Configurations for GNU Emacs

Takaaki Ishikawa

Table of Contents

1. はじめに

- この文章は、org2dokuwiki.pl を使って生成しています。
- init.el 本体は、github に公開しています。
- init.org から init.el、init.pdf、init.odt、wiki を生成しています。
- 一部の関数定義を utility.el に分離しています。
- コピペだけで動かなかった場合は、ページ最下部のコメント欄にご意見をどうぞ。

2. 基本設定

この設定群を利用するために設定です。おそらく記述しておかないと動きません。

2.1. init.el のヘッダ

```
;;;; Configurations for Emacs
;;;;                                          Takaaki ISHIKAWA  <takaxp@ieee.org>
;;;; Cite: http://www.mygooglest.com/fni/dot-emacs.html (GREAT!)
```

2.2. cl を使う

http://lisperblog.blogspot.com/2008/12/blog-post_18.html

```
(eval-when-compile (require 'cl-lib))
```

2.3. リストで関数を渡せる autoload-if-found を使う

http://d.hatena.ne.jp/jimo1001/20090921/1253525484

eval-after-load とのペアでマクロ化したバージョンもある（次章参照）。

http://e-arrows.sakura.ne.jp/2010/03/macros-in-emacs-el.html

```
(defun autoload-if-found (functions file &optional docstring interactive type)
```

```
  "set autoload iff. FILE has found."
  (if (not (listp functions))
      (setq functions (list functions)))
  (and (locate-library file)
       (progn
         (dolist (function functions)
           (autoload function file docstring interactive type))
         t )))
```

2.4. [eval-after-autoload-if-found.el] 遅延読み込み

Twitter でぼやいていたら @cvmat さんが降臨して次のマクロを作ってくださいました。感謝感謝。

https://gist.github.com/3513287

autoload-if-found で遅延読み込みすると、eval-after-load と組み合わせるので、どうしてもインデントが増えてしまう。

例えば、cycle-buffer を遅延読み込みしたい場合、 `setq` で変数を書き換えするために随分とインデントが進んでいます。

```
(when (autoload-if-found
       '(cycle-buffer cycle-buffer-backward) "cycle-buffer" nil t)
  (with-eval-after-load "cycle-buffer"
    (setq cycle-buffer-allow-visible t)
    (setq cycle-buffer-show-length 12)
    (setq cycle-buffer-show-format '(" <(%s)>" . " %s"))))
```

これを、次の eval-after-autoload-func.el を使うと、非常にシンプルになります。行数も桁数もスッキリです。

```
(eval-after-autoload-if-found
 '(cycle-buffer cycle-buffer-backward) ;; autoload で反応させる関数
 "cycle-buffer" nil t nil      ;; 反応させた関数のコールで読むパッケージ指定
 '(;; パッケージ読み込み後の設定
   (setq cycle-buffer-allow-visible t)
   (setq cycle-buffer-show-length 12)
   (setq cycle-buffer-show-format '(" <(%s)>" . " %s"))))
```

戻り値を判定して、グローバルなキーアサインもできます。存在しないパッケージの関数呼び出しを明示的に防ぐには有効です。 `hook` も同様です。

```
(when (eval-after-autoload-if-found
       '(cycle-buffer cycle-buffer-backward) "cycle-buffer" nil t nil
       '((setq cycle-buffer-allow-visible t)
         (setq cycle-buffer-show-length 12)
         (setq cycle-buffer-show-format '(" <(%s)>" . " %s"))
         ;; パッケージのキーアサインはこちら
         ;; (define-key xxx-map (kbd "q") 'hoge)
         ))
  ;; グローバルはこちら
  (global-set-key (kbd "M-]") 'cycle-buffer)
  (global-set-key (kbd "M-[") 'cycle-buffer-backward)
  ;; パッケージに紐付いたフックはこちらへ
  ;; (add-hook 'xxx-hook 'hogehoge)
  ;;
  ;; ビルドインではない mode の auto-mode-alist 設定も必要ならこちら
  ;; (push '("\\.hoge$" . hoge-mode) auto-mode-alist)
  )
```

2.4.1. マクロ版

https://gist.github.com/3499459

```
(defmacro eval-after-autoload-if-found
    (functions file &optional docstring interactive type &rest after-body)
  "Set up autoload and eval-after-load for FUNCTIONS iff. FILE has found."
  `(let* ((functions ,functions)
          (docstring ,docstring)
          (interactive ,interactive)
          (type ,type)
          (file ,file))
     (when (locate-library file)
       (mapc (lambda (func)
               (autoload func file docstring interactive type))
             (if (listp functions)
                 functions
               (list functions)))
       ,@(when after-body
           `((eval-after-load file '(progn ,@after-body))))
       t)))
```

2.4.2. 関数版

関数版にリスト disabled-packages を追加しました。これは、事前にリストに Lisp ファイル名を入れておくと、一切の設定をスキップするものです。 eval-after-atoload-if-found を定義する前に次のような変数を設定しておきます。バイトコンパイルしていないファイルに書いて置けば、パッケージの ON/OFF を簡単に制御できます。

```
(setq disabled-packages ;; 追加されていないものは、有効
      '(("org" . t) ;; 無効
        ("web-mode" . nil) ;; 有効
        ("helm-config" . t))) ;; 無効
```

https://gist.github.com/3513287

```
(defun eval-after-autoload-if-found
    (functions file &optional docstring interactive type after-body)
  "Set up autoload and eval-after-load for FUNCTIONS iff. FILE has found."
  (let ((enabled t)
        (package nil))
    (when (boundp 'disabled-packages)
      (dolist (package disabled-packages)
        (when (and (stringp (car package)) (equal file (car package)))
          (when (cdr package)
            (setq enabled nil)))))
    (when enabled ;; if disabled then return nil
      (when (locate-library file)
        (mapc (lambda (func)
                (autoload func file docstring interactive type))
              (if (listp functions)
                  functions
                (list functions)))
        (when after-body
          (eval-after-load file `(progn ,@after-body)))
        t))))
```

2.5. [library-p] load-path にライブラリがあるかを判定

パッケージが load-path に存在していて使える状態にあるかを調べます。もし存在しなければ、メッセージバッファに [NOT FOUND]を刻みます。

libraries には複数を指定でき、すべてが使える状態の場合のみ t が返ります。

"org" を渡したり、 '("org" "helm") を渡したりできます。

```
(defun library-p (libraries)
  "Return t when every specified library can be located. "
  (let ((result t))
    (mapc (lambda (library)
            (unless (locate-library library)
              (message "--- [NOT FOUND] %s" library)
              (setq result nil)))
          (if (listp libraries)
              libraries
            (list libraries)))
    result))
```

2.6. 警告の抑制

起動時に警告が出てうっとうしい場合に使います。起動直後に呼ばれるように、 .emacs の上の方に書いておくとよいと思います。

http://d.hatena.ne.jp/kitokitoki/20100425/p1

```
(setq byte-compile-warnings
      '(free-vars unresolved callargs redefine obsolete noruntime
                  cl-functions interactive-only make-local))
```

2.7. 起動時間の計測

emacs-init-time を実行すると、Emacs の起動にかかった時間が表示されます。個人的にはミリ秒表示が好きなので、手を加えます。元ネタはすぎゃーんメモからです。感謝。

```
(add-hook 'after-init-hook
          (lambda ()
            (message "--- Emacs booting time: %.0f [msec]"
                     (* 1000
                        (float-time (time-subtract
                                     after-init-time
                                     before-init-time))))))
```

2.8. C-x C-c で容易に Emacs を終了させないように質問する

y-or-n-p を指定するだけです。

```
(setq confirm-kill-emacs 'y-or-n-p)
```

以前は、 C-x C-c を以下の関数に割り当てて、任意の質問文で入力を求めていました。

```
;;; Cite: http://flex.ee.uec.ac.jp/texi/emacs-jp/emacs-jp_12.html
;;; Cite: http://d.hatena.ne.jp/Ubuntu/20090417/1239934416
;; A simple solution is (setq confirm-kill-emacs 'y-or-n-p).
(defun confirm-save-buffers-kill-emacs (&optional arg)
  "Show yes or no when you try to kill Emacs"
  (interactive "P")
  (cond (arg (save-buffers-kill-emacs))
```

```
        (t
         (when (yes-or-no-p "Are you sure to quit Emacs now? ")
           (save-buffers-kill-emacs)))))
```

2.8.1. キーバインド

Show yes or no when you try to kill Emacs

```
(global-set-key (kbd "C-x C-c") 'confirm-save-buffers-kill-emacs)
```

2.9. Messages 出力を封じるためのマクロ

shut-up.el というマクロがあり、現在はそちらを使っています。

```
(defun hoge ()
  (interactive)
  (let ((message-log-max nil))
    (shut-up (recentf-save-list))))
```

（参考）Emacs - エコーエリアや **Messages** バッファにメッセージを表示させたくない - Qiita

非常に強力です。自分は、 recentf-save-list を find-file-hook にぶら下げていますが、そのままだと org-agenda の初回実行時にたくさんのメッセージが出てしまうところ、このマクロを介すだけで抑制可能です。 message-log-max で制御できるのがすごい。

```
(defmacro with-suppressed-message (&rest body)
  "Suppress new messages temporarily in the echo area and the `*Messages*'
buffer while BODY is evaluated."
  (declare (indent 0))
  (let ((message-log-max nil))
    `(with-temp-message (or (current-message) "") ,@body)))
```

3. コア設定

Emacs を操作して文書編集する上で欠かせない設定です。

3.1. 言語／文字コード

徹底的に UTF-8 に合わせます。

save-buffer-coding-system を設定すると、 buffer-file-coding-system の値を無視して、指定した save-buffer-coding-system の値でバッファを保存する。つまり、 buffer-file-coding-system に統一するなら設定不要。

set-default-coding-systems か prefer-coding-system を設定すると、同時に file-name-coding-system=、=set-terminal-coding-system=、=set-keyboard-coding-system も同時に設定される。=prefer-coding-system= は、文字コード自動判定の最上位判定項目を設定する。

set-buffer-file-coding-system は、X とのデータやりとりを設定する。

```
(prefer-coding-system 'utf-8-unix)
(set-language-environment "Japanese")
(set-locale-environment "en_US.UTF-8") ; "ja_JP.UTF-8"
(set-default-coding-systems 'utf-8-unix)
```

```
(set-selection-coding-system 'utf-8-unix)
(set-buffer-file-coding-system 'utf-8-unix)

(set-clipboard-coding-system 'utf-8) ; included by set-selection-coding-system
(set-keyboard-coding-system 'utf-8) ; configured by prefer-coding-system
(set-terminal-coding-system 'utf-8) ; configured by prefer-coding-system
(setq buffer-file-coding-system 'utf-8) ; utf-8-unix
(setq save-buffer-coding-system 'utf-8-unix) ; nil
(set-buffer-process-coding-system 'utf-8 'utf-8)
(setq process-coding-system-alist
      (cons '("grep" utf-8 . utf-8) process-coding-system-alist))
```

3.2. 日本語入力

Emacs23 用にインラインパッチを適用している場合に使います。
Lion でも使える自分用にカスタマイズした inline-patch を使っています。

Emacs24 用には、Mavericks 対応したパッチを使っています。

Emacs24.5 用はこちら。

```
(when (fboundp 'mac-add-key-passed-to-system)
  (setq default-input-method "MacOSX")
  (mac-add-key-passed-to-system 'shift))
```

3.3. [ag.el] 検索

検索には `The Silver Searcher` を使います。あらかじめインストールしておく必要があります。MacPorts の場合、the_silver_searcher の名称で頒布されています。 `exec-path` に `/opt/local/bin` が含まれていることを確認してください。

```
the_silver_searcher @0.18.1 (textproc)
A code-searching tool similar to ack, but faster.
```

カスタマイズした関数を `C-M-f` にぶら下げています。

```
(when (eval-after-autoload-if-found
       '(my:ag) "ag" nil t nil
       '((setq ag-highlight-search t)
         (setq ag-reuse-buffers t) ;; nil=別ウィンドウが開く
         (setq ag-reuse-window t)  ;; nil=結果を選択時に別ウィンドウに結果:を出す
         ;; q でウィンドウを抜ける
         (define-key ag-mode-map (kbd "q") 'delete-window)

         (defun my:ag ()
           (interactive)
           (call-interactively 'ag)
           (switch-to-buffer-other-frame "*ag search*"))))

  (global-set-key (kbd "C-M-f") 'my:ag))
```

3.4. 基本キーバインド

次の機能にキーバインドを設定する。

- Cmd+V でペースト（Mac 用）
- Cmd と Option を逆にする（Mac 用）
- 削除

```
(when (eq window-system 'ns)
  (global-set-key (kbd "M-v") 'yank)
  (setq ns-command-modifier 'meta)
  (setq ns-alternate-modifier 'super)
  (global-set-key [ns-drag-file] 'ns-find-file) ; D&D for Emacs23
  (setq ns-pop-up-frames nil)) ; D&D for Emacs23
(global-set-key [delete] 'delete-char)
(global-set-key [kp-delete] 'delete-char)
```

3.5. ナローイングするか

ナローイングを有効にする。デフォルトは、ナローイングを知らないユーザが「データが消えた！」と勘違いしないように、無効になっている。

Org でナローイングを使う場合は、特に設定しなくてもよい。

```
(put 'narrow-to-region 'disabled nil)
```

3.6. バッファの終わりでの newline を禁止する

```
;; Avoid adding a new line at the end of buffer
(setq next-line-add-newlines nil)
```

3.7. 常に最終行に一行追加する

```
;; Limit the final word to a line break code (automatically correct)
(setq require-final-newline t)
```

3.8. 長い文章を右端で常に折り返す

```
(setq truncate-lines nil)
(setq truncate-partial-width-windows nil)
```

3.9. バッファが外部から編集された場合に自動で再読み込みする

`auto-save-buffers` を使っていれば、バッファは常に保存された状態になるため、revert が即座に反映される。適宜バックアップツールと組み合わせないと不安な場合もあるかも。

```
(global-auto-revert-mode 1)
```

3.10. [uniquify.el] 同じバッファ名が開かれた場合に区別する

ビルトインの `uniquify` を使います。

```
(setq uniquify-buffer-name-style 'post-forward-angle-brackets)
```

3.11. マウスで選択した領域を自動コピー

マウスで選択すると、勝手にペーストボードにデータが流れます。

```
(setq mouse-drag-copy-region t)
```

3.12. パッケージ管理

Cask+Pallet の環境を採用しました。それまでは、特定のディレクトリに必要な elisp をダウンロードしておいたり、git から取り寄せて、それらを load-path に設定するスクリプトを準備するなど、個人的なルールで運用してきましたが、希望の機能を Cask が提供しているので、Emacs24.4 になるタイミングで移行しました。

ただし、頒布元が危ういようなファイルはやはり個人で管理しておきたいので、Cask で管理する対象は、MEPLA 経由で入手可能なメンテナンスが行き届いたパッケージに限定しています。また、普通の使い方（casl.el を読み込んで初期化）をしていると、起動時に少し時間を要するので、所定のディレクトリに Cask で取り寄せたすべてのファイルをコピーして、そのディレクトリだけを load-path で指定するという使い方もしています。今のところ大きな問題は生じていません。

3.12.1. Cask のセットアップ

以下は自分用のメモです。

1. curl -fsSkL https://raw.github.com/cask/cask/master/go | python
2. ~/.cask/bin に PATH を通す (see .zshenv, export PATH="${HOME}/.cask/bin:{$PATH}")
3. cask upgrade
4. cd ~/.emacs.d
5. cask init ;; ~/.emacs.d/Cask が存在しない場合だけ実行
6. cask install

3.12.2. load-path を一箇所にして起動を高速化

Cask を使うと、個々のパッケージが独立に load-path に設定されます。これにより依存関係がスッキリするわけですが、数が増えると起動時間が遅くなります。重いです。自分の例では、800[ms]のオーバーヘッドでした。これを避けるには、load-path を一箇所に集約することが効きます。オーバーヘッドは約 100[ms]まで集約できました。場合によっては依存関係に問題が生じる可能性がありますが、今のところは問題になっていません。

1. `~/.emacs.d/.cask/package` なるフォルダを作る
2. `~/.emacs.d/.cask/24.4.1/elpa/*/*` と
   `~/.emacs.d/.cask/24.4.1/elpa/*/lisp/*` をすべて上記フォルダにコピー
3. ~/.emacs で、~/.emacs.d/.cask/package を load-path に設定し、Cask は読み込まない

`M-x lis-packges` を使って新しいパッケージをインストールする時だけ、以下のフラグを nil に書き換えて Emacs を起動します。

```
(if t
    (load-path-setter '("~/.emacs.d/.cask/package") 'load-path)
  (when (require 'cask "~/.cask/cask.el" t) (cask-initialize)) ;; 800[ms]
  (when (require 'pallet nil t) (pallet-mode t)))
```

Cask で新しいパッケージを導入したり、既存のパッケージを更新したら、その都度、package ディレクトリにコピーします。手動でやると面倒なので、次のようなスクリプトで対処します。

```
#!/bin/sh
CASKPATH=/Users/taka/.emacs.d/.cask
```

```
VERSION=24.5.1
SUBDIR=package
if [ -d "$CASKPATH/$SUBDIR" ]; then
    echo "--- Remove $CASKPATH/$SUBDIR"
    rm -rf $CASKPATH/$SUBDIR
fi
mkdir -p $CASKPATH/$SUBDIR

cd ~/.emacs.d
echo "--- Cask install"
cask install

echo "--- Cask update"
cask update

echo "--- Copying elisp files"
/bin/cp -rf /Users/taka/.emacs.d/.cask/$VERSION/elpa/*/* $CASKPATH/$SUBDIR
cd $CASKPATH/$SUBDIR

echo "--- Done"
```

3.13. インデント

```
(setq-default tab-width 2)
(setq-default indent-tabs-mode nil)
(setq indent-line-function 'insert-tab)
;; (add-hook 'org-mode-hook
;;           '(lambda()
;;              (setq indent-line-function 'insert-tab)))
```

3.14. [aggressive-indent] 即時バッファ整形

特定のメジャーモードで、とにかく整形しまくります。MELPA から入手できます。

```
(when (eval-after-autoload-if-found
       '(aggressive-indent-mode) "aggressive-indent")

  (add-hook 'emacs-lisp-mode-hook 'aggressive-indent-mode)
  (add-hook 'lisp-mode-hook 'aggressive-indent-mode)
  (add-hook 'perl-mode-hook 'aggressive-indent-mode)
  (add-hook 'c-mode-common-hook 'aggressive-indent-mode)
  (add-hook 'python-mode-hook 'aggressive-indent-mode)
  (add-hook 'nxml-mode-hook 'aggressive-indent-mode)
  (add-hook 'web-mode-hook 'aggressive-indent-mode))
```

3.15. ファイルリンクを辿る時に確認のメッセージを出さない

そのまま辿ってファイルオープンします。

```
(setq vc-follow-symlinks t)

;; autorevert.el の読み込みが必要
(setq auto-revert-check-vc-info t)
```

4. カーソル移動

カーソルの移動は、次のポリシーに従っています。デフォルトでは C-v/M-v で上下移動になっているが、M-v は windows のペーストに対応するので混乱を招くので使っていませ

ん。ページスクロールは標準の cua-base.el に記載されています。

行移動	C-n/C-p
ページ移動（スクロール）	M-n/M-p
ウィンドウ移動	C-M-n/C-M-p
バッファ切り替え	M-]/M-[

4.1. バッファ内のカーソル移動

先頭に移動、最終行に移動、ページ単位の進む、ページ単位の戻る、行数を指定して移動

```
(global-set-key (kbd "C-M-t") 'beginning-of-buffer)
(global-set-key (kbd "C-M-b") 'end-of-buffer)
;; Backward page scrolling instead of M-v
(global-set-key (kbd "M-p") 'scroll-down)
;; Frontward page scrolling instead of C-v
(global-set-key (kbd "M-n") 'scroll-up)
;; Move cursor to a specific line
(global-set-key (kbd "C-c g") 'goto-line)
```

4.2. バッファ間のカーソル移動

C-c o の代わりに、ウィンドウの移動をワンアクションで行う。

```
(global-set-key (kbd "C-M-p") '(lambda () (interactive) (other-window -1)))
(global-set-key (kbd "C-M-n") '(lambda () (interactive) (other-window 1)))
```

4.3. スクロールを制御

一行づつスクロールさせます。デフォルトではバッファの端でスクロールすると、半画面移動します。また、上下の端にカーソルがどのくらい近づいたらスクロールとみなすかも指定できます。

http://marigold.sakura.ne.jp/devel/emacs/scroll/index.html

非 ASCII 文字を扱っているときに一行づつスクロールしない場合は、scroll-conservatively の値を 1 ではなく大きい数字にすると直るかもしれません。

http://www.emacswiki.org/emacs/SmoothScrolling

scroll-margin を指定すると、カーソルがウィンドウの端から離れた状態でスクロールされます。

```
;; Scroll window on a line-by-line basis
(setq scroll-conservatively 1000)
(setq scroll-step 1)
(setq scroll-margin 0) ; default=0
```

スクロール時のジャンプが気になる場合は次のパッケージを使うとよいです。

http://adamspiers.org/computing/elisp/smooth-scrolling.el

```
(when (require 'smooth-scrolling nil t)
  (setq smooth-scroll-margin 1))
```

```
(when (autoload-if-found
       '(smooth-scrolling) "smooth-scrolling" nil t)
  (eval-after-load "smooth-scrolling"
    '(progn
       (setq smooth-scroll-margin 1))))
```

4.4. スクロールで表示を重複させる行数

```
;; Scroll window on a page-by-pabe basis with N line overlapping
(setq next-screen-context-lines 1)
```

4.5. [SmoothScroll.el] カーソル固定でスクロールする

https://raw.github.com/takaxp/EmacsScripts/master/SmoothScroll.el
https://github.com/pglotov/EmacsScripts/blob/master/SmoothScroll.el

カーソル位置と行を固定してバッファを背景スクロールできます。

オリジナルのままだとコンパイル時に警告がでるので、 `line-move-visual` で書き換えています。残念ながら最近は使っていません。

```
(eval-after-autoload-if-found
 '(scroll-one-up scroll-one-down) "smoothscroll")

(autoload-if-found
 '(scroll-one-up scroll-one-down) "smoothscroll" nil t)
```

