

# Configurations for GNU Emacs

*Takaaki Ishikawa*

## Table of Contents

- [1. はじめに](#)
- [2. 起動設定](#)
  - [2.1. init.el のヘッダ](#)
  - [2.2. init ファイルの読み込み時間計測](#)
    - [2.2.1. Emacs 起動時の呼び出し順](#)
  - [2.3. 起動時間の計測](#)
  - [2.4. GC サイズの最適化](#)
  - [2.5. cl を使う](#)
  - [2.6. 警告の抑制](#)
  - [2.7. エラー表示の抑制](#)
  - [2.8. バッファの保存を静かにする](#)
  - [2.9. \[my-load-package-p\] 設定の読み込みフラグを確認する](#)
  - [2.10. \[autoload-if-found\] 関数をリストで渡す autoload 設定](#)
  - [2.11. \[future-time-p\] 指定時刻が今日の未来の時刻かを判定](#)
  - [2.12. \[input-focus-p\] フォーカス判定](#)
  - [2.13. \[postpone.el\] Emacs 起動してから使い始めるタイミングで実行する](#)
  - [2.14. \[shut-up.el\] Messages 出力を封じる \( 制御する \)](#)
    - [2.14.1. with-suppressed-message \[OBSOLETE\]](#)
  - [2.15. \[exec-path-from-shell.el\] PATH 設定をシェルから継承する \[OBSOLETE\]](#)
  - [2.16. \[eval-after-autoload-if-found\] 遅延読み込み \[OBSOLETE\]](#)
    - [2.16.1. 関数版](#)
    - [2.16.2. マクロ版](#)
- [3. コア設定](#)
  - [3.1. 言語 / 文字コード](#)
  - [3.2. 日本語入力](#)
  - [3.3. 基本キーバインド](#)
  - [3.4. ナローイングするか](#)
  - [3.5. バッファの終わりでの newline を禁止する](#)
  - [3.6. 常に最終行に一行追加する](#)
  - [3.7. 長い文章を右端で常に折り返す](#)
  - [3.8. マウスで選択した領域を自動コピー](#)
  - [3.9. compilation buffer でのデータ表示で自動するスクロールする](#)
  - [3.10. C-x C-c で容易に Emacs を終了させないように質問する](#)
  - [3.11. パッケージ管理](#)
    - [3.11.1. \[cask-mode.el\] モード設定](#)

### 3.11.2. Cask のセットアップ

### 3.11.3. load-path を一箇所にして起動を高速化

### 3.11.4. [paradox.el] パッケージ選択画面の改善

## 3.12. インデント

### 3.13. 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

### 3.14. 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

### 3.15. マウススクロールをピクセル単位にする

### 3.16. [aggressive-indent.el] 即時バッファ整形

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

### 3.18. [ws-butler.el] 不要なスペースを自動除去

### 3.19. [ag.el] 検索

### 3.20. NS ビルド用設定 [MACOS]

### 3.21. EMP ビルド用設定 [MACOS]

## 4. カーソル移動

### 4.1. バッファ内のカーソル移動

### 4.2. バッファ間のカーソル移動

### 4.3. スクロールを制御

### 4.4. スクロールで表示を重複させる行数

### 4.5. [smooth-scroll.el] 滑らかなスクロール

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

### 4.7. [bm.el] カーソル位置をブックマークして追う

### 4.8. [centered-cursor-mode.el] カーソル位置をバッファ中央に固定

### 4.9. [smart-mark] C-g 後に元の場所へカーソルを戻す

### 4.10. [syntax-subword.el] M-f で移動する位置をより密にする

### 4.11. [goto-chg.el] 編集箇所を簡単に辿る

### 4.12. DONE [back-button] マークをたどる [OBSOLETE]

### 4.13. DONE [point-undo.el] カーソル位置を簡単にたどる [OBSOLETE]

### 4.14. DONE [SmoothScroll.el] カーソル固定でスクロールする [OBSOLETE]

## 5. 編集サポート

### 5.1. エンターキーの挙動

### 5.2. Yank 時に装飾を取る

### 5.3. 矩形編集 / 連番入力

### 5.4. ファイル保存時に時間を記録する

### 5.5. 選択リージョンを使って検索

### 5.6. ChangeLog モード

### 5.7. テキストモード

### 5.8. C/C++モード

### 5.9. C#モード

### 5.10. Info モード

### 5.11. R モード

### 5.12. nXML モード

### 5.13. yaml モード

### 5.14. json モード

- [5.15. javascript モード](#)
- [5.16. csv モード](#)
- [5.17. ascii モード](#)
- [5.18. java モード](#)
- [5.19. es モード](#)
- [5.20. gnuplot モード](#)
- [5.21. markdown-mode モード](#)
- [5.22. cmake モード](#)
- [5.23. Web/HTML モード \[WEB\]](#)
- [5.24. PO モード](#)
- [5.25. スペルチェック](#)
- [5.26. リアルタイムスペルチェック](#)
- [5.27. リージョン内の文字をカウントする](#)
- [5.28. 世界時計を使う](#)
- [5.29. \[latex-math-preview.el\] TeX 数式をプレビュー \[TEX\]](#)
- [5.30. \[yatex.el\] YaTeX で Tex 編集 \[TEX\]](#)
- [5.31. \[auxtex.el\] AUCTEX で Tex 編集 \[TEX\]](#)
- [5.32. \[yasnipet.el\] Emacs 用のテンプレートシステム](#)
- [5.33. \[osx-dictionary.el\] macOS の dictionary.app で辞書をひく](#)
- [5.34. \[describe-number.el\] 16 進数などを確認](#)
- [5.35. \[emmet-mode.el\] zencoding の後継 \[WEB\]](#)
- [5.36. \[web-beautify.el\] ソースコード整形 \[WEB\]](#)
- [5.37. \[smartparens.el\] 対応するカッコの挿入をアシスト](#)
- [5.38. \[replace-from-region.el\] 選択領域を別の文字列に置き換える](#)
- [5.39. \[selected.el\] リージョン選択時のアクションを制御](#)
- [5.40. \[helm-selected.el\] selecte.el のアクション候補を絞り込み](#)
- [5.41. TODO \[isolate.el\] ブラケット等の入力をアシスト](#)
- [5.42. TODO \[git-complete.el\] GIT grep を使う補完エンジン](#)
- [5.43. TODO \[tiny.el\] 連番入力をサポート](#)
- [5.44. TODO \[bratex.el\] LaTeX 数式のブラケット入力をサポート](#)
- [5.45. DONE \[iedit.el\] バッファ内の同じ文字列を一度に編集する \[OBSOLETE\]](#)
- [5.46. DONE \[lookup.el\] 辞書 \[OBSOLETE\]](#)
- [5.47. DONE \[cacoo\] Cacao で描く \[OBSOLETE\]](#)
- [5.48. DONE \[zencoding-mode\] HTML 編集の高速化 \[OBSOLETE\]](#)
- [5.49. DONE \[sdc.el\] 英辞郎で英単語を調べる \[OBSOLETE\]](#)
- [5.50. DONE \[dictionary.app\] macOS の dictionary.app で COBUILD5 をひく \[OBSOLETE\]](#)
  - [5.50.1. マイナーモード化](#)
  - [5.50.2. キーバインド](#)
- [6. 表示サポート](#)
  - [6.1. キーコマンド入力中に入力過程をミニバッファに反映する](#)
  - [6.2. \[delight.el\] モードラインのモード名を短縮する](#)
  - [6.3. モードラインの Narrow を短くする](#)
  - [6.4. モードラインの節約 \( VC-mode 編 \)](#)

## 6.5. モードラインの色をカスタマイズする

### 6.5.1. 色セット例

## 6.6. 行番号の表示制限を拡張する

## 6.7. visible-bell のカスタマイズ

## 6.8. 常に `scratch` を表示して起動する

## 6.9. スクロールバーを非表示にする

## 6.10. ツールバーを非表示にする

## 6.11. 起動時のスプラッシュ画面を表示しない

## 6.12. カーソル行の行数をモードラインに表示する

## 6.13. カーソル行の関数名をモードラインに表示する

## 6.14. 行番号をバッファに表示する

## 6.15. 時刻をモードラインに表示する

## 6.16. 対応するカッコをハイライトする

## 6.17. 全角スペースと行末タブ / 半角スペースを強調表示する

## 6.18. バッファの終わりをフリンジで明示

## 6.19. [migemo.el] ローマ字入力で日本語を検索する

## 6.20. [helm.el] 続・何でも絞り込みインターフェイス

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

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

## 6.23. [calendar.el] カレンダーで週番号を表示する

## 6.24. [which-key] キーバインドの選択肢をポップアップする

## 6.25. [highlight-symbol] 同じ名前のシンボルをハイライトする

## 6.26. [all-the-icons-dired] フォントを使ったアイコンを表示

## 6.27. [eldoc.el] コンテキストに応じてヘルプを表示

## 6.28. [keycast.el] 入力しているキーとコマンドをモードラインに表示

## 6.29. TODO [rainbow-delimiters.el] 対応するカッコに色を付ける

## 6.30. TODO [yascroll.el] フリンジにスクロールバーを出す

## 6.31. TODO [dimmer.el] 現在のバッファ以外の輝度を落とす

## 6.32. DONE バッテリー情報をモードラインに表示する [OBSOLETE]

## 6.33. DONE [mode-icons] 使用中のモード表示をアイコンで代替 [OBSOLETE]

## 6.34. DONE [stock-ticker] 株価をモードラインに表示 [OBSOLETE]

## 6.35. DONE [zlc.el] find-file バッファを zsh ライクにする [OBSOLETE]

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

## 6.37. DONE [guide-key] キーバインドの選択肢をポップアップする [OBSOLETE]

## 6.38. DONE [anything.el] 何でも絞り込みインターフェイス [OBSOLETE]

## 6.39. DONE [diminish.el] モードラインのモード名を短くする [OBSOLETE]

## 7. メディアサポート

### 7.1. [emms.el] メディアプレーヤー

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

### 7.3. [japanlaw.el] Emacs 電子六法

### 7.4. [sunshine.el] 天気を知る

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

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

## 8. 履歴 / ファイル管理

- 8.1. [履歴サイズを大きくする](#)
- 8.2. [Undo バッファを無限に取る](#)
- 8.3. [最近開いたファイルリストを保持](#)
- 8.4. [バッファ保存時にバックアップを生成させない \[BACKUP\]](#)
- 8.5. [特定のファイルを Dropbox 以下にバックアップする \[BACKUP\]](#)
- 8.6. [\[backup-each-save.el\] クラッシュに備える \[BACKUP\]](#)
- 8.7. [\[dired\] ファイラのサポートツール \[DIREL\]](#)
- 8.8. [\[dired-recent.el\] 訪問したディレクトリの履歴を取る \[DIREL\]](#)
- 8.9. [\[osx-trash\] system-move-file-to-trash を有効にする](#)
- 8.10. [\[undo-tree\] 編集履歴をわかりやすくとどる](#)
- 8.11. [\[auto-save-buffers.el\] 一定間隔でバッファを保存する](#)
- 8.12. [\[session.el\] 様々な履歴を保存し復元に利用する](#)
- 8.13. [\[neotree.el\] ディレクトリ情報をツリー表示](#)
- 8.14. [\[helpful.el\] リッチなヘルプページ](#)
- 8.15. [\[keyfreq.el\] コマンドログ](#)
- 8.16. [DONE \[wakatime-mode.el\] WakaTime を利用して作業記録する \[OBSOLETE\]](#)
- 8.17. [DONE \[backup-dir.el\] バックアップファイルを一箇所に集める \[OBSOLETE\]](#)
- 8.18. [DONE バッファ保存時にバックアップファイルを生成する \[OBSOLETE\]](#)
- 8.19. [DONE ミニバッファの履歴を保存しリストアする \[OBSOLETE\]](#)
- 8.20. [DONE Emacs 終了時に開いていたバッファを起動時に復元する \[OBSOLETE\]](#)
- 8.21. [DONE 深夜にバッファを自動整理する \[OBSOLETE\]](#)

