmode:gcr:vela:F42A8A6B-C690-4715-90CB-2207C47C6808`

i. Left Side

`ID: orgmode:gcr:vela:22246934-BE44-4D99-942C-A6DAB4506D65`

A. Row 5

`ID: orgmode:gcr:vela:C00A4E41-0801-4696-86E6-5A1CE1EBB189`

```
(global-set-key (kbd "s-4") #'mc/mark-next-like-this)
(global-set-key (kbd "s-3") #'mc/mark-previous-like-this)
(global-set-key (kbd "s-2") #'mc/mark-all-like-this)
(global-set-key (kbd "s-1") #'mc/edit-lines)
(global-set-key (kbd "s--") #'decrement-integer-at-point)
(global-set-key (kbd "s-+") #'increment-integer-at-point)
```

B. Row 4

```
ID: orgmode:gcr:vela:8F467832-8FC3-42B5-8978-8CF2C1454D5B

(global-set-key (kbd "s-w") #'imenu)
(key-chord-define-global "1o" #'help/1-window)
(key-chord-define-global "2o" #'help/2-window)
(key-chord-define-global "3o" #'help/3-window)
(global-set-key (kbd "s-q") #'kill-buffer)
(global-set-key (kbd "s-Q") #'kill-this-buffer)
(global-set-key (kbd "H-i") #'insert-char)
(global-set-key (kbd "H-p") #'help/insert-datestamp)
(global-set-key (kbd "H-P") #'help/insert-timestamp)
```

C. Row 3

```
ID: orgmode:gcr:vela:6DCD321F-6FDA-4983-9C7C-265D23D1AC4F

(global-set-key (kbd "s-a") #'help/safb-switch-to-previous-buffer)
(global-set-key (kbd "s-d") #'er/expand-region)

(defhydra help/hydra/left-side/global (:color blue
                                            :hint nil)
  "
_1_ reset-font _2_ -font  _3_ +font _4_ ellipsis _5_ UUID _6_ bfr-cdng-systm _7_ grade-level
_q_ apropos _w_ widen _t_ unicode-troll-stopper-mode _u_ ucs-insert  _i_ scrollUp _I_ prevL
_a_ ag  _s_ help/toggle-mac-right-option-modifier _S_ help/toggle-mac-function-modifier _d_
_x_ delete-indentation _c_ fill-paragraph _b_ erase-buffer  _m_ imenu-list _M_ Marked 2 Vie
  ("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)
  ("7" writegood-grade-level)
  ("8" writegood-reading-ease)
  ("9" help/insert-checkmark)
  ("a" ag)
  ("s" help/toggle-mac-right-option-modifier)
  ("S" help/toggle-mac-function-modifier)
  ("x" delete-indentation)
  ("q" hydra-apropos/body)
  ("w" widen)
  ("t" unicode-troll-stopper-mode)
  ("j" org-babel-tangle-jump-to-org)
  ("u" ucs-insert)
  ("i" scroll-down-command :exit nil)
  ("d" dash-at-point)
  ("k" scroll-up-command :exit nil)
  ("I" previous-logical-line :exit nil)
  ("K" next-logical-line :exit nil)
  ("m" imenu-list-minor-mode)
  ("M" help/preview-buffer-file-in-marked-2)
  (";" isearch-toggle-lax-whitespace)
  ("o" toggle-debug-on-error)
  ("p" anzu-query-replace)
  ("[" backward-page :exit nil)
  ("]" forward-page :exit nil)
  ("c" fill-paragraph )
  ("b" erase-buffer))
```

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

Attribution.

```
(defhydra hydra-apropos (:color blue
                                :hint nil)
  "
_a_propos        _c_ommand
_d_ocumentation  _l_ibrary
_v_ariable       _u_ser-option
^ ^          valu_e_"
  ("a" apropos)
  ("d" apropos-documentation)
  ("v" apropos-variable)
  ("c" apropos-command)
  ("l" apropos-library)
  ("u" apropos-user-option)
  ("e" apropos-value))
```

D. Row 2

```
ID: orgmode:gcr:vela:9E95D130-D1EC-445B-9028-24DFA5CCB28A
```

```
(global-set-key (kbd "s-v") #'smex)
(global-set-key (kbd "C-x C-c") #'help/safb-save-buffers-kill-terminal)
(global-set-key (kbd "s-x") #'ido-find-file)
(global-set-key (kbd "s-c") #'ido-switch-buffer)
```

