Contents

1 packages

```
;;; Commentary:
;;; this is only some of the packags.
;;; Some other required packages are loaded in the following postload files.
;;; using prelude-require-package ensures that the packges are loaded
```

```
;;; at the time required, if necessary.

;;; Code:

(require 'prelude-packages)

;; also load prelude-required packages manually.
;; this loads packages which are not (auto-) loaded otherwise.
;; apparently prelude-required packages will be available after restarting emacs twic
(defun prelude-load-require-package (package)
  "Install PACKAGE unless already installed."
  (unless (memq package prelude-packages)
    (add-to-list 'prelude-packages package))
  (unless (package-installed-p package)
    (package-install package))
  (require package))

(defun prelude-load-require-packages (packages)
  "Ensure PACKAGES are installed.
Missing packages are installed automatically."
  (mapc #'prelude-load-require-package packages))

(prelude-load-require-packages
 '(
   moe-theme
   powerline
   multi-term))
```

2 customize location set coordinates for corfu

```
(setq calendar-latitude [39 37 north])
(setq calendar-longitude [19 54 east])
```

3 moe theme powerline delimiter faces

```
;;; Commentary:
;;; moe-theme
```

```
;;; Code:
(prelude-load-require-package 'moe-theme)
(setq powerline-moe-theme t)
(moe-dark)
(require'powerline)
(powerline-default-theme)
(custom-set-faces
 '(info-title-3 ((t (:inherit info-title-4 :foreground "white" :height 1.2))))
 '(info-title-4 ((t (:inherit info-title-4 :foreground "red"))))
 '(font-lock-variable-name-face ((t
                                  (:foreground "turquoise2"))))
 '(font-lock-comment-delimiter-face ((t
                           (:slant italic :foreground "SeaGreen1"))))
 '(font-lock-comment-face ((t
                           (:slant italic :foreground "coral1"))))
 '(mode-line ((t (
                  :background "DarkCyan"
                              :foreground "tomato"
                              :box (:line-width 1 :color "turquoise3")
                              :weight light :height 118 :family "Monospace"))))
 '(aw-leading-char-face ((t
                         (:weight bold :foreground "white" :background "red" :inherit
                                  (aw-mode-line-face))))))
```

4 Theming+Faces

dark theme hl colors and inconsolata font

```
;;; Commentary:

;;; Default font: Inconsolata.
;;; Preferred face colors for region, hl-line, hl-sexp-face.
;;; Work both for zenburn and moe-dark themes.

;;; Code:

(custom-set-faces
 '(default ((t (:family "Inconsolata" :foundry "nil"
                         :width normal :height 120
                         :weight normal :slant normal :underline nil :overline nil
```

```
                        :strike-through nil :box nil
                        :inverse-video nil :foreground "#DCDCCC"
                        :background "#3F3F3F" :stipple nil :inherit nil))))
 '(helm-selection ((t
                    (:underline nil :background "MediumOrchid1" :foreground "white"))
 '(region ((t (:background "thistle4" :foreground nil))))
 ;; following 2 are for moe-light:
 ;; '(hl-line ((t (:background "DarkSlateGray4" :foreground "yellow1"))))
 '(hl-line ((t (:background "red4"))))
 ;; '(hl-sexp-face ((t (:background "linen" :foreground "VioletRed4"))))
 '(hl-sexp-face ((t (:background "gray10"))))
 )

(powerline-moe-theme)
(moe-dark)
```

5 Cursor Beep

```
;;; Commentary:
;;; basic theming, cursor style.

;;; Code:

(setq cursor-type 'bar)   ;; show cursor as thin vertical bar.
(blink-cursor-mode 1)     ;; turn on cursor blinking

(setq visible-bell nil)   ;; instead of ringing a bell ...
(setq ring-bell-function (lambda () ;; .. invert the mode line colors for 1 second
                           (invert-face 'mode-line)
                           (run-with-timer 1 nil 'invert-face 'mode-line)))

(set-cursor-color "tomato")
```

6 Fullscreen toggle and native use

```
;;; Commentary:
;;; enable native fullscreen mode and define key for toggling.
```

```
;;; Code:
(setq ns-use-native-fullscreen nil)
(global-set-key (kbd "H-t") 'toggle-frame-fullscreen)
```

7 load-some-useful-package-avy-jump-etc

```
(prelude-load-require-packages '(avy-zap avy-menu auto-async-byte-compile anzu ace-wi
```

8 god and evil modes

```
(global-set-key (kbd "<f5>") 'god-mode)
(global-set-key (kbd "<f6>") 'evil-mode)
```

9 whitespace and visual line fixes

```
;;; Commentary:
;;; turn off whitespace and turn on visual line modes,
;;; for these main modes:
;;; js, css, web, html, markdown

(defun whitespace-off ()
  "Make turning whitespace mode off a command callable from key."
  (interactive)
  (whitespace-mode -1))

(add-hook 'markdown-mode-hook 'whitespace-off)
(add-hook 'css-mode-hook 'whitespace-off)
(add-hook 'html-mode-hook 'whitespace-off)
(add-hook 'web-mode-hook 'whitespace-off)
(add-hook 'js-mode-hook 'whitespace-off)

(add-hook 'markdown-mode-hook 'visual-line-mode)
(add-hook 'css-mode-hook 'visual-line-mode)
(add-hook 'html-mode-hook 'visual-line-mode)
(add-hook 'web-mode-hook 'visual-line-mode)
(add-hook 'js-mode-hook 'visual-line-mode)
```

10 multiple-cursors

```
(prelude-load-require-packages '(multiple-cursors mc-extras ace-mc))

;; ace-mc
(global-set-key (kbd "C-c )") 'ace-mc-add-multiple-cursors)
(global-set-key (kbd "C-M-)") 'ace-mc-add-single-cursor)

;; multiple-cursors

(global-set-key (kbd "C-S-c C-S-c") 'mc/edit-lines)
(global-set-key (kbd "C->") 'mc/mark-next-like-this)
(global-set-key (kbd "C-<") 'mc/mark-previous-like-this)
(global-set-key (kbd "C-c C-<") 'mc/mark-all-like-this)
;; mc-extras

(define-key mc/keymap (kbd "C-. C-d") 'mc/remove-current-cursor)
(define-key mc/keymap (kbd "C-. d")   'mc/remove-duplicated-cursors)

(define-key mc/keymap (kbd "C-. C-.") 'mc/freeze-fake-cursors-dwim)

(define-key mc/keymap (kbd "C-. =")   'mc/compare-chars)

;; Emacs 24.4+ comes with rectangle-mark-mode.
(define-key rectangle-mark-mode-map (kbd "C-. C-,")
  'mc/rect-rectangle-to-multiple-cursors)

(define-key cua--rectangle-keymap   (kbd "C-. C-,")
  'mc/cua-rectangle-to-multiple-cursors)

(mc/cua-rectangle-setup)
```

11 hl-faces

```
(custom-set-faces
 '(hl-line ((t (:background "gray0"))))
 '(hl-sexp-face ((t (:background "gray10")))))
```

12 helm-swoop

```
(global-ace-isearch-mode +1)
(prelude-load-require-package 'helm-swoop)
(global-set-key (kbd "C-S-s") 'helm-swoop)
```

13 bookmark extensions

```
;;; Commentary:
;;; Better editing of bookmarks perhaps the advantage

;;; Code:
(prelude-load-require-packages '(bm autobookmarks bookmark+))
```

14 EmacsLispModes and whitespace off

```
;;; Commentary:
;;; useful minor modes for emacs-lisp

;;; Code:
(prelude-load-require-packages '(smartparens cl litable icicles hl-sexp))

;;; note: smartparens is preferable to paredit.
(require 'smartparens-config)

(add-hook 'emacs-lisp-mode-hook 'hl-sexp-mode)
(add-hook 'emacs-lisp-mode-hook 'hs-minor-mode)
(global-set-key (kbd "H-l h") 'hs-hide-level)
(global-set-key (kbd "H-l s") 'hs-show-all)

(add-hook 'emacs-lisp-mode-hook 'rainbow-delimiters-mode)
(add-hook 'emacs-lisp-mode-hook 'whitespace-off)

(add-hook 'emacs-lisp-mode-hook 'smartparens-mode)
;; (add-hook 'emacs-lisp-mode-hook 'turn-on-whitespace-mode)
(add-hook 'emacs-lisp-mode-hook 'auto-complete-mode)
(add-hook 'emacs-lisp-mode-hook 'turn-on-eldoc-mode)
```

```
;; H-C-i:
(define-key emacs-lisp-mode-map (kbd "H-i") 'icicle-imenu-command)
```

