

Emacs Initialization

John Jacobsen

November 20, 2014

Contents

1	Introduction	2
2	Utilities	2
3	Package management	2
4	Manually installed packages	3
5	Startup	3
6	Initial keybindings	3
7	Remembering Where We Were Last Time Around	3
8	General Lisp stuff	3
9	Stuff for Editing Emacs Lisp	4
10	Ruby stuff	4
11	Backups	4
12	File Completion	4
13	Autocomplete Mode	4
14	λ	5
15	Highlighting of long lines	5
16	Lots of keybindings	5
17	Clojure setup	7
17.1	Key bindings special to Midje facts	7
17.2	Correcting single-whitespaced toplevel forms	9
17.3	Mode line hack	10
18	Stuff for running shells within Emacs	10
18.1	Path Magic	10
18.2	Moar Shells	10
18.3	Kill shell buffers quickly	11

19 Stuff related to configuring Emacs-in-a-window	11
20 Common Lisp	11
21 Magit stuff	12
22 Org Mode	12
23 Blogging	13
24 Tidying up	15

1 Introduction

Here's my Emacs init code. Some of it was adapted from Matthew Wampler-Doty. Other things have been borrowed from Sacha Chua and other places around the Internetz.

You'll find this code highly focused on Clojure and Org Mode. I have a lot of unconventional keybindings since I didn't bother to learn all the classic ones when I started with Emacs (sometime before *cough* 1997).

2 Utilities

Matt wrote a Clojure-like `filter` function. Probably I'll write more like it as I need them, since I am more used to Clojure than Emacs Lisp. (Why else would one bother with a "programmable programming language"¹, if not to tailor one's language to one's liking?)

```
(defun filter (pred lst)
  "Use PRED to filter a list LST of elements."
  (delq nil (mapcar (lambda (x) (and (funcall pred x) x)) lst)))
```

3 Package management

MELPA is a bit too bleeding edge; since CIDER has broken for me before after upgrading, I use the more stable releases from `melpa-stable` only. To get back MELPA(-bleeding-edge), swap in the following:

```
;; (add-to-list 'package-archives
;;            '("MELPA" . "http://melpa.milkbox.net/packages/") t)
```

Define which remote archives to pull from, and which packages to use:

```
(require 'package)
(add-to-list
 'package-archives
 '("melpa-stable" . "http://melpa-stable.milkbox.net/packages/") t)
(add-to-list 'package-archives
 '("marmalade" . "http://marmalade-repo.org/packages/") t)
(package-initialize)
(defvar my-packages)
(setq my-packages
 '(company auto-complete autopair ac-cider cider color-theme
  zenburn-theme diminish goto-last-change main-line maxframe
```

¹John Foderaro, CACM, Sept. 1991.

```
clojure-mode epl popup rainbow-delimiters smex undo-tree
flycheck flycheck-hdevtools org git-timemachine paredit
auto-indent-mode slamhound lorem-ipsuM midje-mode
hungry-delete metar helm htmlize magit expand-region))
```

Install any missing packages:

```
(let ((uninstalled-packages (filter (lambda (x) (not (package-installed-p x)))
  my-packages)))
  (when (and (not (equal uninstalled-packages '()))
    (y-or-n-p (format "Install packages %s?" uninstalled-packages)))
    (package-refresh-contents)
    (mapc 'package-install uninstalled-packages)))
```

4 Manually installed packages

See http://ergoemacs.org/emacs/emacs_installing_packages.html.

```
(add-to-list 'load-path "~/.emacs.d/hand-installed/")
```

5 Startup

Some configuration to make things quieter on start up:

```
(setq inhibit-splash-screen t
  initial-scratch-message nil)
```

6 Initial keybindings

At some point I took it into my head to map most of my personal key bindings to combinations starting with `\C-o`. We have to set it free from its usual use (`open-line`) before we can use it as a prefix.

```
(global-unset-key "\C-o")
```

7 Remembering Where We Were Last Time Around

```
(require 'saveplace)
(setq-default save-place t)
(setq save-place-file "~/.emacs.d/saved-places")
```