## 9. 開発サポート

- 9.1. [便利キーバインド](#)
- 9.2. [\[gist.el\] Gist インターフェイス](#)
- 9.3. [\[doxymacs.el\] Doxygen のコメントを簡単に入力する](#)
- 9.4. [\[matlab.el\] Matlab 用の設定](#)
- 9.5. [\[flycheck.el\] 構文エラー表示](#)
- 9.6. [\[auto-complete.el\] 自動補完機能](#)
- 9.7. [\[auto-complete-clang.el\] オムニ補完](#)
  - 9.7.1. [References](#)
- 9.8. [\[origami.el\] 関数の折りたたみ](#)
- 9.9. [\[quickrun.el\] お手軽ビルド](#)
- 9.10. [\[ggtags.el\] タグジャンプ](#)
- 9.11. [\[0xc.el\] N進数変換](#)
- 9.12. [\[hexl.el\] バイナリファイルを開く](#)
- 9.13. [\[uuid.el\] UUID の生成](#)
- 9.14. [\[package-lint.el\] MEPLA 登録用 Lint](#)
- 9.15. [\[projectile.el\] ディレクトリ単位でファイル群をプロジェクト扱いする](#)
- 9.16. [TODO \[EditorConfig\] コードスタイルの強制](#)
- 9.17. [TODO \[cov\] カバレッジの状態をフリンジで確認](#)
- 9.18. [TODO \[magit.el\] Git クライアント](#)
- 9.19. [TODO \[format-all.el\] コード整形](#)

[9.20. TODO \[emr.el\] リファクタリング](#)

[9.21. DONE \[hideshowvis.el\] 関数の表示 / 非表示 \[OBSOLETE\]](#)

## [10. Org Mode](#)

[10.1. 基本設定](#)

[10.2. contribution を使う](#)

[10.3. iCal との連携](#)

[10.4. スピードコマンド](#)

[10.5. face 関連](#)

[10.6. TODO キーワードのカスタマイズ](#)

[10.7. ImageMagick を使って様々な画像をインライン表示する](#)

[10.8. README を常に org-mode で開く](#)

[10.9. ソースブロック等の大文字記述を一括で小文字に変える](#)

[10.10. イベント通知](#)

[10.11. org-mode でアラーム管理](#)

[10.12. \[org-capture\] 高速にメモを取る](#)

[10.13. \[org-agenda\] タスク / 予定管理](#)

[10.14. \[orgbox.el\] スケジュール追加のわかりやすい入力](#)

[10.15. \[appt.el\] アラーム設定](#)

[10.16. \[org-refile\] org ツリーの高速移動](#)

[10.17. \[org-babel\] Org バッファでソースコードを扱う](#)

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

[10.19. \[org-tree-slide\] Org Mode でプレゼンテーション](#)

[10.20. \[org-tree-slide\] クロックインとアウトを自動化する](#)

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

[10.22. \[org-tree-slide\] ヘッドラインをリッチにする](#)

[10.23. \[calfw-org.el\] calfw に org の予定を表示する](#)

[10.24. \[org-odt\] ODT 形式に出力](#)

[10.25. \[ox-twbs\] Twitter Bootstrap 互換の HTML 出力](#)

[10.26. \[org-crypt\] ツリーを暗号化する](#)

[10.27. \[org-mac-link\] 外部アプリから情報を取る](#)

[10.28. \[org-download\] org バッファに D&D でファイルを保存](#)

[10.29. \[org-grep\] org ファイルを grep する](#)

[10.30. \[ox-reveal\] ナイスな HTML5 プレゼンテーション出力](#)

[10.31. \[org-dashboard\] 進捗をプログレスバーで確認](#)

[10.32. \[org-clock-today\] 今日の総作業時間をモードラインに表示](#)

[10.33. \[org-random-todo\] ランダムにタスクを選ぶ](#)

[10.34. \[ox-hugo\] Org ファイルから Hogo の記事をエクスポートする](#)

[10.35. \[ox-html\] HTML 見出しへのリンクを固定する](#)

[10.36. TODO \[org-tree-slide\] BEGIN\\_SRC と END\\_SRC を消して背景色を変える](#)

[10.37. TODO \[org-fstree\] ディレクトリ構造を読み取る](#)

[10.38. TODO \[ox\] 出力形式の拡張](#)

[10.39. TODO Parers3.app からリンクを取得する](#)

[10.40. TODO Papers3.app のリンクを開けるようにする](#)

- [10.41. TODO \[org-attach\] 外部ファイルを紐付ける](#)
- [10.42. TODO \[org-recent-headings\] 訪問したツリーを記録し簡単に再訪問可能にする](#)
- [10.43. TODO \[orgnav\] ツリーの検索インタフェース](#)
- [10.44. TODO \[toc-org\] 目次の挿入](#)
- [10.45. TODO \[org-review.el\] レビューフローのサポート](#)
- [10.46. REV2 \[org-screenshot\] スクリーンショットを貼り付ける](#)
- [10.47. DONE \[org-autolist\] ブリッツの入力を簡単に \[OBSOLETE\]](#)
- [10.48. DONE \[MobileOrg\] iOS との連携 \[OBSOLETE\]](#)
- [10.49. DONE \[org-export-generic\] エクスポート機能を拡張する \[OBSOLETE\]](#)
- [10.50. DONE org-mode の latex エクスポート関数をオーバーライド \[OBSOLETE\]](#)
- 11. フレーム / ウィンドウ制御
  - [11.1. 起動時の設定](#)
  - [11.2. 複数フレーム対応](#)
  - [11.3. \[moom.el\] キーボードでフレームの場所を移す](#)
    - [11.3.1. TODO 最大化時にモードラインを消す](#)
  - [11.4. \[winner.el\] ウィンドウ構成の履歴を辿る](#)
  - [11.5. TODO \[shackle.el\] ポップアップウィンドウの制御 \[ONGOING\]](#)
    - [11.5.1. checkdoc.el 用の追加設定](#)
  - [11.6. \[e2wm.el\] 二画面表示 \[OBSOLETE\]](#)
  - [11.7. \[tabbar-ruler\] バッファをタブ切り替え可能に \[OBSOLETE\]](#)
  - [11.8. \[elscreen.el\] Emacs バッファをタブ化 \[OBSOLETE\]](#)
  - [11.9. \[popwin.el\] ポップアップウィンドウの制御 \[OBSOLETE\]](#)
- 12. フォント / 配色関連
  - [12.1. 正規表現を見やすくする](#)
  - [12.2. 設定ファイルを見やすくする](#)
  - [12.3. カーソル行に色をつける](#)
  - [12.4. カーソルを点滅させない](#)
  - [12.5. カーソル位置のフォントを確認](#)
  - [12.6. フォント設定](#)
    - [12.6.1. フォントのインストール方法](#)
    - [12.6.2. フォントチェック用コード](#)
  - [12.7. 行間を制御する](#)
  - [12.8. パッチをカラフルに表示する](#)
  - [12.9. 背景を黒系色にする](#)
  - [12.10. 日中と夜中でテーマを切り替える](#)
  - [12.11. 時間帯を指定して起動時にテーマを切り替える](#)
  - [12.12. \[rainbow-mode.el\] 配色のリアルタイム確認](#)
    - [12.12.1. 色一覧](#)
  - [12.13. \[edit-color-stamp\] Qt 経由でカラーピッカーを使う](#)
  - [12.14. TODO \[volatile-highlights\] コピペした領域を強調](#)
- 13. ユーティリティ関数
  - [13.1. \[pomodoro.el\] ポモドーロの実践](#)
  - [13.2. \[pomodoro.el\] 続・ポモドーロの実践](#)



[13.3. \[google-this.el\] 単語をグーグル検索](#)  
[13.4. \[lingr.el\] チャットルームに自動参加](#)  
[13.5. \[multi-term.el\] ターミナル](#)  
[13.6. \[osx-lib.el\] OSX 用ユーティリティ \[MACOS\]](#)  
[13.7. iterm2 を Emacs から呼び出したい \[MACOS\]](#)  
[13.8. カレントバッファのあるディレクトリをターミナルで表示](#)  
[13.9. \[gif-screencast.el\] ユーザアクションのたびにスクショを取る](#)  
[13.10. \[utility.el\] 自作してテスト中の便利関数群](#)  
[13.11. \[manage-minor-mode.el\] マイナーモードの視覚的な管理](#)  
[13.12. TODO \[password-store.el\] パスワード管理](#)  
[13.13. TODO \[network-watch.el\] ネットワークインターフェイスの状態を監視](#)  
[14. おわりに](#)

## 1. はじめに

- [init.org](#) から [init.el](#) , [init.pdf](#) , [init.odt](#) 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 .
- [init.el](#) 本体は , [github](#) に公開しています .
- コピペだけで動かなかった場合は , wiki の [コメント欄](#) にご意見をどうぞ .
- `obsolete` タグのある設定は , あくまで個人的に不使用になったものです .

## 2. 起動設定

基本的な設定群です . 一部の関数は記述しておかないと動かない可能性があります .

普通は `init.el` に様々な設定を Emacs に読ませますが , 私は諸事情から次のようにブートシーケンスを制御しています .

1. `.emacs` の読み込み
2. `init-env.el` の読み込み
3. `init.el` の読み込み

`.emacs` は , 基本的に `init-env.el` を読み込むだけです . ただ , ちょっとしたコードを試したい時は , `.emacs` に書いてしまいます . 気に入ったら , `init.el` に移します .



init-env.el には、パスの設定、起動時間計測のためのフラグ群、パッケージ管理用の補助関数を設定しています。そして、init-env.el の最後に init.el を require しています。

これ以外に、バッチモードでバイトコンパイルするための init-eval.el も使います。このファイルには、バイトコンパイル時だけ必要なパッケージを列挙しています。

- [emacs.d/.emacs at master · takaxp/emacs.d](#)
- [emacs.d/init-env.el at master · takaxp/emacs.d](#)
- [emacs.d/init-eval.el at master · takaxp/emacs.d](#)
- [emacs.d/init-ad.el at master · takaxp/emacs.d](#)

## 2.1. init.el のヘッダ

```
;; -*- lexical-binding: t -*-  
;; Configurations for Emacs  
;;  
;; Takaaki ISHIKAWA <takaxp@ieee.org>  
;; see also https://takaxp.github.io/init.html
```

## 2.2. init ファイルの読み込み時間計測

次の関数を init.el の最初に記載して、オプション付きで after-init-hook に追加します。すると、init ファイルの読み込み時間を計測できます (GUI のフレーム生成を含まない)。別途設定する emacs-init-time の派生関数とは違い、after-init-hook で実行される処理の時間もわかります。また、ほかの処理も微妙に入るので、あくまで目安です。

```
(defconst my-before-load-init-time (current-time))  
(defun my-load-init-time ()  
  "Loading time of user init files including time for `after-init-hook'."  
  (let ((time1 (float-time  
                (time-subtract after-init-time my-before-load-init-time)))  
        (time2 (float-time  
                (time-subtract (current-time) my-before-load-init-time))))  
    (message (concat "Loading init files: %.0f [msec], "  
                    "of which %.f [msec] for `after-init-hook'.")  
             (* 1000 time1) (* 1000 (- time2 time1))))  
  (add-hook 'after-init-hook #'my-load-init-time t))
```

after-init-hook の値をチェックして、my-load-init-time が最後に配置されていることに留意します。以下は、after-init-hook の値の例です。

```
(session-initialize recentf-mode my-emacs-init-time my-load-init-time)
```

