MINIMAL EMACS CONFIGURATION

Andrea Crotti

08 Luglio 2010

Contents

1 **DONE See if it's possible to have python auto completion without ropemacs** 1

2 **DONE Write a function that setups all the possible paths automatically** 1

3 **TODO Make the table of software versions regenerating every time** 1

4 **DONE See if it's worthy to byte compile everything or not** 1

5 **TODO Write a dependency tree of the various chapters in the configuration** 2

6 **TODO Add options to create encrypted org-files for storing password and more delicate informations** 2

7 **TODO Define defcustom stuff instead of constants for setting up variables** 2

8 **TODO Understand why sometimes auto-complete disables by itself** 2

9 **TODO See how to use variables like special-buffer-regexps to have a better control over the buffer windows** 2

10 **TODO Everything that should be changed by the user must be put in a defcustom** 2

11 Assumptions **2**

12 Links **2**

13 Things not working in emacs 22 **3**

14 Some documentation **3**
14.1 How to extend emacs in general 3

15 Aliasing and other useful shortcuts **3**

16 User customization **5**

17 Private settings **5**
17.1 Private 5
17.2 Custom settings 5

18 Prerequisites **5**
18.1 Constants and some global settings 5
18.2 Some useful functions 6
18.2.1 Make path 6
18.2.2 Make fortune 6
18.2.3 Gen path dirs 6
18.2.4 Google map it 7
18.2.5 New line 7
18.2.6 Error switch 8
18.2.7 Swap windows 8
18.2.8 Rename file and buffer 9
18.2.9 Move buffer file 9
18.2.10 Delete current file 9
18.2.11 Open git files 10
18.2.12 Camelizing 10
18.2.13 Find project 11
18.2.14 Select current line 12
18.3 Reload this configuration 13

19 Operating system detection **13**
19.1 Mac configuration 13

20 Buffer management **14**
20.1 TempBuf . . . 14
20.2 Uniquify . . . 14
20.3 Other nice packages . . . 14
20.4 Minibuffer nice stuff . . . 14

21 Package management **14**
21.1 Byte compilation . . . 15

22 Visualization **15**
22.1 Menu and tool-bar disabling . . . 15
22.2 Elscreen . . . 15
22.3 Fullscreen mode . . . 15
22.4 Winner mode . . . 16
22.5 Color theme setting . . . 16
22.6 Fringe and stuff todo . . . 16
22.7 Font settings . . . 17

23 General useful things **17**
23.1 Searching info . . . 17
23.2 Kill ring stuff . . . 17
23.3 Translations . . . 17
23.3.1 Org babel . . . 17
23.3.2 Spelling functions . . . 18

24 Org mode **19**
24.1 Setting up what happens when closing a task . . . 19
24.2 General TODO keywords . . . 19
24.3 Enforcing TODO dependencies . . . 19
24.4 Info configuration . . . 19
24.5 Clock configuration . . . 19
24.6 Add eventually . . . 19
24.6.1 Org agenda blacklist list . . . 20
24.7 Remember mode . . . 20
24.7.1 Setting up . . . 20
24.7.2 Define templates . . . 20
24.8 Notes . . . 20
24.9 Other org babel modes . . . 21
24.10 Org functionalities in other modes . . . 21

25 Tags **21**
25.1 Etags-select . . . 21
25.2 Extending functions . . . 21

26 Yasnippet **22**

27 Eldoc mode **23**

28 Auto complete **23**
28.1 Importing all packages . . . 23
28.2 Setting up . . . 23
28.3 Setting up generic sources . . . 23
28.4 Define allowed modes . . . 24
28.5 Elisp configuration . . . 24

29 Predictive modes **24**

30 Cedet **24**
30.1 Use cedet and gloabally ede-mode for projects . . . 25
30.2 Options for semantic . . . 25
30.3 Hook for inline documentation setting local keys . . . 25
30.4 Using semanticdb . . . 25

31 Programming **26**
31.1 Internationalization . . . 26
31.2 To spell mode . . . 26
31.3 Web nice utilities . . . 27
31.3.1 Gist . . . 27
31.4 C-mode . . . 27
31.5 Python . . . 27
31.5.1 Python mode . . . 27
31.6 Haskell mode . . . 27
31.7 Prolog . . . 28
31.8 Nesc . . . 28
31.9 Java . . . 29
31.9.1 Javadoc help . . . 29
31.9.2 Jdee settings . . . 29
31.10 Changelog settings and time . . . 29
31.11 Doc . . . 30
31.11.1 Doxymacs . . . 30
31.12 Applescript mode . . . 30

31.13Web programming 30
31.14Lua 30
31.15Fixme mode 30
31.16Yaml 30
31.17Go mode 31
31.18Django modes 31

32 Security **31**

33 Latex **31**
33.1 Configuring Auctex 31
33.2 Accessing to latex symbols 32

34 Mail settings **32**
34.1 General settings for message creation 32
34.2 Setting up gmail smtp server 32
34.3 Setting up completion over the addresses with the Mac address book 33
34.4 Setting default sending modality 33

35 Gnus settings **33**
35.1 Server settings 33
35.2 Old messages settings 34
35.3 Appearance 34
35.4 Avoid the annoying saving of the .news file 34
35.5 **TODO** Setting up some posting styles and other nice settings 34

36 Revision control systems **34**
36.1 Magit 34
36.2 Function to enable revert mode when in a git repository . . . 34

37 General settings **35**
37.1 Mode for startup 35
37.2 Showing more things 35
37.3 IDO mode 35
37.3.1 Use IDO when possible 35
37.4 Windmove 36
37.5 Workarounds 36

38 Flymake **36**
38.1 Setting up flymake 36
38.2 Adding errors to modeline 37
38.3 Flymake for python 37

39 Customizable variables **38**

40 Server stuff **38**

41 Fun **38**
41.1 Fortune settings 38
41.2 Some mac tricks 38
41.2.1 Open the terminal 38
41.2.2 Growl popup 39

42 Global keys settings **40**

43 External configuration files **41**

1 DONE See if it's possible to have python auto completion without ropemacs

CLOSED: *2010-05-09 Dom 14:24*

2 DONE Write a function that setups all the possible paths automatically

CLOSED: *2010-05-04 Mar 11:41*

3 TODO Make the table of software versions regenerating every time

4 DONE See if it's worthy to byte compile everything or not

CLOSED: *2010-05-24 Lun 14:01*

- CLOSING NOTE *2010-05-24 Lun 14:01*
  Solved nicely with byte-compile-cache, some cases it could not work

5 TODO Write a dependency tree of the various chapters in the configuration

6 TODO Add options to create encrypted org-files for storing password and more delicate informations

7 TODO Define defcustom stuff instead of constants for setting up variables

8 TODO Understand why sometimes auto-complete disables by itself

9 TODO See how to use variables like special-buffer-regexps to have a better control over the buffer windows

10 TODO Everything that should be changed by the user must be put in a defcustom

Add also custom keys for all the global-key-set combinations

11 Assumptions

- $HOME/.emacs.d/scripts is in the $PATH
- flymake is installed (distributed with emacs 23 also)
- in general probably working only with emacs 23

All the other libraries configured here are always included in the standard distribution or in **conf**.