15 untangle tangle export babel from master file

```
;;; Commentary:
;;; org-el-untangle:
;;; import muliple el files from one folder into one org mode file.
;;; org-el-tangle-sections
;;; export each sections' emacs-lisp block to a separate file.

;;; Code:

(defun org-el-import-all-files (directory)
  "Import muliple el files from one folder into one org mode file."
  (interactive "D")
  (let
      ((filename (concat "MASTER-FILE-" (format-time-string "%y%m%d") ".org"))
       (files (file-expand-wildcards (concat directory "*.el")))
       (target-buffer))
    ;; (message (concat (file-truename directory) filename))
    (find-file filename)
    (erase-buffer)
    (setq target-buffer (current-buffer))
    (insert "#+STARTUP: overview\n")
    (goto-char (point-max))
    (mapc 'org-el-import-1-file files)))

(defun org-el-import-1-file (fname)
  "Insert file FNAME into the master org file.
Create org header and SRC block from data in FNAME file."
  (message fname)
  (save-excursion
    (let*
        ((fname-base (substring (file-name-base fname) 4 nil))
         found body-start body-end body)
      (find-file fname)
      (goto-char (point-min)) ;; in case we are already editing the buffer!
```

```
      (setq found
            (search-forward fname-base (line-end-position 1) t 1))
      (cond
       (found
        (forward-line 1)
        (setq body-start (point)))
       (t (setq body-start (point-min))))
      (setq found
            (search-forward (format "provide '%s" fname-base) nil t 1))
      (cond
       (found (setq body-end (line-beginning-position)))
       (t (setq body-end (point-max))))
      (setq body (buffer-substring body-start body-end))
      (kill-buffer (current-buffer))
      (with-current-buffer target-buffer
        (goto-char (point-max))
        (insert (replace-regexp-in-string
                 "  " " "
                 (format "\n* %s\n"
                         (replace-regexp-in-string "_" " " fname-base))))
        (insert "\n#+BEGIN_SRC emacs-lisp\n")
        (insert body)
        (insert "#+END_SRC")))))

(defun org-el-export-all-sections ()
  "Export each sections' emacs-lisp block to a separate file.
Add header and footer parts required by flycheck."
  (interactive)
  (let
      ((index 0)
       (root-dir (file-name-directory (buffer-file-name)))
       buffers)
    ;;; First delete old entries, before creating new ones.
    ;;; Prevent duplicate entries due to renumbering.
    (mapc 'delete-file (file-expand-wildcards (concat root-dir "*.el")))
    (org-map-entries 'org-el-export-1-section)
    (mapc 'kill-buffer buffers)))

(defun org-el-export-1-section ()
  "Export this sections' emacs-lisp block to a separate file.
```

```
Add header and footer parts required by flycheck.
Skip sections marked with COMMENT."
  (let* (body-element
         (element (cadr (org-element-at-point)))
         (title (plist-get element :title))
         (commented (plist-get element :commentedp))
         (filename))
    ;; skip commented sections
    (unless commented
      (setq index (+ 1 index))
      (search-forward "#+BEGIN_SRC")
      (setq body-element (cadr (org-element-at-point)))
      ;; (message
      ;;  (replace-regexp-in-string " " "_" (plist-get element :title)))
      ;; (message "%s" body-element)
      (setq title (replace-regexp-in-string " " "_" title))
      (setq filename (format "%03d_%s.el" index title))
      (find-file filename)
      (erase-buffer)
      (insert (format ";;; %s --- %s"
                      title
                      (format-time-string "%F %r\n")))
      (goto-char (point-max))
      (insert (plist-get body-element :value))
      (goto-char (point-max))
      (insert (format "(provide '%s)\n;;; %s ends here" title filename))
      (save-buffer)
      (setq buffers (cons (current-buffer) buffers))
      (kill-buffer))))

(eval-after-load 'org
  '(progn
     ;; Note: This keybinding is in analogy to the default keybinding:
     ;; C-c . -> org-time-stamp
     (define-key org-mode-map (kbd "C-c C-M-e") 'org-el-export-all-sections)))
```

16 cpp makefile

```
;;; Gcc and makefile support
```

```
;; G++ code here

 (global-set-key "\C-xc" 'compile)
 (setq make-backup-files 'nil)
 ;;(setq default-major-mode 'text-mode)
 (setq text-mode-hook 'turn-on-auto-fill)
 ;;(set-default-font "-misc-fixed-medium-r-normal--15-140-*-*-c-*-*-1")
 (setq auto-mode-alist (cons '("\\.cxx$" . c++-mode) auto-mode-alist))
 (setq auto-mode-alist (cons '("\\.hpp$" . c++-mode) auto-mode-alist))
 (setq auto-mode-alist (cons '("\\.tex$" . latex-mode) auto-mode-alist))

;;(require 'font-lock)
;;(add-hook 'c-mode-hook 'turn-on-font-lock)
;;(add-hook 'c++-mode-hook 'turn-on-font-lock)
```

17 SuperCollider

```
;;; Commentary:
;; Basic setup for using SuperCollider in EMACS

;; (add-to-list 'load-path "~/.emacs.d/personal/packages/sclang/")
;; (load-file "~/.emacs.d/personal/packages/sclang/sclang.el")
;; (load-file "~/.emacs.d/personal/packages/sc-snippets/sc-snippets.el")
(require 'sclang) ;; must be made available through links in personal/packages
;; (require 'sc-snippets) ;; replaced by postload file

;;; Directory of SuperCollider support, for quarks, plugins, help etc.
(defvar sc_userAppSupportDir
  (expand-file-name "~/Library/Application Support/SuperCollider"))

;; Make path of sclang executable available to emacs shell load path

;; For Version 3.6.6:
(add-to-list
 'exec-path
 "/Applications/SuperCollider/SuperCollider.app/Contents/Resources/")
```

```
;; For Version 3.7:
(add-to-list
 'exec-path
 "/Applications/SuperCollider/SuperCollider.app/Contents/MacOS/")

;; Global keyboard shortcut for starting sclang
(global-set-key (kbd "C-c M-s") 'sclang-start)
;; overrides alt-meta switch command
(global-set-key (kbd "C-c W") 'sclang-switch-to-workspace)
```