### 2.2.1. Emacs 起動時の呼び出し順

Emacs は、以下の順番で起動します。

1. Set before-init-time = (current-time) and Set after-init-time = nil
2. Initializes the window
3. Run before-init-hook
4. Set my-before-load-init-time = (current-time) => 独自変数で計測
5. Load user's init files
6. Set after-init-time = (current-time)
7. Run after-init-hook
8. Run emacs-startup-hook

`emacs-init-time` は、GUI 生成と、GUI 関連 el の読み込み、`before-init-hook` で実行する関数の実行時間、およびユーザ定義の `init` ファイル群の読み込み時間の合計値となる。

すでに `after-init-time` で時間計測が完了しているので、その後に続く `after-init-hook` と `emacs-startup-hook` に登録された関数の実行時間は `emacs-init-time` の値に含まれない。

より端的に言えば、`emacs-init-time = before-init-time - after-init-time` である。

一方、`my-before-load-init-time` と `after-init-time` を使って計算する算出する起動時間には、GUI 生成と GUI 関連 el の読み込み、`before-init-hook` にある関数の実行が含まれていない。

see also [Startup Summary - GNU Emacs Lisp Reference Manual](#)

## 2.3. 起動時間の計測

M-x `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，手を加えて表示を変えます．元ネタは[すぎやーんメ](http://lisperblog.blogspot.com/2008/12/blog-post_18.html) [王](#)からです．感謝．

```
(defun my-emacs-init-time ()
  "Emacs booting time in msec."
  (message "Emacs booting time: %.0f [msec] = `emacs-init-time'."
    (* 1000
      (float-time (time-subtract
                    after-init-time
                    before-init-time))))
  (add-hook 'after-init-hook #'my-emacs-init-time))
```

ついでに，`%.1f` で出力されるいつもの `emacs-init-time` をハック．

```
(defun ad:emacs-init-time ()
  "Return a string giving the duration of the Emacs initialization."
  (interactive)
  (let ((str
        (format "%.3f seconds"
          (float-time
            (time-subtract after-init-time before-init-time)))))
    (if (called-interactively-p 'interactive)
        (message "%s" str)
        str)))
(advice-add 'emacs-init-time :override #'ad:emacs-init-time)
```

## 2.4. GC サイズの最適化

起動時に発生するガベージコレクションを防ぎます．起動後に使用しているメモリサイズを超えていれば良さ気．`garbage-collection-messages` を設定しておくで，ガベージコレクションが生じる時にメッセージが出るようになります．

```
(setq gc-cons-threshold 134217728) ;; 128MB
(setq garbage-collection-messages t)
```

## 2.5. `cl` を使う

- [http://lisperblog.blogspot.com/2008/12/blog-post\\_18.html](http://lisperblog.blogspot.com/2008/12/blog-post_18.html)

一般的には次のように記載すれば OK です．

```
(eval-when-compile
  (require 'cl-lib nil t))
```

しかし，いくつかのパッケージは，バイトパイル時の警告を回避するために，`cl-lib` と同様に `eval-when-compile` の中に記述する必要が出てきます．`init.el` の成長に伴

い，対象パッケージが増えたので，別ファイル([init-eval.el](#))に括り出しました．バイトコンパイルするときだけ，`init-eval.el` を読み込みます．

例えば，`init.el` のコンパイルを次のようにします．

```
/Applications/Emacs.app/Contents/MacOS/Emacs -l ~/.emacs -l
~/Dropbox/emacs.d/config/init-eval.el -batch -f batch-byte-compile-if-not-done
~/Dropbox/emacs.d/config/init.el
```

これなら `init.el` に `eval-when-compile` を書くのと同じに見えますが，そうしてしまうと，バイトコンパイルしない状態の `init.el` を読み込むと，`eval-when-compile` の中身がすべて実行されてしまい，起動が重くなってしまいます．

そもそも `init.org` を編集対象にしている，`init.el` を手動でバイトコンパイルすることはないので，私のフローには合っています．

```
(eval-when-compile
  (require 'init-eval nil t))
```

## 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ういと思います．

- <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))
(setq ad-redefinition-action 'accept)
```

## 2.7. エラー表示の抑制

普段使いでは要らないので抑制します．

```
(setq debug-on-error nil)
```

## 2.8. バッファの保存を静かにする

ビルトインの `file.el` にある変数を使うと，バッファ保存時に表示されるメッセージが減り "Saving file ..." がメッセージバッファに記録されなくなります．バッファ保存時以外で，メッセージ出力を抑制したい場合は，`shut-up.el` が便利です．

```
(setq save-silently t) ;; No need shut-up.el for saving files.
```

## 2.9. [my-load-package-p] 設定の読み込みフラグを確認する

autoload-if-found のサポート関数です .

my-loading-packages 変数に , 設定を無視するパッケージを記述しておく , autoload-if-found 内でそれらを読み込まないようにします . 読み込まなかったパッケージは , Messages バッファにそのことを報告します . autoload-if-found は本来の実装ではなく , my-loading-packages を参照するように手を加えています .

```
(defun my-load-package-p (file)
  (let ((enabled t))
    (when (boundp 'my-loading-packages)
      (dolist (package my-loading-packages)
        (let ((name (car package))
              (flag (cdr package)))
          (when (and (stringp name)
                    (equal file name)
                    (not flag))
            (setq enabled nil)
            (message "--- '%s' was NOT loaded intentionally" name))))))
    enabled))
```

## 2.10. [autoload-if-found] 関数をリストで渡す autoload 設定

この autoload-if-found の初出は , [dot.emacs](#) です .

autoload-if-found と with-eval-after-load の組み合わせが基本です . 最初に when 判定をすることで , パッケージが未インストールの状態に各種設定をスキップするので安全です .

```
(defvar my-skip-autoload-file-check t)
(defun autoload-if-found (functions file &optional docstring interactive type)
  "set autoload iff. FILE has found."
  (when (or my-skip-autoload-file-check
            (and (my-load-package-p file)
                 (locate-library file)))
    (dolist (f functions)
      (autoload f file docstring interactive type))
    t))
```

この init.el は , 基本的にこの autoload-if-found に依存しています . 遅延ロードに基づく高速化において必須の関数ですが , 特にこだわらない人は use-package.el に依存するのがオススメです . 以下 , autoload-if-found を使った設定のテンプレートです .

```
(when (autoload-if-found
      '(f1 f2) ;; 関数リスト . 要素の関数を呼ぶ時に , パッケージが読み込まれる
      "package-name" nil t) ;; 第 4 引数まで指定すると , M-x で補完対象になる .

  ;; (0) バイトコンパイル時に警告がでたら , 対応するパッケージを記載
```

```
;; 現在は、バイトコンパイル時だけ、必要なパッケージの require を全て記載した
;; init-eval.el を読み込む方式に変更．ここでは eval-when-compile を使わない．
;; (eval-when-compile
;;   (require 'package-name nil t))

;; (1) パッケージが存在すれば、Emacs 起動時に読み込む設定群
(push '("\.hoge$" . package-name-mode) auto-mode-alist)
(add-hook 'after-init-hook #'f1)
(global-set-key (kbd "r") 'f2)
(autoload-if-found '(hoge1 hoge2) "hoge" nil t)

;; (2) 遅延読み込みする設定群
;; autoload-if-found で遅延ロードする関数
(with-eval-after-load "package-name"
  (setq v1 t)
  (setq v2 nil)
  (define-key package-name-map (kbd "q") 'f1))

;; 関連するパッケージの遅延ロード
(with-eval-after-load "package-name2"
  (setq v3 t)
  (setq v4 nil))

;; 起動時に実行せず、最初に任意の発行するコマンドまで遅延ロードさせる
;; see https://github.com/takaxp/postpone
(with-eval-after-load "postpone"
  (setq v5 nil)))
```

このカスタマイズされた `autoload-if-found` は引数にリストのみを受け付けます．欠点は、リスト内の全ての関数に個別の `docstring` を設定できないため、ヘルプ画面で何の関数なのかの情報があまり表示されないことです．ただ、`require` でパッケージを読み込んでしまえば、本来の `docstring` の内容が正しく反映されるので、すべての情報を確認できるようになります．また、`interactive` を `t` にすれば、`M-x` で当該関数がリストアップされるようになります．したがって、`(docstring interactive) = (nil t)` をデフォルトで指定すれば OK です．`type` には、`macro` や `keymap` を指定して、関数以外の情報をオートロード対象にします．

コンパイル時に `Warning: the function 'google-this' is not known to be defined.` のようなメッセージが出る時には、`autoload-if-found` の `functions` に当該関数を追加すると、警告されなくなります．`declare-function` を追記せずに済みます．

`locate-library` は、高速化を求める状況では、意外と高コストです．なので、それらを評価しないためのフラグ `skip-file-checking` を追加しました．各パッケージの設定は、基本的に `with-eval-after-load` に括られていることを前提にすれば、起動が停止する自体は避けられるので、悪くない選択です．もし変なことになってしまっても、`skip-file-checking` を `nil` にすれば、従来のようにパッケージの存在を確認し、安全に起動できます．

## 2.11. [future-time-p] 指定時刻が今日の未来の時刻かを判定

今日の時刻に限定して，指定時刻が過去の時間かどうかを判定します．run-at-time が想定通りに動かず，起動時に run-at-time に登録した関数が走ってしまうので，この判定が non-nil の時だけタイマー登録します．time を HH:MM の書式で与えます．指定時刻と現在時刻が同じ場合は，過去とみなします．

```
(defun future-time-p (time)
  "Return non-nil if provided TIME formed of \"10:00\" is the future time."
  (not (time-less-p
        (apply 'encode-time
                (let ((t1 (decode-time))
                    (t2 (parse-time-string time)))
                  (setf (nth 0 t1) 0)
                  (setf (nth 1 t1) (nth 1 t2))
                  (setf (nth 2 t1) (nth 2 t2))
                  t1))
        (current-time))))
;; (when (future-time-p "10:00") (run-at-time...))
```

以下は古い実装．指定時刻と現在時刻が同じ場合は，タイマー登録の動作確認が楽だったこともあり，未来とみなしていた．

```
(defun passed-clock-p (target)
  (let ((hour nil)
        (min nil)
        (current-hour nil)
        (current-min nil))
    (when (string-match "\\([0-2]?[0-9]\\):\\([0-5][0-9]\\)" target)
      (setq hour (substring target (match-beginning 1) (match-end 1)))
      (setq min (substring target (match-beginning 2) (match-end 2)))
      (setq current-hour (format-time-string "%H" (current-time)))
      (setq current-min (format-time-string "%M" (current-time)))
      (< (+ (* (string-to-number hour) 60)
            (string-to-number min))
         (+ (* (string-to-number current-hour) 60)
            (string-to-number current-min)))))
```

## 2.12. [input-focus-p] フォーカス判定

フォーカスが当たっているのかを判定するための関数です．