8 General Lisp stuff

Rainbow delimiters for all programming major modes:

```
(require 'rainbow-delimiters)
(add-hook 'prog-mode-hook 'rainbow-delimiters-mode)
```

Show paren balancing nicely:

```
(require 'paren)
(set-face-background 'show-paren-match "white")
(add-hook 'prog-mode-hook 'show-paren-mode)
```

9 Stuff for Editing Emacs Lisp

I add a hook for evaluating the expression just before point; I've played with `auto-indent-mode` and `flycheck-mode` but tired of them. I do want `paredit` though (and therefore don't want `autopair-mode`).

```
(define-key emacs-lisp-mode-map (kbd "<s-return>") 'eval-last-sexp)

;;(add-hook 'emacs-lisp-mode-hook 'flycheck-mode)
;;(add-hook 'emacs-lisp-mode-hook 'auto-indent-mode)
(add-hook 'emacs-lisp-mode-hook
  (lambda ()
    (paredit-mode 1)
    (autopair-mode 0)))
```

10 Ruby stuff

Sometimes I have to write Rails or Ruby code. You might, too. Do two space indents for Ruby code:

```
(setq ruby-indent-level 2)
```

11 Backups

Tell Emacs to write backup files to their own directory, and make backups even for files in revision control:

```
(setq backup-directory-alist
  '(("." . ,(expand-file-name
    (concat user-emacs-directory "backups")))))

(setq vc-make-backup-files t)
```

12 File Completion

Trying out Helm²:

```
(require 'helm-config)
(helm-mode)
```

13 Autocomplete Mode

This is one of several available auto-complete modes for Emacs. It's basic but I've found it more reliable than `company-mode`.

```
(require 'auto-complete)
(add-hook 'prog-mode-hook 'auto-complete-mode)
```

²<http://emacs-helm.github.io/helm/>

14 λ

Make lambda show as λ :

```
(font-lock-add-keywords
 'emacs-lisp-mode
 '(("\\(\\(lambda\\)\\)\\>"
   (0 (prog1 ()
        (compose-region (match-beginning 1)
                        (match-end 1)
                        ?))))))
```

15 Highlighting of long lines

```
(defun highlight-long-lines ()
  "Turn on highlighting of long lines."
  (interactive)
  (highlight-lines-matching-regexp "\\{81\\}" 'hi-pink))
```

```
(defun unhighlight-long-lines ()
  "Turn off highlighting of long lines."
  (interactive)
  (unhighlight-regexp "^.*\\(?:\\.\\{81\\}\\)\\.*$"))
```

```
;(global-set-key "\C-oH" 'highlight-long-lines)
;(global-set-key "\C-oh" 'unhighlight-long-lines)
```