18 SuperCollider-utils

```
;;; Commentary:
;;; emacs  commands for doing useful things in supercollider.
;;; Includes newest version of snippets library.

;;; Code:
;; (sclang-eval-string string &optional print-p)
;; (defun dired-get-filename (&optional localp no-error-if-not-filep)
;; Requires Buffers class of sc-hacks lib.

;; Disable switching to default SuperCollider Workspace when recompiling SClang
(setq sclang-show-workspace-on-startup nil)

;; minor modes SuperCollider

;;; note: Replacing paredit with smartparens
(prelude-load-require-packages
 '(smartparens rainbow-delimiters hl-sexp auto-complete))

(require 'smartparens-config)

;;; paredit
;; NOTE: hs-minor, electric-pair: package names?

;; (add-hook 'sclang-mode-hook 'sclang-extegnsions-mode) ;; still problems with this
(add-hook 'sclang-mode-hook 'smartparens-mode)
(add-hook 'sclang-mode-hook 'rainbow-delimiters-mode)
```

```
(add-hook 'sclang-mode-hook 'hl-sexp-mode)
(add-hook 'sclang-mode-hook 'hs-minor-mode)
(add-hook 'sclang-mode-hook 'electric-pair-mode)
;; (add-hook 'sclang-mode-hook 'yas-minor-mode)
(add-hook 'sclang-mode-hook 'auto-complete-mode)
;; (add-hook 'sclang-mode-hook 'hl-paren-mode)

;; Own bindings for hide-show minor mode:
(add-hook 'sclang-mode-hook
          (lambda()
            (local-set-key (kbd "H-b b") 'hs-toggle-hiding)
            (local-set-key (kbd "H-b H-b")  'hs-hide-block)
            (local-set-key (kbd "H-b a")    'hs-hide-all)
            (local-set-key (kbd "H-b H-a")  'hs-show-all)
            (local-set-key (kbd "H-b l")  'hs-hide-level)
            (local-set-key (kbd "H-b H-l")  'hs-show-level)
            (hs-minor-mode 1)
            (visual-line-mode 1)))

(global-set-key (kbd "H-w") 'sclang-clear-and-switch-to-workspace)

(defun sclang-clear-and-switch-to-workspace ()
  "Shortcut for clear post window and switch to workspace."
  (interactive)
  (sclang-clear-post-buffer)
  (sclang-switch-to-workspace))

(defun dired-load-audio-buffer (&optional preview)
  "Load file at cursor in dired to sc audio buffer.  If PREVIEW then play when loaded
  (interactive "P")
  (sclang-eval-string
   (if preview
       (format "\"%s\",previewBuffer"
               (dired-get-filename))
     (format "\"%s\".loadBuffer"
             (dired-get-filename)))
   t))

(defun dired-add-startup-file (&optional preview)
  "Add the file to the list of startup files.  If PREVIEW then only test loading but
```

```
  (interactive "P")
  (let ((paths (dired-get-marked-files)))
    (dolist (path paths)
      (message path)
      (sclang-eval-string
       (if preview
           (format "\"%s\".previewCode;\n" path)
         (format "\"%s\".addCode;\n" path))
       t))))

(eval-after-load 'dired
  '(progn
     ;; Note: This keybinding is in analogy to the default keybinding:
     ;; C-c . -> org-time-stamp
     (define-key dired-mode-map (kbd "C-c C-b") 'dired-load-audio-buffer)
     (define-key dired-mode-map (kbd "C-c C-s") 'dired-add-startup-file)))

;; (global-set-key (kbd "H-d b") 'dired-load-audio-buffer)

(defun org-sclang-eval-babel-block ()
  "Evaluate current babel code block as sclang code."
  (interactive)
  (let*
      ((element (cadr (org-element-at-point)))
       (code (plist-get element :value)))
    (sclang-eval-string code t)))

(eval-after-load 'org
  '(progn
     ;; Note: This keybinding is in analogy to the default keybinding:
     ;; C-c . -> org-time-stamp
     (define-key org-mode-map (kbd "C-c C-/") 'org-sclang-eval-babel-block)))

  ;;; key chords for sclang
(defun sclang-2-windows ()
  "Reconfigure frame to this window and sclang-post-window."
  (interactive)
  (delete-other-windows)
  (sclang-show-post-buffer))
```

```
;; (defun sclang-plusgt ()
;;   "Insert +>."
;;   (interactive)
;;   (insert "+>"))

;; (defun sclang-ltplus ()
;;   "Insert <+."
;;   (interactive)
;;   (insert "<+"))

;; (defun sclang-xgt ()
;;   "Insert *>"
;;   (interactive)
;;   (insert "*>"))

(defun scundelify ()
  "Blah."
  (interactive)
  (save-excursion
    (goto-char (point-min))
    (while (re-search-forward "\n//:" nil t)
      (replace-match "\n\)\n//:")
      (goto-char (line-end-position 2))
      (goto-char (line-beginning-position 1))
      (insert "\(\n")
      (goto-char (line-beginning-position 1))
      (delete-blank-lines))
    (goto-char (point-min))
    (re-search-forward "\)\n//:" nil t)
    (replace-match "\n://:")))

(defun sclang-get-current-snippet ()
  "Return region between //: comments in sclang, as string.
If the beginning of line is '//:+', then fork the snippet as routine.
If the beginning of line is '//:*', then wrap the snippet in loop and fork."
  (save-excursion
    (goto-char (line-end-position)) ;; fix when starting from point-min
    (let (
          (snippet-begin (search-backward-regexp "^//:" nil t))
          snippet-end
```

```
            snippet
            snippet-head
            (prefix ""))
        (unless snippet-begin
          (setq snippet-begin (point-min))
          (setq prefix "//:\n"))
        (setq sclang-snippet-is-routine nil)
        (setq sclang-snippet-is-loop nil)
        (goto-char snippet-begin)
        (setq snippet-head (buffer-substring-no-properties (point) (+ 4 (point))))
        (if (equal snippet-head "//:+") (setq sclang-snippet-is-routine t))
        (if (equal snippet-head "//:*") (setq sclang-snippet-is-loop t))
        (goto-char (line-end-position))
        (setq snippet-end (search-forward-regexp "^//:" nil t))
        (if snippet-end
            (setq snippet-end (line-beginning-position))
          (setq snippet-end (point-max)))
        (concat prefix
                (buffer-substring-no-properties snippet-begin snippet-end)))))

(defun sclang-cut-current-snippet ()
  "Return region between //: comments in sclang, as string, and cut it out."
  (interactive)
  (save-excursion
    (goto-char (line-end-position)) ;; fix when starting from point-min
    (let (
          (snippet-begin (search-backward-regexp "^//:" nil t))
          snippet-end
          snippet
          (prefix ""))
      (unless snippet-begin
        (setq snippet-begin (point-min))
        (setq prefix "//:\n"))
      (goto-char (line-end-position))
      (setq snippet-end (search-forward-regexp "^//:" nil t))
      (if snippet-end
          (setq snippet-end (line-beginning-position))
        (setq snippet-end (point-max)))
      (setq snippet (concat prefix
                            (buffer-substring-no-properties snippet-begin snippet-end
```

```
      (kill-region snippet-begin snippet-end))))

(defun sclang-transpose-snippet-down ()
  "Transpose this snippet with the one following it."
  (interactive)
  (sclang-cut-current-snippet)
  (sclang-goto-next-snippet)
  (insert "\n")
  (yank)
  (delete-blank-lines)
  (re-search-backward "^//:")
  (goto-char (line-end-position 2)))

(defun sclang-transpose-snippet-up ()
  "Transpose this snippet with the one preceding it."
  (interactive)
  (sclang-cut-current-snippet)
  (re-search-backward "^//:")
  (yank)
  (re-search-backward "^//:")
  (goto-char (line-end-position 2)))

(defun sclang-eval-current-snippet ()
  "Evaluate the current snippet in sclang.
A snippet is a block of code enclosed between comments
starting at the beginning of line and with a : following immediately after '//'.
If the beginning of line is '//:+', then fork the snippet as routine.
If the beginning of line is '//:*', then wrap the snippet in loop and fork."
  (interactive)
  (let* (sclang-snippet-is-routine
         sclang-snippet-is-loop
         (snippet (sclang-get-current-snippet)))
    (if sclang-snippet-is-routine
        (setq snippet (format "{\n %s\n }.fork" snippet)))
    (if sclang-snippet-is-loop
        (setq snippet (format "{\n loop {\n %s \n} \n }.fork" snippet)))
    (sclang-eval-string snippet t)))

(defun sclang-goto-next-snippet ()
  "Go to the next snippet."
```

```
  (interactive)
  (goto-char (sclang-end-of-snippet))
  (goto-char (line-end-position 2))
  (goto-char (line-beginning-position)))

(defun sclang-goto-previous-snippet ()
  "Go to the previous snippet."
  (interactive)
  (goto-char (line-end-position))
  (let ((pos (search-backward-regexp "^//:" nil t)))
    (if (and pos (> pos 1)) (goto-char (1- pos)))
    (setq pos (search-backward-regexp "^//:" nil t))
    (cond