12 Links

- A guided tour of emacs features
- nice emacs lisp intro

13 Things not working in emacs 22

- autoload stuff from ORG-clock
- remember mode (not present)

14 Some documentation

14.1 How to extend emacs in general

example about advising functions

- If you can use the mode hooks provided by the author, use them instead.
- If there is a bug in the original mode, just fix it in the original code and submit a patch.
- If there is a new feature you want, add it to the original mode and submit a patch. Talk to the author, work with him, and it will most likely end up in the next release.
- If you are an Emacs developer, working on Emacs itself, or one of the modes shipped with Emacs, never use advice. It's the least maintainable method of extending Emacs with the exception of a pure fork, and since you're working on Emacs itself, it's not a fork.

- If your patch is not accepted, or you know that what you want is fringe enough or hackish enough to not warrant submitting a patch, only then should you use advice or fork the project.

15 Aliasing and other useful shortcuts

```
(defalias 'eb 'eval-buffer)
(defalias 'er 'eval-region)
(defalias 'yes-or-no-p 'y-or-n-p)
(defalias 'rs 'replace-string)
(defalias 'qs 'query-replace)
(defalias 'ac 'auto-complete-mode)
(defalias 'go 'google-search-it)
(defalias 'gs 'google-search-selection)
(defalias 'spell 'flyspell-mode)
(defalias 'spell-prog 'flyspell-prog-mode)
(defalias 'dml 'delete-matching-lines)
(defalias 'bb 'bury-buffer)
(defalias 'elm 'emacs-lisp-mode)

(defalias 'ys 'yas/reload-all)
(defalias 'yv 'yas/visit-snippet-file)

(defalias 'ascii 'org-export-as-ascii)
(defalias 'html 'org-export-as-html-and-open)
(defalias 'pdf 'org-export-as-pdf-and-open)
(defalias 'box 'comment-box)
(defalias 'rb 'revert-buffer)

(defalias 'sh 'shell)

(defalias 'ws 'whitespace-mode)
(defalias 'bu 'browse-url)

(defalias 'mem 'doxymacs-insert-member-comment)
(defalias 'fun 'doxymacs-insert-function-comment)
(defalias 'file 'doxymacs-insert-file-comment)

;; Those below are my favourite themes
```

```
(defalias 'black 'color-theme-hober)
(defalias 'blue 'color-theme-deep-blue)
(defalias 'grey 'color-theme-black-on-gray)
(defalias 'blipp 'color-theme-blippblopp)
(defalias 'high 'color-theme-high-contrast)
(defalias 'billw 'color-theme-billw)
(defalias 'coal 'color-theme-charcoal-black)

(defalias 'batt 'display-battery-mode)

(defun get-some-messages ()
  (interactive)
  (gnus-summary-rescan-group 1000))
;; gnus
(defalias 'gg 'get-some-messages)
(defalias 'jd 'javadoc-lookup)
(defalias 'br 'babel-region-default)
(defalias 'git 'open-git-files)

(defalias 'fold 'senator-fold-tag-toggle)
```

16 User customization

```
(defgroup miniconf nil
  "miniconf"
  :group 'miniconf
  :prefix "mini-")
```

17 Private settings

17.1 Private

In this file you can store your own personal settings

```
;; not complain if not existing
(if (file-exists-p "private.el")
    (load-file "private.el"))
```

17.2 Custom settings

```
(defcustom custom-file
  (concat base "custom.el")
  "customization file"
  :type 'string
  :group 'miniconf)
```

18 Prerequisites

18.1 Constants and some global settings

```
(defcustom default-closing-char ";"
  "default closing char, change in newline-force-close-alist if needed"
  :type 'string
  :group 'miniconf)

;; TODO: use a defcustom instead
(defcustom newline-force-close-alist
  '((python-mode . ":")
    (jython-mode . ":")
    (prolog-mode . ".")
    (latex-mode . " \\\\")
    (org-mode . " \\\\")
    (tuareg-mode . ";;")
    (html-mode . " <br>"))
  "Closing char for different modes"
  :type 'list
  :group 'miniconf)
```

18.2 Some useful functions

We suppose that the global variable **conf** has been already set from the outside.

18.2.1 Make path

```
(defun make-path (path)
  (concat conf path))
```

18.2.2 Make fortune

Print below a fortune cookie if the command is present in the system.

```
(defun make-fortune ()
(interactive)
(let ((beg (point)))
  (insert (shell-command-to-string "fortune"))
  (end-of-paragraph-text)))
```

18.2.3 Gen path dirs

All all the directories in the first level of the configuration directory to the load path.

```
;; TODO: make it more general
(defun gen-path-dirs ()
  "Add to load path all the subdirectories of first level"
  (interactive)
  (message "adding all directories in the first level to the load-path")
  (dolist (dir (directory-files conf t))
    (if (and
         (file-directory-p dir)
         (not (file-symlink-p dir)))
        (add-to-list 'load-path dir))))

(gen-path-dirs)
```

18.2.4 Google map it

Search an address in google map

```
(defun google-map-it (address)
  "get the map of the given address"
  (interactive "sSearch for: ")
  (let
      ((base "http://maps.google.it/maps?q=%s"))
    (browse-url (format base (url-hexify-string address)))))
```

18.2.5 New line

Those functions are inspired by textmate

```
;; My own functions
(defun newline-force()
  "Goes to newline leaving untouched the rest of the line"
  (interactive)
  (end-of-line)
  (newline-and-indent))

(defun newline-force-close()
  "Same as newline-force but putting a closing char at end"
  (interactive)
  (end-of-line)
  (let ((closing-way (assoc major-mode newline-force-close-alist))
        closing-char)
    ;; Setting the user defined or the constant if not found
    (if (not closing-way)
        (progn
          (message "closing char not defined for this mode, using default")
          (setq closing-char default-closing-char))
      (setq closing-char (cdr closing-way)))
    (when (not (bobp))
      ;; if we're at beginning of buffer, the backward-char will beep
      ;; :( This works even in the case of narrowing (e.g. we don't
      ;; look outside of the narrowed area.
      ;; FIXME: looking-at not working as expected
      (when (not (looking-at closing-char))
        (insert closing-char))
      (newline-force))))
```

18.2.6 Error switch

Useful function to toogle on and off the debug mode

```
(defun err-switch()
  "switch on/off error debugging"
  (interactive)
  (if debug-on-error
      (setq debug-on-error nil)
    (setq debug-on-error t))
  (message "debug-on-error now %s" debug-on-error))
```

18.2.7 Swap windows

```
;; someday might want to rotate windows if more than 2 of them
(defun swap-windows ()
  "If you have 2 windows, it swaps them." (interactive) (cond ((not (= (count-windows)
                                                                       (t
                                                                        (let* ((w1 (first (windo
                                                                               (w2 (second (win
                                                                               (b1 (window-buff
                                                                               (b2 (window-buff
                                                                               (s1 (window-star
                                                                               (s2 (window-star
                                                                          (set-window-buffer w1
                                                                          (set-window-buffer w2
                                                                          (set-window-start w1 
                                                                          (set-window-start w2 
```