```
(defvar window-focus-p t)
(with-eval-after-load "postpone"
  (defun window-focus-p ()
    (if window-focus-p t nil))
  (add-hook 'focus-in-hook (lambda () (setq window-focus-p t)))
  (add-hook 'focus-out-hook (lambda () (setq window-focus-p nil)))))
```

## 2.13. [postpone.el] Emacs 起動してから使い始めるタイミングで実行する

- [takaxp/postpone: Call functions just one time at your first action in Emacs](#)



helm-config に紐付けていた遅延呼び出し関数および設定を，自作の postpone.el に紐付けました．helm-config に紐付けていた時は，M-x を初めて使う時に複数の設定が読み込まれましたが，今回の設定では，Emacs 起動後の最初のアクション時に読み込まれます（一部のコマンドを除く）．これには M-x も含まれます．この init.el に記載された遅延読み込みは約 660[ms]にも及びます．通常起動では約 400[ms]を要している（コンソール起動の場合は，約 100[ms]）ので，もし遅延読み込みしなければ，Emacs の起動が 1 秒を超えてしまうことになります．

起動後，即座に Emacs 終了するシーンでは，postpone.el に紐付けた設定が活性化すると無駄になります．それを回避するために，this-command の値を確認して，条件に合う場合だけ postpone.el が読み込まれる（結果的に紐付いている全ての設定が活性化する）ようにしています．

```
(if (not (locate-library "postpone"))
    (message "postpone.el is NOT installed.")
    (autoload 'postpone-kicker "postpone" nil t)
    (defun my-postpone-kicker ()
      (interactive)
      (unless (memq this-command ;; specify commands for exclusion
                    '(self-insert-command
                      save-buffers-kill-terminal
                      exit-minibuffer))
        (message "Activating postponed packages...")
        (let ((t1 (current-time)))
          (postpone-kicker 'my-postpone-kicker)
          (setq postpone-init-time (float-time
                                     (time-subtract (current-time) t1))))
        (message "Activating postponed packages...done")))
    (add-hook 'pre-command-hook #'my-postpone-kicker))
```

## 2.14. [shut-up.el] Messages 出力を封じる（制御する）

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．非常に強力です．

- [cask/shut-up: Emacs, shut up would you!](#)

```
(defvar shutup-p nil)
(with-eval-after-load "postpone"
  (unless noninteractive
    (postpone-message "shut-up")))
(setq shutup-p (when (require 'shut-up nil t) t))
(setq message-log-max 5000) ;; メッセージバッファの長さ
```

### 2.14.1. with-suppressed-message

[OBSOLETE]

以下はそれ以前のアプローチ．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．

- [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](#)

```
(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)))
```

## 2.15. [exec-path-from-shell.el] PATH 設定をシェルから継承する [OBSOLETE]

- [purcell/exec-path-from-shell: Make Emacs use the \\$PATH set up by the user's shell](#)

外部プログラムのサポートを得て動くパッケージは，設定の過程で「プログラムが見つからない」と怒られることがしばしばあります．exec-path-from-shell は，シェルに設定した PATH の情報を継承して exec-path や PATH を設定してくれます．私は起動時に環境を切り替えることがあるので使ってませんが，使われている方は多いようです．

```
(when (and (require 'exec-path-from-shell nil t)
  (memq window-system '(mac ns)))
  (exec-path-from-shell-initialize))
```

## 2.16. [eval-after-autoload-if-found] 遅延読み込み

[OBSOLETE]

現在は使っていません．非推奨です．

with-eval-after-load とのペアでマクロ化したバージョンです．ただ，生成されるバイトコードに問題がありそうなので，最近は使っています．長らくお世話になっております．になりましたが，現在は，autoload-if-found と with-eval-after-load の組み合わせを使っています．

Twitter でぼやいていたら [@cvmatt](#) 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のが欠点です．

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"))))
```

これをスッキリさせるために `eval-after-autoload-if-found` を導入します．記述がシンプルになり，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"))))
```

さらに戻り値を判定して，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ために有効です．hook 系の登録も同様です．

```
(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 #'hoge)
;;
;; ビルドインではない mode の auto-mode-alist 設定も必要ならここに記述
;; (push '("\.hoge$" . hoge-mode) auto-mode-alist)
)
```

なお，第四引数 (`functions file docstring interactive`) まで指定すれば，`M-x` の呼び出し候補に `functions` で指定した関数が補完表示されます．

### 2.16.1. 関数版

関数版にリスト `my-loading-packages` を追加しました．このリストに事前に 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おけば，パッケージの ON/OFF を簡単に制御できます．

ただ問題点として，最後の引数に入れる関数がバイトコンパイル時に展開されないようで，出来上がったバイトコードが高速化に寄与しているのか不明です．簡易的な実験ではだいぶ差があるようで，最近では `autoload-if-found`，`with-eval-after-load`，

eval-when-compile の組み合わせで設定を書いて、遅延ロードと高速読み込みを実現しています。my-loading-packages は autoload-if-found に組み込みました。

```
(setq my-loading-packages ;; 追加されていない場合は標準で読み込む
      '(("web-mode" . nil) ;; 読み込まない
        ("org" . t))) ;; 読み込む
```

<https://gist.github.com/3513287>

```
;; https://github.com/zk-phi/setup
;; (when (require 'setup nil t)
;;   (setup-initialize))
(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))
    (message "--- %s" file)
    (when (and (boundp 'my-loading-packages) my-loading-packages)
      (dolist (package my-loading-packages)
        (let ((name (car package))
              (flag (cdr package)))
          (when (and (stringp name) (equal file name))
            (unless flag
              (setq enabled nil)
              (message "--- A setting for `%s' was NOT loaded explicitly"
                       name))))))
    ;; if disabled then return nil.
    (when (and enabled (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)))
```

## 2.16.2. マクロ版

以下はマクロ版です。引数の渡し方が関数と少し違うので要注意です。

<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))
            after-body)))
```

```
(if (listp functions)
    functions
    (list functions)))
,@(when after-body
    `((eval-after-load file '(progn ,@after-body))))
t)))
```

## 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` が `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") ;; will take 20-30[ms]
(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)
```

### 3.2. 日本語入力

NSビルド用の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](#) .
- Emacs25.2 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))

(when (eq system-type 'gnu/linux)
  (global-set-key (kbd "<hiragana-katakana>") 'toggle-input-method)
  (push "/usr/share/emacs/site-lisp/anthy" load-path)
  (push "/usr/share/emacs/site-lisp/emacs-mozc" load-path)
  (set-language-environment "Japanese"))

(if (require 'mozc nil t)
    (progn
      (setq default-input-method "japanese-mozc")
      (custom-set-variables
        '(mozc-candidate-style 'overlay)))

    (when (require 'anthy nil t) ;; sudo yum install emacs-anthy-el
      ;; if get error
      (load-file "/usr/share/emacs/site-lisp/anthy/leim-list.el")
      (setq default-input-method 'japanese-anthy)))
```

### 3.3. 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 (memq window-system '(mac ns))
  (when (boundp 'ns-command-modifier)
    (setq ns-command-modifier 'meta))
  (when (boundp 'ns-alternate-modifier)
    (setq ns-alternate-modifier 'super))
  (when (boundp 'ns-pop-up-frames)
    (setq ns-pop-up-frames nil))
  (global-set-key (kbd "M-v") 'yank)
  (global-set-key [ns-drag-file] 'ns-find-file))
(global-set-key [delete] 'delete-char)
(global-set-key [kp-delete] 'delete-char)
```

### 3.4. ナローイングするか

ナローイングを有効にします . 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 , デフォルトでは無効になっています .

Org Mode でナローイングを使う場合は , 特に設定しなくても OK です .

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

`fancy-narrow` を使うと，通常のアローイングではバッファ上で表示しなくなる領域を目立たないように残すことができます．

```
(autoload-if-found
 ' (fancy-narrow-to-region
    fancy-widen
    org-fancy-narrow-to-block
    org-fancy-narrow-to-element
    org-fancy-narrow-to-subtree)
 "fancy-narrow" nil t)
```

### 3.5. バッファの終わりで *newline* を禁止する

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

### 3.6. 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)
```

### 3.7. 長い文章を右端で常に折り返す

```
(with-eval-after-load "postpone"
 (setq truncate-lines nil)
 (setq truncate-partial-width-windows nil))
```

### 3.8. 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)
```

### 3.9. *compilation buffer* でのデータ表示で自動スクロールする

`nil` のままだと，出力が続いてもスクロールされないので，自動的にスクロールされるように設定．

```
(setq compilation-scroll-output t)
```

### 3.10. *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` もしくは `yes-or-no-p` を指定するだけです．

```
(setq confirm-kill-emacs 'yes-or-no-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す．



```
;; 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))))))

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

### 3.11. 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。

#### 3.11.1. [cask-mode.el] モード設定

- [Wilfred/cask-mode: Major mode for editing Cask files](#)

```
(when (autoload-if-found '(cask-mode) "cask-mode" nil t)
  (push '("/Cask/" . cask-mode) auto-mode-alist))
```

#### 3.11.2. Cask のセットアップ

以下は自分用のメモです。

1. curl -fsSkL <https://raw.githubusercontent.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

### 3.11.3. 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]まで削減できました．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/<emacs-version>` 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/<emacs-version>` 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．`load-path-setter` は独自関数です（普通に `add-to-list` で追加するのと同じです）

```
(defconst cask-package-dir
  (format "~/.emacs.d/.cask/package/%s" emacs-version))
(if t
  (load-path-setter `(&,(cask-package-dir) 'load-path)
  (when (or (require 'cask "~/.cask/cask.el" t)
            (require 'cask "/usr/local/opt/cask/cask.el" t)) ;; Homebrew
    (when (fboundp 'cask-initialize) (cask-initialize)) ;; 800[ms]
    (when (require 'pallet nil t)
      (when (fboundp 'pallet-mode) (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．アルファリリースなどに対応するときなど，少し調整が必要です．

```
#!/bin/sh

LOADPATH=$HOME/.emacs.d/lisp
SOURCEPATH=$HOME/Dropbox/emacs.d
EVALWCOMPILE="$SOURCEPATH/config/init-eval.el"

while getopts t:hd opt
do
case ${opt} in
t)

```

```

TARGET=${OPTARG};;
d)
echo "--- Remove byte compiled files."
rm -rf $LOADPATH/*.elc
rm -rf $SOURCEPATH/config/*.elc
rm -rf ${HOME}/devel/git/org-mode/lisp/*.elc
exit 1;;
h)
echo "TARGET LIST:";
echo "  actex";
echo "  org-sync";
echo "  yatex";
echo "e.g."
echo "> $0 -t anything"
exit 1;;
\?)
exit 1;;
esac
exit 0
done

if [ ! -f $HOME/.zsh.local ]; then
    echo "~/.zsh.local does NOT exist."
    exit 0
fi

if [ ! -d $HOME/.emacs.d ]; then
    echo "~/.emacs.d does NOT exist."
    exit 0
fi

EMACS=`which emacs`
if [ $HOSTTYPE = "intel-mac" ]; then
    TARGETV=head # 25.1, 25.2, 25.3, 26.1, head
    EMACS="/Users/taka/devel/emacs/bin/$TARGETV/Emacs.app/Contents/MacOS/Emacs"
    if [ $TARGETV = "head" ]; then
        EMACS="/Applications/Emacs.app/Contents/MacOS/Emacs"
    fi
fi
# $EMACS -l ~/.emacs --script ~/Dropbox/config/tangle.el
$EMACS -q --script ~/Dropbox/config/tangle.el

echo "--- Starting batch-byte-compiles with GNU Emacs ($TARGETV).\"

COMMAND=\"$EMACS -l ~/.emacs -l $EVALWCOMPILE -batch -f batch-byte-compile-if-not-done\"

rm -rf $LOADPATH/*.elc
rm -rf $SOURCEPATH/config/*.elc

cp -rf $SOURCEPATH/*.el $LOADPATH
cp -rf $SOURCEPATH/config/init.el $LOADPATH
cp -rf $SOURCEPATH/config/utility.el $LOADPATH

$COMMAND $LOADPATH/init.el
$COMMAND $LOADPATH/utility.el
#$COMMAND $SOURCEPATH/config/init.el $SOURCEPATH/config/utility.el $SOURCEPATH/*.el

# raw
# /Applications/Emacs.app/Contents/MacOS/Emacs -l ~/.emacs -l