     (pos
      (goto-char pos)
      (goto-char (1+ (line-end-position)))
      (goto-char (line-beginning-position)))
     (t
      (goto-char (point-min))))
    ;; (re-search-backward "^//:")
    ))

(defun sclang-eval-next-snippet ()
  "Go to the next snippet and evaluate it."
  (interactive)
  (sclang-goto-next-snippet)
  (sclang-eval-current-snippet))

(defun sclang-eval-previous-snippet ()
  "Go to the previous snippet and evaluate it."
  (interactive)
  (sclang-goto-previous-snippet)
  (sclang-eval-current-snippet))

(defun sclang-duplicate-current-snippet ()
  "Insert a copy the current snippet below itself."
  (interactive)
  (let ((snippet (sclang-get-current-snippet)))
    (goto-char (line-end-position))
    (goto-char (sclang-end-of-snippet))
```

```
    (if (eq (point) (point-max)) (insert "\n"))
    (insert snippet)))

(defun sclang-copy-current-snippet ()
  "Copy the current snippet into the kill ring."
  (interactive)
  (let ((snippet (sclang-get-current-snippet)))
    (kill-new snippet)))

(defun sclang-region-select-current-snippet ()
  "Select region between //: comments in sclang."
  (save-excursion
    (goto-char (line-end-position)) ;; fix when starting from point-min
    (let (
          (snippet-begin (search-backward-regexp "^//:" nil t))
          snippet-end
          snippet
          snippet-head)
      (unless snippet-begin
        (setq snippet-begin (point-min)))
      (goto-char snippet-begin)
      (goto-char (line-end-position))
      (setq snippet-end (search-forward-regexp "^//:" nil t))
      (if snippet-end
          (setq snippet-end (line-beginning-position))
        (setq snippet-end (point-max)))
      (goto-char snippet-begin)
      (push-mark snipet-end)
      (setq mark-active t))))

(defun sclang-cut-current-snippet ()
  "Kill the current snippet, storing it in kill-ring."
  (sclang-region-select-current-snippet)
  (kill-region (mark) (point)))

(defun sclang-end-of-snippet ()
  "Return the point position of the end of the current snippet."
  (save-excursion
    (let ((pos (search-forward-regexp "^//:" nil t)))
      (if pos (line-beginning-position) (point-max)))))
```

```
(defun sclang-beginning-of-snippet ()
  "Return the point position of the beginning of the current snippet."
  (save-excursion
    (goto-char (line-end-position))
    (let ((pos (search-backward-regexp "^//:" nil t)))
      (if pos pos (point-min)))))

(defun sclang-insert-snippet-separator (&optional before)
  "Insert snippet separator //: at beginning of line."
  (interactive "P")
  (cond
   (before
    (goto-char (line-beginning-position))
    (insert "//:\n"))
   (t
    (goto-char (line-end-position))
    (insert "\n//:"))
   ))

(defun sclang-insert-snippet-separator+ (&optional before)
  "Insert snippet separator //:+ at beginning of line."
  (interactive "P")
  (cond (before
         (goto-char (line-beginning-position))
         (insert "//:+\n"))
        (t
         (goto-char (line-end-position))
         (insert "\n//:+"))
        ))

(defun sclang-insert-snippet-separator* (&optional before)
  "Insert snippet separator //:* at beginning of line."
  (interactive "P")
  (cond (before
         (goto-char (line-beginning-position))
         (insert "//:*\n"))
        (t
         (goto-char (line-end-position))
         (insert "\n//:*"))
```

```
        ))

(defun sclang-server-plot-tree ()
  "Open plotTree for default server."
  (interactive)
  (sclang-eval-string "Server.default.plotTree"))

(defun sclang-server-meter ()
  "Open i/o meter for default server."
  (interactive)
  (sclang-eval-string "Server.default.meter"))

(defun sclang-server-scope ()
  "Open scope for default server."
  (interactive)
  (sclang-eval-string "Server.default.scope"))

(defun sclang-server-freqscope ()
  "Open frequency scope for default server."
  (interactive)
  (sclang-eval-string "Server.default.freqscope"))

(defun sclang-startupfiles-gui ()
  "Open StartupFile gui."
  (interactive)
  (sclang-eval-string "StartupFiles.gui"))

(defun sclang-audiofiles-gui ()
  "Open AudioFiles gui."
  (interactive)
  (sclang-eval-string "AudioFiles.gui"))

(defun sclang-players-gui ()
  "Open Players gui."
  (interactive)
  (sclang-eval-string "PlayerGui.gui"))

(defun sclang-extensions-gui ()
  "Open gui for browsing user extensions classes and methods.
  Type return on a selected item to open the file where it is defined."
```

```
  (interactive)
  (sclang-eval-string "Class.extensionsGui;"))

(defun sclang-nevent-gui ()
  "Open gui displaying contents of current Nenvir."
  (interactive)
  (sclang-eval-string "NeventGui.gui;"))

(eval-after-load 'sclang
  (progn
    ;; these are disabled by sclang-bindings:
    ;; (define-key sclang-mode-map (kbd "C-c C-p t") 'sclang-server-plot-tree)
    ;; (define-key sclang-mode-map (kbd "C-c C-p m") 'sclang-server-meter)
     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;; sc-hacks gui commands:
    (define-key sclang-mode-map (kbd "C-h g s") 'sclang-startupfiles-gui)
    (define-key sclang-mode-map (kbd "C-h g a") 'sclang-audiofiles-gui)
    (define-key sclang-mode-map (kbd "C-h g p") 'sclang-players-gui)
    (define-key sclang-mode-map (kbd "C-h g e") 'sclang-extensions-gui)
    (define-key sclang-mode-map (kbd "C-h g n") 'sclang-nevent-gui)

     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;; Server state visualisation utilities
    ;; TODO: Check and re-assign these commands for consistency with
    ;; default sclang commands C-c C-p x:
    (define-key sclang-mode-map (kbd "C-c P p") 'sclang-server-plot-tree)
    (define-key sclang-mode-map (kbd "C-c P m") 'sclang-server-meter)
    (define-key sclang-mode-map (kbd "C-c P s") 'sclang-server-scope)
    (define-key sclang-mode-map (kbd "C-c P f") 'sclang-server-freqscope)
 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
     ;;;;;;;;;;;;;;;;;       snippet commands      ;;;;;;;;;;;;;;;;;
    ;; eval current snippet              M-C-x
    ;; goto next snippet                 M-n
    ;; goto previous snippet             M-p
    ;; eval previous snippet             M-P
    ;; eval next snippet                 M-N
    ;; duplicate current snippet         M-D
    ;; copy current snippet              M-C
    ;; select region of current snippet  M-R
    ;; cut current snippet               M-T
```

```
;; transpose snippet down             C-M-N
;; transpose snippet up               C-M-P

(define-key sclang-mode-map (kbd "M-X") 'sclang-eval-current-snippet)
(define-key sclang-mode-map (kbd "C-M-x") 'sclang-eval-current-snippet)
(define-key sclang-mode-map (kbd "M-n") 'sclang-goto-next-snippet)
(define-key sclang-mode-map (kbd "M-p") 'sclang-goto-previous-snippet)
(define-key sclang-mode-map (kbd "M-N") 'sclang-eval-next-snippet)
(define-key sclang-mode-map (kbd "M-P") 'sclang-eval-previous-snippet)
(define-key sclang-mode-map (kbd "M-D") 'sclang-duplicate-current-snippet)
(define-key sclang-mode-map (kbd "M-C") 'sclang-copy-current-snippet)
(define-key sclang-mode-map (kbd "M-R") 'sclang-region-select-current-snippet)
(define-key sclang-mode-map (kbd "M-T") 'sclang-cut-current-snippet)
(define-key sclang-mode-map (kbd "C-M-N") 'sclang-transpose-snippet-down)
(define-key sclang-mode-map (kbd "C-M-P") 'sclang-transpose-snippet-up)


;; (define-key sclang-mode-map (kbd "M-C-.") 'sclang-duplicate-current-snippet)
;; (define-key sclang-mode-map (kbd "M-n") 'sclang-goto-next-snippet)
;; (define-key sclang-mode-map (kbd "M-N") 'sclang-eval-next-snippet)
;; (define-key sclang-mode-map (kbd "M-C-S-n") 'sclang-move-snippet-down)
;; (define-key sclang-mode-map (kbd "M-p") 'sclang-goto-previous-snippet)
;; (define-key sclang-mode-map (kbd "M-P") 'sclang-eval-previous-snippet)
;; (define-key sclang-mode-map (kbd "M-C-S-p") 'sclang-move-snippet-up)X

(define-key sclang-mode-map (kbd "H-=") 'sclang-insert-snippet-separator+)
(define-key sclang-mode-map (kbd "H-8") 'sclang-insert-snippet-separator*)

 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Miscellaneous
(define-key sclang-mode-map (kbd "C-S-c c") 'sclang-clear-post-buffer)

(key-chord-define sclang-mode-map "11" 'sclang-2-windows)
;; (key-chord-define sclang-mode-map "'" 'sclang-plusgt)
;; (key-chord-define sclang-mode-map ";;" 'sclang-ltplus)
;; (key-chord-define sclang-mode-map "\\\\" 'sclang-xgt)
))
```