18.2.8 Rename file and buffer

- **TODO** Add something VCS related for moving away files

```
(defun rename-file-and-buffer (new-name)
  "Renames both current buffer and file it's visiting to NEW-NAME." (interactive "
  (let ((name (buffer-name))
        (filename (buffer-file-name)))
    (if (not filename)
        (message "Buffer '%s' is not visiting a file!" name)
      (if (get-buffer new-name)
          (message "A buffer named '%s' already exists!" new-name)
        (progn   (rename-file name new-name 1)   (rename-buffer new-name)        (
```

18.2.9 Move buffer file

```
(defun move-buffer-file (dir)
  "Moves both current buffer and file it's visiting to DIR." (interactive "DNew direct
  (let* ((name (buffer-name))
         (filename (buffer-file-name))
         (dir
          (if (string-match dir "\\(?:/\\|\\\\)$")
              (substring dir 0 -1) dir))
         (newname (concat dir "/" name)))
```

```
    (if (not filename)
        (message "Buffer '%s' is not visiting a file!" name)
      (progn         (copy-file filename newname 1)  (delete-file filename)  (set-visi
```

18.2.10 Delete current file

```
(defun delete-current-file ()
  "Delete the file associated with the current buffer."
  (interactive)
  (let (currentFile)
    (setq currentFile (buffer-file-name))
    (when (yes-or-no-p (format "Delete file % s and kill buffer? " currentFile))
      (kill-buffer (current-buffer))
      (delete-file currentFile)
      (message "Deleted file: %s " currentFile))))
```

18.2.11 Open git files

Run **git ls-files** and visits all the buffer given from it

```
(defun open-git-files ()
  "Visit all the files in the current git project"
  (interactive)
  (dolist
      (file (ls-git-files))
    (message "Opening %s" file)
    ;; we have to keep the original position
    (save-excursion (find-file file))))

(defun dired-git-files ()
  (interactive)
  (dired (ls-git-files)))

(defun ls-git-files ()
  (if
      (file-exists-p ".git")
      (split-string (shell-command-to-string "git ls-files"))
    (message "not a git repo")
    nil))
```

18.2.12 Camelizing

(un)Camelizing allows to convert quickly function/variables names from camelized to non camelized mode.

```
(defun mapcar-head (fn-head fn-rest list)
  "Like MAPCAR, but applies a different function to the first element."
  (if list
      (cons (funcall fn-head (car list)) (mapcar fn-rest (cdr list)))))

(defun camelize (s)
  "Convert under_score string S to CamelCase string."
  (mapconcat 'identity (mapcar
                        '(lambda (word) (capitalize (downcase word)))
                        (split-string s "_")) ""))

(defun camelize-method (s)
  "Convert under_score string S to camelCase string."
  (mapconcat 'identity (mapcar-head
                        '(lambda (word) (downcase word))
                        '(lambda (word) (capitalize (downcase word)))
                        (split-string s "_")) ""))

(defun un-camelcase-string (s &optional sep start)
  "Convert CamelCase string S to lower case with word separator SEP.
    Default for SEP is a hyphen \"-\".
    If third argument START is non-nil, convert words after that
    index in STRING."
  (let ((case-fold-search nil))
    (while (string-match "[A-Z]" s (or start 1))
      (setq s (replace-match (concat (or sep "-")
                                     (downcase (match-string 0 s)))
                             t nil s)))
    (downcase s)))
```

We also have camel case mode which makes moving in camelized words smarter

```
(autoload 'camelCase-mode "camelCase-mode")
(defcustom camelCase-modes
  '(python-mode-hook java-mode-hook c-mode-hook nesc-mode-hook)
```

```
  "Modes where camelizing is allowed"
  :type 'list
  :group 'miniconf)

(dolist (hook camelCase-modes)
  (add-hook hook 'camelCase-mode))
```

18.2.13 Find project

This functions are take from textmate.el and are used to check if we're on a project of some kind. Not used at the moment.

```
;; When it's a git project we can use a grep over git ls-files
;; same thing for mercurial
;; check also with the Makefiles in general if we can do something like this
;; In this way is too simplicistic

(defvar *project-roots*
  '(".git" ".hg" "Rakefile" "Makefile" "README" "build.xml")
  "The presence of any file/directory in this list indicates a project root.")

(defun root-match(root names)
  (member (car names) (directory-files root)))

(defun root-matches(root names)
  (if (root-match root names)
      (root-match root names)
    (if (eq (length (cdr names)) 0)
        'nil
      (root-matches root (cdr names)))))

;; should return also the type and the certainty level
(defun find-project-root (&optional root)
  "Determines the current project root by recursively searching for an indicator."
  (interactive)
  (when (null root)
    (setq root default-directory))
  (cond
   ((root-matches root *project-roots*)
    (expand-file-name root))
```

```
   ((equal (expand-file-name root) "/") nil)
   (t
    ;; recursive call
    (find-project-root (concat (file-name-as-directory root) "..")))))

(find-project-root)
```

18.2.14 Select current line

```
(defun select-line ()
  "If the mark is not active, select the current line.
Otherwise, expand the current region to select the lines the region touches."
  (interactive)
  (if mark-active ;; expand the selection to select lines
      (let ((top (= (point) (region-beginning)))
            (p1 (region-beginning))
            (p2 (region-end)))
        (goto-char p1)
        (beginning-of-line)
        (push-mark (point))
        (goto-char p2)
        (unless (looking-back "\n")
          (progn
            (end-of-line)
            (if (< (point) (point-max)) (forward-char))))
        (setq mark-active t
              transient-mark-mode t)
        (if top (exchange-point-and-mark)))
    (progn
      (beginning-of-line)
      (push-mark (point))
      (end-of-line)
      (if (< (point) (point-max)) (forward-char))
      (setq mark-active t
            transient-mark-mode t))))
```

18.3 Reload this configuration

```
(defun reload-conf ()
  (interactive)
```

```
  (org-babel-load-file "miniconf.org"))
```

19 Operating system detection

```
(defconst linux (string-match "linux" system-configuration))
(defconst mac (string-match "apple" system-configuration))
(defconst win (string-match "win" system-configuration))
```

19.1 Mac configuration

This will setup the Command key ot be used as meta. I also add the path normally used for macports to the exec-path.

```
(if mac
    (progn
      (add-to-list 'exec-path "/opt/local/bin")
      (setq ns-alternate-modifier (quote none))
      (setq ns-command-modifier (quote meta))))
```

20 Buffer management

See also this nice article for a better filtering of buffers while switching.