16 Lots of keybindings

Many of these are extremely old, having followed me from machine to machine over the years. Some could probably be deleted.

```
(global-set-key [S-deletechar] 'kill-ring-save)
;; Set up the keyboard so the delete key on both the regular keyboard
;; and the keypad delete the character under the cursor and to the right
;; under X, instead of the default, backspace behavior.
(global-set-key [delete] 'delete-char)
(global-set-key [kp-delete] 'delete-char)
```

```
(define-key function-key-map "\e[1~" [find])
(define-key function-key-map "\e[2~" [insertchar])
(define-key function-key-map "\e[3~" [deletechar])
(define-key function-key-map "\e[4~" [select])
(define-key function-key-map "\e[5~" [prior])
(define-key function-key-map "\e[6~" [next])
(define-key global-map [select] 'set-mark-command)
(define-key global-map [insertchar] 'yank)
(define-key global-map [deletechar] 'kill-region)
```

```
(global-unset-key "\C- ")
```

```

(global-set-key [?\C- ] 'other-window)
(global-set-key "\C-oW" (lambda ()
  (interactive)
  (org-babel-load-file (concat user-emacs-directory "org/init.org"))))
(global-set-key "\C-A" 'split-window-horizontally)
(global-set-key "\C-oa" 'split-window-vertically)
(global-set-key "\C-K" 'kill-line)
(global-set-key "\C-os" 'isearch-forward-regexp)
(global-set-key "\C-oD" 'find-name-dired)
(global-set-key "\C-xS" 'sort-lines)
(global-set-key "\C-w" 'backward-kill-word)
(global-set-key "\C-x\C-k" 'kill-region)
(global-set-key "\C-c\C-k" 'kill-region)
(global-set-key "\C-ok" 'comment-region)
(global-set-key "\C-ou" 'uncomment-region)
(global-set-key "\C-on" 'er/expand-region)
(global-set-key "\C-oe" 'eval-current-buffer)
(global-set-key "\C-od" 'delete-horizontal-space)
(global-set-key "\C-ob" 'backward-word)
(global-set-key "\C-oq" 'query-replace-regexp)
(global-set-key "\C-oL" 'lorem-ipsum-insert-paragraphs)
(global-set-key "\C-]" 'fill-region)
(global-set-key "\C-ot" 'beginning-of-buffer)
(global-set-key "\C-oT" 'toggle-window-split)
(global-set-key "\C-N" 'enlarge-window)
(global-set-key "\C-o\C-n" 'enlarge-window-horizontally)
(global-set-key "\C-oc" 'paredit-duplicate-closest-sexp)
(global-set-key "\C-ol" 'goto-line)
(global-set-key "\C-ob" 'end-of-buffer)
(global-set-key "\C-op" 'fill-region)
(global-set-key "\C-og" 'save-buffers-kill-emacs)
(global-set-key "\C-od" 'downcase-region)
(global-set-key "\C-oR" 'indent-region)
(global-set-key "\C-or" 'rgrep)
(global-set-key "\C-L" 'delete-other-windows)
(global-set-key "\C-B" 'scroll-down)
(global-set-key "\C-F" 'scroll-up)
(global-set-key "\C-V" 'save-buffer)
(global-set-key "\C-R" 'isearch-forward)
(global-set-key "\C-^" 'wnt-alog-add-entry)
(global-set-key "\C-T" 'set-mark-command)
(global-set-key "\C-Y" 'yank)
(global-set-key "\C-D" 'backward-delete-char-untabify)
(global-set-key "\C-\\\" 'shell)
(global-set-key "\C-oi" 'quoted-insert)
(global-set-key "\e[1~" 'isearch-forward)
(global-set-key [select] 'set-mark-command)
(global-set-key [insertchar] 'yank)
(global-set-key [deletechar] 'kill-region)
(global-set-key "\C-\\\" 'shell)

```

```
(global-set-key "\C-oi" 'quoted-insert)
(global-set-key "\e[1~" 'isearch-forward)
(global-set-key [select] 'set-mark-command)
(global-set-key [insertchar] 'yank)
(global-set-key [deletechar] 'kill-region)
(global-set-key (kbd "s-0") 'org-todo-list)
```

Shortcuts for jumping directly into most commonly-used buffers:

```
(global-set-key "\C-o0" (lambda ()
  (interactive)
  (find-file "~/Dropbox/org/toplevel.org")))
(global-set-key "\C-oE" (lambda ()
  (interactive)
  (find-file "~/.emacs.d/org/init.org")))
```

Keyboard shortcuts for joining lines before and after point (thanks to <http://whattheemacs.com/> for the (join-line -1) trick):

```
(global-set-key (kbd "M-j")
  (lambda () (interactive) (join-line -1)))
(global-set-key "\C-oo" 'join-line)
```

Show trailing whitespace, ‘cause *we hates it...*

```
(setq-default show-trailing-whitespace t)
```

17 Clojure setup

Don’t go to REPL buffer when starting Cider:

```
(setq cider-repl-pop-to-buffer-on-connect nil)
```

17.1 Key bindings special to Midje facts

Set up Midje fact with mark inserted at beginning of comment text (refill as needed in appropriate columns, using C-oF).

```
(global-set-key "\C-of" (lambda ()
  (interactive)
  (insert "(fact"                                     "\"\"\"n\n  )")
  (backward-char 6)
  (set-mark (point)))))
```

Perform the refill operation for the text string in a Midje fact:

```
(global-set-key "\C-oF" (lambda ()
  (interactive)
  (set-left-margin (mark) (point) 37)
  (fill-region (mark) (point)))))
```