19 tidal

```
(prelude-load-require-package 'haskell-mode)
(prelude-load-require-package 'tidal)
(setq tidal-interpreter "/usr/local/bin/ghci")
```

20 calc-time-zones

```
;;; Commentary:
;; Add some useful time zones
(require 'calc-forms) ;; built-in package
(add-to-list 'math-tzone-names '("JST" 9 0))
(add-to-list 'math-tzone-names '("EEST" 3 0))
```

21 org-mode

```
;;; Commentary:

;; customize some org mode settings
;; define some useful functions

;;; Code:

;;; pretty bullets
;;;   (prelude-load-require-package 'org-bullets)
(require 'org-bullets)
(add-hook 'org-mode-hook (lambda () (org-bullets-mode 1)))

;; load util to insert recipes for export customization:
;; (require 'org-export-recipes) ;; is now part of postload!

(setq org-attach-directory (file-truename "~/Documents/org-attachments/"))
(setq org-agenda-sticky t) ;; open agenda and todo views in separate buffers
;; (setq org-agenda-diary-file (file-truename
;;                              (concat iz-log-dir "PERSONAL/DIARY2.txt")))

;; customize looks
```

```
(custom-set-faces
 '(org-block-end-line ((t (:background "#3a5a5f" :foreground "gray99"))) t)
 '(org-level-1 ((t (:family "Helvetica" :height 1.1 :weight bold))))
 '(org-level-2 ((t (:family "Helvetica" :height 1.1 :weight bold))))
 ;; '(org-level-1 ((t (:family "Courier New" :height 1.1 :weight bold))))
 ;; '(org-level-2 ((t (:family "Courier New" :height 1.1 :weight bold))))
 '(org-level-3 ((t (:weight bold :height 1.1))))
 '(org-level-4 ((t (:weight bold :height 1.1))))
 '(org-level-5 ((t (:weight bold :height 1.1))))
 '(org-level-6 ((t (:weight bold :height 1.1))))
 '(org-level-7 ((t (:weight bold :height 1.1))))
 '(org-level-8 ((t (:weight bold :height 1.1))))
 '(org-level-9 ((t (:weight bold :height 1.1)))))

(defun org-set-date (&optional active property)
  "Set DATE property with current time.  Active timestamp."
  (interactive "P")
  (org-set-property
   (if property property "DATE")
   (cond ((equal active nil)
          (format-time-string (cdr org-time-stamp-formats) (current-time)))
         ((equal active '(4))
          (concat "["
                  (substring
                   (format-time-string (cdr org-time-stamp-formats) (current-time))
                   1 -1)
                  "]"))
         ((equal active '(16))
          (concat
           "["
           (substring
            (format-time-string (cdr org-time-stamp-formats) (org-read-date t t))
            1 -1)
           "]"))
         ((equal active '(64))
          (format-time-string (cdr org-time-stamp-formats) (org-read-date t t))))))

(defun org-insert-current-date (arg)
  "Insert current date in format readable for org-capture minibuffer.
If called with ARG, do not insert time."
```

```
  (interactive "P")
  (if arg
      (insert (format-time-string "%e %b %Y"))
    (insert (format-time-string "%e %b %Y %H:%M"))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; make heading movement commands skip initial * marks
(defun org-jump-forward-heading-same-level (&optional do-cycle)
  "Jump forward heading same level, and skip to beginning of heading itself."
  (interactive "P")
  (org-forward-heading-same-level 1)
  (re-search-forward " ")
  (if do-cycle (org-cycle)))

(defun org-jump-backward-heading-same-level (&optional do-cycle)
  "Jump backward heading same level, and skip to beginning of heading itself."
  (interactive "P")
  (org-backward-heading-same-level 1)
  (re-search-forward " ")
  (if do-cycle (org-cycle)))

(defun jump-outline-up-heading (&optional do-cycle)
  "Jump upward heading, and skip to beginning of heading itself."
  (interactive "P")
  (outline-up-heading 1)
  (re-search-forward " ")
  (if do-cycle (org-cycle)))

(defun jump-outline-next-visible-heading ()
  "Jump to next visible heading, and skip to beginning of heading itself."
  (interactive)
  (outline-next-visible-heading 1)
  (re-search-forward " "))

(defun jump-outline-previous-visible-heading ()
  "Jump to previous visible heading, and skip to beginning of heading itself."
  (interactive)
  (outline-previous-visible-heading 1)
  (re-search-forward " "))
```

```
(defun jump-outline-previous-visible-heading-and-cycle ()
  "Jump to previous visible heading, and hide subtree."
  (interactive)
  (outline-previous-visible-heading 1)
  (re-search-forward " ")
  (org-cycle))

(defun jump-outline-next-visible-heading-and-cycle ()
  "Jump to previous visible heading, and hide subtree."
  (interactive)
  (outline-next-visible-heading 1)
  (re-search-forward " ")
  (org-cycle))

(defun org-find-next-src-block ()
  "Search for next #+BEGIN_SRC block header."
  (interactive)
  (re-search-forward "\\#\\+BEGIN_SRC " nil t))
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; This is run once after loading org for the first time
;; It adds some org-mode specific key bindings.
(eval-after-load 'org
  '(progn
     ;; alias for org-cycle, more convenient than TAB
     (define-key org-mode-map (kbd "C-M-]") 'org-cycle)
     ;; Note: This keybinding is in analogy to the default keybinding:
     ;; C-c . -> org-time-stamp
     (define-key org-mode-map (kbd "C-c C-.") 'org-set-date)
     (define-key org-mode-map (kbd "C-M-{") 'backward-paragraph)
     (define-key org-mode-map (kbd "C-M-}") 'forward-paragraph)
     (define-key org-mode-map (kbd "C-c C-S") 'org-schedule)
     (define-key org-mode-map (kbd "C-c C-s") 'sclang-main-stop)
     (define-key org-mode-map (kbd "C-c >") 'sclang-show-post-buffer)
     ;; own additions after org-config-examples below:
     (define-key org-mode-map (kbd "C-M-S-n") 'org-next-src-block)
     (define-key org-mode-map (kbd "C-M-S-p") 'org-show-properties-block)
     (define-key org-mode-map (kbd "C-M-/") 'org-sclang-eval-babel-block)
            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
     ;; from: http://orgmode.org/worg/org-configs/org-config-examples.html
```

```
;; section navigation
(define-key org-mode-map (kbd "M-n") 'jump-outline-next-visible-heading)
(define-key org-mode-map (kbd "C-M-n") 'jump-outline-next-visible-heading-and-cy
;; (define-key org-mode-map (kbd "C-M-N") 'org-find-next-src-block)
(define-key org-mode-map (kbd "M-p") 'jump-outline-previous-visible-heading)
(define-key org-mode-map (kbd "C-M-p") 'jump-outline-previous-visible-heading-an
(define-key org-mode-map (kbd "C-M-f") 'org-jump-forward-heading-same-level)
(define-key org-mode-map (kbd "C-M-b") 'org-jump-backward-heading-same-level)
(define-key org-mode-map (kbd "C-M-u") 'jump-outline-up-heading)
;; table
(define-key org-mode-map (kbd "C-M-w") 'org-table-copy-region)
(define-key org-mode-map (kbd "C-M-y") 'org-table-paste-rectangle)
(define-key org-mode-map (kbd "C-M-l") 'org-table-sort-lines)
;; display images
(define-key org-mode-map (kbd "M-I") 'org-toggle-iimage-in-org)
;; Following are the prelude-mode binding, minus the conflicting table bindings.
;; prelude-mode is turned off for org mode, below.
(define-key org-mode-map (kbd "C-c o") 'crux-open-with)
;; (define-key org-mode-map (kbd "C-c g") 'prelude-google)
;; (define-key org-mode-map (kbd "C-c G") 'crux-github)
;; (define-key org-mode-map (kbd "C-c y") 'prelude-youtube)
;; (define-key org-mode-map (kbd "C-c U") 'prelude-duckduckgo)
;;     ;; mimic popular IDEs binding, note that it doesn't work in a terminal se
(define-key org-mode-map [(shift return)] 'crux-smart-open-line)
(define-key org-mode-map (kbd "M-o") 'crux-smart-open-line)
(define-key org-mode-map [(control shift return)] 'crux-smart-open-line-above)
(define-key org-mode-map [(control shift up)]  'move-text-up)
(define-key org-mode-map [(control shift down)]  'move-text-down)
(define-key org-mode-map [(control meta shift up)]  'move-text-up)
(define-key org-mode-map [(control meta shift down)]  'move-text-down)
;;     ;; the following 2 break structure editing with meta-shift-up / down in o
;;     ;;    (define-key map [(meta shift up)]  'move-text-up)
;;     ;;    (define-key map [(meta shift down)]  'move-text-down)
;;     ;; new substitutes for above:  (these are overwritten by other modes...)
;;     ;; (define-key map (kbd "C-c [")  'move-text-up)
;;     ;; (define-key map (kbd "C-c ]")  'move-text-down)
;;     ;; (define-key map [(control meta shift up)]  'move-text-up)
;;     ;; (define-key map [(control meta shift down)]  'move-text-down)
(define-key org-mode-map (kbd "C-c n") 'crux-cleanup-buffer-or-region)
(define-key org-mode-map (kbd "C-c f") 'crux-recentf-ido-find-file)
```

```
    (define-key org-mode-map (kbd "C-M-z") 'crux-indent-defun)
    (define-key org-mode-map (kbd "C-c u") 'crux-view-url)
    (define-key org-mode-map (kbd "C-c e") 'crux-eval-and-replace)
    (define-key org-mode-map (kbd "C-c s") 'crux-swap-windows)
    (define-key org-mode-map (kbd "C-c D") 'crux-delete-file-and-buffer)
    (define-key org-mode-map (kbd "C-c d") 'crux-duplicate-current-line-or-region)
    (define-key org-mode-map (kbd "C-c M-d") 'crux-duplicate-and-comment-current-lin
    (define-key org-mode-map (kbd "C-c r") 'crux-rename-buffer-and-file)
    (define-key org-mode-map (kbd "C-c t") 'crux-visit-term-buffer)
    (define-key org-mode-map (kbd "C-c k") 'crux-kill-other-buffers)
    ;;     ;; another annoying overwrite of a useful org-mode command:
    ;;     ;; (define-key map (kbd "C-c TAB") 'prelude-indent-rigidly-and-copy-to-cl
    (define-key org-mode-map (kbd "C-c I") 'crux-find-user-init-file)
    (define-key org-mode-map (kbd "C-c S") 'crux-find-shell-init-file)
    ;; replace not functioning 'prelude-goto-symbol with useful imenu-anywhere
    (define-key org-mode-map (kbd "C-c i") 'imenu-anywhere)
    ;;     ;; extra prefix for projectile
    (define-key org-mode-map (kbd "s-p") 'projectile-command-map)
    ;;     ;; make some use of the Super key
    (define-key org-mode-map (kbd "s-g") 'god-local-mode)
    (define-key org-mode-map (kbd "s-r") 'crux-recentf-ido-find-file)
    (define-key org-mode-map (kbd "s-j") 'crux-top-join-line)
    (define-key org-mode-map (kbd "s-k") 'crux-kill-whole-line)
    (define-key org-mode-map (kbd "s-m m") 'magit-status)
    (define-key org-mode-map (kbd "s-m l") 'magit-log)
    (define-key org-mode-map (kbd "s-m f") 'magit-log-buffer-file)
    (define-key org-mode-map (kbd "s-m b") 'magit-blame)
    (define-key org-mode-map (kbd "s-o") 'crux-smart-open-line-above)
    ))

(defun org-next-src-block ()
  "Jump to the next src block using SEARCH-FORWARD."
  (interactive)
  (search-forward "\n#+BEGIN_SRC")
  (let ((block-beginning (point)))
    (org-show-entry)
    (goto-char block-beginning)
    (goto-char (line-end-position 2))))

(defun org-show-properties-block ()
```

```
  "Show the entire next properties block using SEARCH-FORWARD."
  (interactive)
  (search-forward ":PROPERTIES:")
  (let ((block-beginning (point)))
    (org-show-entry)
    (goto-char block-beginning)
    (org-cycle)
    ;; (goto-char (line-end-position 2))
    ;; (org-hide-block-toggle t)
    ))

;; org-mode-hook is run every time that org-mode is turned on for a buffer
;; It customizes some settings in the mode.
(add-hook
 'org-mode-hook
 (lambda ()
   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
   ;; own stuff:
   ;; Make javascript blocks open in sclang mode in org-edit-special
   ;; This is because sclang blocks must currently be marked as javascript
   ;; in order to render properly with hugo / pygments for webite creation.
   (setq org-src-lang-modes (add-to-list 'org-src-lang-modes '("javascript" . sclang)
   (setq org-hide-leading-stars t)
   ;; (org-indent-mode) ;; this results in added leading spaces in org-edit-special
   (visual-line-mode)
   ;; turn off prelude mode because its key bindings interfere with table bindings.
   ;; Instead, the prelude-mode keybindings have been copied to org-mode above,
   ;; minus the unwanted keybindings for tables.
   (prelude-off)
   ;; disable whitespace mode, which was previously disabled by prelude-mode
   (whitespace-mode -1)
   ))

;; (defun org-customize-mode ()
;;   "Customize some display options for ORG-MODE.
;; - map javascript to sclang-mode in babel blocks.
;; - hide extra leading stars for sections.
;; - turn on visual line mode."
;; )
```

```
(global-set-key (kbd "C-c C-x t") 'org-insert-current-date)
```

22 org mode latex export

N.B. Xelatex enables one to use system fonts. These are necessary for languages such as greek, which requires a character set other than latin.

```
;; Use xelatex as latex compiler, thus enabling use of native fonts for greek etc.
(setq org-latex-compiler "xelatex")
```

23 org-mode: Use uppercase UUIDs in Linux for consistency with MacOS when syncing with rslsync

```
(defun org-id-get (&optional pom create prefix)
  "Get the ID property of the entry at point-or-marker POM.

This is a modified version that returns the ID using uppercase letters,
for consistency with MacOS when syncing attachment folders over rslsync.

If POM is nil, refer to the entry at point.
If the entry does not have an ID, the function returns nil.
However, when CREATE is non nil, create an ID if none is present already.
PREFIX will be passed through to `org-id-new'.
In any case, the ID of the entry is returned."
  (org-with-point-at pom
    (let ((id (org-entry-get nil "ID")))
      (cond
       ((and id (stringp id) (string-match "\\S-" id))
        id)
       (create
        (setq id (org-id-new prefix))
        (org-entry-put pom "ID" id)
        (org-id-add-location id (buffer-file-name (buffer-base-buffer)))
        (upcase id))))))
```

24 org-mode todo states

```
(setq org-todo-keywords
       '((sequence "TODO(t)" "|" "DONE(d@)" "CANCELED(c@)")))
```