```

```
~/Dropbox/emacs.d/config/init-eval.el -batch -f batch-byte-compile-if-not-done

# mv $SOURCEPATH/config/init.elc $HOME/.emacs.d
# mv $SOURCEPATH/config/utility.elc $HOME/.emacs.d
# mv $SOURCEPATH/*.elc $HOME/.emacs.d

echo "--- done."
```

### 3.11.4. [paradox.el] パッケージ選択画面の改善

パッケージインストール用のバッファが多機能になります。スターが表示されたり，ミニバッファには様々な情報が表示されるようになります。基本の操作系は同じで，拡張部分は `h` を押すとミニバッファにデイスパッチャが表示されます。

```
(when (autoload-if-found
      '(paradox-list-packages my-list-packages my-setup-cask)
      "paradox" nil t)

  (defvar my-default-load-path nil)
  (defun my-list-packages ()
    "Call \\[paradox-list-packages] if available instead of \\[list-packages]."
    (interactive)
    (setq my-default-load-path load-path)
    (my-setup-cask)
    (if (fboundp 'paradox-list-packages)
        (paradox-list-packages nil)
        (list-packages nil)))

  (defun my-reset-load-path ()
    "Revert `load-path' to `my-default-load-path'."
    (shell-command-to-string "~/Dropbox/emacs.d/bin/update-cask.sh link")
    (setq load-path my-default-load-path)
    (message "--- Reverted to the original `load-path'."))

  (declare-function advice:paradox-quit-and-close "init" (kill))

  (with-eval-after-load "paradox"
    (defun my-setup-cask ()
      "Override `load-path' to use cask."
      (when (or (require 'cask "/usr/local/opt/cask/cask.el" t)
                (require 'cask "~/cask/cask.el" t))
        (setq load-path (cask-load-path (cask-initialize))))))

    (defun advice:paradox-quit-and-close (_kill)
      (my-reset-load-path))
    (advice-add 'paradox-quit-and-close :after #'advice:paradox-quit-and-close)

    (custom-set-variables
      '(paradox-github-token t))
    (when (fboundp 'paradox-enable)
      (paradox-enable))))
```

### 3.12. インデント

オープンソース等で他の人のコードを修正する場合は，以下のような設定は良くないかも

しれません．例えば差分を取ると見た目は変わらないのに，タブとスペースの違いから差分ありと判定されてしまい，意図しない編集履歴が残ることがあります．ただこの問題は，修正対象のファイルが限定されているならば，`M-x tabify` や `M-x untabify` で回避できそうです．

一方，`org-mode` のソースブロックは半角スペース統一されているため，この設定のほうが都合が良いです．

```
(setq-default tab-width 2)
(setq-default indent-tabs-mode nil)
(setq indent-line-function 'insert-tab)

;; (add-hook 'emacs-lisp-mode-hook
;;           (lambda ()
;;             (setq indent-tabs-mode t)
;;             (setq tab-width 8)
;;             (setq indent-line-function 'lisp-indent-line)))
```

### 3.13. 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)
```

### 3.14. 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` が即座に反映されます．適宜バックアップツールと組み合わせないと，バッファが自動更新されてしまうので不安かもしれません．

`postpone` の後に呼んでいるのは，Emacs 起動時の単純な高速化対策です．

```
(with-eval-after-load "postpone"
  (unless noninteractive
    (postpone-message "global-auto-revert-mode")
    (global-auto-revert-mode 1)))
```

### 3.15. マウススクロールをピクセル単位にする

```
(with-eval-after-load "postpone"
  (when (fboundp 'pixel-scroll-mode)
    (pixel-scroll-mode 1))) ;; 26.1
```

### 3.16. `[aggressive-indent.el]` 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．`python-mode` では意図しないインデントになったりします．`web-mode` だと異常に重かったりします．

```
(when (autoload-if-found
      '(aggressive-indent-mode)
      "aggressive-indent" nil t)

      (dolist (hook
                ' ( ;; python-mode-hook
                  ;; nxml-mode-hook
                  ;; web-mode-hook
                  emacs-lisp-mode-hook
                  lisp-mode-hook perl-mode-hook c-mode-common-hook))

              (add-hook hook #'aggressive-indent-mode)))
```

### 3.17. [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。モードラインの表示が変わります。

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

### 3.18. [ws-butler.el] 不要なスペースを自動除去

行末のスペース等が不要に残るのを回避できます。バッファ保存時の自動処理です。

`postpone` の後に呼んでいるのは、Emacs 起動時の単純な高速化対策です。

```
(when (autoload-if-found
      '(ws-butler-mode ws-butler-global-mode)
      "ws-butler" nil t)

      (with-eval-after-load "ws-butler"
        (custom-set-variables
         '(ws-butler-global-exempt-modes
           (append '(org-mode empty-booting-mode)
                     ws-butler-global-exempt-modes))))

      (with-eval-after-load "postpone"
        (unless noninteractive
          (postpone-message "ws-butler-global-mode")
          (ws-butler-global-mode))))
```

### 3.19. [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。helm インタフェースを使う `helm-ag` もあります。

```
(when (and (executable-find "ag")
  (autoload-if-found
    ' (my-ag ag)
    "ag" nil t))

  (autoload-if-found ' (helm-ag) "helm-ag" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-M-f") 'my-ag))

  (with-eval-after-load "ag"
    (custom-set-variables
      ' (ag-highlight-search t)
      ' (ag-reuse-buffers t)          ;; nil: 別ウィンドウが開く
      ' (ag-reuse-window nil)) ;; 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 ()
      "Switch to search result."
      (interactive)
      (call-interactively 'ag)
      (switch-to-buffer-other-frame "*ag search*"))))
```

### 3.20. NS ビルド用設定

[MACOS]

インラインパッチの適用が前提の設定です . M-SPC/S-SPC で日本語 IME の ON/OFF ができるようになります . インラインパッチの情報はリンク先にあります .

- [emacs-25.2 をインラインパッチをあてて使う \( OSX \) - Qiita](#)
- [An inline-patch for Emacs-25.2](#)

```
(when (eq window-system 'ns)
  (global-set-key (kbd "M-SPC") 'my-ns-ime-toggle) ;; toggle-input-method
  (global-set-key (kbd "S-SPC") 'my-ns-ime-toggle) ;; toggle-input-method
  (declare-function my-ns-org-heading-auto-ascii "init" nil)
  (declare-function my-ns-ime-restore "init" nil))

(with-eval-after-load "postpone"
  (when (and (eq window-system 'ns)
    (fboundp 'mac-get-current-input-source))

    (defun my-ns-ime-toggle ()
      "Toggle IME."
      (interactive)
      (if (my-ime-active-p) (my-ime-off) (my-ime-on)))

    (define-key isearch-mode-map (kbd "M-SPC") 'my-ns-ime-toggle)
    (define-key isearch-mode-map (kbd "S-SPC") 'my-ns-ime-toggle)

    (defun my-ns-org-heading-auto-ascii ()
      "IME off, when the cursor on org headings."
      (when (and window-focus-p
```



```

      (eq major-mode 'org-mode)
      (or (looking-at org-heading-regexp)
          (equal (buffer-name) org-agenda-buffer-name)))
    (my-ime-off)))
(run-with-idle-timer 1 t #'my-ns-org-heading-auto-ascii)

(defun my-ns-ime-restore ()
  "Restore the last IME status."
  (if my-ime-last (my-ime-on) (my-ime-off)))
(add-hook 'focus-in-hook #'my-ns-ime-restore))

```

### 3.21. EMP ビルド用設定

[MACOS]

NS ビルド版で生じた日本語入力時のチラつきを避けるために、EMP 版ビルドに(一時期)浮気しました。以下はその時に NS ビルドの振る舞いに近づけるためにがんばった設定です。詳細な情報は、リンク先の記事にあります。

- [Emacs 25.1/25.2 を EMP 版で快適に使う - Qiita](#)

```

(when (eq window-system 'mac)
  (global-set-key (kbd "M-SPC") 'mac-win-ime-toggle)
  (global-set-key (kbd "S-SPC") 'mac-win-ime-toggle)
  (declare-function mac-win-save-last-ime-status "init" nil)
  (declare-function ad:mac-auto-ascii-setup-input-source "init" nil)
  (declare-function mac-win-restore-ime "init" nil)
  (declare-function mac-win-restore-ime-target-commands "init" nil))

(with-eval-after-load "postpone"
  (when (eq window-system 'mac)
    (mac-auto-ascii-mode 1)

    (defvar mac-win-last-ime-status 'off) ;; {'off|'on}
    (defun mac-win-save-last-ime-status ()
      (setq mac-win-last-ime-status
        (if (string-match "\\.\\" (Roman\\|US\\) $" (mac-input-source))
            'off 'on)))
    (mac-win-save-last-ime-status) ;; 初期化

    (defun mac-win-restore-ime ()
      (when (and mac-auto-ascii-mode
                  (eq mac-win-last-ime-status 'on))
        (mac-select-input-source
         "com.google.inputmethod.Japanese.base")))

    (defun ad:mac-auto-ascii-setup-input-source (&optional _prompt)
      "Extension to store IME status"
      (mac-win-save-last-ime-status))
    (advice-add 'mac-auto-ascii-setup-input-source :before
      #'ad:mac-auto-ascii-setup-input-source)

    (defvar mac-win-target-commands
      '(find-file save-buffer other-window delete-window split-window))

    (defun mac-win-restore-ime-target-commands ()
      (when (and mac-auto-ascii-mode
                  (eq mac-win-last-ime-status 'on))

```

```

(mapc (lambda (command)
      (when (string-match
            (format "%s" command) (format "%s" this-command))
        (mac-select-input-source
         "com.google.inputmethod.Japanese.base")))
      mac-win-target-commands)))
(add-hook 'pre-command-hook #'mac-win-restore-ime-target-commands)

;; バッファリストを見るとき
(add-to-list 'mac-win-target-commands 'helm-buffers-list)
;; ChangeLog に行くとき
(add-to-list 'mac-win-target-commands 'add-change-log-entry-other-window)
;; 個人用の関数を使うとき
;; (add-to-list 'mac-win-target-commands 'my-)
;; 自分で作ったパッケージ群の関数を使うとき
(add-to-list 'mac-win-target-commands 'change-frame)
;; org-mode で締め切りを設定するとき
(add-to-list 'mac-win-target-commands 'org-deadline)
;; org-mode で締め切りを設定するとき
;; (add-to-list 'mac-win-target-commands 'org-capture)
;; query-replace で変換するとき
(add-to-list 'mac-win-target-commands 'query-replace)

;; ミニバッファ利用後に IME を戻す
;; M-x でのコマンド選択で IME を戻せる
;; これ移動先で q が効かないことがある
(add-hook 'minibuffer-setup-hook #'mac-win-save-last-ime-status)
(add-hook 'minibuffer-exit-hook #'mac-win-restore-ime)

;; タイトルバーの振る舞いを NS 版に合わせる
(setq frame-title-format (format (if (buffer-file-name) "%f" "%b")))