Append result of evaluating previous expression (Clojure):

```

(defun cider-eval-last-sexp-and-append ()
  "Evaluate the expression preceding point and append result."
  (interactive)
  (let ((last-sexp (cider-last-sexp)))
    ;; we have to be sure the evaluation won't result in an error
    (cider-eval-and-get-value last-sexp)
    (with-current-buffer (current-buffer)
      (insert ";;=>\n"))
    (cider-interactive-eval-print last-sexp)))

(defun cider-format-with-out-str-pprint-eval (form)
  "Return a string of Clojure code that will return pretty-printed FORM."
  (format "(clojure.core/let [x %s] (with-out-str (clojure.pprint/pprint x)))" form))

(defun cider-eval-last-sexp-and-pprint-append ()
  "Evaluate the expression preceding point and append pretty-printed result."
  (interactive)
  (let ((last-sexp (cider-last-sexp)))
    ;; we have to be sure the evaluation won't result in an error
    (with-current-buffer (current-buffer)
      (insert ";;=>\n")
      (insert (cider-eval-and-get-value (cider-format-with-out-str-pprint-eval last-sexp))))))

;; A few paredit things, also from whattheemacs.com:
(defun paredit--is-at-start-of-sexp ()
  (and (looking-at "(\\|\\|\\|\\|")
       (not (nth 3 (syntax-ppss))) ;; inside string
       (not (nth 4 (syntax-ppss)))) ;; inside comment

(defun paredit-duplicate-closest-sexp ()
  (interactive)
  ;; skips to start of current sexp
  (while (not (paredit--is-at-start-of-sexp))
    (paredit-backward))
  (set-mark-command nil)
  ;; while we find sexps we move forward on the line
  (while (and (bounds-of-thing-at-point 'sexp)
              (<= (point) (car (bounds-of-thing-at-point 'sexp)))
              (not (= (point) (line-end-position))))
    (forward-sexp)
    (while (looking-at " ")
      (forward-char)))
  (kill-ring-save (mark) (point))
  ;; go to the next line and copy the sexprs we encountered
  (paredit-newline)
  (yank)
  (exchange-point-and-mark))

```


17.2 Correcting single-whitespace toplevel forms

```
(defun correct-single-whitespace ()
  "Correct single-spaced Lisp toplevel forms."
  (interactive)
  (goto-char 1)
  (while (search-forward-regexp "\\n\\n(" nil t)
    (replace-match "\\n\\n\\n(" t nil)))
  (global-set-key "\C-oQ" 'correct-single-whitespace)

  (add-hook 'clojure-mode-hook
    '(lambda ()
      (paredit-mode 1)
      (highlight-long-lines)
      (define-key clojure-mode-map (kbd "C-c e") 'shell-eval-last-expression)
      (define-key clojure-mode-map (kbd "C-o x") 'cider-eval-defun-at-point)
      (define-key clojure-mode-map (kbd "C-o j") 'cider-jack-in)
      (define-key clojure-mode-map (kbd "C-o J") 'cider-restart)
      (define-key clojure-mode-map (kbd "C-<up>") 'paredit-backward)
      (define-key clojure-mode-map (kbd "C-<down>") 'paredit-forward)
      (define-key clojure-mode-map (kbd "C-o y")
        'cider-eval-last-sexp-and-append)
      (define-key clojure-mode-map (kbd "C-o Y")
        'cider-eval-last-sexp-and-pprint-append)
      (define-key clojure-mode-map (kbd "s-i") 'cider-eval-last-sexp)
      (define-key clojure-mode-map (kbd "C-c x") 'shell-eval-defun)))

;; Minibuffer size
(add-hook 'minibuffer-setup-hook 'my-minibuffer-setup)
(defun my-minibuffer-setup ()
  (set (make-local-variable 'face-remapping-alist)
    '((default :height 1.5))))

;;; Swap window split orientation
;;; (http://emacs.stackexchange.com/questions/318/switch-window-split-orientation-fastest-way):
(defun toggle-window-split ()
  (interactive)
  (if (= (count-windows) 2)
    (let* ((this-win-buffer (window-buffer))
           (next-win-buffer (window-buffer (next-window)))
           (this-win-edges (window-edges (selected-window)))
           (next-win-edges (window-edges (next-window)))
           (this-win-2nd (not (and (<= (car this-win-edges)
                                       (car next-win-edges))
                                   (<= (cadr this-win-edges)
                                       (cadr next-win-edges))))))
      (splitter
        (if (= (car this-win-edges)
              (car (window-edges (next-window))))
          'split-window-horizontally
          'split-window-vertically)))
    (delete-other-windows))
```

```
(let ((first-win (selected-window)))
  (funcall splitter)
  (if this-win-2nd (other-window 1))
  (set-window-buffer (selected-window) this-win-buffer)
  (set-window-buffer (next-window) next-win-buffer)
  (select-window first-win)
  (if this-win-2nd (other-window 1))))))
```