20.1 TempBuf

TempBuf will automatically kill some of the normally useless buffers

```
(require 'tempbuf)

;; Enabling tempbuf mode for some kind of buffers
(add-hook 'dired-mode-hook 'turn-on-tempbuf-mode)
(add-hook 'custom-mode-hook 'turn-on-tempbuf-mode)
(add-hook 'w3-mode-hook 'turn-on-tempbuf-mode)
(add-hook 'Man-mode-hook 'turn-on-tempbuf-mode)
(add-hook 'view-mode-hook 'turn-on-tempbuf-mode)
```

20.2 Uniquify

Uniquify is used to distinguish easily from buffers with the same name.

```
;; Using uniquify for better handling of buffers with same name
(require 'uniquify)
;; Using part of the directory in this case
(setq uniquify-buffer-name-style 'forward)
```

20.3 Other nice packages

```
;; it will remember where were you in the buffer
(require 'saveplace)
```

20.4 Minibuffer nice stuff

```
(setq visible-bell t) ; Turn beep off
(savehist-mode t) ; save also minibuffer history, very useful
```

21 Package management

Auto install is a nice way to install packages from emacswiki repository.
There are other possible ways to manage the emacs packages.

```
; Other autoloads
(autoload 'auto-install-from-emacswiki "auto-install" "auto install from emacswiki" t)
(setq auto-install-directory (concat conf "auto-install/"))
```

21.1 Byte compilation

Byte compilation can be done automatically by this package that keeps a cache of the compiled packages.

It's also possible to put some more files in the blacklist.

```
(require 'byte-code-cache)
;; FIXME: This is still not fixing the recursive stuff
(add-to-list 'bcc-blacklist "eieio")
```

22 Visualization

22.1 Menu and tool-bar disabling

```
(if (fboundp 'scroll-bar-mode) (scroll-bar-mode -1))
(if (fboundp 'tool-bar-mode) (tool-bar-mode -1))
```

22.2 Elscreen

```
(require 'alist)
(setq elscreen-path (concat conf "elscreen/"))
(add-to-list 'load-path (concat elscreen-path "elscreen"))
(add-to-list 'load-path (concat elscreen-path "elscreen-server"))
(add-to-list 'load-path (concat elscreen-path "elscreen-color-theme"))
(require 'elscreen)
(require 'elscreen-color-theme)
(require 'elscreen-server)
```

22.3 Fullscreen mode

Doesn't work yet on OSX 10.6 with emacs 23, works fine with linux and emacs 22 on OSX.

```
(defun full (&optional f)
  (interactive)
  (if
      ;; more checks on the version and the OS should be necessary here
      mac
      (ns-toggle-fullscreen)
    (set-frame-parameter f 'fullscreen
                         (if (frame-parameter f 'fullscreen) nil 'fullboth))))
;; this toogle the fullscreen for every new frame (window) created
(add-hook 'after-make-frame-functions 'full)
```

22.4 Winner mode

Winner mode remember the window configuration and allows you to go back and forth

```
;; enabling winner mode for window reconfiguration
(winner-mode t)
```

22.5 Color theme setting

```
(require 'color-theme)
(eval-after-load "color-theme"
  '(progn
     (color-theme-initialize)))
(coal)
```

22.6 Fringe and stuff todo

Nice but unable to update itself automatically, but only set when the file is visited first time.

```
(defun annotate-todo ()
  "put fringe marker on TODO: lines in the curent buffer"
  (interactive)
  (save-excursion
    ;; TODO: add also other regexps like FIXME or others
    (goto-char (point-min))
    (while (re-search-forward "TODO:" nil t)
      (let ((overlay (make-overlay (- (point) 5) (point))))
        (overlay-put overlay 'before-string (propertize "A"
                                                        'display '(left-fringe right-t


(add-hook 'find-file-hooks 'annotate-todo)
```

22.7 Font settings

Defininig some nice fonts and how to switch between theme. Cycling function definition

```
(setq current "monaco-12")
(setq font-list
      (list "monaco-12" "inconsolata-14" "courier-13"))

(defun cycle-font ()
  "Change font in current frame"
  (interactive)

  (let (fontToUse currentState)
    ;; states starts from 1.
    (setq currentState (if (get this-command 'state) (get this-command 'state) 1))
    (setq fontToUse (nth (1- currentState) font-list))

    (set-frame-parameter nil 'font fontToUse)
    (message "Current font is: %s" fontToUse)
    (put this-command 'state (1+ (% currentState (length font-list))))
    (redraw-frame (selected-frame))))
```

23 General useful things

23.1 Searching info

Look for in google

```
(load-library "google_search")
```

23.2 Kill ring stuff

Sometimes the key ring is not easy to manage, we can browse inside it to see what we saved

```
(require 'browse-kill-ring)
```

23.3 Translations

23.3.1 Org babel

```
(setq babel-preferred-from-language "German")
(setq babel-preferred-to-language "English")

(autoload 'babel-region-default "babel" "translating default" t)
(autoload 'babel-region "babel" "translating a region" t)
(autoload 'babel "babel" "translating interactively" t)
(autoload 'babel-buffer "babel" "translate buffer" t)
```

23.3.2 Spelling functions

```
(setq ispell-dictionary "english")

;; TODO: possible to refactor this code maybe?
(defun en ()
  "Check spelling in english"
   (interactive)
  (ispell-change-dictionary "english")
  (flyspell-mode t))

(defun it ()
  "Check spelling in english"
  (interactive)
  (ispell-change-dictionary "italian")
  (flyspell-mode t))
```

```
(defun fr ()
  "Check spelling in english"
  (interactive)
  (ispell-change-dictionary "french")
  (flyspell-mode t))

(defun de ()
  "Check spelling in english"
  (interactive)
  (ispell-change-dictionary "german")
  (flyspell-mode t))
```

24 Org mode

24.1 Setting up what happens when closing a task

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

24.2 General TODO keywords

```
(setq org-todo-keywords
      '((sequence "TODO(t)" "FEEDBACK(f)" "VERIFY(v)" "|" "DONE(d)" "DELEGATED(D)")))
```

24.3 Enforcing TODO dependencies

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

24.4 Info configuration

```
(add-to-list 'Info-default-directory-list "~/.emacs.d/org-mode/doc/")
```

24.5 Clock configuration

```
;; Clock configuration
(setq org-clock-persist t)
(org-clock-persistence-insinuate)
```

24.6 Add eventually

This hook enables to expand your KB very easily, every time you create a new org-file it will check if it's already in the agenda and asks to add it. Disable this if you don't plan to use org mode and its agenda

```
(defun org-add-eventually()
  "Adding a file to org-agenda when saved"
  (interactive)
  (if
      (and
       (string= major-mode "org-mode")
       (not (member (abbreviate-file-name buffer-file-name) org-agenda-files)))
      (if
          (yes-or-no-p "add the file to agenda?")
          (org-agenda-file-to-front))))

(add-hook 'before-save-hook 'org-add-eventually)
```

24.6.1 Org agenda blacklist list

Having to say "n" every time for a file that we don't want to add to the agenda can be annoying, so every time we say no we call another function.

```
(defun org-agenda-add-to-blacklist ()
  (setq org-agenda-blacklist
        (add-to-list 'org-agenda-blacklist (abbreviate-file-name buffer-file-name)))
  (customize-save-variable org-agenda-blacklist org-agenda-blacklist))
```

24.7 Remember mode

Org mode can be used in conjunction with remember mode to keep track of repetitive things to remember.