4.5.1. キーバインド

```
(global-set-key (kbd "s-<up>") 'scroll-one-down)
(global-set-key (kbd "s-<down>") 'scroll-one-up)
```

4.6. [smooth-scroll.el] 滑らかなスクロール

良い感じです。ススススっとスクロールします。

```
(eval-after-autoload-if-found
 '(smooth-scroll) "smooth-scroll" nil t nil
    '((smooth-scroll-mode t)
      (setq smooth-scroll/vscroll-step-size 6)
      (setq smooth-scroll/hscroll-step-size 6)))
```

4.7. [point-undo.el] カーソル位置を簡単にたどる

autoload や autoload-if-found で定義すると、使いたい時に履歴が取れていないのでよろしくないです。起動時に有効化します。 bm.el で明示的にマーカーを残して履歴をたどる方が気に入っているので、最近は point-undo を使っていません。

```
(require 'point-undo nil t)
```

4.7.1. キーバインド

シングルキーを割り当てておくと使いやすいです。

```
;; [point-undo.el] Move the cursor to the previous position
(global-set-key (kbd "<f7>") 'point-undo)
;; [point-undo.el] Redo of point-undo
```

```
(global-set-key (kbd "S-<f7>") 'point-redo)
```

4.8. [cycle-buffer.el] カレントバッファの表示切り替え

http://www.emacswiki.org/emacs/download/cycle-buffer.el

Cycle-buffer を使うと、バッファの履歴をスライドショーのようにたどれます。ミニバッファに前後の履歴が表示されるので、何回キーを押せばいいかの目安になります。それを超える場合には、おとなしくバッファリストを使います。直近数件のバッファをたどるのに便利です。

```
(eval-after-autoload-if-found
 '(cycle-buffer cycle-buffer-backward) "cycle-buffer" nil t nil
    '((setq cycle-buffer-allow-visible t)
      (setq cycle-buffer-show-length 12)
      (setq cycle-buffer-show-format '(" <(%s)>" . " %s"))))
```

4.8.1. キーバインド

```
(global-set-key (kbd "M-]") 'cycle-buffer)
(global-set-key (kbd "M-[") 'cycle-buffer-backward)
```

4.9. [bm.el] カーソル位置をブックマークして追う

bm.el は、カーソル位置をブックマークしておくためのツールです。 `point-undo` と比較して、ユーザが明示的に位置を保存でき、見た目にも使いやすいです。以下の例では、`org-mode` のツリー内にブックマークがある時にも、上手い具合に表示ができるように調整してあります。カーソル移動は、順方向（ `bm-next` ）にだけ使っています。

```
(when (eval-after-autoload-if-found
       '(bm-toggle my:bm-next bm-buffer-save bm-buffer-restore
                   bm-buffer-save-all bm-repository-save
                   bm-repository-load) "bm" nil t nil
                   '((setq-default bm-buffer-persistence t) ;; t
                     (setq bm-repository-file "~/Dropbox/emacs.d/.bookmark")
                     ;; autoload との組み合わせでは無意味
                     ;;（after-init-hook を利用せよ）
                     ;; (setq bm-restore-repository-on-load t)
                     (setq bm-persistent-face 'bm-face)
                     (defun my:bm-toggle ()
                       "bm-toggle with updating history"
                       (interactive)
                       (bm-toggle)
                       (bm-save))

                     (defun my:bm-next ()
                       "bm-next with org-mode"
                       (interactive)
                       (bm-next)
                       (when (and (equal major-mode 'org-mode)
                                  (not (org-before-first-heading-p)))
                         (widen)
                         (org-overview)
                         (org-reveal)
                         (org-cycle-hide-drawers 'all)
                         (org-show-entry)
                         (show-children)
                         (org-show-siblings)))))
```

```
  (add-hook 'after-init-hook 'bm-repository-load)
  (add-hook 'find-file-hook 'bm-buffer-restore)
  (add-hook 'after-revert-hook 'bm-buffer-restore)
  (add-hook 'kill-buffer-hook 'bm-buffer-save)
  (add-hook 'after-save-hook 'bm-buffer-save)
  (add-hook 'vc-before-checkin-hook 'bm-buffer-save)
  (add-hook 'kill-emacs-hook 'bm-save)

  (global-set-key (kbd "<f10>") 'my:bm-toggle)
  (global-set-key (kbd "<C-f10>") 'my:bm-next))
```

4.10. [centered-cursor-mode.el] カーソル位置を中央に固定

`isearch-mode` の時だけ有効にしています。

```
(when (eval-after-autoload-if-found
       '(centered-cursor-mode) "centered-cursor-mode")

  (add-hook 'isearch-mode-hook
            '(lambda () (interactive) (centered-cursor-mode 1)))
  (add-hook 'isearch-mode-end-hook
            '(lambda () (interactive) (centered-cursor-mode -1))))
```

5. 編集サポート

5.1. 矩形編集／連番入力

Built-in の cua-base.el（CUA-mode）を使う。矩形選択は、領域選択後 `cua-toggle-rectangle-mark` でもできるが、 24.4 からは、 `C-x SPC` を押下すると矩形モードに入り直感的に矩形選択ができるようになっています。

```
(require 'cua-base)
(cua-mode 1)
(setq cua-enable-cua-keys nil)
```

矩形選択した後に、 `M-n` を押すと、連番をふれる。開始値、増加値を入力してから、hoge%03d.pgm などとすれば、hoge001、hoge002、、、と入力される。これと、org-mode の表機能（ `C-c |` で選択部分を簡単に表にできる）を組み合わせれば、連番で数値をふったテーブルを容易に作れる。

なお、標準の rect.el に以下の機能が実装されている。

矩形切り取り	C-x r k
矩形削除	C-x r d
矩形貼り付け	C-x r y
矩形先頭に文字を挿入	C-x r t
矩形を空白に変換する	C-x r c

5.2. Yank 時に装飾を取る

```
(setq yank-excluded-properties t)
```

5.3. ファイル保存時に時間を記録する

Built-in の time-stamp.el を使う。

バッファの保存時にタイムスタンプを記録する。以下の設定では、バッファの先頭から10 行以内に、"Last Update: " があると、"Last Update: 2011-12-31@12:00"のようにタイムスタンプが記録される。

```
;; org-tre-slide が有効ならタイムスタンプを更新しない
;;  （Undo 範囲が限定されてしまうため）

;; #+UPDATE 用
(when (eval-after-autoload-if-found
       '(update-stamp) "update-stamp" nil t nil
       '((setq update-stamp-start "UPDATE:[ \t]*")
         (setq update-stamp-format "%02H:%02M:%02S")
         (setq update-stamp-end "$")
         (setq update-stamp-line-limit 10))) ; def=8

  (add-hook 'before-save-hook
            '(lambda ()
               (if (boundp 'org-tree-slide-mode)
                   (unless org-tree-slide-mode) (update-stamp))
               (update-stamp))))

;; #+DATE 用
(when (eval-after-autoload-if-found
       '(time-stamp) "time-stamp" nil t nil
       '((setq time-stamp-start "DATE:[ \t]*")
         (setq time-stamp-format "%04y-%02m-%02d")
         (setq time-stamp-end "$")
         (setq time-stamp-line-limit 10) ; def=8
         ))

  (add-hook 'before-save-hook
            '(lambda ()
               (if (boundp 'org-tree-slide-mode)
                   (unless org-tree-slide-mode) (time-stamp))
               (time-stamp))))
(add-hook 'before-save-hook 'time-stamp)
(with-eval-after-load "time-stamp"
  (setq time-stamp-start "Last Update: ")
  (setq time-stamp-format "%04y-%02m-%02d@%02H:%02M")
  (setq time-stamp-end "$")
  (setq time-stamp-line-limit 10)) ; def=8
```

5.4. 選択リージョンを使って検索

検索語をミニバッファに入力するのが面倒なので、リージョンをそのまま検索語として利用します。

http://dev.ariel-networks.com/articles/emacs/part5/

```
(defadvice isearch-mode
    (around isearch-mode-default-string
```

```
            (forward &optional regexp op-fun recursive-edit word-p) activate)
  (if (and transient-mark-mode mark-active (not (eq (mark) (point))))
      (progn
        (isearch-update-ring (buffer-substring-no-properties (mark) (point)))
        (deactivate-mark)
        ad-do-it
        (if (not forward)
            (isearch-repeat-backward)
          (goto-char (mark))
          (isearch-repeat-forward)))
    ad-do-it))
```

5.5. ChangeLog モード

```
(setq user-full-name "Your NAME")
(setq user-mail-address "your@address.com")

(add-hook 'change-log-mode-hook
          '(lambda()
             (setq tab-width 4)
             (setq left-margin 4)))
```

5.6. テキストモード

http://d.hatena.ne.jp/NeoCat/20080211

とは言っても、Org-mode を知ってから .txt もテキストモードで開かなくなったので、ほぼ無意味な設定となりました。しかも、 nxml-mode で TAB が効かなくなる現象が起きているので、以下の設定はしない方がよさげ。

```
(add-hook 'text-mode-hook
          '(lambda()
             (setq tab-width 4)
             (setq indent-line-function 'tab-to-tab-stop)
             (setq tab-stop-list
                   '(4 8 12 16 20 24 28 32 36 40 44 48 52 56 60
                       64 68 72 76 80))))
```

5.7. C/C++モード

```
(push '("\\.h$" . c++-mode) auto-mode-alist)
```

5.8. C#モード

```
(when (eval-after-autoload-if-found
       '(csharp-mode) "csharp-mode" "Major mode for editing C# mode.")

  (push '("\\.cs$" . csharp-mode) auto-mode-alist))
```

5.9. Info モード

Org-mode の日本語翻訳済み info を読むための設定。翻訳プロジェクトで頒布しています。

```
(eval-after-autoload-if-found
 '(info org-info-ja) "info" nil t nil
 '((add-to-list 'Info-additional-directory-list
                (expand-file-name "~/devel/mygit/org-ja/work/"))
```

```
   (defun org-info-ja (&optional node)
     "(Japanese) Read documentation for Org-mode in the info system.
    With optional NODE, go directly to that node."
     (interactive)
     (info (format "(org-ja)%s" (or node ""))))))
```

5.10. R モード

```
(when (eval-after-autoload-if-found
       '(R-mode R) "ess-site" "Emacs Speaks Statistics mode")
  (push '("\\.[rR]$" . R-mode) auto-mode-alist))
```

5.11. nXML モード

```
(add-hook 'nxml-mode-hook
          '(lambda ()
             (define-key nxml-mode-map "\r" 'newline-and-indent)
             (setq auto-fill-mode -1)
             (setq nxml-slash-auto-complete-flag t)
             (setq tab-width 1)
             (setq nxml-child-indent 1)
             (setq indent-tabs-mode t)
             (setq nxml-attribute-indent 0)))
```

5.12. yaml モード

```
(when (eval-after-autoload-if-found
       '(yaml-mode) "yaml-mode")
  (push '("\\.yml$" . yaml-mode) auto-mode-alist))
```

5.13. json モード

```
(when (eval-after-autoload-if-found
       '(json-mode) "json-mode")
  (push '("\\.json$" . json-mode) auto-mode-alist))
```

5.14. javascript モード

```
(when (eval-after-autoload-if-found '(js2-mode) "js2-mode")
  (push '("\\.js$" . js2-mode) auto-mode-alist))

(when (eval-after-autoload-if-found '(ac-js2-mode) "ac-js2")
  (add-hook 'js2-mode-hook 'ac-js2-mode))

(if (executable-find "tern")
    (when (eval-after-autoload-if-found
           '(tern-mode) "tern" nil t nil
           '((tern-mode 1)
             (when (require 'tern-auto-complete nil t)
               (tern-ac-setup))))
      (add-hook 'js2-mode-hook 'tern-mode))

  (message "--- tern is NOT installed in this system."))
```

5.15. csv モード

```
(when (eval-after-autoload-if-found
       '(csv-mode) "csv-mode")
  (push '("\\.csv$" . csv-mode) auto-mode-alist))
```

5.16. ascii モード

カーソル下の文字のアスキーコードを別ウィンドウでリアルタイムに確認できます。

```
(eval-after-autoload-if-found '(ascii-on ascii-off) "ascii")
```

5.17. スペルチェック

Built-in の ispell を使う。チェックエンジンは、aspell を利用する。

'ns	sudo port install aspell aspell-dict-en
'x32	installer.exe and aspell-en from http://aspell.net/win32/

```
;;; Use aspell for spell checking instead of ispell.
(when (executable-find "aspell")
  (eval-after-autoload-if-found
   '(ispell-region) "ispell" nil t nil
   '((setq-default ispell-program-name "aspell")
     (when (eq window-system 'w32)
       (setq-default ispell-program-name
                     "C:/Program Files/Aspell/bin/aspell.exe"))
     ;;(setq ispell-grep-command "grep")
     ;; for English and Japanese mixed

     (add-to-list 'ispell-skip-region-alist '("[^\000-\377]"))
     (setq ispell-dictionarry "english")
     (setq ispell-personal-dictionary "~/.emacs.d/.aspell.en.pws")

     ;; This will also avoid an IM-OFF issue for flyspell-mode.
     ;;  (setq ispell-aspell-supports-utf8 t)
     ;;  (setq ispell-encoding8-command t)
     (setq ispell-local-dictionary-alist
           '((nil "[a-zA-Z]" "[^a-zA-Z]" "'" t
                  ("-d" "en" "--encoding=utf-8") nil utf-8))))))
```

5.17.1. キーバインド

```
;; Spell checking within a specified region
(global-set-key (kbd "C-c f 5") 'ispell-region)
```

5.18. リアルタイムスペルチェック

Built-in の flyspell.el を使います。

重いので現在は使っていません。

http://www.morishima.net/~naoto/fragments/archives/2005/12/20/flyspell/

```
(dolist
```

```
    (hook
     '(text-mode-hook change-log-mode-hook c++-mode-hook
                      latex-mode-hook org-mode-hook))
  (add-hook hook (lambda () (flyspell-mode 1))))

(add-hook 'c++-mode-hook
          (lambda () (flyspell-prog-mode)))

;; Auto complete との衝突を回避
(ac-flyspell-workaround)
```

5.19. [latex-math-preview.el] TeX 数式をプレビュー

- http://www.emacswiki.org/emacs/latex-math-preview.el
- http://transitive.info/software/latex-math-preview/

以下の設定では、 数式で <f7> を押すとプレビューが走り、さらに <f7> を押すとプレビューウィンドウを閉じるように動作します。通常、=q= でプレビューを閉じられます。

```
(when (eval-after-autoload-if-found
       '(latex-math-preview-expression
         latex-math-preview-insert-symbol
         latex-math-preview-save-image-file
         latex-math-preview-beamer-frame)
       "latex-math-preview" nil t nil
       '((define-key latex-math-preview-expression-mode-map (kbd "<f7>")
           'latex-math-preview-delete-buffer)))

  (global-set-key (kbd "<f7>") 'latex-math-preview-expression))
```

5.20. [po-mode.el] 翻訳ファイルの編集

http://www.emacswiki.org/emacs/PoMode
http://www.emacswiki.org/emacs/po-mode+.el

```
;;(autoload 'po-mode "po-mode+" nil nil)
;;(autoload 'po-mode "po-mode" nil t)
(when (eval-after-autoload-if-found
       '(po-mode) "po-mode")
  (push '("\\.po[tx]?\\'\\|\\.po\\$" . po-mode) auto-mode-alist))
```

5.21. [word-count.el] リージョン内の文字をカウントする

有効な頒布元に変更があった。word-count.el から新しい頒布元にたどりつける。

```
(when (eval-after-autoload-if-found
       '(word-count-mode) "word-count" "Minor mode to count words.")
  (global-set-key (kbd "M-+") 'word-count-mode))
```

と思ったら、ビルドインの simple.el に十分な機能なのがあった。

```
(global-set-key (kbd "M-=") 'count-words)
```

5.22. [yatex.el] YaTeX モード

```
(when (eval-after-autoload-if-found
       '(yatex-mode) "yatex" "Yet Another LaTeX mode" t nil
```

```
      '((setq YaTeX-kanji-code 4))) ;; 1=Shift JIS, 2=JIS, 3=EUC, 4=UTF-8
  (push '("\\.tex$" . yatex-mode) auto-mode-alist)
  ;; Disable auto line break
  (add-hook 'yatex-mode-hook
            '(lambda () (setq auto-fill-function nil))))
```

5.23. [wclock.el] 世界時計

http://pastelwill.jp/wiki/doku.php?id=emacs

```
(eval-after-autoload-if-found 'wclock "wclock")
```

5.24. [yasnippet.el] Emacs 用のテンプレートシステム

https://github.com/capitaomorte/yasnippet

- http://yasnippet-doc-jp.googlecode.com/svn/trunk/doc-jp/index.html
- http://d.hatena.ne.jp/IMAKADO/20080401/1206715770
- http://coderepos.org/share/browser/config/yasnippet
- https://github.com/RickMoynihan/yasnippet-org-mode

Org-mode との衝突を避ける

```
(when (eval-after-autoload-if-found
       '(yas-global-mode yas-minor-mode)
       "yasnippet" nil t nil
       '((setq yas-verbosity 2)
         (setq yas-snippet-dirs
               (list "~/Dropbox/emacs.d/yas-dict"
                     'yas-installed-snippets-dir)) ;; for Cask
         (custom-set-variables '(yas-trigger-key [tab]))
         (yas-global-mode 1)
         ))

  (dolist (hook
           (list 'perl-mode-hook
                 'c-mode-common-hook 'js2-mode-hook 'org-mode-hook
                 'python-mode-hook))
    (add-hook hook 'yas-minor-mode)))
(when (eval-after-autoload-if-found
       '(yas-minir-mode yas-expand yas-org-very-safe-expand)
       "yasnippet" nil t nil
       '((setq yas-verbosity 2)
         (setq yas-snippet-dirs
               (list "~/Dropbox/emacs.d/yas-dict"
                     'yas-installed-snippets-dir))
         (custom-set-variables '(yas-trigger-key [tab]))
         (yas-global-mode 1)

         (defun yas-org-very-safe-expand ()
           (let ((yas-fallback-behavior 'return-nil)) (yas-expand)))))

  (dolist (hook (list 'perl-mode-hook 'c-mode-common-hook))
    ;;                        'emacs-lisp-mode-hook))
    (add-hook hook 'yas-minor-mode))

  (add-hook 'org-mode-hook
```

```
            '(lambda ()
               (yas-minor-mode)
               ;; org-cycle (<TAB>) との衝突を避ける
               (setq yas-trigger-symbol [tab])
               (add-to-list 'org-tab-first-hook 'yas-org-very-safe-expand)
               (define-key yas-keymap [tab] 'yas-next-field))))
```

5.25. [sdic.el] 英辞郎で英単語を調べる

http://www.namazu.org/~tsuchiya/sdic/index.html

Emacs から辞書を使う。lookup を使う方法もあるが、Emacs から使うのは英辞郎に限定。

```
(when (autoload-if-found
       '(sdic-describe-word sdic-describe-word-at-point)
       "sdic" nil t)
  (eval-after-load "sdic"
    '(progn
       (setq sdic-face-color "#3333FF")
       (setq sdic-default-coding-system 'utf-8)
       ;; Dictionary (English => Japanese)
       (setq sdic-eiwa-dictionary-list
             '((sdicf-client "~/Dropbox/Dic/EIJIRO6/EIJI-128.sdic")))
       ;; Dictionary (Japanese => English)
       (setq sdic-waei-dictionary-list
             '((sdicf-client "~/Dropbox/Dic/EIJIRO6/WAEI-128.sdic"))))))

(eval-after-autoload-if-found
 '(sdic-describe-word sdic-describe-word-at-point) "sdic" nil t nil
 '((setq sdic-face-color "#3333FF")
   (setq sdic-default-coding-system 'utf-8)
   ;; Dictionary (English => Japanese)
   (setq sdic-eiwa-dictionary-list
         '((sdicf-client "~/Dropbox/Dic/EIJIRO6/EIJI-128.sdic")))
   ;; Dictionary (Japanese => English)
   (setq sdic-waei-dictionary-list
         '((sdicf-client "~/Dropbox/Dic/EIJIRO6/WAEI-128.sdic")))))
```

5.25.1. キーバインド

```
;; カーソルの位置の英単語の意味を調べる
(global-set-key (kbd "C-M-w") 'sdic-describe-word-at-point)
;; ミニバッファに英単語を入れて英辞郎を使う
(global-set-key (kbd "C-c w") 'sdic-describe-word)
```

5.26. MacOS の dictionary.app で辞書をひく

`osx-dictionary` なるパッケージが存在します。さくさくと高速に動作します。

```
(when (eval-after-autoload-if-found
       '(osx-dictionary-search-pointer osx-dictionary-search-input)
       "osx-dictionary" nil t nil
       '((setq osx-dictionary-dictionary-choice "英辞郎 第七版")))
  (global-set-key (kbd "C-M-w") 'osx-dictionary-search-pointer))
```

COBUILD5 をデフォルトで使うには、次のサイト参照してください。

- Collins COBULD 5 を Dictionary.app で利用できるようにする

私の場合は、できあがった辞書を `/Library/Dictionaries/` 以下に置いています。そ

の状態で dictionary.app の設定で辞書の優先順位を変えることで、常に COBUILD5 の情報を引っ張り出せます。もしくは、 `osx-dictionary-dictionary-choice` で辞書名を指定します。

5.27. MacOS の dictionary.app で COBUILD5 の辞書をひく（旧）

OS 標準の辞書アプリ（dictionary.app）を経由して、バッファに COBUILD5 のデータを流し込むことができます。

- 辞書(Dictionary).app を使い倒そう

以下の関数を準備します。

```
(defun dictionary ()
  "dictionary.app"
  (interactive)

  (let ((editable (not buffer-read-only))
        (pt (save-excursion (mouse-set-point last-nonmenu-event)))
        beg end)

    (if (and mark-active
             (<= (region-beginning) pt) (<= pt (region-end)) )
        (setq beg (region-beginning)
              end (region-end))
      (save-excursion
        (goto-char pt)
        (setq end (progn (forward-word) (point)))
        (setq beg (progn (backward-word) (point)))
        ))

    (let ((word (buffer-substring-no-properties beg end))
          ;;              (win (selected-window))
          (tmpbuf " * dict-process*"))
      (pop-to-buffer tmpbuf)
      (erase-buffer)
      (insert "Query: " word "\n\n")
      (start-process "dict-process" tmpbuf "dict.py" word)
      (goto-char 0)
      ;;         (select-window win)
      )))
```