17.3 Mode line hack

Shorten clojure-mode in mode line³.

```
(defmacro rename-modeline (package-name mode new-name)
  '(eval-after-load ,package-name
    '(defadvice ,mode (after rename-modeline activate)
      (setq mode-name ,new-name))))

(rename-modeline "clojure-mode" clojure-mode "Clj")
```

18 Stuff for running shells within Emacs

18.1 Path Magic

Smooth the waters for starting processes from the shell. “Set up Emacs’ ‘exec-path’ and PATH environment variable to match the user’s shell. This is particularly useful under Mac OSX, where GUI apps are not started from a shell⁴.”

```
(defun set-exec-path-from-shell-PATH ()
  (interactive)
  (let ((path-from-shell
        (replace-regexp-in-string
         "[ \\t\\n]*$" ""
         (shell-command-to-string "$SHELL --login -i -c 'echo $PATH'"))))
    (setenv "PATH" path-from-shell)
    (setq exec-path (split-string path-from-shell path-separator))))
```

18.2 Moar Shells

Create shell in new buffer when needed, rather than just loading up the existing shell buffer.

```
(defun create-shell-in-new-buffer ()
  (interactive)
  (let ((currentbuf (get-buffer-window (current-buffer))))
    (newbuf (generate-new-buffer-name "*shell*")))
    (generate-new-buffer newbuf)
    (set-window-dedicated-p currentbuf nil)
    (set-window-buffer currentbuf newbuf)
    (shell newbuf)))

(global-set-key "\C-oS" 'create-shell-in-new-buffer)
```

³From <http://whattheemacs.com/>

⁴See <http://stackoverflow.com/questions/8606954/path-and-exec-path-set-but-emacs-does-not-find-executable>

18.3 Kill shell buffers quickly

“With this snippet, [a second] press of C-d will kill the buffer. It’s pretty nice, since you then just tap C-d twice to get rid of the shell and go on about your merry way⁵”

```
(defun comint-delchar-or-eof-or-kill-buffer (arg)
  (interactive "p")
  (if (null (get-buffer-process (current-buffer)))
      (kill-buffer)
      (comint-delchar-or-maybe-eof arg)))

(add-hook 'shell-mode-hook
  (lambda ()
    (define-key shell-mode-map
      (kbd "C-d") 'comint-delchar-or-eof-or-kill-buffer)))
```

19 Stuff related to configuring Emacs-in-a-window

When running GUI Emacs (i.e. on OS-X, which is the only way I run Emacs these days anyways), set the theme to Zenburn, turn off visual noise, fix up the PATH for shells, and allow resizing of window.

```
(when window-system
  (load-theme 'zenburn t)
  (tool-bar-mode -1)
  (scroll-bar-mode -1)
  (set-exec-path-from-shell-PATH)
  (global-set-key (kbd "s-=") 'text-scale-increase)
  (global-set-key (kbd "s--") 'text-scale-decrease))
```

Don’t pop up newly-opened files in a new frame – use existing one:

```
(setq ns-pop-up-frames nil)
```

20 Common Lisp

I haven’t done too much Common Lisp programming yet, but have just played around. So far I find Emacs integration to be at least as good as with Clojure. Here I mimic two of the keybindings I use most from Clojure.

```
;; (require 'slime-autoloads)
;; (setq inferior-lisp-program "/usr/local/bin/sbcl")
;; (setq slime-contribs '(slime-fancy))
;; (add-hook 'lisp-mode-hook
;;           '(lambda ()
;;             (paredit-mode 1)
;;             (highlight-long-lines)
;;             (define-key lisp-mode-map (kbd "C-o j") 'slime)
;;             (define-key lisp-mode-map (kbd "s-i")
;;               'slime-eval-last-expression)))
```

⁵From <http://whattheemacs.com>.