24.7.1 Setting up

```
(require 'remember)
(require 'org-remember)
(org-remember-insinuate)
```

24.7.2 Define templates

```
(setq org-remember-templates
      '(
        ("Note" ?n "* " "~/Documents/pycon/notes.org")))
        ;; ("Note" ?n "* " "~/org/notes.org")
        ;; ("Homeworks" ?h "* TODO %^{homework|german|functional|database|scientific|g
        ;; ("TOBUY" ?b "* TODO %^{what you want}\n %t" "~/org/tobuy.org")))
```

24.8 Notes

```
;; Defining a setup where org-mode takes care of remember notes
(setq org-directory "~/org/")
(setq org-default-notes-file (concat org-directory "notes.org"))
```

24.9 Other org babel modes

```
(require 'org-babel-dot)       ;; dot
(require 'org-babel-haskell)   ;; haskell, haskell-mode, inf-haskell
(require 'org-babel-python)    ;; python, and python-mode
(require 'org-babel-ditaa)
;; (require 'org-babel-ruby)      ;; ruby, irb, ruby-mode, and inf-ruby
(require 'org-babel-sql)       ;; none
(require 'org-babel-sh)
```

24.10 Org functionalities in other modes

```
(setq org-struct-hooks
      '(message-mode-hook
        mail-mode-hook))

(dolist (hook org-struct-hooks)
  (add-hook hook 'turn-on-orgstruct)
  (add-hook hook 'turn-on-orgtbl))
```

25 Tags

25.1 Etags-select

This extension to etags helps choosing from equal names of functions.

```
(require 'etags-select)
```

25.2 Extending functions

This functions help to look for the TAGS file in the filesystem when is not in the same working directory

```
(defun jds-find-tags-file ()
  "recursively searches each parent directory for a file named 'TAGS' and returns the
path to that file or nil if a tags file is not found. Returns nil if the buffer is
not visiting a file"
  (progn
    (defun find-tags-file-r (path)
      "find the tags file from the parent directories"
      (let* ((parent (file-name-directory path))
             (possible-tags-file (concat parent "TAGS")))
        (cond
         ((file-exists-p possible-tags-file) (throw 'found-it possible-tags-file))
         ((string= "/TAGS" possible-tags-file) (error "no tags file found"))
         (t (find-tags-file-r (directory-file-name parent))))))

    (if (buffer-file-name)
        (catch 'found-it
          (find-tags-file-r (buffer-file-name)))
      (error "buffer is not visiting a file"))))

(defun jds-set-tags-file-path ()
  "calls `jds-find-tags-file' to recursively search up the directory tree to find
a file named 'TAGS'. If found, set 'tags-table-list' with that path as an argument
otherwise raises an error."
  (interactive)
  (setq tags-table-list (list (jds-find-tags-file))))

;; delay search the TAGS file after open the source file
(add-hook 'emacs-startup-hook
          '(lambda () (jds-set-tags-file-path)))
```

26 Yasnippet

```
(require 'yasnippet)

(setq yas/root-directory
```

```
      (mapcar 'make-path
              '("yasnippet/" "my-snippets/")))

;; Maybe needed to set to fixed for some modes
(setq yas/indent-line 'auto)

(yas/initialize)

(setq yas/ignore-filenames-as-triggers nil)

(mapc 'yas/load-directory yas/root-directory)

;; don't make backups in the snippet folder, they mess up yasnippet
(add-to-list 'backup-directory-alist '("/my-snippets/" . "/tmp/"))
```

27 Eldoc mode

Show the documentation of some functions directly in the minibuffer.

```
(require 'eldoc)
;; Maybe better a direct activation??
(dolist (hook '(python-mode-hook
                c-mode-hook
                ruby-mode-hook
                lisp-interaction-mode-hook
                emacs-lisp-mode-hook))
  (add-hook hook 'turn-on-eldoc-mode))
```

28 Auto complete

28.1 Importing all packages

```
;;; Require
(require 'auto-complete)
;; Various configurations
(require 'auto-complete-config)
(require 'auto-complete-extension nil t) ;optional
(require 'auto-complete-yasnippet nil t) ;optional
(require 'auto-complete-semantic nil t)  ;optional
```

28.2 Setting up

```
;; Generic setup.
(global-auto-complete-mode t)          ;enable global-mode

(setq ac-auto-start 3)                 ;automatically start
(setq ac-override-local-map nil)       ;don't override local map

(define-key ac-complete-mode-map "\C-n" 'ac-next)
(define-key ac-complete-mode-map "\C-p" 'ac-previous)
```

28.3 Setting up generic sources

```
(setq-default ac-sources
              (append ac-sources '(ac-source-semantic)))
```

28.4 Define allowed modes

```
(setq ac-modes
      '(python-mode
        emacs-lisp-mode
        c-mode
        nesc-mode
        lisp-interaction-mode
        java-mode
        org-mode
        html-mode
        xml-mode))

(add-to-list 'ac-trigger-commands 'org-self-insert-command) ; if you want enable auto-
```

28.5 Elisp configuration

```
(dolist (hook (list
               'emacs-lisp-mode-hook
               'lisp-interaction-mode
               ))
  (add-hook hook '(lambda ()
                    (add-to-list 'ac-sources 'ac-source-symbols))))
```

29 Predictive modes

```
(autoload 'predictive-mode "predictive" "predictive" t)
(set-default 'predictive-auto-add-to-dict t)
(setq predictive-main-dict 'dict-english
      predictive-auto-learn t
      predictive-add-to-dict-ask nil
      predictive-use-auto-learn-cache nil
      predictive-which-dict t)
```

30 Cedet

See gentle introduction to cedet for a nicer tutorial

30.1 Use cedet and gloabally ede-mode for projects

```
(require 'cedet)
;; This is not working with the stupid code cache stuff
;; (global-ede-mode t)
(require 'semantic)
```

30.2 Options for semantic

```
(semantic-mode t)
(global-semantic-stickyfunc-mode 1)
(global-semantic-idle-completions-mode 1)
(global-semantic-decoration-mode 1)
(global-semantic-highlight-func-mode 1)
(global-semantic-idle-summary-mode 1)
(global-semantic-highlight-edits-mode 1)
;; enable working on nesc-code, a superset of C, add another language to semantic inst
(add-to-list 'semantic-new-buffer-setup-functions
             '(nesc-mode . semantic-default-c-setup))
```

30.3 Hook for inline documentation setting local keys

```
(defun my-c-like-cedet-hook ()
  (local-set-key [(control return)] 'semantic-ia-complete-symbol)
  (local-set-key "\C-c?" 'semantic-ia-complete-symbol-menu)
  (local-set-key "\C-c>" 'semantic-complete-analyze-inline)
```

```
  (local-set-key "\C-cj" 'semantic-ia-fast-jump)
  (local-set-key "\C-cq" 'semantic-ia-show-doc)
  (local-set-key "\C-cs" 'semantic-ia-show-summary)
  (local-set-key "\C-cp" 'semantic-analyze-proto-impl-toggle))

(defun my-c-only-cedet-hook ()
  (local-set-key "." 'semantic-complete-self-insert)
  (local-set-key ">" 'semantic-complete-self-insert))
```