;; なおテーマを切り替えたら、face の設定をリロードしないと期待通りにならない
(when (require 'hl-line nil t)
  (custom-set-faces
   ;; 変換前入力時の文字列用 face
   `(mac-ts-converted-text
    (((background dark)) :underline "orange"
     :background ,(face-attribute 'hl-line :background))
    (t (:underline "orange"
      :background
      ,(face-attribute 'hl-line :background)))))
   ;; 変換対象の文字列用 face
   `(mac-ts-selected-converted-text
    (((background dark)) :underline "orange"
     :background ,(face-attribute 'hl-line :background))
    (t (:underline "orange"
      :background
      ,(face-attribute 'hl-line :background))))))

(when (fboundp 'mac-input-source)
  (run-with-idle-timer 3 t 'my-mac-keyboard-input-source))

;; あまりよいアプローチでは無い気がするけど、org-heading 上と agenda では
;; 1 秒アイドルすると、自動的に IME を OFF にする
(defun my-mac-win-org-heading-auto-ascii ()
  (when (and (eq major-mode 'org-mode)

```

```

        (or (looking-at org-heading-regexp)
            (equal (buffer-name) org-agenda-buffer-name)))
    (setq mac-win-last-ime-status 'off)
    (mac-auto-ascii-select-input-source)))
(when (fboundp 'mac-auto-ascii-select-input-source)
  (run-with-idle-timer 1 t 'my-mac-win-org-heading-auto-ascii))

;; EMP 版 Emacs の野良ビルド用独自設定群
;; IME toggle を Emacs 内で有効にする
(defun mac-win-ime-toggle ()
  (interactive)
  (when (fboundp 'mac-input-source)
    (mac-select-input-source
     (concat "com.google.inputmethod.Japanese"
             (if (string-match "\\..base$" (mac-input-source))
                 ".Roman" ".base")))))

;; isearch 中に IME を切り替えると, [I-Search] の表示が消える .
;; (define-key isearch-mode-map (kbd "M-SPC") 'mac-win-ime-toggle)
(define-key isearch-mode-map (kbd "S-SPC") 'mac-win-ime-toggle)

(when (boundp 'mac-win-ime-cursor-type)
  (setq mac-win-ime-cursor-type my-cursor-type-ime-on))
;; minibuffer では↑の背景色を無効にする
(when (fboundp 'mac-min--minibuffer-setup)
  (add-hook 'minibuffer-setup-hook #'mac-min--minibuffer-setup))
;; echo-area でも背景色を無効にする
(when (boundp 'mac-win-default-background-echo-area)
  (setq mac-win-default-background-echo-area t));; *-text の background を無視
;; デバッグ用
(when (boundp 'mac-win-debug-log)
  (setq mac-win-debug-log nil))
;; Testing...
(when (boundp 'mac-win-apply-org-heading-face)
  (setq mac-win-apply-org-heading-face 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ますが、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。

---

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-[
バッファ先頭・末尾	C-M-t/C-M-b

---

## 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．

```
(with-eval-after-load "postpone"
  (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 でもいいですが，ワンアクションで移動できるようが楽です．次のように双方向で使えるように設定しています．

```
(with-eval-after-load "postpone"
  (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

一行ず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ず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．

```
(with-eval-after-load "postpone"
  ;; 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))
```

#### 4.4. スクロールで表示を重複させる行数

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

#### 4.5. [smooth-scroll.el] 滑らかなスクロール

良い感じです．スススッとスクロールします．最初にスクロールする時にパッケージを読み込みます．

```
(when (autoload-if-found
      '(smooth-scroll-mode)
      "smooth-scroll" nil t)

  (with-eval-after-load "postpone"
    (smooth-scroll-mode t))

  (with-eval-after-load "smooth-scroll"
    (custom-set-variables
      '(smooth-scroll/vscroll-step-size 6)
      '(smooth-scroll/hscroll-step-size 6))))
```

#### 4.6. [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．cycle-buffer は古めですが，個人的には気に入っています．

なおビルトインに同様の機能を提供する bs.el があり，bs-cycle-next あるいは bs-cycle-previous でバッファを切り替えられます．

```
(when (autoload-if-found
      '(cycle-buffer cycle-buffer-backward)
      "cycle-buffer" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "M-]") 'cycle-buffer)
    (global-set-key (kbd "M-[") 'cycle-buffer-backward))

  (with-eval-after-load "cycle-buffer"
    (custom-set-variables
      '(cycle-buffer-allow-visible t)
      '(1cycle-buffer-show-length 12)
      '(cycle-buffer-show-format '("< %s >" . " %s")))))
```

## 4.7. [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。

org-mode との連携には、[org-bookmark-heading](#) があります。ただ、私は下記の設定だけでそれほど不自由していません。

```
(when (autoload-if-found
      ' (my-bm-toggle
        my-bm-next bm-buffer-save bm-buffer-restore bm-buffer-save-all
        bm-repository-save bm-repository-load bm-load-and-restore)
      "bm" nil t)

(with-eval-after-load "postpone"
  ;; ファイルオープン時にブックマークを復帰
  (add-hook 'find-file-hook #'bm-buffer-restore)
  (global-set-key (kbd "<f10>") 'my-bm-toggle)
  (global-set-key (kbd "<C-f10>") 'my-bm-next))

(with-eval-after-load "bm"
  (setq-default bm-buffer-persistence t)
  (setq bm-cycle-all-buffers t)
  ;; (setq bm-toggle-buffer-persistence 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-buffer-persistence t)
  (setq bm-persistent-face 'bm-face)
  (setq bm-repository-file
    (expand-file-name "~/Dropbox/emacs.d/.bm-repository"))
  (bm-repository-load)

(defun my-bm-toggle ()
  "bm-toggle with updating history"
  (interactive)
  (let ((bm (concat
              (buffer-name) "::<"
              (if (and (equal major-mode 'org-mode)
                      (not (org-before-first-heading-p)))
                  (nth 4 (org-heading-components))
                  (format "%s" (line-number-at-pos))))))
    (if (bm-bookmark-at (point))
        (bookmark-delete bm)
        (bookmark-set bm)))
    (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))))))
```

#### 4.8. [centered-cursor-mode.el] カーソル位置をバッファ中央に固定

isearch-mode の時だけ有効にしています .

```
(when (autoload-if-found
      '(centered-cursor-mode)
      "centered-cursor-mode" nil t)

  (with-eval-after-load "postpone"
    (add-hook 'isearch-mode-hook
              (lambda () (centered-cursor-mode 1)))
    (add-hook 'isearch-mode-end-hook
              (lambda () (centered-cursor-mode -1))))))
```

#### 4.9. [smart-mark] C-g 後に元の場所へカーソルを戻す

すぐわかる例は , C-x h で全選択して何もせず , C-g する場合です . 通常だとバッファの先頭にカーソルが置き去りにされますが , smart-mark を使うと , 全選択を実行した時の位置に自動的に戻してくれます .

postpone の後に呼んでいるのは , Emacs 起動時の単純な高速化対策です .

- [zhangkaiyulw/smart-mark: Emacs package to restore point after C-g when mark](https://github.com/zhangkaiyulw/smart-mark)

```
(with-eval-after-load "postpone"
  (when (require 'smart-mark nil t)
    (unless noninteractive
      (postpone-message "smart-mark-mode")
      (smart-mark-mode 1))))
```

#### 4.10. [syntax-subword.el] M-f で移動する位置をより密にする

- <https://bitbucket.org/jpkotta/syntax-subword>

```
(with-eval-after-load "postpone"
  (when (require 'syntax-subword nil t)
    (unless noninteractive
      (postpone-message "global-syntax-subword-mode")
      (global-syntax-subword-mode 1))))
```



## 4.11. [goto-chg.el] 編集箇所を簡単に辿る

編集結果を保持したまま編集箇所にカーソルを移すことができます。=C-/= の Undo のような操作で、簡単かつ高速にカーソルを移動できます。

- postpone の後に呼んでいるのは、Emacs 起動時の単純な高速化対策です。

```
(when (autoload-if-found
      '(goto-last-change goto-last-change-reverse)
      "goto-chg" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-,") 'goto-last-change)
    (global-set-key (kbd "C-." ) 'goto-last-change-reverse)))

(with-eval-after-load "flyspell"
  (define-key flyspell-mode-map (kbd "C-,") 'goto-last-change)
  (define-key flyspell-mode-map (kbd "C-." ) 'goto-last-change-reverse)))
```

## 4.12. DONE [back-button] マークをたどる

[OBSOLETE]

現在のバッファと開いている全てのバッファのマークを辿ることができます。いま注目している位置が、マーク全体でどのあたりに位置するのかをミニバッファに表示してくれます。ツールバーを表示している場合は、マークを付けたり辿る用のボタンが追加されます。global-mark-ring-max を設定して、辿れるマークの数を拡張しておきます。

ただ C-x <SPC> が潰れるので要注意です。矩形選択で用いる rectangle-mark-mode に当てられているキーバインドです。

- [rolandwalker/back-button: Visual navigation through mark rings in Emacs](#)

postpone の後に呼んでいるのは、Emacs 起動時の単純な高速化対策です。

```
(when (autoload-if-found
      '(back-button-mode
        back-button-local-forward back-button-global-forward)
      "back-button" nil t)

  (with-eval-after-load "back-button"
    (setq global-mark-ring-max 64)
    (setq back-button-local-forward-keystrokes '("<f10>"))
    (setq back-button-global-forward-keystrokes '("C-<f10>"))
    (define-key back-button-mode-map
      (kbd (car back-button-local-forward-keystrokes))
      'back-button-local-forward)
    (define-key back-button-mode-map
      (kbd (car back-button-global-forward-keystrokes))
      'back-button-global-forward)
    (setq back-button-mode-lighter nil)
    (setq back-button-index-timeout 0)))
```

```
(with-eval-after-load "postpone"
  (unless noninteractive
    (postpone-message "back-button-mode")
    (back-button-mode 1)))
```

### 4.13. **DONE** [point-undo.el] カーソル位置を簡単にたどる [OBSOLETE]

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ありません。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、最近では point-undo を使っていません。シングルキーを割り当てておくと使いやすいです。

```
(when (require 'point-undo nil t)
  ;; [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))
```

### 4.14. **DONE** [SmoothScroll.el] カーソル固定でスクロールする [OBSOLETE]

<https://raw.githubusercontent.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。残念ながら最近では使っていません。

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

  (with-eval-after-load "postpone"
    (global-set-key (kbd "s-<up>") 'scroll-one-down)
    (global-set-key (kbd "s-<down>") 'scroll-one-up)))
```

## 5. 編集サポート

### 5.1. エンターキーの挙動

好みの問題ですかね。標準では C-j が当てられています。

```
(with-eval-after-load "postpone"
  (global-set-key (kbd "RET") 'electric-newline-and-maybe-indent))
```

## 5.2. Yank 時に装飾を取る

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

## 5.3. 矩形編集 / 連番入力

24.4 からは，`rectangle-mark-mode` が使えるようになり，`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．

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

Built-in の `cua-base.el` (CUA-mode) を使うと，矩形選択は，領域選択後 `cua-toggle-rectangle-mark` でもできる．また，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．

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

## 5.4. 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，`"#+date:"` があると，`"#+date: 2011-12-31"` のようにタイムスタンプが記録される．Org Mode 用には `"[2018-10-08 Mon 10:00]"` のような形式でタイムスタンプを記録するようにしている．

```
(when (autoload-if-found
  '(time-stamp my-time-stamp)
  "time-stamp" nil t)
  (with-eval-after-load "postpone"
```

```
(add-hook 'before-save-hook #'my-time-stamp))

(with-eval-after-load "time-stamp"
  (setq time-stamp-start "#\\+date:[ \\t]*")
  (setq time-stamp-end "$")
  (setq time-stamp-line-limit 10) ;; def=8
  (setq time-stamp-default-format "%04y-%02m-%02d")