これでカーソル以下の単語の情報が別ウィンドウに出ます。チェックし終わったら `C-x 1` (delete-other-windows) で表示を閉じます。 `q` で閉じられるようにしたり、ツールチップで表示したりもできるはずです。

マスタカさんのナイスソリューションをまだ試していないので、こちらの方がエレガントかもしれません。

- Emacs で Mac の辞書を sdic っぽく使う
- Emacs から Mac の辞書をお手軽に使う

なお、COBUILD5 の辞書データを dictionary.app で引けるようにするには以下の操作が必要です。

- Collins COBULD 5 を Dictionary.app で利用できるようにする

私の場合は、できあがった辞書を `/Library/Dictionaries/` 以下に置いています。その状態で dictionary.app の設定で辞書の優先順位を変えることで、常に COBUILD5 の情報を引っ張り出せます。

5.27.1. マイナーモード化

q で閉じたくなったのでマイナーモードを作りました。これまで通り、 C-M-w でカーソル下の単語を調べてポップアップで表示。カーソルはその新しいバッファに移しておき、q で閉じられます。新しいバッファ内で別な単語を C-M-w で調べると、同じバッファに結果を再描画します。

マイナーモード化した elisp は、gist で公開しています。

5.27.2. キーバインド

マイナーモード化した dict-app を使う場合は以下のようにします。sdic を使っている人は、sdic 用の設定と衝突しないように気をつけます。

```
(when (eval-after-autoload-if-found
       '(dict-app-search) "dict-app")
  ;; カーソルの位置の英単語の意味を調べる
  (global-set-key (kbd "C-M-w") 'dict-app-search))
```

5.28. [lookup.el] 辞書

最近使っていません。

```
;; .lookup/cache.el
(setq lookup-init-directory "~/env/dot_files/.lookup")

(autoload 'lookup "lookup" nil t)
(autoload 'lookup-region "lookup" nil t)
(autoload 'lookup-word "lookup" nil t)
(autoload 'lookup-select-dictionaries "lookup" nil t)

(setq lookup-search-modules
      '(("default"
         ("ndeb:/Users/taka/Dropbox/Dic/COBUILD5/cobuild" :priority t)
         ("ndeb:/Users/taka/Dropbox/Dic/COBUILD5/wordbank" :priority t)
         ("ndeb:/Users/taka/Dropbox/Dic/LDOCE4/ldoce4" :priority t)
         ("ndeb:/Users/taka/Dropbox/Dic/LDOCE4/bank" :priority t)
         ("ndeb:/Users/taka/Dropbox/Dic/LDOCE4/colloc" :priority t)
         ("ndeb:/Users/taka/Dropbox/Dic/LDOCE4/activ" :priority t))))

(setq lookup-agent-attributes
      '(("ndeb:/Users/taka/Dropbox/Dic/COBUILD5"
         (dictionaries "cobuild" "wordbank"))
        ("ndeb:/Users/taka/Dropbox/Dic/LDOCE4"
         (dictionaries "ldoce4" "bank" "colloc" "activ"))))

(setq lookup-dictionary-attributes
      '(("ndeb:/Users/taka/Dropbox/Dic/COBUILD5/cobuild"
         (title . "COBUILD 5th Edition")
         (methods exact prefix))
        ("ndeb:/Users/taka/Dropbox/Dic/COBUILD5/wordbank"
         (title . "Wordbank")
         (methods))
        ("ndeb:/Users/taka/Dropbox/Dic/LDOCE4/ldoce4"
         (title . "Longman 4th Edition")
         (methods exact prefix))
        ("ndeb:/Users/taka/Dropbox/Dic/LDOCE4/bank"
         (title . "LDOCE4 Examples and Phrases")
         (methods exact prefix menu))
```

```
        ("ndeb:/Users/taka/Dropbox/Dic/LDOCE4/colloc"
         (title . "LDOCE4 Collocation")
         (methods exact prefix))
        ("ndeb:/Users/taka/Dropbox/Dic/LDOCE4/activ"
         (title . "Longman Activator")
         (methods exact prefix menu))))

(setq lookup-default-dictionary-options
      '((:stemmer .  stem-english)))
(setq lookup-use-kakasi nil)

;;; lookup for dictionary (require EB Library, eblook, and lookup.el)
;; package download: http://sourceforge.net/projects/lookup
;; http://lookup.sourceforge.net/docs/ja/index.shtml#Top
;; http://www.bookshelf.jp/texi/lookup/lookup-guide.html#SEC_Top
                                        ;(load "lookup-autoloads") ; for 1.99
                                        ;(autoload 'lookup "lookup" nil t)
                                        ;(autoload 'lookup-region "lookup" nil
t)
                                        ;(autoload 'lookup-word "lookup" nil t)
                                        ;(autoload 'lookup-select-dictionaries
"lookup" nil t)
;; Search Agents
;; ndeb option requries "eblook" command
                                        ; Use expand-file-name!
                                        ;(setq lookup-search-agents `((ndeb ,
(concat homedir "/Dropbox/Dic/COBUILD5"))
                                        ;                            (ndeb ,
(concat homedir "/Dropbox/Dic/LDOCE4"))))
                                        ;(setq lookup-use-bitmap nil)
                                        ;(setq ndeb-program-name
"/usr/bin/eblook")
                                        ;(when (eq window-system 'ns)
                                        ;  (setq ndeb-program-name
"/opt/local/bin/eblook")
                                        ;  (setq ndeb-program-arguments '("-q"
"-e" "euc-jp"))
                                        ;  (setq ndeb-process-coding-system
'utf-8)) ; utf-8-hfs
```

5.28.1. キーバインド

```
(global-set-key (kbd "<f6>") 'lookup-word)
```

5.29. [cacoo] Cacoo で描く

画像をリサイズしてバッファに表示する用途にも使える。

```
(when (autoload-if-found 'toggle-cacoo-minor-mode "cacoo" nil t)
  (global-set-key (kbd "M--") 'toggle-cacoo-minor-mode)
  (eval-after-load "cacoo"
    '(progn
       (require 'cacoo-plugins))))

(when (eval-after-autoload-if-found
       '(toggle-cacoo-minor-mode) "cacoo" nil t nil
       '((require 'cacoo-plugins)))

  (global-set-key (kbd "M--") 'toggle-cacoo-minor-mode))
```

5.30. [iedit] バッファ内の同じ文字列を一度に編集する

iedit.el を使うと、バッファ内の同じ文字列を一度に編集することができる。部分重複のない変数名を置き換えるときに有用な場合がある。

```
(require 'iedit nil t)
```

5.31. [web-mode] HTML 編集

HTML 編集をするなら web-mode がお勧めです。古い HTML モードを使っている方は、移行時期です。以下の my:web-indent-fold では、 タブキーを打つたびにタグでくくられた領域を展開／非表示して整形します。Org-mode っぽい動作になりますが、操作の度にバッファに変更が加わったと判断されるので好みが分かれると思います。自動保存を有効にしているとそれほど気になりません。

```
(when (eval-after-autoload-if-found
       '(web-mode) "web-mode" "web-mode" t nil
       '((defun my:web-indent-fold ()
           (interactive)
           (web-mode-fold-or-unfold)
           (web-mode-buffer-indent)
           (indent-for-tab-command))

         ;; indent
         (setq web-mode-markup-indent-offset 1)

         ;; 色の設定
         (custom-set-faces
          ;; custom-set-faces was added by Custom.
          ;; If you edit it by hand, you could mess it up, so be careful.
          ;; Your init file should contain only one such instance.
          ;; If there is more than one, they won't work right.
          '(default ((t (:background "#FFFFFF" :foreground "#202020"))))
          '(web-mode-comment-face ((t (:foreground "#D9333F"))))
          '(web-mode-css-at-rule-face ((t (:foreground "#FF7F00"))))
          '(web-mode-css-pseudo-class-face ((t (:foreground "#FF7F00"))))
          '(web-mode-css-rule-face ((t (:foreground "#A0D8EF"))))
          '(web-mode-doctype-face ((t (:foreground "#82AE46"))))
          '(web-mode-html-attr-name-face ((t (:foreground "#C97586"))))
          '(web-mode-html-attr-value-face ((t (:foreground "#82AE46"))))
          '(web-mode-html-tag-face ((t (:foreground "##4682ae" :weight bold))))
          '(web-mode-server-comment-face ((t (:foreground "#D9333F")))))
         (define-key web-mode-map (kbd "<tab>") 'my:web-indent-fold)))


  ;; web-mode で開くファイルの拡張子を指定
  (setq auto-mode-alist
        (append '(("\\.phtml\\'" . web-mode)
                  ("\\.tpl\\.php\\'" . web-mode)
                  ("\\.jsp\\'" . web-mode)
                  ("\\.as[cp]x\\'" . web-mode)
                  ("\\.erb\\'" . web-mode)
                  ("\\.mustache\\'" . web-mode)
                  ("\\.djhtml\\'" . web-mode)
                  ("\\.html?\\'" . web-mode))
                auto-mode-alist)))
```

5.32. [zencoding-mode] HTML 編集の高速化（旧）

zencoding でタグ打ちを効率化します。今は emmet-mode を使います。

- http://www.emacswiki.org/emacs/ZenCoding

```
(when (eval-after-autoload-if-found
       '(zencoding-mode zencoding-expand-line)
       "zencoding-mode" "Zen-coding" t nil
       '((define-key zencoding-mode-keymap
           (kbd "M-<return>") 'zencoding-expand-line)))

  (add-hook 'sgml-mode-hook 'zencoding-mode)
  (add-hook 'html-mode-hook 'zencoding-mode)
  (add-hook 'web-mode-hook 'zencoding-mode))
```

5.33. [emmet-mode] zencoding の後継

```
(when (eval-after-autoload-if-found
       '(emmet-mode) "emmet-mode" nil t nil
       '((setq emmet-indentation 2)
         (setq emmet-move-cursor-between-quotes t)))
  (push '("\\.xml\\'" . nxml-mode) auto-mode-alist)
  (push '("\\.rdf\\'" . nxml-mode) auto-mode-alist)
  (add-hook 'sgml-mode-hook 'emmet-mode)
  (add-hook 'nxml-mode-hook 'emmet-mode)
  (add-hook 'css-mode-hook 'emmet-mode)
  (add-hook 'html-mode-hook 'emmet-mode)
  (add-hook 'web-mode-hook 'emmet-mode))
```

5.34. [describe-number] 16 進数などを確認

describe-number.el を使うと、16 進数表示や文字コードを確認できます。

```
(eval-after-autoload-if-found
 '(describe-number describe-number-at-point) "describe-number")
```

6. 表示サポート

6.1. モードラインのモード名を短くする

以前は自作したパッケージを使っていましたが、不具合も多く、調べると diminish.el という素晴らしいパッケージがあったので移行しました。これはマイナーモードの短縮表示なので、メジャーモードは個別にフックで mode-name を書き換えて対応します。use-package.el を使っていると依存関係から自動的にインストールされます。

```
(require 'mode-name-abbrev nil t)
```

diminish.el を使えば、短縮名に書き換えることも、存在自体を消してしまうこともできます。helm だけ行儀が悪いので、後段での設定時に diminish を呼ぶようにしています。

```
(when (require 'diminish nil t)
  (with-eval-after-load "isearch" (diminish 'isearch-mode))
  (with-eval-after-load "smooth-scroll" (diminish 'smooth-scroll-mode))
  (with-eval-after-load "whitespace" (diminish 'global-whitespace-mode))
  (with-eval-after-load "centered-cursor-mode"
    (diminish 'centered-cursor-mode))
```

```
  (with-eval-after-load "volatile-highlights"
    (diminish 'volatile-highlights-mode))
  (with-eval-after-load "aggressive-indent"
    (diminish 'aggressive-indent-mode " Ai"))
  (with-eval-after-load "emmet-mode" (diminish 'emmet-mode " e"))
  (with-eval-after-load "abbrev" (diminish 'abbrev-mode " a"))
  (with-eval-after-load "yasnippet" (diminish 'yas-minor-mode " y"))
  (with-eval-after-load "doxymacs" (diminish 'doxymacs-mode " d"))
  (with-eval-after-load "rainbow-mode" (diminish 'rainbow-mode))
  (with-eval-after-load "guide-key" (diminish 'guide-key-mode))
  (with-eval-after-load "org-autolist" (diminish 'org-autolist-mode))
;;;  (with-eval-after-load "helm" (diminish 'helm-mode " H"))
  )
```

メジャーモードは、単純に各モードの hook で対処します。

```
(add-hook 'c-mode-hook '(lambda () (setq mode-name "C")))
(add-hook 'js2-mode-hook '(lambda () (setq mode-name "JS")))
(add-hook 'c++-mode-hook '(lambda () (setq mode-name "C++")))
(add-hook 'csharp-mode-hook '(lambda () (setq mode-name "C#")))
(add-hook 'prog-mode-hook '(lambda () (setq mode-name "S")))
(add-hook 'emacs-lisp-mode-hook '(lambda () (setq mode-name "el")))
(add-hook 'python-mode-hook '(lambda () (setq mode-name "py")))
(add-hook 'perl-mode-hook '(lambda () (setq mode-name "pl")))
(add-hook 'web-mode-hook '(lambda () (setq mode-name "W")))
(add-hook 'lisp-interaction-mode-hook '(lambda () (setq mode-name "Lisp")))
```

6.2. モードラインの Narrow を短くする

標準では「Narrow」と表示されますが、「N」に短縮します。

```
(defvar my:narrow-display " N")
(setq mode-line-modes
      (mapcar
       (lambda (entry)
         (if (and (stringp entry)
                  (string= entry "%n"))
             '(:eval (if (and (= 1 (point-min))
                              (= (1+ (buffer-size)) (point-max))) ""
                       my:narrow-display)) entry))
       mode-line-modes))
```

6.3. モードラインの節約（VC-mode 編）

定形で表示されている Git を消します。

```
(with-eval-after-load "vc-hooks"
  (setcdr (assq 'vc-mode mode-line-format)
          '((:eval (replace-regexp-in-string "^ Git" " " vc-mode)))))
```

6.4. モードラインの色をカスタマイズする

```
(set-face-attribute 'mode-line nil :overline "#203e6f" :box nil)
(set-face-foreground 'mode-line "#203e6f")
(set-face-background 'mode-line "#b2cefb")
(set-face-attribute 'mode-line-inactive nil :overline "#94bbf9" :box nil)
(set-face-foreground 'mode-line-inactive  "#94bbf9")
(set-face-background 'mode-line-inactive "#d8e6fd")
```

6.4.1. 色セット例

- 青／白

	background	foreground	overline
active	558BE2	FFFFFF	566f99
inactive	94bbf9	EFEFEF	a4bfea

- 青

	background	foreground	overline
active	b2cefb	203e6f	203e6f
inactive	94bbf9	94bbf9	94bbf9

- 緑

	background	foreground	overline
active	b1fbd6	206f47	206f47
inactive	95f9c7	95f9c7	95f9c7

6.5. visible-bell のカスタマイズ

http://www.emacswiki.org/emacs/MilesBader を参考にカスタマイズしていました。現在は後継パッケージ（http://www.emacswiki.org/emacs/echo-bell.el）があり、MELPA から取れます。

visibl-bell を使うと、操作ミスで発生するビープ音を、視覚的な表示に入れ替えられます。ただ、デフォルトではバッファ中央に黒い四角が表示されて少々鬱陶しいので、ミニバッファの点滅に変更します。

```
(eval-after-autoload-if-found
 '(echo-area-bell) "echo-area-bell" nil t nil
 '((setq visible-bell t)
   (setq ring-bell-function 'echo-area-bell)))
;; パッケージ（echo-bell）の場合
(when (require 'echo-bell nil t)
  (echo-bell-mode 1)
  (setq echo-bell-string "")
  (setq echo-bell-background "#FFDCDC")
  (setq echo-bell-delay 0.1))
```

ビープ音も無しかつ視覚効果も無くすには、次のようにします。

see http://yohshiy.blog.fc2.com/blog-entry-171.html

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

6.6. 常に scratch を表示して起動する

session.el や desktop.el を使っていても、いつも *scratch* バッファを表示する。
そうじゃないと安心できない人向け。

```
;; Start Emacs with scratch buffer even though it call session.el/desktop.el
(add-hook 'emacs-startup-hook '(lambda () (switch-to-buffer "*scratch*")))
```

もしくは、以下でも OK かも。

```
(setq initial-buffer-choice t)
```

さらに、使われるメジャーモードと表示する文字列も制御できます。

```
(setq initial-major-mode 'text-mode)
(setq initial-scratch-message
      (concat "                                                                "
              (format-time-string "%Y-%m-%d (%a.)") "\n"
              "--------------------------------------------------"
              "--------------------------\n"))

(when (require 'buffer nil t)
  (setq initial-scratch-message nil)
  (setq initial-major-mode 'buffer-mode)
  (set-face-attribute 'header-line nil :overline nil :underline "#203e6f")
  (set-face-foreground 'header-line "#203e6f")
  (set-face-background 'header-line "#d8e6fd")

  (defvar my:header-clock
    (setq header-line-format
          (concat
           "  Buffers                                                       "
           (format-time-string "%Y-%m-%d (%a.)")))))

(global-set-key (kbd "C-M-s")
                '(lambda () (interactive) (switch-to-buffer "*scratch*")))
```

6.7. バッテリー情報をモードラインに表示する

```
;; Show battery information on the mode line.
(display-battery-mode t)
```

6.8. スクロールバーを非表示にする

スクロールバーを非表示にするには、nil を指定します。
右側に表示したい場合は、'right とします。

```
;; Show scroll bar or not
(set-scroll-bar-mode nil) ; 'right
```

6.9. ツールバーを非表示にする

ツールバーは使わないので非表示にします。

```
;; Disable to show the tool bar.
(tool-bar-mode 0)
```

6.10. 起動時のスプラッシュ画面を表示しない

```
;; Disable to show the splash window at startup
(setq inhibit-startup-screen t)
```

6.11. カーソル行の行数をモードラインに表示する

```
;; Show line number in the mode line.
(line-number-mode t)
```

6.12. カーソル行の関数名をモードラインに表示する

- emacs24.3 で重く感じるので外している。

```
;; Show function name in the mode line.
(which-function-mode t)
```

6.13. 時刻をモードラインに表示する

```
;; Show clock in in the mode line
(setq display-time-format "%H:%M")
(setq display-time-default-load-average nil)
(display-time-mode t)
```

6.14. 対応するカッコをハイライトする

Built-in の paren.el が利用できる。拡張版として mic-paren.el があり、現在はこれを利用している。

```
(when (eval-after-autoload-if-found
       '(paren-activate) "mic-paren" nil t nil
       '((setq paren-sexp-mode nil)
         (set-face-foreground 'paren-face-match "#FFFFFF")
         ;; Deep blue: #6666CC, orange: #FFCC66
         (set-face-background 'paren-face-match "66CC66")))

  (paren-activate))
```

paren.el の場合は以下の設定。

```
(setq show-paren-delay 0)
(show-paren-mode t)
;; (setq show-paren-style 'expression) ; カッコ内も強調
                                       ;(set-face-background 'show-paren-
match-face "#5DA4ff") ; カーソルより濃い青
(set-face-background 'show-paren-match-face "#a634ff")
(set-face-foreground 'show-paren-match-face "#FFFFFF")
(set-face-underline-p 'show-paren-match-face nil)
(setq show-paren-style 'parenthesis)
```

6.15. 全角スペースと行末タブ／半角スペースを強調表示する

http://ubulog.blogspot.jp/2007/09/emacs_09.html

英語で原稿を書く時に全角スペースが入っているを苦労するので、強調表示して編集中でも気づくようにします。また、行末のタブや半角スペースも無駄なので、入り込まないように強調しています。

```
;; スペース
(defface my:face-b-1
  '((t (:background "gray" :bold t :underline "red")))
  nil :group 'font-lock-highlighting-faces)
;; タブだけの行
(defface my:face-b-2
  '((t (:background "orange" :bold t :underline "red")))
  nil :group 'font-lock-highlighting-faces)
;; 半角スペース
(defface my:face-b-3 '((t (:background "orange")))
  nil :group 'font-lock-highlighting-faces)

(defvar my:face-b-1 'my:face-b-1)
(defvar my:face-b-2 'my:face-b-2)
(defvar my:face-b-3 'my:face-b-3)
(defadvice font-lock-mode (before my:font-lock-mode ())
  (font-lock-add-keywords
   major-mode
   ;; "[\t]+$" 行末のタブ
   '(("　" 0 my:face-b-1 append)
     ;;       ("[ ]+$" 0 my:face-b-3 append)
     ("[\t]+$" 0 my:face-b-2 append))))
(ad-enable-advice 'font-lock-mode 'before 'my:font-lock-mode)
(ad-activate 'font-lock-mode)
```

6.16. [migemo.el] ローマ字入力で日本語を検索する

http://0xcc.net/migemo/#download

以下は、cmigemo を使う設定です。

```
(when
    (eval-after-autoload-if-found
     '(migemo-init) "migemo" nil t nil
     '((setq completion-ignore-case t) ;; case-independent
       (setq migemo-command "cmigemo")
       (setq migemo-options '("-q" "--emacs" "-i" "\a"))
       (setq migemo-dictionary "/usr/local/share/migemo/utf-8/migemo-dict")
       (setq migemo-user-dictionary nil)
       (setq migemo-regex-dictionary nil)
       (setq migemo-use-pattern-alist t)
       (setq migemo-use-frequent-pattern-alist t)
       (setq migemo-pattern-alist-length 1024)
       (setq migemo-coding-system 'utf-8-unix)))

  (when (executable-find "cmigemo")
    (add-hook 'isearch-mode-hook 'migemo-init)))
```