30.4 Using semanticdb

```
(global-semanticdb-minor-mode)
(semanticdb-enable-gnu-global-databases 'c-mode)
(semanticdb-enable-gnu-global-databases 'java-mode)
(semanticdb-enable-gnu-global-databases 'jde-mode)
(semanticdb-enable-gnu-global-databases 'python-mode)
```

31 Programming

31.1 Internationalization

Editing po files

```
(autoload 'po-mode "po-mode+"
  "Major mode for translators to edit PO files" t)

(add-to-list 'auto-mode-alist
             '("\\.po$" . po-mode))

(add-to-list 'auto-mode-alist
             '("\\.pot$" . po-mode))


;; to automatically find out the coding system
(modify-coding-system-alist 'file "\\.po\\'\\|\\.po\\."
                            'po-find-file-coding-system)
(autoload 'po-find-file-coding-system "po-mode")
```

31.2 To spell mode

Most of the programming languages we can have syntax checking on the comments and strings. Flyspell-prog-mode is just for this

```
(defcustom to-spell-langs
  '(emacs-lisp-mode-hook python-mode-hook c-mode-hook nesc-mode-hook java-mode-hook jd
  "Set of programming modes for which I want to enable spelling in comments and string
  :type 'list
  :group 'miniconf)
;; (setq to-spell-langs

(dolist (lang-hook to-spell-langs)
  (add-hook  lang-hook 'flyspell-prog-mode))
```

31.3 Web nice utilities

31.3.1 Gist

Use simply **gist-buffer** or **gist-region** to paste code online.

```
(require 'gist)
```

31.4 C-mode

```
(require 'c-eldoc)
(setq c-default-style
      '((java-mode . "java")
        (awk-mode . "awk")
        (other . "cc-mode")))

;; FIXME: eldoc mode, not working correctly apparently
;; See http://www.emacswiki.org/emacs/CEldocMode for more info
(add-hook 'c-mode-hook 'c-turn-on-eldoc-mode)
;; adding the hook from cedet
(add-hook 'c-mode-common-hook 'my-c-like-cedet-hook)

(add-hook 'c-mode-hook 'my-c-only-cedet-hook)
```

31.5 Python

31.5.1 Python mode

Possible configuration for auto completion with ropemacs Ropmacs though is not up to date and needs and may not work correctly

```
;; TODO: check why is not working with the autoload
(load-library "python-mode")
(add-to-list 'auto-mode-alist '("\\.py$" . python-mode))
(add-to-list 'interpreter-mode-alist '("python" . python-mode))
(autoload 'doctest-mode "doctest-mode" "doc test python mode" t)
```

31.6 Haskell mode

```
(add-to-list 'auto-mode-alist '("\\.hs$" . haskell-mode))
(autoload 'haskell-mode "haskell-mode" "haskell mode" t)
(autoload 'turn-on-haskell-doc-mode "haskell-doc" "haskell doc mode" t)
(autoload 'turn-on-haskell-indent "haskell-indent" "haskell indent facilities" t)

(add-hook 'inf-haskell "inf-haskell" "inf-haskell" t)
(add-hook 'hs-lint "hs-lint" "haskell checker" t)

;; here some haskell variables
(setq haskell-doc-show-global-types t)
(setq haskell-program-name "ghci")
                                        ; where haskell-hoogle is loaded?

;; enabled to get indentation over if-then-else
(setq haskell-indent-thenelse 1)

;; If nothing found pass the control
(add-hook 'haskell-mode-hook
          '(lambda ()
             (require 'haskell-doc) ; Is this the only way?
             (require 'haskell-indent)
             (turn-on-haskell-doc-mode)
             (turn-on-haskell-indentation)
             ;; This would be very nice but it conflicts with yasnippet
             (define-key haskell-mode-map [tab] 'haskell-indent-cycle)
             (define-key haskell-mode-map "\C-ch" 'haskell-hoogle)
```

```
            (define-key haskell-mode-map "\C-cl" 'hs-lint)
            (make-variable-buffer-local 'yas/trigger-key)
            (setq yas/trigger-key [tab])
            (define-key yas/keymap [tab] 'yas/next-field)))
```

31.7 Prolog

```
(autoload 'run-prolog "prolog" "Start a Prolog sub-process." t)
(autoload 'prolog-mode "prolog" "Major mode for editing Prolog programs." t)
(autoload 'mercury-mode "prolog" "Major mode for editing Mercury programs." t)
(setq prolog-system 'swi)
(add-to-list 'auto-mode-alist '("\\.pl$" . prolog-mode))
```

31.8 Nesc

```
(autoload 'nesc-mode "nesc" nil t)
(add-to-list 'auto-mode-alist '("\\.nc$" . nesc-mode))
```

31.9 Java

31.9.1 Javadoc help

```
(autoload 'javadoc-lookup       "javadoc-help" "Look up Java class in Javadoc."   t)
(autoload 'javadoc-help         "javadoc-help" "Open up the Javadoc-help menu."   t)
(autoload 'javadoc-set-predefined-urls  "javadoc-help" "Set pre-defined urls."    t)
```

31.9.2 Jdee settings

```
(add-to-list 'load-path (concat conf "jdee/lisp"))

(autoload 'jde-mode "jde" "jde mode" t)

;; In this way we only load if really necessary
(add-hook 'jde-mode-hook
          '(lambda ()
             (require 'ecb)
             (setq indent-tabs-mode nil)))

;; (defun turn-on-font-lock-if-enabled ()
;;   "set up to make jdee shut up")
```

```
;; TODO: put some conditional stuff for the different operating systems
;; make it more general usage
(setq jde-jdk-registry
      '(("1.6" . "/System/Library/Frameworks/JavaVM.framework/Versions/1.6/")
        ("1.5" . "/System/Library/Frameworks/JavaVM.framework/Versions/1.5/")
        ("1.3.1" . "/System/Library/Frameworks/JavaVM.framework/Versions/1.3.1/")))

(setq jde-jdk '("1.6" . "/System/Library/Frameworks/JavaVM.framework/Versions/1.6/"))

(setq bsh-jar "/opt/local/share/java/bsh.jar")
```

31.10 Changelog settings and time

```
;; for changelogs
(setq add-log-always-start-new-record 1)
(add-hook 'before-save-hook 'time-stamp)
(setq time-stamp-format "%02d-%02m-%:y, %02H:%02M")
```

31.11 Doc

31.11.1 Doxymacs

```
(require 'doxymacs)
```

31.12 Applescript mode

```
(add-to-list 'auto-mode-alist
             '("\\.applescript$" . applescript-mode))
(autoload 'applescript-mode "applescript-mode" "mode for applescript files" t)
```