25 org calfw

```
  ;;; Commentary:
  ;;; use calfw package to display agenda in calendar-grid format
  ;;; Provide commands for generation of entries on current date on calendar grid

  ;;; Code:
;; (require 'calfw-org)
;; (require 'calfw-cal)

(prelude-load-require-packages '(calfw calfw-org calfw-cal))

(setq calendar-christian-all-holidays-flag t)

(setq org-capture-use-agenda-date t)

(setq cfw:org-overwrite-default-keybinding t)

(defun org-calfw-here (&optional arg)
  "Open calfw on the file of the present buffer."
  (interactive "P")
  (when (and (buffer-file-name) (eq major-mode 'org-mode))
    (if arg
        (setq org-agenda-files (list (buffer-file-name)))
      (add-to-list 'org-agenda-files (buffer-file-name))))
  ;; (org-log-here (buffer-file-name) t)
  (cfw:open-org-calendar))

;; (defun cfw:org-capture (prefix)
;;   "Overwrite original to run own cfw:org-capture-at-date instead."
;;   (interactive "P")
;;   (cfw:org-journal-at-date prefix))

(defun cfw:org-journal-at-date-from-cursor (prefix)
```

```
  "Run org-journal-new-entry with ORG-OVERRIDING-DEFAULT-TIME from cursor."
  (interactive "P")
  (with-current-buffer  (get-buffer-create cfw:calendar-buffer-name)
    (let* ((pos (cfw:cursor-to-nearest-date))
           (org-overriding-default-time
            (encode-time 0 0 7
                         (calendar-extract-day pos)
                         (calendar-extract-month pos)
                         (calendar-extract-year pos))))
      (org-journal-new-entry prefix org-overriding-default-time)
      (unless prefix
        (org-insert-time-stamp org-overriding-default-time t)
        (backward-word)
        (backward-word)
        (paredit-forward-kill-word)
        (paredit-forward-kill-word)))))

(defun cfw:org-journal-entry-for-now (prefix)
  "Run org-journal-new-entry with date+time timestamp from current time."
  (interactive "P")
  (with-current-buffer  (get-buffer-create cfw:calendar-buffer-name)
    (let* ((pos (cfw:cursor-to-nearest-date))
           (org-overriding-default-time (apply 'encode-time (decode-time))
                                        ;; (encode-time 0 0 7
                                        ;;              (calendar-extract-day pos)
                                        ;;              (calendar-extract-month pos)
                                        ;;              (calendar-extract-year pos))
                                        ))
      (org-journal-new-entry prefix org-overriding-default-time)
      (org-insert-time-stamp org-overriding-default-time t))))

(defun org-jump-to-refile-target ()
  "Make org-refile with prefix available as command.
  Also, always update refile targets before running org-refile.
  This ensures that files changed / created recently will be taken into account."
  (interactive)
  (org-iz-make-refile-targets)
  (org-refile '(4)))

(global-set-key (kbd "M-C-g") 'org-jump-to-refile-target)
```

```
(global-set-key (kbd "C-c c c") 'org-calfw-here)
(global-set-key (kbd "C-c C J") 'cfw:org-journal-entry-for-now)
;; journal entry for Now (current date and time at time of command)
(define-key
  cfw:calendar-mode-map "N" 'cfw:org-journal-entry-for-now)
;; journal entry for Here (date at cursor on calfw buffer)
(define-key
  cfw:calendar-mode-map "H" 'cfw:org-journal-at-date-from-cursor)


;; (define-key
;;   cfw:calendar-mode-map "C" 'cfw:org-journal-entry-for-now)
;; (define-key
;;   cfw:calendar-mode-map "c" 'cfw:org-journal-at-date-from-cursor)

(provide '018_calfw)
  ;;; 018_calfw.el ends here
```

26 org-split-hugo

```
;;; Commentary:
;;; Utilities for blog + website editing with HUGO

;;; org-hugo-autosplit: split an entire org-file into subfiles for export to hugo.
;;; The contents of any section that has a property "filename" will be
;;; exported under the same directory as the source file.
;;; the filename property gives the filename.
;;; the heading becomes title property in yaml front-matter.
;;; the weight is set according to the order of the exported sections.
;;;
;;; Sections with property "foldername" set a subfolder for saving
;;; subsequent file sections.
;;; Folder path is constructed by concatenating a cumulative list of subfolders.
;;; The foldername property sets the component of the folder path
;;; in the corresponding depth of subsections in a list of path components.
;;; Construct _index.md from the name of the folder section.
;;; Increment a folder_index variable to set weight for folder _index.md.

;;; Code:
```

```
;;; provides commands for hugo config, page creation, publish and
(prelude-load-require-packages '(easy-hugo dash)) ;; dash list utility functions

;;; org-check-agenda-file stops the file creation process
;;; and therefore must be redefined here.
;;; Consequences of overwriting it are not yet checked, but seem irrelevant.
(defun org-check-agenda-file (file)
  "Make sure FILE exists.  If not, ask user what to do."
  (unless (file-exists-p file)
    (message "Ignoring non-existent agenda file: %s"
             (abbreviate-file-name file))))

(defun org-hugo-autosplit ()
  "Auto-export sections marked with filename property after each save."
  (interactive)
 (add-hook 'after-save-hook
           (lambda ()
             (org-hugo-export)
             ;; (message "hugo export to individual files done")
             )
           'append 'local)
 (message "This buffer will now export to hugo section files after each save."))

(defun org-hugo-export ()
  "Split 1st level sections with filename property to files.
Add front-matter for hugo, including automatic weights."
  (interactive)
  (let*
      ((root-dir (file-name-directory (buffer-file-name)))
       (cleanup-list (file-expand-wildcards (concat root-dir "*")))
       (path root-dir)
       (folder-components)
       (index 0)
       folderindex ;; initialized from index upon first folder
       buffers-to-delete)
    (mapc
     (lambda (path)
       (if
           (file-directory-p path)
```

```
            (delete-directory path t))
          (if (string-match-p
               "[[:digit:]]+-"
               (file-name-nondirectory path))
              (delete-file path)))
       cleanup-list)
      (org-map-entries
       '(org-split-1-file-or-folder)
       t 'file 'archive 'comment)
      (mapc (lambda (buffer)
              ;; (message "killing buffer: %s" buffer)
              (set-buffer-modified-p nil)
              (kill-buffer buffer))
            buffers-to-delete)
      (message "Exported %d files" index)))

(defun org-split-1-file-or-folder ()
  "Helper function for org-hugo-export.
First compute the path based on levels and previous input.
Then export the file using the path.
Files are automatically stored in nested folders corresponding
to the level of the section that is the source of the file.
The path for the nested folders is stored in variable FOLDER-COMPONENTS.
Level 1 corresponds to root level of the base folder path.
Therefore for level 1 the string to add to the folder path is the
empty string \"\".
Level 2 has default contents (\"section1/\"), Etc.
The default names section1, section2 etc. can be overwritten
by setting the property foldername of any section.
This overwrites the foldername at the corresponding level position
in the FOLDER-COMPONENTS list."
  (let*
      ((filename (org-entry-get (point) "filename"))
       (foldername (org-entry-get (point) "foldername"))
       (element (cadr (org-element-at-point)))
       (title (plist-get element :title))
       (level (plist-get element :level))
       contents
       ;; (num-folders (- level 1))
       )
```

```
    ;; (message "PREPARING TOPLEVEL EXPORT OF: %s" filename)
    (when (or foldername filename)
      (setq index (+ 1 index))
      ;; add missing default levels to folder-components list
      (when(< (length folder-components) level)
        (setq folder-components
              (append folder-components
                      (--map (list (format "section%d/" it) (format "section%d/" it))
                             (-iterate '1+ level (- level (length folder-components))
    (when foldername ;; set folder name at corresponding level of folder-components
        (setq folder-components
              (-replace-at (1- level)
                           (list (concat foldername "/") title)
                           folder-components)))
    (when filename
      ;; (message "the filename is: %s" filename)
      (setq contents (buffer-substring
                      (plist-get element :contents-begin)
                      (plist-get element :contents-end)))
      (org-hugo-make-folders)
      (org-hugo-export-section-2-file))))

(defun org-hugo-make-folders ()
  "Make subfolders for this level and provide index file for each."
  (when (> level 1)
    ;; (message "testing %s" folder-components)
    (let
        ((sub-components (-take (1- level) folder-components))
         (dir root-dir))
      (make-directory (concat
                       root-dir
                       (apply 'concat (-map 'car sub-components)))
                      t)
      (-each
          sub-components
        (lambda (component)
          (setq dir (concat dir (car component)))
          (let
              ((section-name (cadr component))
               (header-file (concat dir "_index.md")))
```

```
            (find-file header-file)
            (erase-buffer)
            (insert (format "+++\ntitle = \"%s\"\nweight = %d\n+++\n\n"
                            section-name index))
            (insert "__See next page ->__\n")

            (save-buffer)
            (kill-buffer)))))))

(defun org-hugo-export-section-2-file ()
  "Export current \"org-mode\" section to org-mode file.
Add yaml header for hugo."
  ;; (message "EXPORTING!!! filename: %s, level: %d, foldername: %s, components: %s"
  ;;          filename level foldername folder-components)
  ;; (message "subfolderpath: %s"
  ;;          (apply 'concat (-take (1- level) (-map 'car folder-components))))
  ;; (message "full path: %s"
  ;;          (concat
  ;;           root-dir
  ;;           (apply 'concat (-take (1- level) (-map 'car folder-components)))
  ;;           filename ".org"))
  ;; (goto-char (plist-get element :begin))
  ;; (org-copy-subtree)
  (find-file (format "%s%03d-%s.org"
                     (concat
                      root-dir
                      (apply 'concat (-take (1- level) (-map 'car folder-components))
                      )
                     index filename))
  (erase-buffer)
  ;; (message "I am now in buffer: %s" (current-buffer))
  (insert contents)
  (goto-char (point-min))
  (re-search-forward ":END:")
  (kill-region (point-min) (point))
  (insert
   (format "+++\ntitle = \"%s\"\nweight = %d\n+++" title index))
  (-dotimes level (lambda (n) (org-map-entries 'org-promote)))
  (save-buffer)
  (setq buffers-to-delete (cons (current-buffer) buffers-to-delete)))
```

```
(defun org-hugo-select-filenames ()
  "Build sparse tree with entries whose property filename is set."
  (interactive)
  (org-match-sparse-tree nil "filename={[^]}"))

(eval-after-load 'org
  '(progn
     (define-key org-mode-map (kbd "C-c C-h C-e") 'org-hugo-export)
     (define-key org-mode-map (kbd "C-c C-h C-a") 'org-hugo-autosplit)
     (define-key org-mode-map (kbd "C-c C-h C-/") 'org-hugo-select-filenames)))
```