  (defun my-time-stamp ()
    (setq time-stamp-format
      (if (eq major-mode 'org-mode)
          "[%04y-%02m-%02d %3a %02H:%02M]"
          time-stamp-default-format))
    (if (boundp 'org-tree-slide-mode)
        (unless org-tree-slide-mode
            (time-stamp))
        (time-stamp))))
```

## 5.5. 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/>

```
(with-eval-after-load "postpone"
  (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)))
```

## 5.6. ChangeLog モード

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

(add-hook 'change-log-mode-hook
  (lambda()
    (if (require 'orgalist nil t)
        (when (boundp 'orgalist-mode)
            (orgalist-mode 1))
        (orgstruct-mode))
    (setq tab-width 4)
    (setq left-margin 4)))
```

## 5.7. 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))))
```

## 5.8. C/C++モード

```
(when (autoload-if-found
  '(modern-c++-font-lock-mode)
  "modern-cpp-font-lock" nil t)
  (push '("\.h$" . c++-mode) auto-mode-alist)
  (add-hook 'c++-mode-hook #'modern-c++-font-lock-mode))
```

## 5.9. C#モード

```
(when (autoload-if-found
  '(csharp-mode)
  "csharp-mode" "Major mode for editing C# mode." nil t)
  (push '("\.cs$" . csharp-mode) auto-mode-alist))
```

## 5.10. 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．

```
(when (autoload-if-found
  '(info org-info-ja)
  "info" nil t)

  (with-eval-after-load "info"
    (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 "")))))
```

## 5.11. R モード

```
(when (autoload-if-found
      '(R-mode R)
      "ess-site" "Emacs Speaks Statistics mode" nil t)

      (push '("\.rR$" . R-mode) auto-mode-alist))
```

## 5.12. nXML モード

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

## 5.13. yaml モード

```
(when (autoload-if-found
      '(yaml-mode)
      "yaml-mode" nil t)

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

## 5.14. json モード

```
(when (autoload-if-found
      '(json-mode)
      "json-mode" nil t)

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

## 5.15. javascript モード

```
(when (autoload-if-found
      '(js2-mode)
      "js2-mode" nil t)

  (with-eval-after-load "js2-mode"
    (require 'js2-refactor nil t)
    (push '("\.js$" . js2-mode) auto-mode-alist)

    (when (autoload-if-found
          '(ac-js2-mode ac-js2-setup-auto-complete-mode)
          "ac-js2" nil t)
      (add-hook 'js2-mode-hook #'ac-js2-mode))

    (if (executable-find "tern")
        (when (autoload-if-found
              '(tern-mode)
              "tern" nil t)
          (with-eval-after-load "tern"
```

```
(tern-mode 1)
;; tern-command shall be overwritten by actual path
(setq tern-command `("node" , (executable-find "tern")))
(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.)))))
```

## 5.16. csv モード

```
(when (autoload-if-found
      '(csv-mode)
      "csv-mode" nil t)

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

## 5.17. 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 .

```
(autoload-if-found '(ascii-on ascii-off) "ascii" nil t)
```

## 5.18. java モード

```
(when (autoload-if-found
      '(cc-mode)
      "cc-mode" nil t)

  (push '("\\.pde$" . java-mode) auto-mode-alist) ;; Processing
  (push '("\\.java$" . java-mode) auto-mode-alist))
```

## 5.19. es モード

- <https://github.com/dakrone/es-mode>

[ElasticSearch](#) のクエリを編集します . org-mode との連携もできます .

```
(when (autoload-if-found
      '(es-mode)
      "es-mode" nil t)

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

## 5.20. gnuplot モード

```
(when (autoload-if-found
      '(gnuplot-mode)
      "gnuplot-mode" nil t)

  (push '("\\.plt$" . gnuplot-mode) auto-mode-alist))
```

## 5.21. markdown-mode モード

- <http://jblevins.org/projects/markdown-mode/>

```
(when (autoload-if-found
      '(markdown-mode)
      "markdown-mode" nil t)

(push '("\\.markdown$" . markdown-mode) auto-mode-alist)
(push '("\\.md$" . markdown-mode) auto-mode-alist))
```

## 5.22. cmake モード

```
(if (executable-find "cmake")
    (when (autoload-if-found
          '(cmake-mode)
          "cmake-mode" nil t)

      (add-to-list 'auto-mode-alist '("CMakeLists\\.txt$" . cmake-mode))
      (add-to-list 'auto-mode-alist '("\\.cmake$" . cmake-mode)))
    (message "--- cmake is NOT installed."))
```

## 5.23. Web/HTML モード

[WEB]

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 (autoload-if-found
      '(web-mode)
      "web-mode" "web-mode" t)

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

(with-eval-after-load "web-mode"
  (define-key web-mode-map (kbd "<tab>") 'my-web-indent-fold)

  (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.
 '(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"))))))
```

## 5.24. PO モード

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

## 5.25. 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 . そして , hunspell に移行した .

---

'ns	sudo port install aspell aspell-dict-en
'x32	installer.exe and aspell-en from <a href="http://aspell.net/win32/">http://aspell.net/win32/</a>

---

- [英語の文章を aspell でスペルチェック - Qiita](#)

コマンドラインから aspell を使う時は ,

```
aspell -l en -c <file>
```

とすると , ~/.aspell.en.pws を個人辞書として暗黙的に設定し , スペルチェックをしてくれる . hunspell が使える環境ならば , 優先して使います . さらに ,

~/.aspell.conf に、次を書いておきます。

```
lang en_US

(when (autoload-if-found
      '(ispell-region ispell-complete-word)
      "ispell" nil t)

      (with-eval-after-load "postpone"
        ;; Spell checking within a specified region
        (global-set-key (kbd "C-c f 7") 'ispell-region)
        ;; 補完候補の表示 (flyspell が使える時はそちらを優先して <f7> にする .
        (global-set-key (kbd "<f7>") 'ispell-word))
        ;; (if (autoload-if-found '(helm-ispell) "helm-ispell" nil t)
        ;;     #'helm-ispell #'ispell-word)))

(with-eval-after-load "ispell"
  (setq ispell-encoding8-command t)
  ;; for English and Japanese mixed
  (add-to-list 'ispell-skip-region-alist '("[^\000-\377]+"))
  ;; http://endlessparentheses.com/ispell-and-org-mode.html
  (add-to-list 'ispell-skip-region-alist '("^#\|\\+begin_src" . "^#\|\\
+end_src"))
  (add-to-list 'ispell-skip-region-alist '("~" "~"))
  (add-to-list 'ispell-skip-region-alist '("=" "="))
  (add-to-list 'ispell-skip-region-alist '("org-property-drawer-re"))

  (cond
    ((executable-find "hunspell")
     (setenv "LC_ALL" "en_US")
     ;; (message "--- hunspell loaded.")
     (setenv "DICPATH"
"/Applications/LibreOffice.app/Contents/Resources/extensions/dict-en")
     (if shutup-p
       ;; 必要 . しかも ispell-program-name 指定の前で .
       (shut-up (ispell-change-dictionary "en_US" t))
       (ispell-change-dictionary "en_US" t))
     (setq-default ispell-program-name (executable-find "hunspell"))
     (setq ispell-local-dictionary "en_US")
     (setq ispell-dictionary ispell-local-dictionary)
     ;; Not regal way, but it's OK (usually ispell-local-dictionary-alist)

     (setq ispell-local-dictionary-alist
      '(("ja_JP" "[[:alpha:]]" "[^[:alpha:]]" "'" nil
        ("-d" "en_US") nil utf-8)
        ("en_US" "[[:alpha:]]" "[^[:alpha:]]" "'" nil
        ("-d" "en_US") nil utf-8)))
     (setq ispell-hunspell-dictionary-alist ispell-local-dictionary-alist)
     (setq ispell-personal-dictionary "~/Dropbox/emacs.d/.hunspell.en.dic"))

    ((executable-find "aspell")
     ;; (message "--- aspell loaded.")
     (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-dictionary "english")
     ;; This will also avoid an IM-OFF issue for flyspell-mode.
     ;; (setq ispell-aspell-supports-utf8 t) ;; Obsolete
     (setq ispell-local-dictionary-alist
```

```

'((nil "[a-zA-Z]" "[^a-zA-Z]" "" t
   ("-d" "en" "--encoding=utf-8") nil utf-8)))
(setq ispell-personal-dictionary "~/Dropbox/emacs.d/.aspell.en.pws")
)

(t
 nil)))

```

## 5.26. リアルタイムスペルチェック

Built-in の [flyspell.el](#) を使います。flyspell は内部で ispell を読み込んでいるので、辞書機能自体はそちらの設定が使われます。

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

```

(when (autoload-if-found
      '(flyspell-prog-mode flyspell-mode)
      "flyspell" nil t)

  (with-eval-after-load "postpone"
    (defvar major-mode-with-flyspell
      '(text-mode change-log-mode latex-mode yatex-mode
        git-commit-mode org-mode))
    (defvar major-mode-with-flyspell-prog
      '(c-mode-common emacs-lisp-mode perl-mode python-mode))
    (defvar my-flyspell-target-modes
      (append major-mode-with-flyspell
        major-mode-with-flyspell-prog))

    ;; バッファ内の全てをチェック対象にするモードの hook に flyspell 起動を登録
    (dolist (hook major-mode-with-flyspell)
      (add-hook (intern (format "%s-hook" hook))
        (lambda () (flyspell-mode 1))))

    ;; コメント行のみをチェック対象にする
    (dolist (hook major-mode-with-flyspell-prog)
      (add-hook (intern (format "%s-hook" hook))
        #'flyspell-prog-mode)))

  (with-eval-after-load "flyspell"
    ;; C-; をオーバーライド
    (define-key flyspell-mode-map (kbd "C-;") 'comment-dwim)
    (setq flyspell-duplicate-distance 0)
    (setq flyspell-mode-line-string " F")
    ;; (setq flyspell-large-region 200)
    (set-face-attribute 'flyspell-duplicate nil
      :foreground "#EA5506" :bold t
      :background nil :underline t)
    (set-face-attribute 'flyspell-incorrect nil
      :foreground "#BA2636" :bold nil
      :background nil :underline t)

    ;; ispell-complete-word のキーバインドを上書き
    (when (and (require 'helm nil t)
      (require 'flyspell-correct-helm nil t))
      (global-set-key (kbd "<f7>") 'flyspell-correct-word-generic)))

```

```
;; Auto complete との衝突を回避
(with-eval-after-load "auto-complete"
  (ac-flyspell-workaround))

(defun my-flyspell-ignore-nonascii (beg end _info)
  "incorrect 判定を ASCII に限定"
  (string-match "[^!-~]" (buffer-substring beg end)))
(add-hook 'flyspell-incorrect-hook #'my-flyspell-ignore-nonascii)

(defun my-flyspell-on ()
  (cond
    ((memq major-mode major-mode-with-flyspell)
     (turn-on-flyspell))
    ((memq major-mode major-mode-with-flyspell-prog)
     (flyspell-prog-mode))
    (t
     nil)))

(defun my-flyspell-off ()
  (when (memq major-mode my-flyspell-target-modes)
    (turn-off-flyspell)))

;; [FIXME] nextstep+inline-patch 版で flyspell すると、日本語 nyuu のようになる場合
;; があるので、それを回避 (IME が ON になったら一時的に flyspell を止める)
(add-hook 'my-ime-off-hook #'my-flyspell-on)
(add-hook 'my-ime-on-hook #'my-flyspell-off))
```

## 5.27. リージョン内の文字をカウントする

ビルドインの `simple.el` に十分な機能なのがある。

```
(with-eval-after-load "postpone"
  (global-set-key (kbd "M-=") 'count-words))
```