31.13 Web programming

Enabling nxhtml mode

```
(load (concat conf "nxhtml/autostart.el"))

;;(add-to-list 'auto-mode-alist '("\\.mak$" . html-mode))
;;(add-to-list 'auto-mode-alist '("\\.mako$" . html-mode))
(add-to-list 'auto-mode-alist '("\\.mak$" . mako-html-mumamo-mode))
(add-to-list 'auto-mode-alist '("\\.mako$" . mako-html-mumamo-mode))

(setq mumamo-chunk-coloring 2)
```

31.14 Lua

```
;; lua mode
(autoload 'lua-mode "lua-mode" "mode for lua" t)
```

31.15 Fixme mode

This is a mode to highlight stuff, adding some more modes

```
(require 'fixme-mode)
;;TODO: make it working everywhere automatically if possible
(add-to-list 'fixme-modes 'org-mode)
```

31.16 Yaml

```
(autoload 'yaml-mode "yaml-mode" "mode for yaml" t)
(add-to-list 'auto-mode-alist
             '("\\.yaml$" . yaml-mode))
```

31.17 Go mode

```
(autoload 'go-mode "go-mode" "go mode" t)
(add-to-list 'auto-mode-alist
             '("\\.go$" . go-mode))
```

31.18 Django modes

This mode is derived from html and helps writing django templates

```
(autoload 'django-html-mode "django-html-mode" "mode for django templates" t)
(add-to-list 'auto-mode-alist
             '("views" . django-html-mode))
```

32 Security

```
(require 'epa)
(epa-file-enable)
```

33 Latex

33.1 Configuring Auctex

Auctex is much more powerful than the default latex mode, enabling it using pdf as default mode

```
(add-to-list 'load-path (concat conf "auctex"))
(autoload 'latex-mode "auctex" "latex mode" t)
(autoload 'preview-latex "preview-latex" "preview latex in buffer" t)

(setq TeX-auto-save t)
(setq TeX-parse-self t)
(setq LaTeX-command "latex")
(setq TeX-PDF-mode t)
(setq TeX-master nil)

;; using flyspell also here
(add-hook 'latex-mode-hook 'turn-on-flyspell)
```

33.2 Accessing to latex symbols

```
(setq latex-symbols-file
      (expand-file-name "~/howto_guide/languages/latex/symbols-a4.pdf"))

(defvar latex-command-program
  (cond
   (mac "open")
   (linux "evince"))
  "latex program to execute for viewing pdf")

(defun latex-symbols ()
  "open the latex symbols file"
  (interactive)
  (if (file-exists-p latex-symbols-file)
      (shell-command (concat latex-command-program " " latex-symbols-file))
    (message "file not found")))
```

34 Mail settings

34.1 General settings for message creation

```
;; setting where the mail is coming from
(setq mail-setup-with-from t)

;; This is just to enable flyspell in mail-mode
;; FIXME: check if this dirty hack is still needed
(defvar message-signature-separator "^-- *$" "\
    Regexp matching the signature separator.")
```

34.2 Setting up gmail smtp server

Make sure you configure correctly your .authinfo for login and password

```
(setq send-mail-function 'smtpmail-send-it
      message-send-mail-function 'smtpmail-send-it
      smtpmail-starttls-credentials
      '(("smtp.gmail.com" 587 nil nil))
      smtpmail-auth-credentials
      (expand-file-name "~/.authinfo")
      smtpmail-default-smtp-server "smtp.gmail.com"
      smtpmail-smtp-server "smtp.gmail.com"
      smtpmail-smtp-service 587
      ;; This can be commented out for a less verbose output
      smtpmail-debug-info t)

(require 'smtpmail)
```

34.3 Setting up completion over the addresses with the Mac address book

```
(if mac
    (progn
      (require 'external-abook)
       (setq external-abook-command "contacts -lf '%%e\t%%n' %s")
       ;; TODO: check if it's dynamic enough
       (eval-after-load "message"
         '(progn
            (add-hook 'mail-mode-hook
```

```
                        '(lambda ()
                           (define-key message-mode-map "\C-c\t" 'external-abook-try-
```

34.4 Setting default sending modality

```
(setq mail-user-agent 'sendmail-user-agent)
```

35 Gnus settings

35.1 Server settings

```
(setq gnus-select-method '(nntp "news.gmane.org"))
;; Set also comp.* hierarchy
(setq gnus-secondary-select-methods
      '(
        ;; Configuration for http://www.eternal-september.org/
        (nntp "eternal"
              (nntp-authinfo-file "~/.authinfo")
              (nntp-address "news.eternal-september.org")
              (nntp-port-number 119))))
```

35.2 Old messages settings

```
(setq gnus-large-newsgroup 500)
(setq gnus-fetch-old-headers nil)
```

35.3 Appearance

```
;; Changing modeline to include also the date of the message
(setq gnus-summary-line-format "%U%R%z%I%(%[%4L: %-23,23f%]%) %s--%d\n")
```

35.4 Avoid the annoying saving of the .news file

```
(add-hook 'gnus-started-hook
          (lambda ()
            (when (buffer-live-p gnus-dribble-buffer)
              (with-current-buffer gnus-dribble-buffer
                (setq buffer-save-without-query t)))))
```

35.5 TODO Setting up some posting styles and other nice settings

36 Revision control systems

36.1 Magit

Nice interface for git.

```
(require 'magit)
```

36.2 Function to enable revert mode when in a git repository

It's nice to enable auto-revert-mode automatically on files which are surely in a git repository. To do this we can simply add a hook to find-file-hook

```
(defun is-git-file ()
  "Return nil unless the file is in the git files"
  (if
      (member (file-name-nondirectory buffer-file-name)
              (split-string  (shell-command-to-string "git ls-files")))
      (auto-revert-mode t)))

(add-hook 'find-file-hook 'is-git-file)
```

37 General settings

37.1 Mode for startup

```
(setq initial-major-mode 'python-mode)
```

37.2 Showing more things

```
(display-time-mode 1)
(transient-mark-mode 1)
(setq inhibit-startup-message t)
(setq initial-scratch-message nil)

(show-paren-mode t)
(column-number-mode t)
;; always truncate lines (useful for netbook), not working yet in ORG MODE
(setq truncate-lines nil)
```

```
;; Setting indent-tabs-mode for only spaces
(setq-default indent-tabs-mode nil)
```

37.3 IDO mode

```
(require 'ido)
(ido-mode t)
;; otherwise it will try to connect to old servers all the time
(setq ido-enable-tramp-completion t)
(setq ido-enable-flex-matching nil)
;; regexp matching also
(setq ido-enable-regexp nil)
(setq ido-use-url-at-point t)
```