6.17. [anything.el] 何でも絞り込みインターフェイス（旧）

- helm に移行しました。

http://svn.coderepos.org/share/lang/elisp/anything-c-moccur/trunk/anything-c-moccur.el
http://d.hatena.ne.jp/IMAKADO/20080724/1216882563

```
(when (eval-after-autoload-if-found
       '(anything-other-buffer anything-complete anything-M-x
                               anything-c-moccur-occur-by-moccur)
       "anything-startup" nil t nil
       '((require 'anything-c-moccur nil t)
```

```
        ;;  (setq moccur-split-word t)
        ;;  (setq anything-c-locate-options `("locate" "-w"))

        ;; M-x install-elisp-from-emacswiki recentf-ext.el
        ;; http://www.emacswiki.org/cgi-bin/wiki/download/recentf-ext.el
        ;;  (autoload-if-found 'recentf-ext "recentf-ext" nil t)
        ;;              (require 'recentf-ext nil t)

        (when (require 'migemo nil t)
          (setq moccur-use-migemo t))
        ;; M-x anything-grep-by-name
        (setq anything-grep-alist
              '(("Org-files" ("egrep -Hin %s *.org" "~/Dropbox/org/"))
                (".emacs.d" ("egrep -Hin %s *.el" "~/.emacs.d/"))
                ("ChangeLog" ("egrep -Hin %s ChangeLog" "~/"))))))
  ;; ("Spotlight" ("mdfind %s -onlyin ~/Dropbox/Documents/Library/" ""))))

  (defun my:anything ()
    (interactive)
    (anything-other-buffer
     '(anything-c-source-recentf
       anything-c-source-file-name-history
       anything-c-source-buffers
       anything-c-source-emacs-commands
       anything-c-source-locate)
     " *my:anything*"))

  (defun my:anything-buffer ()
    (interactive)
    (anything-other-buffer
     '(anything-c-source-buffers)
     " *my:anthing-buffer*"))

  (when (eq window-system 'ns)
    (defun my:anything-spotlight ()
      "Spotlight search with anything.el"
      (interactive)
      (anything-other-buffer
       '(anything-c-source-mac-spotlight)
       " *anything-spotlight*")))

  (setq anything-candidate-number-limit 50) ; 50
  (setq anything-input-idle-delay 0.1)      ; 0.1
  (setq anything-idle-delay 0.5)            ; 0.5
  (setq anything-quick-update nil))        ; nil

(when (autoload-if-found
       '(anything-other-buffer anything-complete
                               anything-M-x anything-c-moccur-occur-by-moccur)
       "anything-startup" nil t)

  (defun my:anything ()
    (interactive)
    (anything-other-buffer
     '(anything-c-source-recentf
       anything-c-source-file-name-history
       anything-c-source-buffers
       anything-c-source-emacs-commands
       anything-c-source-locate)
     " *my:anything*"))
```

```
  (defun my:anything-buffer ()
    (interactive)
    (anything-other-buffer
     '(anything-c-source-buffers)
     " *my:anthing-buffer*"))

  (when (eq window-system 'ns)
    (defun my:anything-spotlight ()
      "Spotlight search with anything.el"
      (interactive)
      (anything-other-buffer
       '(anything-c-source-mac-spotlight)
       " *anything-spotlight*")))

  (eval-after-load "anything-startup"
    '(progn

       (require 'anything-c-moccur nil t)
       ;;  (setq moccur-split-word t)
       ;;  (setq anything-c-locate-options `("locate" "-w"))

       ;; M-x install-elisp-from-emacswiki recentf-ext.el
       ;; http://www.emacswiki.org/cgi-bin/wiki/download/recentf-ext.el
       ;;  (autoload-if-found 'recentf-ext "recentf-ext" nil t)
       ;;             (require 'recentf-ext nil t)

       (when (require 'migemo nil t)
         (setq moccur-use-migemo t))

       ;; M-x anything-grep-by-name
       (setq anything-grep-alist
             '(("Org-files" ("egrep -Hin %s *.org" "~/Dropbox/org/"))
               (".emacs.d" ("egrep -Hin %s *.el" "~/.emacs.d/"))
               ("ChangeLog" ("egrep -Hin %s ChangeLog" "~/"))))
       ;; ("Spotlight" ("mdfind %s -onlyin ~/Dropbox/Documents/Library/" "")))))

       (setq anything-candidate-number-limit 50) ; 50
       (setq anything-input-idle-delay 0.1)      ; 0.1
       (setq anything-idle-delay 0.5)            ; 0.5
       (setq anything-quick-update nil))))       ; nil
```

6.17.1. キーバインド

普通に anything-startup を呼んでいる場合には、anything-M-x を設定する必要はない。

```
;; Show ibuffer powered by anything
;;  (with-eval-after-load "anything-startup"
(global-set-key (kbd "M-x") 'anything-M-x)
(global-set-key (kbd "C-c o") 'anything-c-moccur-occur-by-moccur)
(global-set-key (kbd "C-M-r") 'my:anything)
(global-set-key (kbd "C-M-s") 'my:anything-spotlight)
(global-set-key (kbd "C-x C-b") 'my:anything-buffer)
;;)
```

6.18. [helm.el] 続・何でも絞り込みインターフェイス

```
(eval-after-autoload-if-found
 '(helm-M-x helm-locate helm-recentf helm-buffers-list
            helm-occur helm-swoop helm-flycheck)
```

```
"helm-config" nil t nil
'((helm-mode 1)
  (when (require 'diminish nil t)
    (diminish 'helm-mode " H"))
  (when (require 'helm-swoop nil t)
    (set-face-attribute
     'helm-swoop-target-line-face nil :background "#FFEDDC")
    (set-face-attribute
     'helm-swoop-target-word-face nil :background "#FF5443"))
  (when (require 'helm-google nil t)
    (setq helm-google-tld "co.jp"))
  ;;        (require 'recentf-ext nil t)
  ;; helm-find-files を呼ばせない
  (add-to-list 'helm-completing-read-handlers-alist '(find-file . nil))
  ;; helm-mode-ag を呼ばせない
  (add-to-list 'helm-completing-read-handlers-alist '(ag . nil))
  ;; helm-mode-org-set-tags を呼ばせない
  (add-to-list 'helm-completing-read-handlers-alist '(org-set-tags . nil))
  (setq helm-display-source-at-screen-top nil)
          ;;;         (setq helm-display-header-line nil)
  ;; helm-autoresize-mode を有効にしつつ 30% に固定
  (helm-autoresize-mode 1)
  (setq helm-autoresize-max-height 30)
  (setq helm-autoresize-min-height 30)
  (set-face-attribute 'helm-source-header nil
                      :height 1.0 :family "Verdana" :weight 'normal
                      :foreground "#666666" :background "#DADADA")
  (when (eq window-system 'ns)
    (setq helm-locate-command "mdfind -name %s %s")
    )

  ;; この修正が必要
  (when (require 'helm-migemo nil t)
    (defun helm-compile-source--candidates-in-buffer (source)
      (helm-aif (assoc 'candidates-in-buffer source)
          (append source
                  `((candidates
                     . ,(or (cdr it)
                            (lambda ()
                              ;; Do not use `source' because other plugins
                              ;; (such as helm-migemo) may change it
                              (helm-candidates-in-buffer
                               (helm-get-current-source)))))
                    (volatile) (match identity)))
        source)))
  ))
```

6.18.1. キーバインド

```
(when (eval-after-autoload-if-found
       '(helm-M-x helm-locate helm-recentf helm-buffers-list
                  helm-occur helm-swoop helm-flycheck)
       "helm-config")

  (global-set-key (kbd "C-c f f") 'helm-locate)
  (global-set-key (kbd "C-x C-b") 'helm-buffers-list)
  (global-set-key (kbd "M-s") 'helm-swoop)
  (global-set-key (kbd "C-M-r") 'helm-recentf)
  (global-set-key (kbd "C-c o") 'helm-occur)
  (global-set-key (kbd "M-x") 'helm-M-x))
```

6.19. [stripe-buffer.el] テーブルの色をストライプにする

stripe-buffer.el を使います。重くツリーが多い Org バッファだと激重になる可能性があります。

```
(when (eval-after-autoload-if-found
       'org-mode "org" "Org Mode" t nil
       '((require 'stripe-buffer nil t)))
  (add-hook 'org-mode-hook 'turn-on-stripe-table-mode))
```

6.20. [rainbow-delimiters] 対応するカッコに色を付ける

複数のカッコが重なる言語では、カッコの対応関係がひと目で理解し難い場合があります。 rainbow-delimiters を使うと、対応するカッコを七色に色付けして見やすくできます。デフォルトだと色がパステル調で薄いので、パラメータを追加して調整します。

org-block 内でうまく動かないようなので、本格導入は様子見中です。

```
(with-eval-after-load "rainbow-delimiters"
  ;; https://yoo2080.wordpress.com/2013/12/21/small-rainbow-delimiters-
tutorial/
  (require 'cl-lib)
  (require 'color)
  (cl-loop
   for index from 1 to rainbow-delimiters-max-face-count
   do
   (let ((face (intern (format "rainbow-delimiters-depth-%d-face" index))))
     (cl-callf color-saturate-name (face-foreground face) 50))))

(add-hook 'prog-mode-hook
          '(lambda ()
             (unless (equal (buffer-name) "*scratch*")
               (rainbow-delimiters-mode))))
```

6.21. [git-gutter-fringe] 編集差分をフレーム端で視覚化

git-gutter-fringe を使えば、フレーム幅の変化を気にしなくて済みます。とりあえず、org-mode を初回起動するときに紐づけています。

```
(when (eval-after-autoload-if-found
       '(git-gutter-mode) "git-gutter-fringe" "Org Mode" t nil
       '((setq git-gutter:lighter "")
         ;; "!"
         (fringe-helper-define 'git-gutter-fr:modified nil
           "...XX..."
           "...XX..."
           "...XX..."
           "...XX..."
           "...XX..."
           "........"
           "...XX..."
           "...XX...")
         ;; "+"
         (fringe-helper-define 'git-gutter-fr:added nil
           "........"
           "...XX..."
           "...XX..."
           ".XXXXXX."
           ".XXXXXX."
```

```
            "...XX..."
            "...XX..."
            "........")
          ;; "-"
          (fringe-helper-define 'git-gutter-fr:deleted nil
            "........"
            "........"
            "........"
            ".XXXXXX."
            ".XXXXXX."
            "........"
            "........"
            "........")
          (setq git-gutter-fr:side 'left-fringe)
          (set-face-foreground 'git-gutter-fr:added    "#FF2600")
          (set-face-foreground 'git-gutter-fr:modified "orange")
          (set-face-foreground 'git-gutter-fr:deleted  "medium sea green")))

  (add-hook 'emacs-lisp-mode-hook 'git-gutter-mode)
  (add-hook 'lisp-mode-hook 'git-gutter-mode)
  (add-hook 'perl-mode-hook 'git-gutter-mode)
  (add-hook 'python-mode-hook 'git-gutter-mode)
  (add-hook 'c-mode-common-hook 'git-gutter-mode)
  (add-hook 'nxml-mode-hook 'git-gutter-mode)
  (add-hook 'web-mode-hook 'git-gutter-mode))
```

6.22. [zlc.el] find-file バッファを zsh ライクにする

ファイル選択を zsh ライクに変更できます。

```
(when (require 'zlc nil t)
  ;; http://d.hatena.ne.jp/mooz/20101003/p1
  (zlc-mode 1)
  (set-face-attribute 'zlc-selected-completion-face nil
                      :foreground "#000000" :background "#9DFFD2" :bold t)
  (setq zlc-select-completion-immediately t)
  (let ((map minibuffer-local-map))
    ;; like menu select
    (define-key map (kbd "<down>") 'zlc-select-next-vertical)
    (define-key map (kbd "<up>") 'zlc-select-previous-vertical)
    (define-key map (kbd "<left>") 'zlc-select-next)
    (define-key map (kbd "<right>") 'zlc-select-previous)
    ;; reset selection
    (define-key map (kbd "C-c") 'zlc-reset)))
```

6.23. [japanese-holidays] カレンダーをカラフルにする

ビルドインの holidays と、 japanese-holidays を使います。土日祝日に色を着けます。土曜日と日曜祝日で異なる配色にできます。

```
(with-eval-after-load "calendar"
  (add-hook 'calendar-today-visible-hook 'calendar-mark-today)

  (when (require 'japanese-holidays nil t)
    (setq calendar-holidays
          (append japanese-holidays
                  holiday-local-holidays holiday-other-holidays))
    (setq mark-holidays-in-calendar t)
    (setq japanese-holiday-weekend-marker
```

```
          '(holiday nil nil nil nil nil japanese-holiday-saturday))
    (setq japanese-holiday-weekend '(0 6))
    (add-hook 'calendar-today-visible-hook 'japanese-holiday-mark-weekend)
    (add-hook 'calendar-today-invisible-hook 'japanese-holiday-mark-weekend)))
```

6.24. [guide-key] キーバインドの選択肢をポップアップする

自分用の関数にキーバインドを付けたのはいいけど、覚えられない時に使っています。以下の例では、 `helm` もしくは `org` が読み込まれた時についでに有効化し、 `C-c f` を押して、 0.5 秒経つと、その後ろに続くキーの一覧がポップします。すでに覚えたキーバインドならば、0.5 秒以内に打てるでしょうから、ポップ表示無しで通常通りにコマンドが発行します。色分けも効くのでわかりやすいです。

```
(when (eval-after-autoload-if-found
       '(guide-key-mode) "guide-key" nil t nil
       '((setq guide-key/guide-key-sequence '("C-c f"))
         (setq guide-key/popup-window-position 'bottom)
         (setq guide-key/idle-delay 0.5)
         (setq guide-key/highlight-command-regexp
               '(("my:" . "red")
                 ("takaxp:" . "blue")))
         (guide-key-mode 1)))

  (with-eval-after-load "helm" (require 'guide-key nil t))
  (with-eval-after-load "org" (require 'guide-key nil t)))
```

7. メディアサポート

7.1. [bongo.el] Emacs のバッファで音楽ライブラリを管理する

[iTunes の代わりに Emacs を使う]

autoload を設定すると、 `*.bango-playlist` や `*.bongo-library` から起動できないので、明示的に require している。なお、bongo-mplayer を使う場合、bongo を先に require するとうまく動作しない（bongo.el の最後で、bongo-mplayer が provide されているからだと思われる）。

以下の設定では、autoload で使いつつ、=M-x init-bongo= でプレイリストを読み込んでいる。これならば、Emacs 起動時は軽量で、かつ、プレイリストの訪問で Bongo を開始できる。

```
;;  (require 'bongo)
(when (autoload-if-found 'bongo "bongo-mplayer" nil t)
  (defun init-bongo ()
    (interactive)
    (bongo)
    (find-file "~/Desktop/next/Tidy/hoge.bongo-playlist"))
  (with-eval-after-load "bongo-mplayer"
    ;; Volume control
    ;;          (require volume.el nil t)
    (setq bongo-mplayer-extra-arguments '("-volume" "1"))
    ;; Avoid error when editing bongo buffers
    (setq yank-excluded-properties nil)
    ;; Use mplayer
    (setq bongo-enabled-backends '(mplayer))))

(eval-after-autoload-if-found
 '(bongo) "bongo-mplayer" nil t nil
```

```
'(;; Volume control
  ;;          (require volume.el nil t)
  (setq bongo-mplayer-extra-arguments '("-volume" "1"))
  ;; Avoid error when editing bongo buffers
  (setq yank-excluded-properties nil)
  ;; Use mplayer
  (setq bongo-enabled-backends '(mplayer))

  (defun init-bongo ()
    (interactive)
    (bongo)
    (find-file "~/Desktop/next/Tidy/hoge.bongo-playlist"))))
```

org-player.el を使えば、org-mode のバッファから Bongo を操作できる。

```
(eval-after-autoload-if-found 'org-mode "org-player")
```

音量コントロールには、volume.el が必要です。設定がうまくいかないので保留中

```
(autoload 'volume "volume" "Tweak your sound card volume." t)
```

7.2. [GoogleMaps.el] GoogleMaps を Emacs 内で使う

http://julien.danjou.info/software/google-maps.el

`M-x gogole-maps` で起動します。

```
(eval-after-autoload-if-found
 '(google-maps) "google-maps" nil t nil
 '((require 'org-location-google-maps nil t)))
```

`+/-` でズーム、 矢印 で移動、 `q` で終了します。また、 `w` で URL を取得してコピー、 `t` で地図の種別を変更できます。

Org-mode を使っている場合には、 `C-c M-L` で表示されるプロンプトで検索すると、プロパティにそのキーワードが記録されます。後から `C-c M-l` すれば、いつでも地図を表示できるようになります。

7.3. [org-google-weather.el] org-agenda に天気を表示する

残念ながら Google API が変更になり動かなくなったそうです。

http://julien.danjou.info/software/google-weather.el

```
(require 'google-weather nil t)
(when (require 'org-google-weather nil t)
  '(org-google-weather-use-google-icons t))
```

8. 履歴／ファイル管理

8.1. Undo バッファを無限に取る

```
(setq undo-outer-limit nil)
```

8.2. [undo-tree] 編集履歴をわかりやすくたどる

Undo のツリーが表示され、履歴をたどれます。 `C-x u` と `q` に対して、フレームサイズの変更を紐付けています。また、 `auto-save-buffers` が org-files をどんどん保存して

記録してしまうので、ツリーを選んでいる時に auto-save-buffers が発動するのを別途抑制しています。加えて、 org-tree-slide でナローイングしていると、タイムスタンプが記録される時に履歴が辿れなくなるので、 org-tree-slide が有効の時は、タイムスタンプを押させないように別途制限を加えています。

```
(when (eval-after-autoload-if-found
       '(my:undo-tree-visualize) "undo-tree" nil t nil
       '((global-undo-tree-mode)
         (setq undo-tree-mode-lighter nil) ;; モードライン領域を節約
         (defun my:undo-tree-visualizer-quit ()
           (interactive)
           (undo-tree-visualizer-quit)
           (delete-window)
           (set-frame-width (selected-frame) 80))
         (defun my:undo-tree-visualize ()
           (interactive)
           (set-frame-width (selected-frame) 163)
           (undo-tree-visualize))

         (define-key undo-tree-visualizer-mode-map (kbd "q")
           'my:undo-tree-visualizer-quit)
         ;; undo-tree-map にも必要
         (define-key undo-tree-map (kbd "C-x u")
           'my:undo-tree-visualize)))

  (global-set-key (kbd "C-x u") 'my:undo-tree-visualize))
```

8.3. バッファ保存時にバックアップファイルを生成する

バッファが保存されるとき、必ずバックアップを生成する。

```
;; Backup the buffer whenever the buffer is saved
(global-set-key (kbd "C-x C-s")
                '(lambda () (interactive) (save-buffer 16)))
```

8.4. バッファ保存時にバックアップを生成させない

```
;; *.~
(setq make-backup-files nil)
;; .#*
(setq auto-save-default nil)
;; auto-save-list
(setq auto-save-list-file-prefix nil)
```

8.5. ミニバッファの履歴を保存しリストアする

```
(when (eval-after-autoload-if-found
       '(savehist-mode) "savehist" nil t nil
       '((setq savehist-file "~/Dropbox/emacs.d/.history")))
  (savehist-mode 1))
```

8.6. 履歴サイズを大きくする

t で無限大に指定する。

```
(setq history-length 2000)
```

8.7. Emacs 終了時に開いていたバッファを起動時に復元する

Built-in の desktop.el を使う。

org バッファを CONTENT view で大量に開いていると、再起動が非常に遅くなるので利用を中止した。代替手段として、session.el と recentf の組み合わせがある。最近利用したファイルとそのカーソル位置が保持されるため、最後に訪問していたファイルを比較的簡単に復元できる。頻繁に復元するバッファには、別途キーバインドを割り当てておけば問題ない。

```
(eval-after-autoload-if-found
 '(desktop-save desktop-clear desktop-load-default desktop-remove)
 "desktop" nil t nil
 '((desktop-save-mode 1)
   (setq desktop-files-not-to-save "\\(^/tmp\\|^/var\\|^/ssh:\\)")))
```

8.8. 最近開いたファイルリストを保持

Built-in の recentf.el を使う。

http://d.hatena.ne.jp/tomoya/20110217/1297928222

session.el でも履歴管理できるが、anything のソースとして使っているので併用している。

起動直後から有効にするので、autolad-if-load で括る必要はない。

recentf-auto-cleanup を 'mode などにすると起動時にファイルのクリーニングが行われるてしまうので、='never= で回避し、アイドルタイマーなどで対応する。これだけで 50[ms]ほど起動を高速化できる。

```
(when (eval-after-autoload-if-found
       '(rencetf-mode recentf-save-list-without-msg) "recentf" nil t nil
       '((defun recentf-save-list-without-msg ()
           (interactive)
           (let ((message-log-max nil))
             (if (require 'shut-up nil t)
                 (shut-up (recentf-save-list))
               (recentf-save-list)))
           (message ""))
         (defun recentf-cleanup-without-msg ()
           (interactive)
           (let ((message-log-max nil))
             (if (require 'shut-up nil t)
                 (shut-up (recentf-cleanup))
               (recentf-cleanup)))
           (message ""))
         (add-hook 'focus-out-hook 'recentf-save-list-without-msg)
         (add-hook 'focus-out-hook 'recentf-cleanup-without-msg)

         (setq recentf-max-saved-items 2000)
         (setq recentf-save-file
               (expand-file-name "~/.emacs.d/recentf"))
         (setq recentf-auto-cleanup 'never)
         (setq recentf-exclude
               '(".recentf" "^/tmp\\.*"
                 "^/private\\.*" "^/var/folders\\.*" "/TAGS$"))))

  (add-hook 'after-init-hook 'recentf-mode))
```

8.9. 深夜にバッファを自動整理する

http://www.emacswiki.org/emacs-zh/CleanBufferList

```
(when (require 'midnight nil t)
  (setq clean-buffer-list-buffer-names
        (append clean-buffer-list-kill-buffer-names
                '("note.txt")))
  (setq clean-buffer-list-delay-general 1)
  (setq clean-buffer-list-delay-special 10))
```

8.10. [auto-save-buffers.el] 一定間隔でバッファを保存する

http://0xcc.net/misc/auto-save/

起動直後から有効にするので、autolad-if-load で括る必要はない。同じ機能で比較的新しいパッケージに、 real-auto-save があります。ただ、私の場合は、以下のようなモードごとの制御がうまくできなかったので移行していません。

```
(when (and (require 'auto-save-buffers nil t)
           (require 'shut-up nil t))
  (run-with-idle-timer
   1.6 t
   #'(lambda ()
       (cond ((equal major-mode 'undo-tree-visualizer-mode) nil)
             ((equal major-mode 'diff-mode) nil)
             ((string-match "Org Src" (buffer-name)) nil)
             (t (shut-up (auto-save-buffers)))))))
```

8.11. [backup-dir.el] バックアップファイルを一箇所に集める

backup-each-save を使うようになりました。

http://www.emacswiki.org/emacs/BackupDirectory
http://www.northbound-train.com/emacs-hosted/backup-dir.el
http://www.northbound-train.com/emacs.html