27 org-export-recipes

```
;;; Commentary:

;; define concenience function for selecting a recipe file
;; from this folder and insering it in the org file to configure export.

;;; Code:

;; (prelude-load-require-package 'helm)

(require 'helm)

(defconst org-export-recipe-folder (concat
                                    (file-name-directory load-file-name)
                                    "/recipes/"))

(defconst org-export-snippet-folder (concat
                                     (file-name-directory load-file-name)
                                     "/snippets/"))

(defconst org-lisp-snippet-folder (concat
                                      (file-name-directory load-file-name)
                                      "/elisp-snippets/"))

(defun org-export-insert-recipe (snippet-p)
  "Select and insert file from recipes or snippet to org-mode file.
```

```
If SNIPPET-P then insert snippet from snippet folder in text at point.
Else insert recipe from recipe folder at top of file."
  (interactive "P")
  (if snippet-p
      (insert-file-contents
       (helm-read-file-name "select snippet:" :initial-input org-export-snippet-folde
    (save-excursion
      (goto-char 0)
      (insert-file-contents  (helm-read-file-name "select recipe:" :initial-input org

(defun org-load-lisp-snippet (open-p)
  "Load or edit file from elisp-snippets folder.
If OPEN-P then open snippet for editing instead of loading it.
Else insert recipe from recipe folder at top of file."
  (interactive "P")
  (if open-p
      (find-file
     (helm-read-file-name "edit snippet:" :initial-input org-lisp-snippet-folder))
    (load
       (helm-read-file-name "load snippet:" :initial-input org-lisp-snippet-folder)))

(global-set-key (kbd "H-c i") 'org-export-insert-recipe)
(global-set-key (kbd "H-c l") 'org-load-lisp-snippet)
```

28 unset-command-q

```
;;; Commentary:
;;; disable command-q key to avoid inadvertently quitting EMACS.

;;; Code:
(global-set-key (kbd "s-q") nil)
```

29 tiny menu

Design draft for next version:

search/files	
(icicles jump)	goto section
	recent files
	projectile
	dired
	set bookmark
	goto bookmark
	icy on
	icy off
journal/agenda	
	agenda
	todos
	agenda menu
	journal make entry
	journal goto entry
sc lang	
	start sclang
	recompile
	workspace
	post window
	clear post window
sc server	
	set io channels
	boot server
	quit server
	kill servers
sc utils	
	set rec channels
	start recording
	stop recording
	meter
	scope
	freqscope

```
;;; Commentary:

;; test code for using tiny-menu
;; from: https://blog.aaronbieber.com/2016/07/31/org-navigation-revisited.html

;;; Code:
```

```
;; (let ((projectile-switch-project-action 'projectile-find-file))
;;   (projectile-switch-project-by-name "/Users/iani/BitTorrent Sync/000WORKFILES/"))

(prelude-load-require-package 'tiny-menu)

(defun air--org-global-custom-ids ()
  "Find custom ID fields in all org agenda files."
  (let ((files (org-agenda-files))
        file
        air-all-org-custom-ids)
    (while (setq file (pop files))
      (with-current-buffer (org-get-agenda-file-buffer file)
        (save-excursion
          (save-restriction
            (widen)
            (goto-char (point-min))
            (while (re-search-forward "^[ \t]*:CUSTOM_ID:[ \t]+\\(\\S-+\\)[ \t]*$"
                                      nil t)
              (add-to-list 'air-all-org-custom-ids
                           `(,(match-string-no-properties 1)
                             ,(concat file ":" (number-to-string (line-number-at-pos)
    air-all-org-custom-ids))

(defun air-org-goto-custom-id ()
  "Go to the location of a custom ID, selected interactively."
  (interactive)
  (let* ((all-custom-ids (air--org-global-custom-ids))
         (custom-id (completing-read
                     "Custom ID: "
                     all-custom-ids)))
    (when custom-id
      (let* ((val (cadr (assoc custom-id all-custom-ids)))
             (id-parts (split-string val ":"))
             (file (car id-parts))
             (line (string-to-int (cadr id-parts))))
        (pop-to-buffer (org-get-agenda-file-buffer file))
        (goto-char (point-min))
        (forward-line line)
        (org-reveal)
```

```
        (org-up-element)))))

;; The helm menu does not update when changing the org-refile-targets variable like t
;; Switch to icicle mode as a workaround.
(defun air-goto-section ()
  "Set refile targets to current buffer and call org-refile with 1 u prefix."
  (interactive)
  (let ((org-refile-targets `((,buffer-file-name :maxlevel . 10))))
    (icy-mode 1)
    (org-refile '(4))
    (icy-mode -1)))

(defun air-turn-icicles-on ()
  "Turn icicle mode on."
  (interactive)
  (icy-mode 1))

(defun air-turn-icicles-off ()
  "Turn icicle mode off."
  (interactive)
  (icy-mode 0))

(defun air-journal-goto-date ()
  "Jump to journal at date from user"
  (interactive)
  (org-journal-at-date-from-user '(4)))

(defun sclang-kill-all-servers ()
  "Kill all supercollider servers."
  (interactive)
  (sclang-eval-string "Server.killAll" t))

(defvar sclang-num-recording-chans 2
  "Default number of recording channels in supercollider")

(defvar sclang-num-input-chans 2
  "Default number of audio input channels in supercollider")

(defvar sclang-num-output-chans 2
  "Default number of audio output channels in supercollider")
```

```
(defun sclang-sclang-set-io-channels ()
  "Kill all supercollider servers."
  (interactive)
  (setq sclang-num-input-chans
        (eval-minibuffer "number of input channels: " (format "%d" sclang-num-input-c
  (setq sclang-num-output-chans
        (eval-minibuffer "number of output channels: " (format "%d" sclang-num-output
  (sclang-eval-string (format
                       "Server.default.options.numOutputBusChannels = %d"
                       sclang-num-output-chans) t)
  (sclang-eval-string (format
                       "Server.default.options.numInputBusChannels = %d"
                       sclang-num-input-chans) t))

(defun sclang-start-recording ()
  "Kill all supercollider servers."
  (interactive)
  (setq sclang-num-recording-chans
        (eval-minibuffer "number of channels to record: " (format "%d" sclang-num-rec
  (sclang-eval-string (format "Server.default.record(%d)" sclang-num-recording-chans)

(defun sclang-stop-recording ()
  "Kill all supercollider servers."
  (interactive)
  (sclang-eval-string "Server.killAll" t))

(defun air-tiny-menu ()
    "My custom tiny menu."
    (interactive)
    (let ((tiny-menu-items
           '(("search/files" ("search/files"
                              ((?g "goto-section" air-goto-section)
                               (?r "recent files" crux-recentf-ido-find-file)
                               (?p "projectile" helm-projectile-switch-project)
                               (?d "dired" dired)
                               (?g "goto bookmark" bookmark-jump)
                               (?s "set bookmark" bookmark-set)
                               (?1 "icy on" air-turn-icicles-on)
                               (?0 "icy off" air-turn-icicles-off))))
```

```
            ("journal/agenda" ("journal/agenda"
                               ((?a "agenda" org-agenda-list)
                                (?t "todos" org-todo-list)
                                (?m "agenda menu" org-agenda)
                                (?n "new journal entry" org-journal-at-date-from-use
                                (?g "goto journal entry" air-journal-goto-date))))
            ("sc lang" ("sc lang"
                        ((?s "start" sclang-start)
                         (?q "quit" sclang-stop)
                         (?r "recompile" sclang-recompile)
                         (?w "workspace" sclang-switch-to-workspace)
                         (?p "post window" sclang-show-post-buffer)
                         (?c "clear post window" sclang-clear-post-buffer))))
            ("sc server" ("sc server"
                          ((?i "set io channels" sclang-set-io-channels)
                           (?b "boot" sclang-server-boot)
                           (?q "quit server" sclang-server-quit)
                           (?k "kill all servers" sclang-kill-all-servers))))
            ("sc utils" ("sc utils"
                         ((?1 "start recording" sclang-start-recording)
                          (?0 "stop recording" sclang-stop-recording)
                          (?m "meter" sclang-server-meter)
                          (?s "scope" sclang-server-scope)
                          (?f "freqscope" sclang-server-freqscope)))))))
      (tiny-menu)))
;; (defun air-tiny-menu ()
;;     "My custom tiny menu."
;;     (interactive)
;;     (let ((tiny-menu-items
;;            '(("agenda" ("agenda"
;;                         ((?a "agenda" org-agenda-list)
;;                          (?A "agenda menu" org-agenda)
;;                          (?t "todo" org-todo-list))))
;;              ("workfiles" ("workfiles"
;;                            ((?c "commander" projectile-commander-workfiles)
;;                             ;; (?s "ag" projectile-ag-workfiles)
;;                             (?v "magit" projectile-vc-workfiles)
;;                             (?f "find file" projectile-find-file-workfiles)
;;                             (?r "recent" projectile-recent-files-workfiles)
;;                             (?d "dired" projectile-dired-workfiles)
```

```
;;                                (?D "root dired" projectile-root-dired-workfiles))))
;;              ("projects" ("projects"
;;                           ((?c "commander" projectile-commander-projects)
;;                            ;; (?s "ag" projectile-ag-projects)
;;                            (?v "magit" projectile-vc-projects)
;;                            (?f "find file" projectile-find-file-projects)
;;                            (?r "recent" projectile-recent-files-projects)
;;                            (?d "dired" projectile-dired-projects)
;;                            (?D "root dired" projectile-root-dired-projects))))
;;              ("files" ("files"
;;                        ((?i "icy imenu" icicle-imenu)
;;                         (?l "org jump local" air-refile-goto-current-buffer)
;;                         (?r "recent files" helm-recentf)
;;                         (?j "projects refile jump" org-jump-to-refile-target)
;;                         (?w "Workfiles find file" projectile-find-file-workfiles)
;;                         (?W "Workfiles projectile commander" projectile-commander-
;;              ("stuff"   ("stuff"
;;                          ((?t "Tag"     org-tags-view)
;;                           (?i "ID"      air-org-goto-custom-id)
;;                           (?k "Keyword" org-search-view)
;;                           (?s "SuperCollider" sclang-start))))
;;              ("icicles"   ("icicles"
;;                            ((?1 "icy on" air-turn-icicles-on)
;;                             (?0 "icy off" air-turn-icicles-off))))
;;              ("org-links"    ("Links"
;;                               ((?c "Capture"   org-store-link)
;;                                (?l "Insert"    org-insert-link)
;;                                (?i "Custom ID" air-org-insert-custom-id-link))))))
;;       (tiny-menu)))

;; (setq tiny-menu-items
;;       '(("org-things"   ("Things"
;;                          ((?t "Tag"     org-tags-view)
;;                           (?i "ID"      air-org-goto-custom-id)
;;                           (?k "Keyword" org-search-view)
;;                           (?l "Refile Goto Local" air-refile-goto-current-buffer)
;;                           )))
;;         ("org-links"    ("Links"
;;                          ((?c "Capture"   org-store-link)
;;                           (?l "Insert"    org-insert-link)
```

```
;;                              (?i "Custom ID" air-org-insert-custom-id-link))))))

;; (global-set-key (kbd "C-H-M-t") 'air-tiny-menu)
;; s-m is set by prelude/magit to magit commands that I do not use or plan to use dir
(global-set-key (kbd "H-m") 'air-tiny-menu)
```