E. Row 1

```
ID: orgmode:gcr:vela:4CDDC2CE-646A-4D8B-B5D3-2588FBEFF650
```

F. Unsorted

```
ID: orgmode:gcr:vela:AD2164B2-CB66-48AD-B367-4E0CC406B022
```

VC activities.

```
(define-prefix-command 'help/vc-map)
(global-set-key (kbd "s-r") #'help/vc-map)
(define-key help/vc-map "e" #'help/safb-vc-ediff)
(define-key help/vc-map "d" #'help/safb-vc-diff)
(define-key help/vc-map "u" #'help/safb-vc-revert)
(global-set-key (kbd "s-f") #'help/safb-help/vc-next-action)
```

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

```
(global-set-key (kbd "C-a") #'beginning-of-line-dwim)
```

Occur has 3 cases. I like to use it to explore the unknown.

```
(global-set-key (kbd "H-o") #'help/occur-dwim)
```

Simpler buffer movement.

```
(global-set-key (kbd "s-g") #'help/safb-other-window)
```

Toggle utility buffers (“logical F” key, so left side; “logical J” key on right).

```
(key-chord-define-global "f9" #'help/util-cycle)
```

Hide and show code blocks.

```
(global-set-key (kbd "s-b") #'hs-toggle-hiding)
```

ii. Left & Right Side

`ID: orgmode:gcr:vela:FA2BFDC9-5242-4547-A8A5-6DECC8ED1C1B`

Exploratory programming in EMACS.

- Don't use "dn" for "describe-function" because of "and"-words
- `d.` typically ends sentences

```
(key-chord-define-global "d8" #'describe-function)
(key-chord-define-global "d9" #'describe-variable)
```

Don't use "qi"; "unique".

```
(global-set-key (kbd "s-‘") #'help/comment-or-uncomment)
```

Make `ispell` accessible.

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

Use the default Langtool bindings.

```
(define-prefix-command 'help/langtool-map)
(key-chord-define-global "qk" #'help/langtool-map)
(define-key help/langtool-map "c" #'langtool-check-buffer)
(define-key help/langtool-map "C" #'langtool-correct-buffer)
(define-key help/langtool-map "j" #'langtool-goto-previous-error)
(define-key help/langtool-map "k" #'langtool-show-message-at-point)
(define-key help/langtool-map "l" #'langtool-goto-next-error)
(define-key help/langtool-map "q" #'langtool-check-done)
```

Via comments:

```
(key-chord-define-global "TH" (lambda () (interactive) (insert "Th")))
```

Easily kill buffers. Can't use "df" because of "PDF".
Can't use "fr" because of "fright" and "France".
Can't use "dc" because of `cd (change directory)`.
Can't use "gr" brecause of "Grant" and "Great.
Go to a word.

```
(key-chord-define-global "fj" #'avy-goto-word-1)
(key-chord-define-global "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: orgmode:gcr:vela:16040443-9099-42C1-A7FB-90C0DDC9F8EE
```

Try to reserve the right side for mode-specific activities.

iv. Exceptions

```
ID: orgmode:gcr:vela:EBBB727C-6110-4F7B-A2DC-45E9833EBEFE
```

Return.
Do smart new line inside, indenting given the mode.

```
(help/not-on-gui (global-set-key (kbd "s-RET") #'help/smart-open-line))
(help/on-gui (global-set-key (kbd "s-<return>") #'help/smart-open-line))
```

Scroll the whole buffer by one line keeping the cursor with it.

```
(global-set-key (kbd "M-n") (kbd "C-u 1 C-v"))
(global-set-key (kbd "M-p") (kbd "C-u 1 M-v"))
```

Use a nicer `eval-expression` approach.

```
(global-set-key (kbd "s-:") #'my-eval-expression)
```

```
(global-set-key (kbd "s-C-n") #'next-line)
(global-set-key (kbd "C-n") #'next-logical-line)
(global-set-key (kbd "s-C-p") #'previous-line)
(global-set-key (kbd "C-p") #'previous-logical-line)
```

Ansu.

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