起動直後から有効にするので、autolad-if-load で括る必要はない。

```
(make-variable-buffer-local 'backup-inhibited)
(setq backup-files-store-dir "~/.emacs.d/backup")
(unless (file-directory-p backup-files-store-dir)
  (message "!!! %s does not exist. !!!" backup-files-store-dir)
  (sleep-for 1))
(when (require 'backup-dir nil t)
  (when (file-directory-p backup-files-store-dir)
    ;; backup path
    (setq bkup-backup-directory-info '((t "~/.emacs.d/backup" ok-create)))
    ;; for tramp
    (setq tramp-bkup-backup-directory-info bkup-backup-directory-info)
    ;; generation properties
    (setq delete-old-versions t
          kept-old-versions 0
          kept-new-versions 5
          version-control t)))
```

8.12. [backup-each-save] クラッシュに備える

直近のファイルを常にバックアップします。 backup-dir.el でも良いですが、バック

アップの目的が、バッファ編集中に emacs が落ちる時の保険ならば、 backup-each-save の方が適切な場合があります。以下の例では、すべてのファイルを保存の度に保存しつつ、 emacs 終了時に 7 日前までのバックアップファイルをすべて削除するようにしています。

```
(when (require 'backup-each-save nil t)
  (setq backup-each-save-mirror-location "~/.emacs.d/backup")
  (setq backup-each-save-time-format "%y-%m-%d_%M:%S")
  (setq backup-each-save-size-limit 1048576)
  (add-hook 'after-save-hook
            '(lambda () (unless (equal (buffer-name) "recentf")
                          (backup-each-save)))))

(when (eval-after-autoload-if-found
       '(recursive-delete-backup-files delete-backup-files) "utility")

  ;; backup-each-save が作るファイルのうち条件にあうものを終了時に削除
  (add-hook 'kill-emacs-hook
            #'(lambda () (recursive-delete-backup-files 7))))
```

8.13. [session.el] 様々な履歴を保存し復元に利用する

http://emacs-session.sourceforge.net/

- 入力履歴の保持（検索語、表示したバッファ履歴）
- 保存時のカーソル位置の保持
- キルリングの保持
- 変更が加えられたファイル履歴の保持

M-x session-save-session

session-undo-check を指定していると、保存時ではなくバッファを閉じるときの状態を保持する。

Org-mode と併用する場合は、my:org-reveal-session-jump の設定が必須。

```
(when (eval-after-autoload-if-found
       'session-initialize "session" nil t nil
       '((add-to-list 'session-globals-exclude 'org-mark-ring)
         ;; Change save point of session.el
         (setq session-save-file
               (expand-file-name "~/Dropbox/emacs.d/.session"))
         (setq session-initialize '(de-saveplace session keys menus places)
               session-globals-include '((kill-ring 100)
                                         (session-file-alist 100 t)
                                         (file-name-history 200)
                                        search-ring regexp-search-ring))
         (setq session-undo-check -1)))

  (add-hook 'after-init-hook 'session-initialize))
;; FIXME
;;  (setq session-set-file-name-exclude-regexp
;;        "^/private/\\.\\*"))
;;          "[/\\]\\.overview\\|[/\\]\\.session\\|News[/\\]\\|^/private\\.*\\|
^/var/folders\\.*"))
```

次はテスト中。org バッファを開いたらカーソル位置を org-reveal したいが、time-stamp などと組み合わせたり、org-tree-slide と組み合わせていると、うまくいかない。バッファを

表示した時に org-reveal (C-c C-r) を打つのをサボりたいだけなのだが。。。

http://www.emacswiki.org/emacs/EmacsSession

```
(when (autoload-if-found 'session-initialize "session" nil t)
  (add-hook 'after-init-hook 'session-initialize)
  (eval-after-load "session"
    '(progn
       ;; For Org-mode
       (defun my:maybe-reveal ()
         (interactive)
         (when (and (or (memq major-mode '(org-mode outline-mode))
                        (and (boundp 'outline-minor-mominor-de)
                             outline-minor-mode))
                    (outline-invisible-p))
           (if (eq major-mode 'org-mode)
               (org-reveal)
             (show-subtree))))

       (defun my:org-reveal-session-jump ()
         (message "call!")
         (when (and (eq major-mode 'org-mode)
                    (outline-invisible-p))
           (org-reveal)))

       ;; C-x C-/
       (add-hook 'session-after-jump-to-last-change-hook
                 'my:maybe-reveal))))
```

8.14. [wakatime-mode.el] WakaTime を利用して作業記録する

1. https://www.wakati.me/（API 発行とログ GUI 表示）
2. https://github.com/wakatime/wakatime（ログ記録用スクリプト）
3. https://github.com/nyuhuhuu/wakatime-mode（Emacs 用プラグイン）

利用開始前に、ログ表示サイトでルールをカスタマイズしておくとよい。例えば、拡張子が `.org` なファイルの場合、言語設定を `Text` にする、という具合に。すると、グラフ表示がわかりやすくなる。

```
  (when (require 'wakatime-mode nil t)
    (setq wakatime-api-key "<insert your own api key>")
    (setq wakatime-cli-path "/Users/taka/Dropbox/emacs.d/bin/wakatime-cli.py")
    ;; すべてのバッファで訪問時に記録を開始
;     (global-wakatime-mode)
    )
```

9. 開発サポート

9.1. 便利キーバインド

```
(global-set-key (kbd "C-;") 'comment-dwim) ;; M-; is the defualt
(global-set-key (kbd "C-c c") 'compile)
```

9.2. [gist.el] Gist インターフェイス

```
(eval-after-autoload-if-found '(gist) "gist")
```

9.3. [doxymacs.el] Doxygen のコメントを簡単に入力する

http://doxymacs.sourceforge.net/

```
(when (eval-after-autoload-if-found
       'doxymacs-mode "doxymacs" nil t nil
       '((setq doxymacs-doxygen-style "JavaDoc")
         (add-hook 'font-lock-mode-hook
                   '(lambda () (interactive)
                      (when (or (eq major-mode 'c-mode)
                                (eq major-mode 'c++-mode))
                        (doxymacs-font-lock))))
         (define-key doxymacs-mode-map (kbd "C-c C-s") 'ff-find-other-file)))

  (add-hook 'c-mode-common-hook 'doxymacs-mode))
```

9.4. [matlab.el] Matlab 用の設定

```
(when (and (eq window-system 'ns) (> emacs-major-version 23))
  (when (eval-after-autoload-if-found
         '(matlab-mode matlab-shell) "matlab")
    (push '("\\.m$" . matlab-mode) auto-mode-alist)))
```

9.5. [flycheck.el] 構文エラー表示

auto-complete より前に hook 設定しておくと余計なエラーが出ないようです。

```
(when (eval-after-autoload-if-found
       '(flycheck-mode) "flycheck" nil t nil
       '(;;http://qiita.com/senda-akiha/items/cddb02cfdbc0c8c7bc2b
         (require 'helm-flycheck nil t)
         (when (require 'flycheck-pos-tip nil t)
           '(custom-set-variables
             '(flycheck-display-errors-function
               #'flycheck-pos-tip-error-messages)))))

  (add-hook 'js2-mode-hook 'flycheck-mode)
  (add-hook 'c-mode-common-hook 'flycheck-mode)
  (add-hook 'perl-mode-hook 'flycheck-mode)
  (add-hook 'python-mode-hook 'flycheck-mode))
```

9.6. [auto-complete.el] 自動補完機能

http://cx4a.org/software/auto-complete/manual.ja.html

- 辞書データを使う（ `ac-dictionary-directories` ）
- auto-complete.el, auto-complete-config.el, fuzzy.el, popup.el を使う。
- 日本語マニュアル
- ac-auto-start を 4 にしておけば、3 文字までは TAB を yasnippet に渡せる。

Org-mode ユーザにとって `TAB` は非常に重要なコマンド。そこに `auto-complete` と `yasnippet` が `TAB` を奪いに来るので、住み分けが重要になる。=ac-auto-start= を=4=にすると、<s=TAB= によるソースブロックの短縮入力を `yasnippet` で実行できる（この目的だけならば=3=を指定してもいい）。<sys などと 4 文字入力すると、=auto-complete= が動いて <system> などを補完してくれる。もちろん、見出しで `TAB` を押すときには、ツリーの表示／非表示の切り替えになる。

情報源については、オンラインマニュアルを参照のこと。

```
(when (eval-after-autoload-if-found
       '(ac-default-setup ac-org-mode-setup)
       "auto-complete" nil t nil
       '((require 'auto-complete-config nil t)
         (ac-config-default)
         ;; 追加のメジャーモードを設定
         (add-to-list 'ac-modes 'objc-mode)
         (add-to-list 'ac-modes 'org-mode)
         ;; ac-modes にあるメジャーモードで有効にする
         ;;lisp, c, c++, java, perl, cperl, python, makefile, sh, fortran, f90
         (global-auto-complete-mode t)
         ;; 辞書
         (add-to-list 'ac-dictionary-directories "~/.emacs.d/ac-dict")
         ;; history
         (setq ac-comphist-file "~/Dropbox/config/ac-comphist.dat")
         ;; n 文字以上で補完表示する（"<s TAB" の場合 yasnippet が呼ばれる）
         (setq ac-auto-start 4)
         ;; n 秒後にメニューを表示
         (setq ac-auto-show-menu 1.0)
         ;; ツールチップの表示
         (setq ac-use-quick-help t)
         (setq ac-quick-help-delay 2.0)
         (setq ac-quick-help-height 10)
         ;; C-n/C-p でメニューをたどる
         (setq ac-use-menu-map t)
         ;; TAB で補完（org-mode でも効くようにする）
         (define-key ac-completing-map [tab] 'ac-complete)
         ;; RET での補完を禁止
         (define-key ac-completing-map "\r" nil)
         ;; 補完メニューの表示精度を高める
         (setq popup-use-optimized-column-computation nil)
         ;;(setq ac-candidate-max 10)

         (defun ac-org-mode-setup ()
           ;;             (message " >> ac-org-mode-setup")
           (setq ac-sources '(
                              ;;; ac-source-abbrev ; Emacs の略語
                              ;;; ac-source-css-property ; heavy
                              ac-source-dictionary ; 辞書
                              ac-source-features
                              ac-source-filename
                              ac-source-files-in-current-dir
                              ac-source-functions
                              ;;; ac-source-gtags
                              ;;; ac-source-imenu
                              ;;; ac-source-semantic
                              ;;; ac-source-symbols
                              ;;; ac-source-variables
                              ;;; ac-source-yasnippet
                              )))

         (defun ac-default-setup ()
           ;;             (message " >> ac-default-setup")
           (setq ac-sources '(ac-source-abbrev
                              ac-source-dictionary
                              ac-source-words-in-same-mode-buffers)))
         ))

  (dolist (hook (list 'python-mode-hook 'perl-mode-hook 'objc-mode-hook))
```

```
    (add-hook hook 'ac-default-setup))
  ;; *scratch* バッファでは無効化
  (add-hook 'lisp-mode-hook
            '(lambda () (unless (equal "*scratch*" (buffer-name))
                          (ac-default-setup))))
  (add-hook 'org-mode-hook 'ac-org-mode-setup))
```

9.7. [auto-complete-clang.el] オムニ補完

C++バッファでメソッドを補完対象とする。try-catch を使っている場合、 `-fcxx-exceptions` オプションが必要で、これはプリコンパイルヘッダを生成する時も同じだ。ここ示す設定では、 `~/.emacs.d/` 以下に `stdafx.pch` を生成する必要があり、以下のコマンドを用いてプリコンパイルヘッダを生成する。ヘッダファイルのパスを適切に与えれば、Boost や自作のライブラリも補完対象に設定できる。

現状では、補完直後にデフォルトの引数がすべて書き込まれてしまう。なんかうまいことしたいものだ。

```
clang -cc1 -x c++-header -fcxx-exceptions ./stdafx.h -emit-pch -o ./stdafx.pch
-I/usr/local/include -I/usr/local/include/netpbm
```

以下の設定は、先に `auto-complete.el` に関する設定を読み込んでいることを前提としている。

```
(when (eval-after-autoload-if-found
       '(auto-complete ac-cc-mode-setup) "auto-complete" nil t nil
       '((require 'auto-complete-clang nil t)
         ;; ac-cc-mode-setup のオーバーライド
         (defun ac-cc-mode-setup ()
           (setq ac-clang-prefix-header "~/.emacs.d/stdafx.pch")
           (setq ac-clang-flags '("-w" "-ferror-limit" "1"
                                  "-fcxx-exceptions"))
           (setq ac-sources '(ac-source-clang
                              ac-source-yasnippet
                              ac-source-gtags)))))

  (add-hook 'c-mode-common-hook 'ac-cc-mode-setup))
```

次のコードを hoge.cpp として保存し、 `v` と `t` について補完できれば、 `STL` と `Boost` のプリコンパイルヘッダが有効になっていることを確認できる。

```
#include <iostream>
#include <vector>
#include <boost/timer.hpp>

int main(){
  std::vector<int> v;
  v; // ここ
  boost::timer t;
  cout << t; // ここ
  return 1;
}
```

9.7.1. 参考サイト

- http://d.hatena.ne.jp/kenbell1988/20120428/1335609313
- http://d.hatena.ne.jp/whitypig/20110306/1299416655

- http://d.hatena.ne.jp/yano-htn/?of=30
- http://www.nomtats.com/2010/11/auto-completeelemacs.html
- http://www.plugmasters.com.br/plugfeed/post/73768/awesome-cc-autocompletion-in-emacs

9.8. [hideshowvis.el] 関数の表示／非表示

- http://www.emacswiki.org/emacs/hideshowvis.el
- org.el（約 2 万行）を開くために約 2 分必要なため、最近は使っていません。

```
(when (and (eq window-system 'ns) (> emacs-major-version 23))
  (when (eval-after-autoload-if-found
         '(hideshowvis-enable hideshowvis-minor-mode) "hideshowvis" nil t nil
         '((define-key hideshowvis-mode-map (kbd "C-(") 'hs-hide-block)
           (define-key hideshowvis-mode-map (kbd "C-)") 'hs-show-block)))

    (add-hook 'emacs-lisp-mode-hook
              '(lambda () (unless (equal "*scratch*" (buffer-name))
                            (hideshowvis-enable))))
    (dolist (hook (list 'perl-mode-hook 'c-mode-common-hook))
      (add-hook hook 'hideshowvis-enable))))
```

9.9. [quickrun.el] お手軽ビルド

カレントバッファで編集中のソースコードをビルド・実行して、別バッファに結果を得ます。

```
(when (eval-after-autoload-if-found
       '(quickrun) "quickrun")

  (add-hook 'c-mode-common-hook
            '(lambda () (define-key c++-mode-map (kbd "<f5>") 'quickrun)))
  (add-hook 'python-mode-hook
            '(lambda () (define-key python-mode-map (kbd "<f5>") 'quickrun)))
  (add-hook 'perl-mode-hook
            '(lambda () (define-key perl-mode-map (kbd "<f5>") 'quickrun))))
```

10. Org Mode

10.1. 基本設定

```
(when (eval-after-autoload-if-found
       'org-mode "org" "Org Mode" t nil
       '((require 'org-extension nil t)
         (require 'org-habit)
         (require 'org-mobile)

         ;;      (when (require 'pomodoro nil t)
         ;;        (pomodoro:start nil))

         ;; C-c & が yasnippet にオーバーライドされているのを張り替える
         (define-key org-mode-map (kbd "C-c 4") 'org-mark-ring-goto)

         ;; Set checksum program path for windows
         (when (eq window-system 'w32)
           (setq org-mobile-checksum-binary "~/Dropbox/do/cksum.exe"))
```

```
        ;; org ファイルの集中管理
        (setq org-directory "~/Dropbox/org/")

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

        ;; Toggle inline images display at startup
        (setq org-startup-with-inline-images t)

        ;; dvipng
        (setq org-export-with-LaTeX-fragments t)

        ;; orgバッファ内の全ての動的ブロックを保存直前に変更する
        ;; (add-hook 'before-save-hook 'org-update-all-dblocks)

        ;; アーカイブファイルの名称を指定
        (setq org-archive-location "%s_archive::")

        ;; タイムスタンプによるログ収集設定
        (setq org-log-done t) ; t ではなく、'(done), '(state) を指定できる

        ;; ログをドロアーに入れる
        (setq org-log-into-drawer t)

        ;; アンダースコアをエクスポートしない（_{}で明示的に表現できる）
        (setq org-export-with-sub-superscripts nil)

        ;; #+OPTIONS: \n:t と同じ
        (setq org-export-preserve-breaks t)

        ;; タイマーの音
        ;; (lsetq org-clock-sound "");

        ;; helm を立ち上げる
        (require 'helm-config nil t)

        ;; - を優先。親のブリッツ表示を継承させない
        (setq org-list-demote-modify-bullet
              '(("+" . "-")
                ("*" . "-")
                ("1." . "-")
                ("1)" . "-")
                ("A)" . "-")
                ("B)" . "-")
                ("a)" . "-")
                ("b)" . "-")
                ("A." . "-")
                ("B." . "-")
                ("a." . "-")
                ("b." . "-")))
        ))

  (push '("\\.txt$" . org-mode) auto-mode-alist))
```

10.2. contribution を使う

```
(setq load-path (append '("~/devel/git/org-mode/contrib/lisp") load-path))
```

10.3. iCal との連携

```
(with-eval-after-load "org"
  ;; ~/Dropbox/Public は第三者に探索される可能性があるので要注意
  ;; default = ~/org.ics
  ;; C-c C-e i org-export-icalendar-this-file
  ;; C-c C-e I org-export-icalendar-all-agenda-files
  ;; C-c C-e c org-export-icalendar-all-combine-agenda-files
  ;; (setq org-combined-agenda-icalendar-file "~/Dropbox/Public/orgAgenda.ics")

  (setq org-icalendar-combined-agenda-file "~/Desktop/org-ical.ics")

  ;; iCal の説明文
  (setq org-icalendar-combined-description "OrgMode のスケジュール出力")
  ;; カレンダーに適切なタイムゾーンを設定する（google 用には nil が必要）
  (setq org-icalendar-timezone "Asia/Tokyo")

  ;; DONE になった TODO はアジェンダから除外する
  (setq org-icalendar-include-todo t)
  ;; （通常は、<>--<> で区間付き予定をつくる。非改行入力で日付が Note に入らない）
  (setq org-icalendar-use-scheduled '(event-if-todo))
        ;;; DL 付きで終日予定にする：締め切り日（スタンプで時間を指定しないこと）
  ;; (setq org-icalendar-use-deadline '(event-if-todo event-if-not-todo))
  (setq org-icalendar-use-deadline '(event-if-todo))

  (when (require 'ox-icalendar nil t)
    (defun my:ox-icalendar ()
      (interactive)
      (let ((temp-agenda-files org-agenda-files))
        (setq org-agenda-files '("~/Dropbox/org/org-ical.org"))
        ;; org-icalendar-export-to-ics を使うとクリップボードが荒れる
        (org-icalendar-combine-agenda-files)
        (setq org-agenda-files temp-agenda-files)
        ;; Dropbox/Public のフォルダに公開する
        ;;            (shell-command
        ;;             (concat "cp " org-icalendar-combined-agenda-file " "
        ;;                     org-icalendar-dropbox-agenda-file))
        (if (eq 0 (shell-command
                   (concat "scp -o ConnectTimeout=5 "
                           org-icalendar-combined-agenda-file " "
                           org-ical-file-in-orz-server)))
            (message "Uploading... [DONE]")
          (message "Uploading... [MISS]"))
        (my:ox-icalendar-cleanup)
        ))

    (defun my:ox-icalendar-cleanup ()
      (interactive)
      (when (file-exists-p
             (expand-file-name org-icalendar-combined-agenda-file))
        (shell-command-to-string
         (concat "rm -rf " org-icalendar-combined-agenda-file))))
    ;;      (add-hook 'focus-out-hook 'my:ox-icalendar-cleanup)
    ))
```

10.4. スピードコマンド

```
(with-eval-after-load "org"
  (setq org-use-speed-commands t)
  (add-to-list 'org-speed-commands-user '("d" org-todo "DONE"))
```

```
  (add-to-list 'org-speed-commands-user '("P" my:proportional-font-toggle))
  (add-to-list 'org-speed-commands-user
               '("$" call-interactively 'org-archive-subtree)))
```

10.5. Pomodoro

http://orgmode.org/worg/org-gtd-etc.html

```
(with-eval-after-load "org"
  (add-to-list 'org-modules 'org-timer)
  (setq org-timer-default-timer "25")
  ;; (add-hook 'org-clock-in-hook
  ;;         '(lamda ()
  ;;                 (if (not org-timer-current-timer)
  ;;                     (org-timer-set-timer '(16)))))

  (setq growl-pomodoro-default-task-name "doing the task")
  (setq growl-pomodoro-task-name 'growl-pomodoro-default-task-name)

  (defun set-growl-pomodoro-task-name ()
    (interactive "P")
    (setq growl-pomodoro-task-name
          (read-from-minibuffer "Task Name: " growl-pomodoro-default-task-
name)))
  (add-hook 'org-timer-set-hook 'set-growl-pomodoro-task-name)

  (defun growl-pomodoro-timer ()
    (interactive)
    (shell-command-to-string
     (concat "growlnotify -s -a Emacs -t \"++ Pomodoro ++\" -m \""
             "The end of " growl-pomodoro-task-name "!\""))
    (shell-command-to-string
                                        ;    (concat "say The end of " growl-
pomodoro-task-name)
     (concat "say -v Kyoko " growl-pomodoro-task-name)
     ))
  (add-hook 'org-timer-done-hook 'growl-pomodoro-timer))
```