37.3.1 Use IDO when possible

We can advice the **completing-read** default function to use IDO when it's possible

```
(defvar ido-enable-replace-completing-read t
  "If t, use ido-completing-read instead of completing-read if possible.

    Set it to nil using let in around-advice for functions where the
    original completing-read is required.  For example, if a function
    foo absolutely must use the original completing-read, define some
    advice like this:

    (defadvice foo (around original-completing-read-only activate)
      (let (ido-enable-replace-completing-read) ad-do-it))")

;; Replace completing-read wherever possible, unless directed otherwise
(defadvice completing-read
  (around use-ido-when-possible activate)
  (if (or (not ido-enable-replace-completing-read) ; Manual override disable ido
          (boundp 'ido-cur-list)) ; Avoid infinite loop from ido calling this
      ad-do-it
    (let ((allcomp (all-completions "" collection predicate)))
      (if allcomp
          (setq ad-return-value
                (ido-completing-read prompt
                                     allcomp
```

```
                                        nil require-match initial-input hist def))
        ad-do-it))))
```

37.4 Windmove

```
(defcustom windmove-key
  'shift
  "key for moving between windows"
  :type 'symbol
  :group 'miniconf)

(windmove-default-keybindings windmove-key)
```

37.5 Workarounds

Compiling on emacs 23.2 often gives some strange errors, this is to avoid them

```
(setq warning-suppress-types nil)
```

38 Flymake

38.1 Setting up flymake

```
(require 'flymake)

(defun activate-flymake ()
  "Activates flymake when real buffer and you have write access"
  (if (and
       (buffer-file-name)
       (file-writable-p buffer-file-name))
      (flymake-mode t)))
```

38.2 Adding errors to modeline

With this the error output of othe current line will appear right below in the modeline

```
(defun my-flymake-show-help ()
  (when (get-char-property (point) 'flymake-overlay)
    (let ((help (get-char-property (point) 'help-echo)))
      (if help (message "%s" help)))))
```

```
(add-hook 'post-command-hook 'my-flymake-show-help)
```

38.3 Flymake for python

We check the errors given by 3 different programs:

- epylint (which runs pylint and make it more parsable)
- pyflakes
- pep8

The errors or warnings appear right in the source code.

```
(defun flymake-python-init ()
  (let* ((temp-file (flymake-init-create-temp-buffer-copy
                     'flymake-create-temp-inplace))
         (local-file (file-relative-name
                      temp-file
                      (file-name-directory buffer-file-name))))
    (list "pycheckers" (list local-file))))

(add-to-list 'flymake-allowed-file-name-masks
             '("\\.py\\'" flymake-python-init))

;; Using function is preferred when quoting functions
;; (add-hook 'python-mode-hook (function activate-flymake))
```

39 Customizable variables

Still thinking how to manage this for best usage

```
(defgroup variables nil "private variables")
(defcustom private "private" "file with private settings"
:group 'variables)
```

40 Server stuff

This will start the server for the GUI version of emacs, make sure you set up correctly the emacsclient, for example on the mac

```
alias emacsclient='/Applications/Emacs.app/Contents/MacOS/bin/emacsclient'
# and then setup your $EDITOR to emacsclient...

(if window-system
    (progn
      (require 'server)
      (server-start)))
```

41 Fun

41.1 Fortune settings

```
;; TODO: make it a defcustom also
(setq fortune-dir "/opt/local/share/games/fortune/")
```

41.2 Some mac tricks

41.2.1 Open the terminal

```
(defun mac-open-terminal ()
  (interactive)
  (let ((dir ""))
    (cond
     ((and (local-variable-p 'dired-directory) dired-directory)
      (setq dir dired-directory))
     ((stringp (buffer-file-name))
      (setq dir (file-name-directory (buffer-file-name))))
     )
    (do-applescript
     (format "
tell application \"Terminal\"
  activate
  try
    do script with command \"cd %s\"
  on error
    beep
  end try
end tell" dir))
    ))
```

41.2.2 Growl popup

This function can be pretty nice during presentations, it will popup the last pressed key via growl.

```
   (defun growl-popup (msg)
  "Pop up a growl notification with MSG, or display an Emacs message.
The \"growlnotify\" program is used if ‘window-system’ is non-nil and
the program is found in ‘exec-path’; otherwise ‘message’ is used."
  (interactive)
  (if (and window-system (executable-find "growlnotify"))
      (shell-command (concat "growlnotify -a /Applications/Emacs.app/ -m "
                             (shell-quote-argument msg)))
    (message msg)))

(defun popup-last ()
  (interactive)
  (let
      ((last-key (key-description (this-command-keys))))
    ;; check if we don't have a "stupid" sequence
    (unless
        (= (length (this-command-keys-vector)) 1)
        (growl-popup last-key))))
```

Now we also create two other functions to enable and disable it

```
(setq growl-mode nil)

(defun growl ()
  (interactive)
  (if (not growl-mode)
      (progn
        (message "enabling growl mode notification")
        (add-hook 'pre-command-hook 'popup-last)
        (setq growl-mode t))
    (progn
      (setq-default pre-command-hook (remq 'popup-last pre-command-hook))
      (message "disabling growl mode notification")
      (setq growl-mode nil))))
```

42 Global keys settings

```
;; compile facilities
(global-set-key [f5] 'recompile)

;; newline like textmate
(global-set-key (kbd "M-RET") 'newline-force)
(global-set-key [M-S-return] 'newline-force-close)

;; cvs stuff
(global-set-key "\C-xg" 'magit-status)

;; org keys
(global-set-key "\C-c\C-l" 'org-annotate-file)
(global-set-key "\C-cr" 'org-remember)
(global-set-key "\C-ca" 'org-agenda)
(global-set-key "\C-c\C-x\C-o" 'org-clock-out)
(global-set-key "\C-c\C-x\C-i" 'org-clock-in)

;; senator
(global-set-key "\M-." 'semantic-complete-jump-local)
(global-set-key "\M-?" 'semantic-ia-fast-jump)

;; overriding defualt not so smart visualization
(global-set-key "\C-x\C-b" 'ibuffer) ;; manage buffers with ibuffer

;; visualization
(global-set-key [f11] 'full)

;; splitting windows facilitites
;; TODO: pass to a sensible splitting instead
(global-set-key [f2] 'split-window-horizontally)
(global-set-key [f1] 'delete-window)

(global-set-key (kbd "<C-f9>") 'cycle-font)

;; elscreen nice stuff
(global-set-key (kbd "M-<left>") 'elscreen-previous)
(global-set-key (kbd "M-<right>") 'elscreen-next)
```

```
;; narrows to the actual function or class analyzed
;; C-x n w to widen again
(global-set-key "\C-xnn" 'semantic-narrow-to-tag)

(global-set-key [(meta shift l)] 'select-line)

;; TODO: the senator stuff should be enabled only where senator actually works!!
(global-set-key [f6] 'senator-fold-tag-toggle)

(global-set-key (kbd "M-n") 'senator-next-tag)
(global-set-key (kbd "M-p") 'senator-previous-tag)
```

43 External configuration files

Some packages use to store their configuration in external files and not in elisp code. For example:

- gnus:
  - ~/.newsrc
  - ~/.newsrc-dribble (an open buffer saving actual status)
  - ~/News (all the message fetched and so on)
- javadoc-help
  - ~/.javadoc-help (configuration about the sdks)