21 Magit stuff

Bind key for quick Git status:

```
(global-set-key "\C-om" 'magit-status)
```

22 Org Mode

General setup:

```
(require 'org)
```

Set Clojure backend for literate programming.

```
(setq org-babel-clojure-backend 'cider)
(require 'ob-clojure)
```

Show source code highlighting in code blocks:

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

Allow alphabetical plain lists (a., A., a), A)).

```
(setq org-list-allow-alphabetical t)
```

Put clock in/out timestamps into drawer, so they stay hidden when expanding items.

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

Don't ask for confirmation before evaluating code in these languages (**use at your own risk**):

```
(defun my-org-confirm-babel-evaluate (lang body)
  (and
    (not (string= lang "lisp"))
    (not (string= lang "emacs-lisp"))
    (not (string= lang "clojure"))))
(setq org-confirm-babel-evaluate 'my-org-confirm-babel-evaluate)
```

Clock in/out based on TODO state changes⁶.

```
(eval-after-load 'org
  '(progn
    (defun wicked/org-clock-in-if-starting ()
      "Clock in when the task is marked STARTED."
      (when (and (string= org-state "STARTED")
        (not (string= org-last-state org-state)))
        (org-clock-in)))
    (add-hook 'org-after-todo-state-change-hook
      'wicked/org-clock-in-if-starting)
    (defadvice org-clock-in (after wicked activate)
      "Set this task's status to 'STARTED'."
      (org-todo "STARTED"))
    (defun wicked/org-clock-out-if-waiting ()
      "Clock out when the task is marked WAITING."
```

⁶From <http://sachachua.com/blog/2007/12/clocking-time-with-emacs-org/>

```

    (when (and (string= org-state "WAITING")
    (equal (marker-buffer org-clock-marker) (current-buffer))
    (< (point) org-clock-marker)
    (> (save-excursion (outline-next-heading) (point))
    org-clock-marker)
    (not (string= org-last-state org-state)))
    (org-clock-out)))
    (add-hook 'org-after-todo-state-change-hook
    'wicked/org-clock-out-if-waiting)))

```

Add Markdown export functionality (<http://stackoverflow.com/questions/22988092/emacs-org-mode-export>)

```

(eval-after-load "org"
  '(require 'ox-md nil t))

```

Log when an item goes to DONE state:

```

(setq org-log-done t)

```

Refile things sensibly based on where they occur in original outline:

```

(setq org-refile-targets (quote ((nil :maxlevel . 10)
  (org-agenda-files :maxlevel . 10))))
(setq org-refile-use-outline-path t)
(setq org-outline-path-complete-in-steps nil)
(setq org-refile-allow-creating-parent-nodes (quote confirm))

```

GTD-style TODO states:

```

(setq org-todo-keywords
  '((sequence "TODO" "STARTED" "WAITING" "SOMEDAY" "DONE")))

```

Where to find agenda files:

```

(setq org-agenda-files '("~/Dropbox/org"))

```

Quickly launch agenda:

```

(define-key global-map "\C-ca" 'org-agenda)

```

Use Org's capture system:

```

(setq org-default-notes-file "~/Dropbox/org/toplevel.org")
(define-key global-map "\C-cc" 'org-capture)

```

Export " as “ and ”:

```

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

```

23 Blogging

```

(setq org-sitemap-link-format
  "@@html:<span class='sm-d'>%d</span> &nbsp; &nbsp; &nbsp; <span class='sm-t'>%t</span>@@"
  org-sitemap-html-preamble
  "<link rel=\"stylesheet\" href=\"../css/style.css\" type=\"text/css\" />
  <link rel=\"stylesheet\" href=\"http://code.jquery.com/ui/1.10.3/themes/smoothness/jquery-ui.css\"")