10.6. face 関連

```
(with-eval-after-load "org"
  ;; Font lock を使う
  (global-font-lock-mode 1)
  (add-hook 'org-mode-hook 'turn-on-font-lock)
  ;; ウィンドウの端で折り返す（想定と逆の振る舞い。どこかにバグがある）
  (setq org-startup-truncated nil)
  ;; サブツリー以下の * を略式表示する
  (setq org-hide-leading-stars t)
  ;; Color setting for TODO keywords
  ;; Color for priorities
  ;; (setq org-priority-faces
  ;;  '(("?A" :foreground "#E01B4C" :background "#FFFFFF" :weight bold)
  ;;    ("?B" :foreground "#1739BF" :background "#FFFFFF" :weight bold)
  ;;    ("?C" :foreground "#575757" :background "#FFFFFF" :weight bold)))
  ;; Color setting for Tags

  ;; #CC3333
  (setq org-todo-keyword-faces
        '(("FOCUS"   :foreground "#FF0000" :background "#FFCC66")
```

```
          ("BUG"      :foreground "#FF0000" :background "#FFCC66")
          ("OTW"      :foreground "#EE3300" :background "#FFEE99")
          ("CHECK"    :foreground "#FF9900" :background "#FFF0F0" :underline t)
          ("ICAL"     :foreground "#33CC66")
          ("WAIT"     :foreground "#CCCCCC" :background "#666666")
          ("EDIT"     :foreground "#FF33CC")
          ("READ"     :foreground "#9933CC")
          ("MAIL"     :foreground "#CC3300")
          ("PLAN"     :foreground "#FF6600")
          ("REV1"     :foreground "#3366FF")
          ("REV2"     :foreground "#3366FF" :background "#99CCFF")
          ("REV3"     :foreground "#FFFFFF" :background "#3366FF")
          ("SLEEP"     :foreground "#9999CC")))

  ;; (:foreground "#0000FF" :bold t)     ; default. do NOT put this bottom
  (setq org-tag-faces
        '(("Achievement" :foreground "#66CC66")
          ("Report"      :foreground "#66CC66")
          ("Background"  :foreground "#66CC99")
          ("Chore"       :foreground "#6699CC")
          ("Domestic"    :foreground "#6666CC")
          ("Doing"       :foreground "#FF0000")
          ("Ongoing"     :foreground "#CC6666") ; for non scheduled/reminder
          ("Repeat"      :foreground "#CC9999") ; for interval tasks
          ("Mag"         :foreground "#9966CC")
          ("buy"         :foreground "#9966CC")
          ("pay"         :foreground "#CC6699")
          ("secret"      :foreground "#FF0000")
          ("note"        :foreground "#6633CC")
          ("print"       :foreground "#6633CC")
          ("Study"       :foreground "#6666CC")
          ("Implements"  :foreground "#CC9999" :weight bold)
          ("Coding"      :foreground "#CC9999")
          ("Editing"     :foreground "#CC9999" :weight bold)
          ("work"        :foreground "#CC9999" :weight bold)
          ("Survey"      :foreground "#CC9999" :weight bold)
          ("Home"        :foreground "#CC9999" :weight bold)
          ("Open"        :foreground "#CC9999" :weight bold)
          ("Blog"        :foreground "#FF33CC" :background "#9966CC")
          ("Test"        :foreground "#FF0000" :weight bold)
          ("drill"       :foreground "#66BB66" :underline t)
          ("DEBUG"       :foreground "#FFFFFF" :background "#9966CC")
          ("EVENT"       :foreground "#FFFFFF" :background "#9966CC")
          ("Thinking"    :foreground "#FFFFFF" :background "#96A9FF")
          ("Schedule"    :foreground "#FFFFFF" :background "#FF7D7D")
          ("INPUT"       :foreground "#FFFFFF" :background "#CC6666")
          ("OUTPUT"      :foreground "#FFFFFF" :background "#66CC99")
          ("CYCLE"       :foreground "#FFFFFF" :background "#6699CC")
          ("weekend"     :foreground "#FFFFFF" :background "#CC6666")
          ("Log"         :foreground "#008500"))))
;;#5BDF8D
```

10.7. TODO キーワードのカスタマイズ

キーワードには日本語も使えます。

```
(with-eval-after-load "org"
  (setq org-todo-keywords
        '((sequence "TODO(t)" "FOCUS(f)" "ICAL(c)" "|" "DONE(d)")
          (sequence "BUG(b)" "READ(r)" "EDIT(e)" "MAIL(m)" "PLAN(p)" "|")
```

```
          (sequence "CHECK(C)" "OTW(o)" "WAIT(w)" "SLEEP(s)" "|")
          (sequence "REV1(1)" "REV2(2)" "REV3(3)" "|")))

  ;; Global counting of TODO items
  (setq org-hierarchical-todo-statistics nil)
  ;; Global counting of checked TODO items
  (setq org-hierarchical-checkbox-statistics nil)

  ;; block-update-time
  (defun org-dblock-write:block-update-time (params)
    (let ((fmt (or (plist-get params :format) "%Y-%m-%d")))
      (insert "" (format-time-string fmt (current-time)))))

  ;; すべてのチェックボックスの cookies を更新する
  (defun do-org-update-statistics-cookies ()
    (interactive)
    (org-update-statistics-cookies 'all)))
```

10.8. ImageMagick を使って様々な画像をインライン表示する

システムに OpenJPEG と ImageMagick がインストールされていれば、JPEG 2000 などの画像形式もバッファに表示できます。

```
(with-eval-after-load "org"
  (setq org-image-actual-width '(256))
  (add-to-list 'image-file-name-extensions "jp2")
  ;; (add-to-list 'image-file-name-extensions "j2c")
  (add-to-list 'image-file-name-extensions "bmp")
  (add-to-list 'image-file-name-extensions "psd"))
```

次の例では、同じ画像を 2 度インライン表示しようと指定ますが、前者は横幅が 128 ピクセルで表示され、後者は `org-image-actual-width` で指定した 256 ピクセルで表示されます。

```
#+ATTR_HTML: :width 128
[[~/Desktop/lena_std.jp2]]

[[~/Desktop/lena_std.jp2]]

(org-toggle-inline-images)
```

10.9. [org-agenda]

```
(with-eval-after-load "org-agenda"
  ;; sorting strategy
  (setq org-agenda-sorting-strategy
        '((agenda habit-down time-up timestamp-up priority-down category-keep)
          (todo priority-down category-keep)
          (tags priority-down category-keep)
          (search category-keep)))
  ;; Set the view span as day in an agenda view, the default is week
  (setq org-agenda-span 'day)
  ;; アジェンダに警告を表示する期間
  (setq org-deadline-warning-days 2)
  ;; アジェンダビューで FOLLOW を設定
  ;; (setq org-agenda-start-with-follow-mode t)
  ;; Customized Time Grid
  (setq org-agenda-time-grid
        '((daily today require-timed)
          "----------------"
```

```
        (800 1000 1200 1400 1600 1800 2000 2200 2400 2600)))
(setq org-agenda-current-time-string "< d('- ' ) now!")
(setq org-agenda-timegrid-use-ampm t)

;; アジェンダ作成対象（指定しないと agenda が生成されない）
;; ここを間違うと、MobileOrg, iCal export もうまくいかない
(setq org-agenda-files
      '("~/Dropbox/org/org-ical.org" "~/Dropbox/org/next.org"
        "~/Dropbox/org/today.org" "~/Dropbox/org/buffer.org"
        "~/Dropbox/org/stock.org"
        "~/Dropbox/org/work.org" "~/Dropbox/org/research.org"))

(add-hook 'org-finalize-agenda-hook
          '(lambda () (org-agenda-to-appt t '((headline "TODO")))))

;; 移動直後に agenda バッファを閉じる（ツリーの内容は SPACE で確認可）
(org-defkey org-agenda-mode-map [(tab)]
            '(lambda () (interactive)
               (org-agenda-goto)
               (with-current-buffer "*Org Agenda*"
                 (org-agenda-quit))))

;; 特定タグを持つツリーリストを一発移動（org-tags-view, org-tree-slide）
(defvar my:doing-tag "Doing")
(defun my:sparse-doing-tree ()
  (interactive)
  (org-tags-view nil my:doing-tag))
;; Doing タグをトグルする
(defun my:toggle-doing-tag ()
  (interactive)
  (when (eq major-mode 'org-mode)
    (save-excursion
      (save-restriction
        (unless (org-at-heading-p)
          (outline-previous-heading))
        (if (string-match
             (concat ":" my:doing-tag ":") (org-get-tags-string))
            (org-toggle-tag my:doing-tag 'off)
          (org-toggle-tag my:doing-tag 'on))
        (org-reveal)))))

(custom-set-faces
 ;; '(org-agenda-clocking ((t (:background "#300020"))))
 '(org-agenda-structure ((t (:underline t :foreground "#6873ff"))))
 '(org-agenda-date-today ((t (:weight bold :foreground "#4a6aff"))))
 '(org-agenda-date ((t (:weight bold :foreground "#6ac214"))))
 '(org-agenda-date-weekend ((t (:weight bold :foreground "#ff8d1e"))))
 '(org-time-grid ((t (:foreground "#0a4796"))))
 '(org-warning ((t (:foreground "#ff431a"))))
 '(org-upcoming-deadline ((t (:inherit font-lock-keyword-face))))
 )

(define-key org-mode-map (kbd "<f11>") 'my:toggle-doing-tag)
(define-key org-mode-map (kbd "C-<f11>") 'my:sparse-doing-tree))
```

10.10. [appt.el] アラーム設定

- Growl や Terminal Notifier と連携していると、Emacs がバックグラウンドにあってもアラームに気づける。

```
(with-eval-after-load "org"
  ;; アラーム表示を有効にする
  (appt-activate 1)
  ;; window を フレーム内に表示する
  (setq appt-display-format 'window)
  ;; window を継続表示する時間[s]
  (setq appt-display-duration 5)
  ;; ビープ音の有無
  (setq appt-audible nil)
  ;; 何分前から警告表示を開始するか[m]
  (setq appt-message-warning-time 30)
  ;; 警告表示開始から何分ごとにリマインドするか[m]
  (setq appt-display-interval 5)
  ;; モードラインにアラームを表示する
  (setq appt-display-mode-line t)
  ;; org-agenda の内容をアラームに登録する（重い）
  ;; (add-hook 'org-mode-hook
  ;;           '(lambda ()
  ;;              (org-agenda-to-appt t '((headline "TODO")))))
  ;; 定期的に更新する
  (defvar org-agenda-to-appt-activate t)
  (defun my:org-agenda-to-appt ()
    (interactive)
    (org-agenda-to-appt t '((headline "TODO"))))
  (add-hook 'focus-out-hook
            #'(lambda ()
                (when org-agenda-to-appt-activate
                  (my:org-agenda-to-appt)
                  (setq org-agenda-to-appt-activate nil))))
  (run-with-idle-timer
   10 t #'(lambda ()
            (setq org-agenda-to-appt-activate t)))

  (define-key org-mode-map (kbd "C-c f 3") 'my:org-agenda-to-appt))
```

10.11. [org-capture] 高速にメモを取る

```
(when
    (eval-after-autoload-if-found
     '(org-capture) "org-capture" nil t nil
     '(;; 2010-06-13 の形式では、タグとして認識されない
       (defun get-current-date-tags () (format-time-string "%Y%m%d"))
       (setq org-default-notes-file (concat org-directory "next.org"))
       (defvar org-capture-words-notes-file (concat org-directory "words.org"))
       (defvar org-capture-notes-file (concat org-directory "note.org"))
       (defvar org-capture-research-file (concat org-directory "research.org"))
       (defvar org-capture-buffer-file (concat org-directory "buffer.org"))
       (defvar org-capture-today-file (concat org-directory "today.org"))
       (defvar org-capture-ical-file (concat org-directory "org-ical.org"))

       ;; see org.pdf:p73
       (setq org-capture-templates
             `(("t" "TODO 項目を INBOX に貼り付ける" entry
                (file+headline nil "INBOX") "** TODO %?\n\t")
               ("c" "同期カレンダーにエントリー" entry
                (file+headline ,org-capture-ical-file "Schedule")
                "** TODO %?\n\t")
               ("d" "Doing タグ付きのタスクを Inbox に投げる" entry
                (file+headline nil "INBOX")
                "** TODO %? :Doing:\n  - \n")
```

```
                ("l" "本日のチェックリスト" entry
                 (file+headline ,org-capture-today-file "Today")
                 "** FOCUS 本日のチェックリスト %T\n（起床時間の記録）
[[http://www.hayaoki-seikatsu.com/users/takaxp/][早起き日記]] \n（朝食）\n  - [ ]
%?\n（昼食）\n（帰宅／夕食）\n----\n（研究速報）\n  - [ ] \n")
                ("i" "アイディアを書き込む" entry (file+headline nil "INBOX")
                 "** %?\n  - \n\t%U")
                ("b" "Bug タグ付きの TODO 項目を貼り付ける" entry
                 (file+headline nil "INBOX")
                 "** TODO %? :bug:\n %i\n %a %t")
                ("w" ,(concat "英単語を " org-capture-words-notes-file
                              " に書き込む") entry
                              (file+headline ,org-capture-words-notes-file
"WORDS")
                              "** %? :%(get-current-date-tags):\n「」\n  - ")
                ("g" ,(concat "英語ノートを " org-capture-words-notes-file
                              " に書き込む")
                 entry (file+headline ,org-capture-words-notes-file "GRAMMER")
                 "** %? :%(get-current-date-tags):\n\n%U")
                ("T" "時間付きエントリー" entry (file+headline nil "INBOX")
                 "** %? %T--\n")
                ("n" "ノートとして INBOX に貼り付ける" entry
                 (file+headline nil "INBOX")
                 "** %? :note:\n\t%U")
                ("D" "「ドラッカー 365 の金言」をノートする" entry
                 (file+headline ,org-capture-notes-file "The Daily Drucker")
                 "** 「%?」\nDrucker) \n  - \n  - \nACTION POINT:\n  -
\nQUESTION:\n  - \n")
                ("r" ,(concat "研究ノートを " org-capture-research-file
                              " に書き込む")
                 entry (file+headline ,org-capture-research-file "Survey")
                 "** %? :note:\n# \n  - \n\t%U")
                ("`" ,(concat "ノートをバッファ " org-capture-buffer-file
                              " に書き込む")
                 entry (file+headline ,org-capture-buffer-file "Buffers")
                 "** %(get-random-string 16) %U\n\n%?\n\n----")))))
```

10.12. [org-refile]

```
(with-eval-after-load "org-refile"
  (setq org-refile-targets
        (quote (("org-ical.org" :level . 1)
                ("research.org" :level . 1)
                ("work.org" :level . 1)
                ("next.org" :level . 1)
                ("sleep.org" :level . 1)))))
```

10.13. [org-babel] 基本設定

```
(with-eval-after-load "org"
  (setq org-confirm-babel-evaluate nil)
  (setq org-src-fontify-natively t)
  (setq org-src-tab-acts-natively t)
  ;; org-src-window-setup (current-window, other-window, other-frame)
  (setq org-src-window-setup 'current-window)
  (require 'ob-http nil t)
  ;; Add ":results output" after program name
  (org-babel-do-load-languages
```

```
   'org-babel-load-languages
   '((dot . t)
     (C . t)
     (perl . t)
     (sh . t)
     (latex . t)
     (R . t)
     (python . t)))
  ;; (require 'ob-C nil t)
  ;; (require 'ob-perl nil t)
  ;; (require 'ob-sh nil t)
  ;; (require 'ob-python nil t)

  ;; 実装済みの言語に好きな名前を紐付ける
  (add-to-list 'org-src-lang-modes '("cs" . csharp))
  (add-to-list 'org-src-lang-modes '("zsh" . sh)))
```

10.14. [org-babel] ソースブロックの入力キーをカスタマイズ

ソースブロックを入力するときは、 <+ TAB でテンプレートを高速に入力できます。しかし、利用する言語までは指定できないので、特定の内容について対応するコマンドを割り当てて起きます。以下の例を設定として追加すると、 <S+ TAB で emacs-lisp を、<C+ TAB でコメントブロックを指定できます。

```
(with-eval-after-load "org"
  (add-to-list 'org-structure-template-alist
               '("C" "#+BEGIN_COMMENT\n?\n#+END_COMMENT" ""))
  (add-to-list 'org-structure-template-alist
               '("S" "#+BEGIN_SRC emacs-lisp\n?\n#+END_SRC" "<src lang=\"emacs-
lisp\">\n\n</src>")))
```

10.15. [MobileOrg] iOS との連携

http://orgmode.org/manual/Setting-up-the-staging-area.html

```
(with-eval-after-load "org"
  ;;(setq org-mobile-files '("~/Dropbox/org/next.org" "1.org" "2.org"))
  (setq org-mobile-files '("~/Dropbox/org/next.org"))
  ;;(setq org-mobile-force-id-on-agenda-items nil)

  ;; Set a file to capture data from iOS devices
  (setq org-mobile-inbox-for-pull (concat org-directory "captured.org"))

  ;; Upload location stored org files (index.org will be created)
  (setq org-mobile-directory "~/Dropbox/Apps/MobileOrg/")

;;; Menu to push or pull org files using MobileOrg
  (defun org-mobile-sync ()
    (interactive)
    (let
        (org-mobile-sync-type
         (read-from-minibuffer "How do you sync the org files? (pull or push)
"))
      (message "%s" org-mobile-sync-type)
      (cond
       ((string= "pull" org-mobile-sync-type) (org-mobile-pull))
       ((string= "push" org-mobile-sync-type) (org-mobile-push))))))
```

10.16. [org-tree-slide] Org でプレゼンテーション

http://pastelwill.jp/wiki/doku.php?id=emacs:org-tree-slide

```
;; Org-tree-slide
(when (eval-after-autoload-if-found
       'org-tree-slide-mode "org-tree-slide" nil t nil
       '(;; <f8>/<f9>/<f10>/<f11> are assigned to control org-tree-slide
         (define-key org-tree-slide-mode-map (kbd "<f9>")
           'org-tree-slide-move-previous-tree)
         (define-key org-tree-slide-mode-map (kbd "<f10>")
           'org-tree-slide-move-next-tree)
         (org-tree-slide-narrowing-control-profile)
         (setq org-tree-slide-modeline-display 'outside)
         (setq org-tree-slide-skip-outline-level 5)
         (setq org-tree-slide-skip-done nil)))

  (global-set-key (kbd "<f8>") 'org-tree-slide-mode)
  (global-set-key (kbd "S-<f8>") 'org-tree-slide-skip-done-toggle))
```

Doing タグのトグルに `f11` を割り当てたので、コンテンツモードへの切り替えは、異なるキーバインドに変更。

10.17. [org-tree-slide] クロックインとアウトを自動化する

特定のファイルを編集している時、 `org-tree-slide` でフォーカスしたら `org-clock-in` で時間計測を始めて、ナローイングを解く時や次のツリーに移る時に `org-clock-out` で計測を停止するように設定しています。基本的に `org-tree-slide` にある hook に色々とぶら下げるだけです。

```
(with-eval-after-load "org-tree-slide"
  (defun my:org-clock-in ()
    (setq vc-display-status nil) ;; モードライン節約
    (org-clock-in))

  (defun my:org-clock-out ()
    (setq vc-display-status t) ;; モードライン節約解除
    (require 'org-clock nil t)
    (when (org-clocking-p) (org-clock-out)))

  (add-hook 'org-tree-slide-before-move-previous-hook
            'my:org-clock-out)
  (add-hook 'org-tree-slide-before-move-next-hook
            'my:org-clock-out)
  (add-hook 'org-tree-slide-stop-hook 'my:org-clock-out)

  (add-hook 'org-tree-slide-before-narrow-hook
            '(lambda ()
               (when
                   (and (or (equal (buffer-name) "work.org")
                            (equal (buffer-name) "effort.org"))
                        (and (or (eq (org-outline-level) 2)
                                 (eq (org-outline-level) 3))
                             (looking-at (concat "^\\*+ "
                                                 org-not-done-regexp))))
                 (my:org-clock-in)))))
```

10.18. [org-tree-slide] 特定のツリーをプロポーショナルフォントで表示する

ツリーのプロパティに、プロポーショナルで表示するか否かの制御フラグを加えます。ツリーにフォーカス時に PROPORTIONAL 指定がプロパティにあると、そのツリーを動的にプロポーショナルフォントでレンダリングします。変更は下位ツリーの全てに継承しています。

```
(when (eval-after-autoload-if-found
       '(org-tree-slide) "org-tree-slide" nil t nil
       '((defcustom use-proportional-font nil
           "The status of FONT property"
           :type 'boolean
           :group 'org-mode)

         (set-face-attribute 'variable-pitch nil
                             :family "Verdana"
                             :height 125)

         (defun my:proportional-font-toggle ()
           (interactive)
           (setq use-proportional-font (not use-proportional-font))
           (if use-proportional-font
               (org-entry-put nil "FONT" "PROPORTIONAL")
             (org-delete-property "FONT")))

         (add-hook 'org-tree-slide-before-narrow-hook
                   '(lambda ()
                      (if (equal "PROPORTIONAL"
                                 (org-entry-get-with-inheritance "FONT"))
                          (buffer-face-set 'variable-pitch)
                        (buffer-face-mode 0))))
         (add-hook 'org-tree-slide-stop-hook
                   '(lambda ()
                      (buffer-face-mode 0))))))
```

10.18.1. キーバインド

```
(with-eval-after-load "org"
  (define-key org-mode-map (kbd "C-c f p") 'my:proportional-font-toggle))
```

10.19. [org-fstree] ディレクトリ構造を読み取る

```
(eval-after-autoload-if-found
 'org-mode "org" nil t nil
 '((require 'org-fstree nil t)))
```

10.20. [calfw-org] calfw に org の予定を表示する

org-mode の表のようにフェイスを統一しています。 `calfw` を起動する時に、自動的にフレームサイズを拡大するような独自関数をぶら下げています。

```
(when (eval-after-autoload-if-found
       '(my:cfw-open-org-calendar cfw:open-org-calendar)
       "calfw-org" "Rich calendar for org-mode" t nil
       '(
         ;; icalendar との連結
         (setq cfw:org-icalendars '("~/Dropbox/org/org-ical.org"))
```

```
        ;; org で使う表にフェイスを統一
        (setq cfw:fchar-junction ?+
              cfw:fchar-vertical-line ?|
              cfw:fchar-horizontal-line ?-
              cfw:fchar-left-junction ?|
              cfw:fchar-right-junction ?|
              cfw:fchar-top-junction ?+
              cfw:fchar-top-left-corner ?|
              cfw:fchar-top-right-corner ?| )

        (defun my:org-mark-ring-goto-calfw ()
          (interactive)
          (org-mark-ring-goto))

        (defun my:cfw-open-org-calendar ()
          (interactive)
          (change-frame-width-double)
          (cfw:open-org-calendar))

        (defun my:cfw-burry-buffer ()
          (interactive)
          (bury-buffer)
          (change-frame-width-single))

        (defun cfw:org-goto-date ()
          "Move the cursor to the specified date."
          (interactive)
          (cfw:navi-goto-date
           (cfw:emacs-to-calendar (org-read-date nil 'to-time))))

        (define-key cfw:calendar-mode-map (kbd "j") 'cfw:org-goto-date)
        (define-key cfw:org-schedule-map (kbd "q") 'my:cfw-burry-buffer)))

  (global-set-key (kbd "C-c f c") 'my:cfw-open-org-calendar))

;;          (add-hook 'window-configuration-change-hook 'cfw:resize-calendar)
;; (defun cfw:resize-calendar ()
;;   (interactive)
;;   (when (eq major-mode 'cfw:calendar-mode)
;;     (cfw:refresh-calendar-buffer nil)
;;     (message "Calendar resized.")))

;; (defun open-calfw-agenda-org ()
;;   (interactive)
;;   (cfw:open-org-calendar))

;; (setq org-agenda-custom-commands
;;       '(("w" todo "FOCUS")
;;         ("G" open-calfw-agenda-org "Graphical display in calfw"))))))
```