:DATE: *¡2018-01-04 Thu 09:27¿*

30 Dired-hide-details

```
;;; Commentary:
;; HIDE DETAILS WHEN FIRST OPENING DIRED

;; Note: following does not work. Why?
;; (setq dired-hide-details-mode t)

;; Using dired+ opens dired without details per default

;;; Code:

(prelude-load-require-package 'dired+)
```

31 projectile

```
;;; Commentary:
;;; some useful extensions to projectile
;;; helm-projectile
;;; Note: neither perspective nor helm-perspective work for me.

;;; Code:
;; (prelude-load-require-packages '(perspective helm-projectile persp-projectile))
(prelude-load-require-packages '(helm-projectile))
(helm-projectile-on)
(setq projectile-switch-project-action #'projectile-commander)
;; (persp-mode)
;; (require 'persp-projectile)
;; (define-key projectile-mode-map (kbd "s-s") 'projectile-persp-switch-project)
```

32 calendar

```
;;; Commentary:

;; Tweak Emacs built-in calendar

;;; Code:

(require 'calendar)

(global-set-key (kbd "C-c c C-c") 'calendar)

;;; Override old calendar-goto-date to use org-read-date, since
;;; the latter is much more convenient.
;;; Unfortunately will not work if new date is not displayed in current calendar.
(defun calendar-goto-date-org-style (date)
  "Move cursor to DATE."
  (interactive (list (let ((date (org-parse-time-string (org-read-date))))
                       (list
                        (nth 4 date)
                        (nth 3 date)
                        (nth 5 date)))))
  (let ((month (calendar-extract-month date))
        (year (calendar-extract-year date)))
    (if (not (calendar-date-is-visible-p date))
        (calendar-other-month
         (if (and (= month 1) (= year 1))
             2
           month)
         year)))
  (calendar-cursor-to-visible-date date)
  (run-hooks 'calendar-move-hook)
  ;; make cursor visible again (otherwise it disappears:)
  (setq cursor-type "box"))

;;; (global-set-key (kbd "C-c c C-o") 'calendar-goto-date-org-style)

;;; provide 025_calendar
;;; 025_calendar.el ends here
```

33 greek input

```
;; (global-set-key (kbd "C-c C-\\") 'toggle-input-method)

(setq default-input-method "greek")
(global-set-key (kbd "s-;") 'toggle-input-method)
(global-set-key (kbd "s-\\") 'toggle-input-method)
```

34 window and buffer switching ace window

```
;;; Commentary:
;;; move amngst windows and switch window position with cursor keys

(prelude-load-require-package 'buffer-move)
;; (require 'windmove) required by buffermove
;; (winner-mode -1)
(global-set-key (kbd "s-<left>")  'windmove-left)
(global-set-key (kbd "s-<right>") 'windmove-right)
(global-set-key (kbd "s-<up>")    'windmove-up)
(global-set-key (kbd "s-<down>")  'windmove-down)
(global-set-key (kbd "s-S-<up>")     'buf-move-up)
(global-set-key (kbd "s-S-<down>")   'buf-move-down)
(global-set-key (kbd "s-S-<left>")   'buf-move-left)
(global-set-key (kbd "s-S-<right>")  'buf-move-right)
(global-set-key (kbd "C-O")  'ace-window)
(global-set-key (kbd "C-x o")  'ace-window)
(setq aw-keys '(?a ?b ?c ?d ?e ?f ?g ?h ?i ?j ?k ?l ?m ?n ?o ?p ?q))

;; (require 'use-package)
;; (use-package
;;  ace-window
;;  :ensure ace-window
;;  :config (setq aw-keys '(?a ?o ?e ?u ?i ?d ?h ?t ?n ?s))
;;  :bind ("C-x o") . ace-window)
```

35 org-journal

```
;;; Commentary:
```

```
;;; use org-journal for capture globally.
;;; https://github.com/bastibe/org-journal

;;; Code:

(prelude-load-require-package 'org-journal)

;; Create files with .org ending to automatically enable org-mode when loading them:
(setq org-journal-file-format "%Y%m%d.org")

(defun org-journal-new-entry-from-org-timestamp ()
  "Like org-journal-new-entry except read time interactively using org-read-date."
  (interactive)
  (org-journal-new-entry nil (apply 'encode-time (org-parse-time-string (org-read-dat

;; Overwrite custom setting of var:
(setq org-journal-dir (file-truename "~/Documents/000WORKFILES/PERSONAL/journal"))

;; adding own custom var to journal group, using template from journal mode.
(defcustom org-todo-dir (file-truename "~/Documents/000WORKFILES/PERSONAL/TODOS")
  "Directory containing journal entries.
  Setting this will update auto-mode-alist using
  `(org-journal-update-auto-mode-alist)`"
  :type 'string :group 'org-journal
  :set (lambda (symbol value)
         (set-default symbol value)
         (org-journal-update-auto-mode-alist)))

(defcustom org-projects-dir (file-truename "~/Documents/000WORKFILES/PROJECTS_CURRENT
  "Directory containing project entries.
  Setting this will update auto-mode-alist using
  `(org-journal-update-auto-mode-alist)`"
  :type 'string :group 'org-journal
  :set (lambda (symbol value)
         (set-default symbol value)
         (org-journal-update-auto-mode-alist)))
;; provide custom refile targets for todo entries
;; NOTE: This function is also used in custom function org-jump-to-refile-target.
(defun org-iz-make-refile-targets ()
"Provide custom refile targets for todo entries.
```

```
This function is also used in custom function org-jump-to-refile-target."
(setq org-refile-targets
      (append
       (mapcar (lambda (x) (cons x '(:maxlevel . 2)))
               (file-expand-wildcards (concat org-todo-dir "/*.org")))
       (mapcar (lambda (x) (cons x '(:maxlevel . 2)))
               (file-expand-wildcards (concat org-projects-dir "/*.org")))))

;; Include all journal and todo files in agenda:
(setq org-agenda-files `(,org-journal-dir
                         ,org-todo-dir
                         ,org-projects-dir))

(defun org-journal-at-date-from-user (no-entry)
  "Creat journal entry with date from user, NO-ENTRY prefix just opens the file witho
  (interactive "P")
  (let ((time (org-read-date t t)) timestamp)
    (org-journal-new-entry no-entry time)
    (setq timestamp (format-time-string (cdr org-time-stamp-formats) time))
    ;; (if no-entry
    ;;     (insert "\n" timestamp))
    (unless no-entry
     (progn
       (insert
        "\n :PROPERTIES:\n :DATE: "
        timestamp
        " \n :END:\n")
       (previous-line 4)
       (end-of-line)))))

;; Make new-entry keyboard command available also in org-mode:
(global-set-key (kbd "C-c c j") 'org-journal-at-date-from-user)
(global-set-key (kbd "C-c c J") 'org-journal-new-entry-from-org-timestamp)
```

36 re-builder

```
;;; see https://www.masteringemacs.org/article/re-builder-interactive-regexp-builder
(prelude-load-require-package 're-builder)
(setq reb-re-syntax 'string)
```

37 turn-whitespace-off

```
(whitespace-mode -1)
;; (toggle-frame-fullscreen)
```

38 last actions 1 window air tiny menu

```
(delete-other-windows)
(air-tiny-menu)
```