```

```

<link rel="stylesheet" href="https://cdn.jsdelivr.net/bootstrap/3.3.0/css/bootstrap.min.css">
<link rel="stylesheet" href="https://cdn.jsdelivr.net/bootstrap/3.3.0/css/bootstrap-theme.min.c
<script src="http://code.jquery.com/jquery-1.9.1.js"></script>
<script src="http://code.jquery.com/ui/1.10.3/jquery-ui.js"></script>
<script src="https://cdn.jsdelivr.net/bootstrap/3.3.0/js/bootstrap.min.js"></script>
<link rel="shortcut icon" href="../images/favicon.gif">
<div id="my-org-div-home-and-up">
  <a href="index.html">home </a>
  <span class="muted">...</span>
  <a href="sitemap.html"> archive </a>
</div>
org-sitemap-html-postamble
"<div id="disqus_thread"></div>
<script type="text/javascript">
  var disqus_shortname = 'eigenhombrecom'; // required: replace example with your forum shortname
  /* * * DON'T EDIT BELOW THIS LINE * * */
  (function() {
var dsq = document.createElement('script'); dsq.type = 'text/javascript'; dsq.async = true;
dsq.src = 'http://' + disqus_shortname + '.disqus.com/embed.js';
(document.getElementsByTagName('head')[0] || document.getElementsByTagName('body')[0]).appendChild(
  })();
</script>
<noscript>Please enable JavaScript to view the <a href="http://disqus.com/?ref_noscript">comments
<a href="http://disqus.com" class="dsq-brlink">blog comments powered by <span class="logo-disq
<script type="text/javascript">
  var _gaq = _gaq || [];
  _gaq.push(['_setAccount', 'UA-40279882-1']);
  _gaq.push(['_trackPageview']);

  (function() {
    var ga = document.createElement('script');
    ga.type = 'text/javascript';
    ga.async = true;
    ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.google-anal
    var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);
  })();
</script>")

(setq org-publish-project-alist
'(("blog"
  :base-directory "~/Dropbox/org/blog/src"
  :html-extension "html"
  :htmlized-source t
  :exclude "~\\(style\\|theindex\\)"
  :base-extension "org"
  :publishing-directory "~/Dropbox/org/blog/html"
  :publishing-function (org-html-publish-to-html)
  :section-numbers nil
  :auto-sitemap t
  :sitemap-sort-files "chronologically"

```

```

:sitemap-file-entry-format ,org-sitemap-link-format
:html-link-up ""
:html-link-home ""
:sitemap-title "Things you can find here"
:html-head-extra nil
:html-preamble ,org-sitemap-html-preamble
:html-postamble ,org-sitemap-html-postamble)))

```

Keyboard mappings to publish and to open local copy of new blog. Using `\C-oX` forces Org to (re-)publish even unmodified files.

```

(global-set-key "\C-ox" (lambda () (interactive) (org-publish-project "blog")))
(global-set-key "\C-oX" (lambda () (interactive) (org-publish-project "blog" t)))

(defun open-blog-index ()
  (interactive)
  (shell-command (concat "open file://" (expand-file-name "~/Dropbox/org/blog/html/index.html"))))

(global-set-key "\C-oZ" 'open-blog-index)

(defun open-blog-sitemap ()
  (interactive)
  (shell-command (concat "open file://" (expand-file-name "~/Dropbox/org/blog/html/sitemap.html"))))

(global-set-key "\C-oz" 'open-blog-sitemap)

```

I use this little bit of magic to reformat blog posts extracted from Blogger (*delete when no longer needed*).

```

(global-set-key "\C-oV" (lambda ()
  (interactive)
  (search-forward "# layout")
  (beginning-of-line)
  (set-mark-command nil)
  (search-forward "---")
  (comment-or-uncomment-region (region-beginning) (region-end))))

```

24 Tidying up

Be a nicely-behaved module or “feature”:

```

(provide 'init)

```