10.21. [org-export-generic] エクスポート機能を拡張する

`org-set-generic-type` を使うことで、エクスポート機能を好みに拡張できる。contrib の中の `org-export-generic.el` が必要なので注意する。

（注意）次の設定は古い内容。動かないかもしれません。

```
(with-eval-after-load "org"
  (org-set-generic-type
```

```
  "textile"
  '(:file-suffix
    ".textile"
    :key-binding ?T
    :title-format    "Title: %s\n\n"
    ;;   :date-format     "Date: %s\n"
    :date-export nil
    :toc-export      nil
    :author-export   nil
    :tags-export     nil
    :drawers-export  nil
    :date-export     t
    :timestamps-export  t
    :priorities-export  nil
    :todo-keywords-export t
    :body-line-fixed-format "\t%s\n"
                                      ;:body-list-prefix "\n"
    :body-list-format "* %s"
    :body-list-suffix "\n"
    :body-bullet-list-prefix ("* " "** " "*** " "**** " "***** ")
    :body-number-list-format "# %s"
    :body-number-list-suffix "\n"
    :header-prefix ("" "" "### " "#### " "##### " "###### ")
    :body-section-header-prefix ("h1. " "h2. " "h3. " "h4. " "h5. " "h6. ")
    :body-section-header-format "%s"
    :body-section-header-suffix ("\n\n")
    :body-header-section-numbers nil
    :body-header-section-number-format "%s) "
    :body-line-format "%s\n"
    :body-newline-paragraph "\n"
    :bold-format "*%s*"
    :italic-format "_%s_"
    :underline-format "+%s+"
    :strikethrough-format "-%s-"
    :verbatim-format "`%s`"
    :code-format "@%s@"
    :body-line-wrap   75
    :blockquote-start "\n<pre>\n"
    :blockquote-end "\n</pre>\n"
    ))

(org-set-generic-type
 "markdown"
 '(:file-suffix
    ".markdown"
    :key-binding     ?M
    :title-format    "Title: %s\n"
    :date-format     "Date: %s\n"
    :toc-export      nil
    :author-export   t
    :tags-export     nil
    :drawers-export  nil
    :date-export     t
    :timestamps-export  t
    :priorities-export  nil
    :todo-keywords-export t
    :body-line-fixed-format "\t%s\n"
    ;;:body-list-prefix "\n"
    :body-list-format "- %s"
    :body-list-suffix "\n"
    :header-prefix ("" "" "### " "#### " "##### " "###### ")
```

```
     :body-section-header-prefix ("" "" "### " "#### " "##### " "###### ")
     :body-section-header-format "%s\n"
     :body-section-header-suffix (?= ?- "")
     :body-header-section-numbers nil
     :body-header-section-number-format "%s) "
     :body-line-format "%s\n"
     :body-newline-paragraph "\n"
     :bold-format "**%s**"
     :italic-format "_%s_"
     :verbatim-format "`%s`"
     :code-format "`%s`"
     :body-line-wrap   75
     )))
```

org-set-generic-type を .emacs に追記した後、 C-c C-e g <key-binding> とすればよい。 <key-binding> は org-set-generic-type で設定する値である。2 つ目は、Markdown へのエクスポーターである。

10.22. [org-odt] ODT 形式に出力

```
(eval-after-autoload-if-found
 '(ox-odt) "ox-odt" nil t nil
 '((setq org-odt-styles-file
         (concat (getenv "HOME") "/Dropbox/emacs.d/config/style.odt"))
   ;; (setq org-odt-content-template-file
   ;;       (concat (getenv "HOME") "/Dropbox/emacs.d/config/style.ott"))
   (setq org-odt-preferred-output-format "pdf") ;; docx
   ;; ox-odt.el の 自作パッチの変数（DOCSTRING が記述されていない）
   (setq org-odt-apply-custom-punctuation t)
   (setq org-odt-convert-processes
         '(("LibreOffice"
            "/Applications/LibreOffice.app/Contents/MacOS/soffice --headless
--convert-to %f%x --outdir %d %i")
           ("unoconv" "unoconv -f %f -o %d %i")))))
```

10.23. [org-crypt] ツリーを暗号化する

M-x org-encrypt-entry でカーソル位置のツリーを暗号化できます。復号は、 M-x org-decrypt-entry にて。ただし、バッファのバックアップファイルが生成されていることに気をつけてください。自分の場合は、バックアップファイルは外部ホストに同期されない設定にしてあるので、とりあえず問題なしと考えています。

M-x org-encrypt-entries で、特定のタグが付けられたツリーを一括処理することもできますが、私は安全性を考慮して使っていません。

なお、実戦投入には十分なテストをしてからの方がよいでしょう。org バッファを外部ホストと同期している場合、転送先のホストでも暗号化／復号ができるかを確認するべきです。他方のホストでツリーにドロワーが付くと、復号できなくなったりします。その時は慌てずにプロパティのドロワーを削除すれば OK です。

```
(eval-after-autoload-if-found
 '(org-crypt org-encrypt-entry org-decrypt-entry) "org-crypt" "Org Mode" t nil
 '((setq org-crypt-key "<insert your key>")
   ;; org-encrypt-entries の影響を受けるタグを指定
   (setq org-tags-exclude-from-inheritance (quote ("secret")))
   ;; 自動保存の確認を無効に
   (setq org-crypt-disable-auto-save 'nil)))
```

10.24. [org-mac-link] 外部アプリから情報を取る

`org-mac-link` を使うと、外部アプリの表示状態をリンクとして取得して、org バッファに流し込めます。Mac 環境用です。簡単な例では URL で、取得したリンクを `C-c C-o` で開けばブラウザが起動してリンク先が表示できます。同じ話を、ファインダーで表示しているディレクトリ、メーラーで表示していた特定のメール、PDF ビューアで表示していた特定のファイルの特定のページなどで実施できます。対応している外部アプリは、Finder, Mail.app, Outlook, Addressbook, Safari, Firefox, Chrome, そして Skim です。

次のように設定すると、org-mode の時に `C-c c` すればミニバッファにどのアプリからリンク情報を取るか選べます。Chrome には `c` が当たっているので、ブラウジング中に記になる記事があったら Emacs に切り替えて、 `C-c c c` とすると、URL が自動でバッファに入ります。単なる URL ではなく、タイトルで表示されるのでわかりやすいです。

```
(with-eval-after-load "org"
  (add-to-list 'org-modules 'org-mac-iCal)
  (add-to-list 'org-modules 'org-mac-link) ;; includes org-mac-message
  (define-key org-mode-map (kbd "C-c c") 'org-mac-grab-link))
```

10.25. [terminal-notifier] イベント通知

Mac の通知機能を使って、Emacs で発生するイベントをユーザに通知します。 `appt-disp-window` をカスタマイズして `org-notify` を呼ぶことで、 `org-agenda` のアイテムも通知されます。

`org-show-notification-handler` に `terminal-notifier.app` を呼ぶ関数をぶら下げることで、 `org-notify` で簡単に通知機能を使えるようになります。

`terminal-notifier-sound` に `terminal-notifier.app` で指定可能な音声ファイル名を指定すると、通知音を制御できます。

```
(with-eval-after-load "org"
  (defvar terminal-notifier-command
    (executable-find
     "terminal-notifier.app/Contents/MacOS/terminal-notifier")
    "Path to terminal-notifier.
      Download from https://github.com/julienXX/terminal-notifier/releases")

  (defvar terminal-notifier-sound nil)
  (with-eval-after-load "org-clock"
    (when (not org-clock-sound)
      ;; Select from Preferences: { Funk | Glass | ... | Purr | Pop ... }
      (setq terminal-notifier-sound "Pop")))

  (defun terminal-notifier-notify (title message &optional sound)
    "Show a message with `terminal-notifier-command`."
    (if (and sound (not (string= sound "")))
        (start-process "terminal-notifier" "*terminal-notifier*"
                       terminal-notifier-command
                       "-title" title "-message" message
                       "-activate" "org.gnu.Emacs" "-sound" sound) ;; FIXME
      (start-process "terminal-notifier" "*terminal-notifier*"
                     terminal-notifier-command
                     "-title" title "-message" message
                     "-activate" "org.gnu.Emacs")))

  (defvar use-terminal-notifier t)
  (defun terminal-notifier-toggle ()
```

```
    (interactive)
    (setq use-terminal-notifier (not use-terminal-notifier))
    (my:org-agenda-to-appt)
    (message "terminal-notifier: %s" use-terminal-notifier))
  (define-key org-mode-map (kbd "C-c f n") 'terminal-notifier-toggle)

  ;; eval (org-notify "hoge") to test this setting
  (setq org-show-notification-handler
        #'(lambda (message)
            (when use-terminal-notifier
              (terminal-notifier-notify
               "Message from org-mode" message terminal-notifier-sound))))

  (defun advice:appt-disp-window (min-to-app new-time appt-msg)
    "Extension to support org-notify."
    (let ((time-msg (concat "in " min-to-app " min.")))
      (when (string= min-to-app "0")
        (setq time-msg "<= Do Now!!"))
      (org-notify (concat "\\\"" appt-msg "\" " time-msg))))
  (advice-add 'appt-disp-window :before #'advice:appt-disp-window))
```

10.25.1. References

- http://blog.devnode.pl/blog/2012/01/04/get-notified/
- https://github.com/p-m/org-notify/blob/master/org-notify.el
- http://sheephead.homelinux.org/2015/01/10/7220/

10.26. [org-grep] org ファイルを grep する

```
(when (eval-after-autoload-if-found
       '(org-grep) "org-grep" nil t nil
       '((setq org-grep-extensions nil)
         (add-to-list 'org-grep-directories "~/.emacs.d")
         (add-to-list 'org-grep-directories "~/.emacs.d/.cask/package")))

  (global-set-key (kbd "C-M-g") 'org-grep))
```

10.27. READ ME を常に org-mode で開く

```
(when (eval-after-autoload-if-found
       'org-mode "org")
  (push '("[rR][eE][aA][dD][mM][eE]" . org-mode) auto-mode-alist))
```

10.28. Growlnotify と org-mode でアラーム管理

（注）Growlnotify の代わりに `terminal-notifier` を使うこともできます。

growlnotify と org-mode のバッファを組み合わせてアラームリストを管理しています。アラームを org バッファに書き込むだけなので、とても楽です。機能としては、特定の org バッファに、時刻とアラームの内容を表の形式として保存しておくだけで、Emacs が起動している限りにおいて growl がそのアラームを表示してくれます。つまり、アラームリストは org-mode の表で一覧化されているので、管理も楽ですし、見た目もわかりやすいです。

アラームとして解釈される表は、オプション、時刻（HH:MM 形式）、アラーム内容の 3 列で構成していれば OK です。オプションの列に X を入れておくと、growl が Sticky モード

で動作するので、アラームを見逃しません。

アラームは複数登録することができます。不要になったアラームを削除するのは、単純に表から当該の行を削除するだけで済みます。実際のところは、バッファが保存される時にアラームリストの変更が自動的にシステムに反映されるので、余計な作業は不要です。

`set-alarms-from-file` は、utility.el に記述した関数です。

```
(with-eval-after-load "utility"
  (set-alarms-from-file "~/Dropbox/org/today.org")
  (defun my:update-alarms-from-file ()
    (when (string= "today.org" (buffer-name))
      (set-alarms-from-file "~/Dropbox/org/today.org"))))

(when (library-p "utility")
  (add-hook 'after-save-hook 'my:update-alarms-from-file))
```

10.28.1. キーバインド

```
(when (eval-after-autoload-if-found
       'org-mode "org" "Org Mode" t nil
       '(;; (org-transpose-element) が割り当てられているので取り返す。
         (org-defkey org-mode-map "\C-\M-t" 'beginning-of-buffer)

         ;;(define-key org-mode-map (kbd "C-c 1")
         ;;  'org-export-icalendar-combine-agenda-files)
         (define-key org-mode-map (kbd "C-c f 1") 'my:ox-icalendar)
         (defun my:do-org-update-staistics-cookies ()
           (interactive)
           (message "Update statistics ...")
           (do-org-update-statistics-cookies))
         (define-key org-mode-map (kbd "C-c f 2")
           'my:do-org-update-staistics-cookies)
         (define-key org-mode-map (kbd "C-c m") 'org-mobile-sync)
         (define-key org-mode-map (kbd "<f5>") 'org-narrow-to-subtree)
         (define-key org-mode-map (kbd "S-<f5>") 'widen)))

  (global-set-key (kbd "S-<f12>")
                  '(lambda () (interactive) (show-org-buffer "work.org")))
  (global-set-key (kbd "C-M-o")
                  '(lambda () (interactive) (show-org-buffer "next.org")))
  (global-set-key (kbd "C-M-c")
                  '(lambda () (interactive) (show-org-buffer "org-ical.org")))
  (global-set-key (kbd "C-M-9")
                  '(lambda () (interactive) (show-org-buffer "buffer.org")))
  (global-set-key (kbd "C-M-0")
                  '(lambda () (interactive) (show-org-buffer "today.org")))
  (global-set-key (kbd "C-c l") 'org-store-link)
  (global-set-key (kbd "C-c a") 'org-agenda)
  (global-set-key (kbd "C-c r") 'org-capture))
```

10.29. org-mode の latex エクスポート関数をオーバーライド

```
;;; Tex export (org-mode -> tex with beamer class) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; (setq org-export-latex-classes
;;   '(("article"
;;      "\\documentclass[11pt]{article}
;; \\usepackage[AUTO]{inputenc}
;; \\usepackage[T1]{fontenc}
;; \\usepackage{graphicx}
```

```
;; \\usepackage{longtable}
;; \\usepackage{float}
;; \\usepackage{wrapfig}
;; \\usepackage{soul}
;; \\usepackage{amssymb}
;; \\usepackage{hyperref}"
;;       ("\\section{%s}" . "\\section*{%s}")
;;       ("\\subsection{%s}" . "\\subsection*{%s}")
;;       ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
;;       ("\\paragraph{%s}" . "\\paragraph*{%s}")
;;       ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))
;;      ("report"
;;       "\\documentclass[11pt]{report}
;; \\usepackage[AUTO]{inputenc}
;; \\usepackage[T1]{fontenc}
;; \\usepackage{graphicx}
;; \\usepackage{longtable}
;; \\usepackage{float}
;; \\usepackage{wrapfig}
;; \\usepackage{soul}
;; \\usepackage{amssymb}
;; \\usepackage{hyperref}"
;;       ("\\part{%s}" . "\\part*{%s}")
;;       ("\\chapter{%s}" . "\\chapter*{%s}")
;;       ("\\section{%s}" . "\\section*{%s}")
;;       ("\\subsection{%s}" . "\\subsection*{%s}")
;;       ("\\subsubsection{%s}" . "\\subsubsection*{%s}"))
;;      ("book"
;;       "\\documentclass[11pt]{book}
;; \\usepackage[AUTO]{inputenc}
;; \\usepackage[T1]{fontenc}
;; \\usepackage{graphicx}
;; \\usepackage{longtable}
;; \\usepackage{float}
;; \\usepackage{wrapfig}
;; \\usepackage{soul}
;; \\usepackage{amssymb}
;; \\usepackage{hyperref}"
;;       ("\\part{%s}" . "\\part*{%s}")
;;       ("\\chapter{%s}" . "\\chapter*{%s}")
;;       ("\\section{%s}" . "\\section*{%s}")
;;       ("\\subsection{%s}" . "\\subsection*{%s}")
;;       ("\\subsubsection{%s}" . "\\subsubsection*{%s}"))
;;      ("beamer"
;;       "\\documentclass{beamer}
;; \\usepackage[AUTO]{inputenc}
;; \\usepackage{graphicx}
;; \\usepackage{longtable}
;; \\usepackage{float}
;; \\usepackage{wrapfig}
;; \\usepackage{amssymb}
;; \\usepackage{hyperref}"
;;       org-beamer-sectioning)))
```

10.30. [org-autolist] ブリッツの入力を簡単に

ブリッツの入力や削除を Microsoft Word 的にします。

```
(when (eval-after-autoload-if-found
       '(org-autolist-mode) "org-autolist" nil t nil nil)
```

```
  (add-hook 'org-mode-hook #'(lambda () (org-autolist-mode))))
```

11. フォント／配色関連

11.1. 正規表現を見やすくする

```
(set-face-foreground 'font-lock-regexp-grouping-backslash "#66CC99")
(set-face-foreground 'font-lock-regexp-grouping-construct "#9966CC")
```

Emacs Lisp での正規表現入力をサポートするツールには、 `M-x re-builder` や `rx` マクロがある。

- EmacsWiki: Re Builder
- EmacsWiki: rx

11.2. 設定ファイルを見やすくする

`generic-x` を使うと、 `/etc/hosts` や `/etc/apache2.conf` に色を付けられる。

```
(require 'generic-x nil t)
```

11.3. カーソル行に色をつける

```
;; Color of the current line
;; Cite: http://murakan.cocolog-nifty.com/blog/2009/01/emacs-tips-1d45.html
;; see also http://www.emacswiki.org/cgi-bin/emacs/highlight-current-line.el
(global-hl-line-mode t)
(set-face-background 'hl-line "#DEEDFF")
```

11.4. カーソルの色

```
;; Cursor (see also takaxp-mac.el)
;;(add-to-list 'default-frame-alist '(cursor-type . (hbar . 5)))
;;(add-to-list 'default-frame-alist '(cursor-type . bar))
;;(add-hook 'window-configuration-change-hook

(defvar my:cursor-color-ime-on "#91C3FF")
(defvar my:cursor-color-ime-off "#AAAAAA")
(set-cursor-color my:cursor-color-ime-on)

(when (and (eq window-system 'ns) (>= emacs-major-version 24))
  ;; when IME is ON
  (when (fboundp 'mac-set-input-method-parameter)
    (mac-set-input-method-parameter
     "com.google.inputmethod.Japanese.base" 'title "グ"))

  (defun my:update-cursor-color ()
    (interactive)
    (if current-input-method (set-cursor-color my:cursor-color-ime-off)
      (set-cursor-color my:cursor-color-ime-on)))
  ;; (my:update-cursor-color)
  (run-with-idle-timer 10 t 'my:update-cursor-color))

(add-hook 'input-method-activate-hook
          (lambda () (set-cursor-color my:cursor-color-ime-on)))
(add-hook 'input-method-inactivate-hook
```

```
          (lambda () (set-cursor-color my:cursor-color-ime-off)))
```

11.5. カーソルを点滅させない

```
;; Disable cursor blink
(blink-cursor-mode -1)
```

11.6. カーソル位置のフォントを確認

M-x describe-char することで、カーソル位置のフォントの情報がポップで表示される。

11.7. フォント設定

表のような利用環境に対して、個別に設定を施しています。Windows と Linux は安定版の Emacs23 で、Mac は開発版の CocoaEmacs23 です。Mac では Emacs24 でもうまく表示できています。最近は、Migu 2M を気に入って使っています。

	ASCII	日本語
Mac	Monaco	ヒラギノ丸ゴ
Windows	Inconsolata	メイリオ
Linux	Inconsolata	MigMix

http://d.hatena.ne.jp/setoryohei/20110117/1295336454

```
(defvar my:font-size 12)
(defun my:ja-font-setter (spec)
  (set-fontset-font nil 'japanese-jisx0208 spec)
  (set-fontset-font nil 'katakana-jisx0201 spec)
  (set-fontset-font nil 'japanese-jisx0212 spec)
  (set-fontset-font nil '(#x0080 . #x024F) spec)
  (set-fontset-font nil '(#x0370 . #x03FF) spec)
  (set-fontset-font nil 'mule-unicode-0100-24ff spec))
(defun my:ascii-font-setter (spec)
  (set-fontset-font nil 'ascii spec))

(cond
 ;; CocoaEmacs
 ((eq window-system 'ns)
  (when (>= emacs-major-version 23)
    (let
        ;; 1) Monaco, Hiragino/Migu 2M : font-size=12, -apple-hiragino=1.2
        ;; 2) Inconsolata, Migu 2M     : font-size=14,
        ;; 3) Inconsolata, Hiragino    : font-size=14, -apple-hiragino=1.0

        ;; Fonts

        ((font-size my:font-size)
         ;;          ((font-size 28) ; for mirroring presentation (1440x900)
         ;;           (ascii-font "Inconsolata")
         (ascii-font "Monaco")
         (ja-font "Migu 2M"))
      ;; (ja-font "Hiragino Maru Gothic Pro"))
```

```
      (my:ascii-font-setter (font-spec :family ascii-font :size font-size))
      (my:ja-font-setter (font-spec :family ja-font :size font-size)))

    ;; Fix ratio provided by set-face-attribute for fonts display
    (setq face-font-rescale-alist
          '(("^-apple-hiragino.*" . 1.0) ; 1.2
            (".*Migu.*" . 1.2)
            (".*Inconsolata.*" 1.0)
            (".*osaka-bold.*" . 1.0)      ; 1.2
            (".*osaka-medium.*" . 1.0)    ; 1.0
            (".*courier-bold-.*-mac-roman" . 1.0) ; 0.9
            ;; (".*monaco cy-bold-.*-mac-cyrillic" . 1.0)
            ;; (".*monaco-bold-.*-mac-roman" . 1.0) ; 0.9
            ("-cdac$" . 1.0)))            ; 1.3

    ;; Anti aliasing with Quartz 2D
    (setq mac-allow-anti-aliasing t)))

((eq window-system 'w32) ; windows7
 (let
      ((font-size 14)
       (font-height 100)
       (ascii-font "Inconsolata")
       ;; (ja-font "Meiryo UI"))
       (ja-font "メイリオ"))
    (my:ja-font-setter
     (font-spec :family ja-font :size font-size :height font-height))
    (my:ascii-font-setter (font-spec :family ascii-font :size font-size)))
 (setq face-font-rescale-alist '((".*Inconsolata.*" . 1.0))) ; 0.9
 )
(window-system ; for SuSE Linux 12.1
 (let
      ((font-size 14)
       (font-height 100)
       (ascii-font "Inconsolata")
       ;; (ja-font "MigMix 1M")
       (ja-font "Migu 1M"))
    (my:ja-font-setter
     (font-spec :family ja-font :size font-size :height font-height))
    (my:ascii-font-setter (font-spec :family ascii-font :size font-size)))
 (setq face-font-rescale-alist '((".*MigMix.*" . 2.0)
                                 (".*Inconsolata.*" . 1.0))) ; 0.9
 ))
```

11.7.1. フォントのインストール方法

Linux では次のように処理するだけでよく、意外と簡単。

1. ~/.fonts を作成する
2. フォントを 1.のディレクトリに置く
3. fc-cache -fv を実行
4. fc-list でインストールされているかを確認。

なお、Windows では、フォントファイルを右クリックして、インストールを選択するだけで OK。

11.7.2. フォントチェック用コード

サンプルの org ファイル を作って、見た目をチェックしています。バッファ内の桁数チェックや、ASCII が漢字の半分の幅になっているかのチェックが楽になります。

11.8. 行間を制御する

```
(set-default 'line-spacing 0.2)
```

11.9. パッチをカラフルに表示する

Built-in の diff-mode.el をカスタマイズします。

現在試験中。

```
(setq ediff-window-setup-function 'ediff-setup-windows-plain)
```

http://d.hatena.ne.jp/syohex/20111228/1325086893

```
(eval-after-autoload-if-found
 'diff-mode "diff-mode" nil t nil
 '((set-face-attribute 'diff-added-face nil
                       :background nil :foreground "green"
                       :weight 'normal)
   (set-face-attribute 'diff-removed-face nil
                       :background nil :foreground "firebrick1"
                       :weight 'normal)

   (set-face-attribute 'diff-file-header-face nil
                       :background nil :weight 'extra-bold)

   (set-face-attribute 'diff-hunk-header-face nil
                       :foreground "chocolate4"
                       :background "white" :weight 'extra-bold
                       :inherit nil)))
```

11.10. 背景を黒系色にする

```
(custom-set-faces
 '(default ((t
             (:background "black" :foreground "#55FF55")
             ))))
```

11.11. 時間帯を指定して起動時にテーマを切り替える

次の例では、19 時から翌日の朝 5 時までの間に夜用のテーマを使っています。

```
(defun my:night-time-p (begin end)
  (let* ((ch (string-to-number (format-time-string "%H" (current-time))))
         (cm (string-to-number (format-time-string "%M" (current-time))))
         (ct (+ cm (* 60 ch))))
    (if (> begin end)
        (or (<= begin ct) (<= ct end))
      (and (<= begin ct) (<= ct end)))))

(defvar night-time-in 21)
(defvar night-time-out 5)

(when (my:night-time-p (* night-time-in 60) (* night-time-out 60))
```

```
  ;;(when (require 'gotham-theme nil t)
  ;;(load-theme 'gotham t)
  (when (require 'night-theme nil t)
    (load-theme 'night t)
    (setq my:cursor-color-ime-on "#8599ff")))
```

11.12. [rainbow-mode.el] 配色のリアルタイム確認

M-x rainbow-mode とすると、色指定のコードの背景色を、その指定色にリアルタイム変換してくれる。

http://elpa.gnu.org/packages/rainbow-mode.html

```
(when (eval-after-autoload-if-found
       '(rainbow-mode) "rainbow-mode")
  (add-hook 'emmet-mode-hook 'rainbow-mode)
  (add-hook 'emacs-lisp-mode-hook 'rainbow-mode)
  (add-hook 'org-mode-hook 'rainbow-mode))
```

11.12.1. 色一覧

0,6,9,C,F の組み合わせ

#000000 #000033 #000066 #000099 #0000CC #0000FF
#003300 #003333 #003366 #003399 #0033CC #0033FF
#006600 #006633 #006666 #006699 #0066CC #0066FF
#009900 #009933 #009966 #009999 #0099CC #0099FF
#00CC00 #00CC33 #00CC66 #00CC99 #00CCCC #00CCFF
#00FF00 #00FF33 #00FF66 #00FF99 #00FFCC #00FFFF

#330000 #330033 #330066 #330099 #3300CC #3300FF
#333300 #333333 #333366 #333399 #3333CC #3333FF
#336600 #336633 #336666 #336699 #3366CC #3366FF
#339900 #339933 #339966 #339999 #3399CC #3399FF
#33CC00 #33CC33 #33CC66 #33CC99 #33CCCC #33CCFF
#33FF00 #33FF33 #33FF66 #33FF99 #33FFCC #33FFFF

#660000 #660033 #660066 #660099 #6600CC #6600FF
#663300 #663333 #663366 #663399 #6633CC #6633FF
#666600 #666633 #666666 #666699 #6666CC #6666FF
#669900 #669933 #669966 #669999 #6699CC #6699FF
#66CC00 #66CC33 #66CC66 #66CC99 #66CCCC #66CCFF
#66FF00 #66FF33 #66FF66 #66FF99 #66FFCC #66FFFF

#990000 #990033 #990066 #990099 #9900CC #9900FF
#993300 #993333 #993366 #993399 #9933CC #9933FF
#996600 #996633 #996666 #996699 #9966CC #9966FF
#999900 #999933 #999966 #999999 #9999CC #9999FF
#99CC00 #99CC33 #99CC66 #99CC99 #99CCCC #99CCFF
#99FF00 #99FF33 #99FF66 #99FF99 #99FFCC #99FFFF

#CC0000 #CC0033 #CC0066 #CC0099 #CC00CC #CC00FF
#CC3300 #CC3333 #CC3366 #CC3399 #CC33CC #CC33FF
#CC6600 #CC6633 #CC6666 #CC6699 #CC66CC #CC66FF
#CC9900 #CC9933 #CC9966 #CC9999 #CC99CC #CC99FF
#CCCC00 #CCCC33 #CCCC66 #CCCC99 #CCCCCC #CCCCFF

#CCFF00 #CCFF33 #CCFF66 #CCFF99 #CCFFCC #CCFFFF

#FF0000 #FF0033 #FF0066 #FF0099 #FF00CC #FF00FF
#FF3300 #FF3333 #FF3366 #FF3399 #FF33CC #FF33FF
#FF6600 #FF6633 #FF6666 #FF6699 #FF66CC #FF66FF
#FF9900 #FF9933 #FF9966 #FF9999 #FF99CC #FF99FF
#FFCC00 #FFCC33 #FFCC66 #FFCC99 #FFCCCC #FFCCFF
#FFFF00 #FFFF33 #FFFF66 #FFFF99 #FFFFCC #FFFFFF

11.13. [volatile-highlights] コピペした領域を強調

コピペ領域をその直後のみハイライトします。

```
(when (eval-after-autoload-if-found
       '(volatile-highlights-mode) "volatile-highlights" nil t nil
       '((set-face-attribute
          'vhl/default-face nil :foreground "#FF3333" :background "#FFCDCD")
         (volatile-highlights-mode t)

         ;; ふわっとエフェクトの追加（ペースト時の色 => カーソル色 => 本来色）
         (defun my:vhl-change-color ()
           (let
               ((next 0.2)
                (reset 0.5)
                (colors '("#F8D3D7" "#F2DAE1" "#EBE0EB" "#E5E7F5" "#DEEDFF")))
             (dolist (color colors)
               (run-at-time next nil
                            'set-face-attribute
                            'vhl/default-face
                            nil :foreground "#FF3333" :background color)
               (setq next (+ 0.05 next)))
             (run-at-time reset nil 'vhl/clear-all))
           (set-face-attribute 'vhl/default-face
                               nil :foreground "#FF3333"
                               :background "#FFCDCD"))

         (defun my:yank (&optional ARG)
           (interactive)
           (yank ARG)
           (my:vhl-change-color))
         (global-set-key (kbd "M-v") 'my:yank)
         (global-set-key (kbd "C-y") 'my:yank)

         (with-eval-after-load "org"
           (define-key org-mode-map (kbd "C-y")
             '(lambda () (interactive)
                (org-yank)
                (my:vhl-change-color))))))

  (add-hook 'org-mode-hook 'volatile-highlights-mode)
  (add-hook 'emacs-lisp-mode-hook 'volatile-highlights-mode)
  (add-hook 'emmet-mode-hook 'rainbow-mode))
```

12. フレーム／ウィンドウ制御

12.1. 起動時の設定

```
;; To avoid an error setting up the frame width (only for Emacs23)
                                          ;(set-frame-width (selected-frame) 81)
                                          ;(set-frame-width (selected-frame) 80)

;; Default window position to show a Emacs frame
;; Dynabook UX: top=0, left=0, width=80, height=32
(cond
 ((eq window-system 'ns) ; for Macintosh
  (setq initial-frame-alist
        (append
         '((top . 22)  ; Y-pos from (0,0) the height of menu bar is 22pix.
           (left . 0)  ; X-pos from (0,0) ; 420 is the center for MBP
           ;; 26 is the setting for Butler's Docklet
           ;; 837 is the setting for right side for MBP
           (width . 80) ; Width  : character count
           (alpha . (100 90))
           (cursor-height . 16)
           (vertical-scroll-bars . nil)
           ) initial-frame-alist)))

 ((eq window-system 'x) ; for Linux
  (setq initial-frame-alist
        (append
         '((vertical-scroll-bars . nil)
           (top . 0)
           (left . 0)
           (width . 80)
           (height . 38)
           ) initial-frame-alist)))

 (t                     ; for Windows
  (setq initial-frame-alist
        (append
         '((vertical-scroll-bars . nil)
           (top . 0)
           (left . 0)
           (width . 80)
           (height . 26)
           ) initial-frame-alist))))

;; Apply the initial setting to default
(setq default-frame-alist initial-frame-alist)
```

12.2. [elscreen.el] Emacs バッファをタブ化

```
;;; ElScreen (require apel) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Note: change a string in the elscreen.el from "mac" to "ns"
;; 2011-10-26: e2wm's perspective (two) mode is more useful for me.
(load "elscreen" "ElScreen" t)
```

12.3. [e2wm.el] 二画面表示

1. http://github.com/kiwanami/emacs-window-manager/raw/master/e2wm.el

2. http://github.com/kiwanami/emacs-window-layout/raw/master/window-layout.el

```
(eval-after-autoload-if-found
 'e2wm:dp-two "e2wm" nil t nil
 '((setq e2wm:c-two-recipe
         '(- (:lower-size 10)
             (| left right)
             sub))
   (setq e2wm:c-two-winfo
         '((:name left )
           (:name right )
           (:name sub :default-hide t)))

   ;; 高さを画面の最大に矯正
   (when (require 'frame-ctr nil t)
     (setq frame-height-tall (max-frame-height))
     (setq frame-height-small frame-height-tall))

   ;; left, prev
   (setq e2wm:c-two-right-default 'left)

   ;; To avoid rebooting issue when using desktop.el and recentf.el
   (add-hook 'kill-emacs-hook 'e2wm:stop-management)))
```

12.4. [frame-ctr.el] キーボードでフレームの場所を移す

拙作の frame-ctr.el を使います。e2wm.el と frame-cmds.el に依存しています。

frame-ctr.el は、frame-cmds, frame-fns と e2wm.el を利用したアドホックなフレーム管理ツールです。

1. http://www.emacswiki.org/emacs/download/frame-cmds.el
2. http://www.emacswiki.org/emacs/download/frame-fns.el

ビルトインに `face-remap` があり、アスキーフォントは `C-x C--` と `C-x C-=` で拡大縮小を制御できます。以下のキーバインドは、face-remap.el が提供する text-scale-adjust のキーバインドを上書きします。text-scale-adjust をそのまま使うと、日本語フォントが制御されないので、オーバーライドしてしまっても OK だと思います。

```
(defvar frame-ctr-autoloads
  '(move-frame-with-user-specify
    move-frame-to-center move-frame-to-edge-top move-frame-right
    move-frame-left move-frame-to-edge-bottom frame-ctr-open-height-ring
    fit-frame-to-fullscreen set-font-size-input max-frame-height
    reset-font-size increase-font-size decrease-font-size set-font-size))

(when (eval-after-autoload-if-found
       frame-ctr-autoloads "frame-ctr" nil t nil
       '((make-frame-height-ring)
         (cond
          ((eq (display-pixel-width) 1920) (setq fullscreen-fontsize 38))
          ((eq (display-pixel-width) 1440) (setq fullscreen-fontsize 28))
          ((eq (display-pixel-width) 1366) (setq fullscreen-fontsize 19))
          ((eq (display-pixel-width) 800) (setq fullscreen-fontsize 14))
          (t (setq fullscreen-fontsize 12)))

         ;; リングの状態を最新に更新（max-frame-height が変わるため）
         (add-hook 'frame-ctr-after-fullscreen-hook
                   'make-frame-height-ring)))
```

```
  ;; Move the frame to somewhere (default: 0,0)
  (global-set-key (kbd "M-0") 'move-frame-with-user-specify)
  ;; Move the frame to left side of the current position (require 'frame-cmds)
  (global-set-key (kbd "M-1") '(lambda () (interactive) (move-frame-left 40)))
  ;; Move the frame to the center of the window display (require 'frame-ctr)
  (global-set-key (kbd "M-2") 'move-frame-to-center)
  ;; Move the frame to right side of the current position(require 'frame-cmds)
  (global-set-key (kbd "M-3") '(lambda () (interactive)(move-frame-right 40)))
  ;; Move the current frame to the top of the window display
  (global-set-key (kbd "<f1>") 'move-frame-to-edge-top)
  ;; Move the current frame to the bottom of the window display
  (global-set-key (kbd "S-<f1>") 'move-frame-to-edge-bottom)
  ;; Cycle heights
  (global-set-key (kbd "<f2>") 'frame-ctr-open-height-ring)

  (global-set-key (kbd "C-x C-9") 'fit-frame-to-fullscreen)
  (global-set-key (kbd "C-x C-0") 'reset-font-size)
  (global-set-key (kbd "C-=") 'increase-font-size)
  (global-set-key (kbd "C--") 'decrease-font-size))

;; setting for e2wm
(when (eval-after-autoload-if-found
       '(change-frame-width-single
         change-frame-width-double change-frame-double-window
         change-frame-single-window) "frame-ctr-e2wm")

  ;; Set the frame width single size
  ;;  C-u C-x - => e2wm OFF, single size width and double height, move center
  (global-set-key (kbd "C-x -") 'change-frame-single-window)
  ;; Set the frame width double size
  ;;  C-u C-x = => e2wm ON, double size width and height, move to the center
  (global-set-key (kbd "C-x =") 'change-frame-double-window)
  (global-set-key (kbd "C-c f s") 'change-frame-width-single)
  (global-set-key (kbd "C-c f d") 'change-frame-width-double))
```

12.5. [popwin.el] ポップアップウィンドウの制御

https://github.com/m2ym/popwin-el/

popwin:display-buffer を autoload してもうまくいかない。

```
(when (eval-after-autoload-if-found
       '(popwin-mode) "popwin" nil t nil
       '(;; for emacs 24.1
         ;; (setq special-display-function 'popwin:special-display-popup-
window)
         ;; (setq display-buffer-function 'popwin:display-buffer)
         ;; for emacs 24.3
         ;; (setq special-display-alist 'popwin:special-display-popup-window)
         ;; (setq display-buffer-alist 'popwin:display-buffer)
         ;; (push '("*sdic*" :position top) popwin:special-display-config)

         (setq popwin:special-display-config
               (append
                '(("CAPTURE-next.org" :height 10 :position bottom :noselect t)
                  ("CAPTURE-org-ical.org" :height 10 :position bottom :noselect
t)
                  ("*Help*"          :height 20 :position bottom)
                  ("dict-app-result" :height 20 :position bottom)
                  ("*osx-dictionary*" :height 20 :position bottom)
                  ("*wclock*"        :height 10 :position bottom)
```

```
                  ("*Org Agenda*"    :height 10 :position bottom)
                  ("*Org Select*"  :height 10 :position bottom)
                  ("*Occur*"       :height 10 :position bottom)
;;;             (undo-tree-visualizer-buffer-name :height 10 :position top)
;;;             (undo-tree-diff-buffer-name :height 20 :position bottom)
                  ("*eshell*"      :height 10 :position bottom))
                popwin:special-display-config))))
  (popwin-mode 1))
```

13. ユーティリティ関数

13.1. [pomodoro.el] ポモドーロの実践

@syohex さん謹製の pomodoro.el に少しカスタマイズしたおれおれ pomodoro.el を使っています。以下のように設定すると、ポモドーロの残り時間は表示せず、アイコンだけをモードラインに表示できます。残り時間は `M-x pomodoro:mode-line-time-display-toggle` すれば、いつでも表示できます。

`pomodoro:finish-work-hook`、 `pomodoro:finish-rest-hook`、`pomodoro:long-rest-hook` にそれぞれ結びつけてあるのは、Mac のスピーチ機能です。この例では、Kyoko さんが指示を出してくれます。

`M-x pomodoro:start` すると、ポモドーロが始まり、8 時間後に `pomodoro:stop` が呼ばれてポモドーロが終了します。pomodoro は機械的に仕事をしたい人にピッタリです。人によっては GTD よりも取っ付きやすいと思います。

```
(eval-after-autoload-if-found
 '(pomodoro:start) "pomodoro" nil t nil
 '(;; 作業時間終了後に開くファイルを指定しない
   (setq pomodoro:file nil)

   ;; ●だけで表現する（残り時間表示なし）
   (setq pomodoro:mode-line-time-display nil)

   ;; ●の置き換え
   (setq pomodoro:mode-line-work-sign "●")
   (setq pomodoro:mode-line-rest-sign pomodoro:mode-line-work-sign)
   (setq pomodoro:mode-line-long-rest-sign pomodoro:mode-line-work-sign)

   ;; 長い休憩に入るまでにポモドーロする回数
   (setq pomodoro:iteration-for-long-rest 2)

   ;; 作業時間関連
   (setq pomodoro:work-time 120      ; 作業時間
         pomodoro:rest-time 20       ; 休憩時間
         pomodoro:long-rest-time 60 ; 長い休憩時間
         pomodoro:max-iteration 16) ; ポモドーロする回数

   ;; タイマーの表示をノーマルフェイスにする
   (set-face-bold-p 'pomodoro:timer-face nil)

   ;; 作業中（赤）、休憩中（青）、長い休憩中（緑）にする
   (set-face-foreground 'pomodoro:work-face "#F53838")
   (set-face-foreground 'pomodoro:rest-face "#3869FA")
   (set-face-foreground 'pomodoro:long-rest-face "#00B800")

   (defvar my:pomodoro-speak nil)
```

```
  (defun my:pomodoro-speak-toggle ()
    (interactive)
    (setq my:pomodoro-speak (not my:pomodoro-speak)))

  ;; Mac ユーザ向け。Kyoko さんに指示してもらう
  (defvar pomodoro:with-speak nil)
  (when pomodoro:with-speak
    (add-hook 'pomodoro:finish-work-hook
              (lambda ()
                (let ((script
                       (concat "say -v Kyoko "
                               (number-to-string (floor pomodoro:rest-time))
                               "分間、休憩しろ")))
                  (if my:pomodoro-speak
                      (shell-command-to-string script)
                    (message "%s" script)))))

    (add-hook 'pomodoro:finish-rest-hook
              (lambda ()
                (let ((script
                       (concat "say -v Kyoko "
                               (number-to-string (floor pomodoro:work-time))
                               "分間、作業しろ")))
                  (if my:pomodoro-speak
                      (shell-command-to-string script)
                    (message "%s" script)))))

    (add-hook 'pomodoro:long-rest-hook
              (lambda ()
                (let ((script
                       (concat "say -v Kyoko これから"
                               (number-to-string
                                (floor pomodoro:long-rest-time))
                               "分間の休憩です")))
                  (if my:pomodoro-speak
                      (shell-command-to-string script)
                    (message "%s" script))))))

  (defun my:pomodoro-notify ()
    (interactive)
    (shell-command-to-string
     (concat "terminal-notifier -title \"Pomodoro\" -message \""
             "DONE! This time slot has been finished!\"")))
  (add-hook 'pomodoro:finish-work-hook 'my:pomodoro-notify)
  ))
```

13.2. [utility.el] 自作してテスト中の便利関数群

関数定義を別ファイルに分離して、Emacs 起動の高速化を図っています。各関数を autoload の管理下において、必要なときにロードするように設定しています。

```
(defvar utility-autoloads
  '(takaxp:date
    takaxp:window-resizer takaxp:open-file-ring my:update-alarms-from-file
    my:desktop-notify my:daylight-theme my:night-theme
    eval-org-buffer kyoko-mad-mode-toggle
    org2dokuwiki-cp-kill-ring open-current-directory set-alarms-from-file
    reload-ical-export show-org-buffer get-random-string init-auto-install
    add-itemize-head insert-formatted-current-date
    insert-formatted-current-time insert-formatted-signature
```

```
    export-timeline-business export-timeline-private
    browse-url-chrome count-words-buffer do-test-applescript
    delete-backup-files recursive-delete-backup-files describe-timer))

(when (eval-after-autoload-if-found utility-autoloads "utility")
  (global-set-key (kbd "<f12>") 'takaxp:open-file-ring)
  (global-set-key (kbd "C-c t") 'takaxp:date)
  (global-set-key (kbd "C-c f 4") 'takaxp:window-resizer))
```

14. お試し中

14.1. yascroll

- http://d.hatena.ne.jp/m2ym/20110401/1301617991
- 2015-03-15: smooth-scroll との組み合わせで重いため試用停止中

```
(use-package yascroll
  :disabled t
  :init
  (global-yascroll-bar-mode 1)
  (setq yascroll:delay-to-hide 1)
  (setq yascroll:disabled-modes '(org))
  (set-face-foreground 'yascroll:thumb-fringe "#b2cefb")
  (set-face-background 'yascroll:thumb-fringe "#b2cefb")
  )
```

15. provide

```
(provide 'init)
```