以前は、[word-count.el](#) を使用していた。

```
(when (autoload-if-found
      ' (word-count-mode)
      "word-count" "Minor mode to count words." t)
  (global-set-key (kbd "M-=") 'word-count-mode))
```

## 5.28. 世界時計を使う

`M-x display-time-world` で表示されます。次の設定を施して、`shackle.el` と組み合わせれば、表示後に `q` 押下でウィンドウを閉じてカーソルを元のバッファに戻せます。

世界の時刻を確認するために `wclock.el` がありました (参考:

<https://pxaka.tokyo/wiki/doku.php?id=emacs>) が、現在はビルトインの `time.el` に `display-time-world-mode` として吸収されているようです。`display-time-world-buffer-name` に `wclock` が設定されているところが名残と思われます。

```
(with-eval-after-load "time"
  (define-key display-time-world-mode-map "q" 'delete-window))
```

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

[TEX]

- [Files · 775887a89447dd19541b121161cc02e9799d0d3a · latex-math-preview / Latex Math Preview · GitLab](https://files.775887a89447dd19541b121161cc02e9799d0d3a-latex-math-preview-Latex-Math-Preview-GitLab)
- <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 <f6> を押すとプレビューが走り，さらに <f6> 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 (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)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "<f6>") 'latex-math-preview-expression))

  (with-eval-after-load "latex-math-preview"
    (setq latex-math-preview-command-path-alist
          '((latex . "latex")
            (dvi png . "dvi png")
            (dvips . "dvips"))))
  (define-key latex-math-preview-expression-mode-map (kbd "<f6>")
    'latex-math-preview-delete-buffer)))
```

## 5.30. [yatex.el] YaTeX で Tex 編集

[TEX]

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

  (push '("\\.tex$" . yatex-mode) auto-mode-alist)

  ;; Disable auto line break
  (add-hook 'yatex-mode-hook
    (lambda ()
      (setq auto-fill-function nil)))

  (with-eval-after-load "yatex"
    ;; 1=Shift JIS, 2=JIS, 3=EUC, 4=UTF-8
    ;; (setq YaTeX-kanji-code nil)
    (modify-coding-system-alist 'file "\\\.tex$" 'utf-8)))
```

ショートカットの利用を強制するメッセージが出るので，抑制します．

```

(with-eval-after-load "yatex"
  (put 'YaTeX-insert-braces 'begend-guide 2)

  (defun ad:YaTeX-insert-begin-end (env region-mode)
    "Insert \\begin{mode-name} and \\end{mode-name}."
    This works also for other defined begin/end tokens to define the structure."
    (setq YaTeX-current-completion-type 'begin)
    (let* ((ccol (current-column)) beg beg2 exchange
           (_arg region-mode) ;for old compatibility
           (indent-column (+ ccol YaTeX-environment-indent)) (_i 1) _func)
      (if (and region-mode (> (point) (mark)))
        (progn (exchange-point-and-mark)
                (setq exchange t
                      ccol (current-column)
                      indent-column (+ ccol YaTeX-environment-indent))))
      ;;VER2 (insert "\\begin{" env "}" (YaTeX-addin env))
      (setq beg (point))
      (YaTeX-insert-struct 'begin env)
      (setq beg2 (point))
      (insert "\n")
      (indent-to indent-column)
      (save-excursion
        ;;indent optional argument of \begin{env}, if any
        (while (> (point-beginning-of-line) beg)
          (skip-chars-forward "\\s " (point-end-of-line))
          (indent-to indent-column)
          (forward-line -1)))
      (require 'yatexenv)
      (if region-mode
        ;;if region-mode, indent all text in the region
        (save-excursion
          (if (fboundp (intern-soft (concat "YaTeX-enclose-" env)))
              (funcall (intern-soft (concat "YaTeX-enclose-" env))
                       (point) (mark))
              (while (< (progn (forward-line 1) (point)) (mark))
                (if (eolp) nil
                    (skip-chars-forward " \t\n")
                    (indent-to indent-column))))))
        (if region-mode (exchange-point-and-mark))
        (indent-to ccol)
        ;;VER2 (insert "\\end{" env "}\n")
        (YaTeX-insert-struct 'end env)
        (YaTeX-reindent ccol)
        (if region-mode
          (progn
            (insert "\n")
            (or exchange (exchange-point-and-mark)))
          (goto-char beg2)
          (YaTeX-intelligent-newline nil)
          (YaTeX-indent-line))
        (YaTeX-package-auto-usepackage env 'env)
        (if YaTeX-current-position-register
          (point-to-register YaTeX-current-position-register))))

  (advice-add 'YaTeX-insert-begin-end
    :override #'ad:YaTeX-insert-begin-end))

```

### 5.31. [auxtex.el] AUCTEX で Tex 編集

[TEX]

実践未投入です .

```
(when (autoload-if-found
      '(autotex-latexmk)
      "autotex-latexmk" nil t)

(push '("\\.tex$" . autotex-latexmk) auto-mode-alist)

(with-eval-after-load "autotex-latexmk"
  (setq-default TeX-master nil)
  (setq TeX-auto-save t)
  (setq TeX-parse-self t)
  (setq TeX-PDF-mode t)
  (auctex-latexmk-setup)))
```

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

実は余り使いこなせていません...

- <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

↑のサイトで紹介されている回避策とは異なり，新たな `my-yas-expand` を作ることで，`org` バッファのソースブロック中で `TAB` 押下してもエラーを受けないようにしました．ソースコードは `C-c` で開く別バッファで編集します．

↑どうやら [本家で対応](#) がなされたようです．`my-yas-expand` なしで所望の動作になりました．ありがたや，ありがたや．

```
(when (autoload-if-found
      '(yas-minor-mode yas-global-mode)
      "yasnippet" nil t)

(dolist (hook
  (list
    'perl-mode-hook 'c-mode-common-hook 'js2-mode-hook 'org-mode-hook
    'python-mode-hook 'emacs-lisp-mode-hook))
  (add-hook hook #'yas-minor-mode))

(with-eval-after-load "yasnippet"
  (setq yas-verbosity 2)
  (setq yas-snippet-dirs
```

```
(list "~/Dropbox/emacs.d/yas-dict"
      'yas-installed-snippets-dir)) ;; for Cask
(unless noninteractive
  (yas-global-mode 1)))
```

以下は，本家できちんと対応されたので，不要になった．

```
(with-eval-after-load "yasnippet"
  (defun my-yas-expand-src-edit (&optional field)
    "Override `yas-expand'. Kick `org-edit-special' directly in src-block."
    (interactive)
    (cond ((and (equal major-mode 'org-mode)
                 (org-in-src-block-p t))
           (org-edit-special))
          (t
           (yas-expand field))))

  (defun my-yas-expand (&optional field)
    "Disable `yas-expand' in src-block."
    (interactive)
    (cond ((and (equal major-mode 'org-mode)
                 (org-at-heading-p))
           (org-cycle))
          ((and (equal major-mode 'org-mode)
                 (org-in-src-block-p t)
                 (not (and (fboundp 'org-src-edit-buffer-p)
                           (org-src-edit-buffer-p))))
           (org-cycle))
          (t (yas-expand field))))
  (define-key yas-minor-mode-map (kbd "<tab>") 'my-yas-expand))
```

### 5.33. [osx-dictionary.el] 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 (autoload-if-found
      ' (osx-dictionary-search-pointer osx-dictionary-search-input)
      "osx-dictionary" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-M-w") #'osx-dictionary-search-pointer)
    (global-set-key (kbd "C-c f w") #'osx-dictionary-search-input))

  (with-eval-after-load "osx-dictionary"
    (custom-set-variables
      ' (osx-dictionary-dictionary-choice "英辞郎 第七版"))))
```

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 COBUILD 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 .

### 5.34. [describe-number.el] 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 .

```
(autoload-if-found
  '(describe-number describe-number-at-point)
  "describe-number" nil t)
```

### 5.35. [emmet-mode.el] zencoding の後継

[WEB]

```
(when (autoload-if-found
      '(emmet-mode)
      "emmet-mode" nil t nil)

  (push '("\.xml\\"" . nxml-mode) auto-mode-alist)
  (push '("\.rdf\\"" . nxml-mode) auto-mode-alist)
  (dolist (hook
            '(sgml-mode-hook
               nxml-mode-hook css-mode-hook html-mode-hook web-mode-hook))
    (add-hook hook #'emmet-mode))

  (with-eval-after-load "emmet-mode"
    (setq emmet-indentation 2)
    (setq emmet-move-cursor-between-quotes t)))
```

### 5.36. [web-beautify.el] ソースコード整形

[WEB]

ソースコードを読みやすい表示に整形します . バッファの自動時に自動で整形を実施するには , after-save-hook を使えば OK です .

---

JavaScript	M-x web-beautify-js
HTML	M-x web-beautify-html
CSS	M-x web-beautify-css

---

```
(if (executable-find "js-beautify")
  (when (autoload-if-found
        '(js2-mode)
        "js2-mode" nil t)

    (with-eval-after-load "js2-mode"
      (when (require 'web-beautify nil t)
        (define-key js2-mode-map (kbd "C-c b") 'web-beautify-js)
        (define-key js2-mode-map (kbd "C-c b") 'web-beautify-css))))

  (message "--- js-beautify is NOT installed.")
  (message "--- Note: npm -g install js-beautify"))
```

## 5.37. [smartparens.el] 対応するカッコの挿入をアシスト

- <http://smartparens.readthedocs.io/>

postpone の後に呼んでいるのは，Emacs 起動時の単純な高速化対策です．

```
(with-eval-after-load "postpone"
  (when (require 'smartparens nil t)
    (setq-default sp-highlight-pair-overlay nil)
    (setq-default sp-highlight-wrap-overlay nil)
    (setq-default sp-highlight-wrap-tag-overlay nil)
    (sp-local-pair 'org-mode "$" "$")
    (sp-local-pair 'org-mode "~" "~")
    (sp-local-pair 'org-mode "+" "+")
    (sp-local-pair 'org-mode "=" "=")
    (sp-local-pair 'org-mode "_" "_")
    (sp-local-pair 'yatex-mode "$" "$")
    (sp-local-pair 'emacs-lisp-mode "`" "`")
    (sp-pair "`" nil :actions :rem)
    (sp-pair "'" nil :actions :rem)
    (sp-pair "[" nil :actions :rem)
    (unless noninteractive
      (postpone-message "smartparens")
      (smartparens-global-mode))))
```

## 5.38. [replace-from-region.el] 選択領域を別の文字列に置き換える

通常の query-replace は，変更前と変更後の文字列を両方入力しますが，query-replace-from-region は，変更前の文字列を選択領域から自動抽出し，さらに同じ単語を変更後の文字列の候補として事前入力してくれます．したがって，文字列選択，query-replace-from-region 呼び出し，文字列を一部変更，実行，というフローになり簡略化されます．さらに，selected.el を使うことで，文字列選択後の query-replace-from-region 呼び出しは，シングルキーの押下で実現できます（私の場合は 5 の押下）．

文字列選択は，C-M-SPC で簡単にできるので，つまり，C-M-SPC, 5，文字列を一部変更，実行となり，通常の query-replace よりもタイプ量を圧倒的に減らせます．

iedit.el の方が好みの人が多いかもしれません．

- <https://www.emacswiki.org/emacs/replace-from-region.el>

```
(autoload-if-found
  ' (query-replace-from-region query-replace-regexp-from-region)
  "replace-from-region" nil t)
```

## 5.39. [selected.el] リージョン選択時のアクションを制御

選択した後に右クリック的な感じでリージョンに対するアクションを制御できます．選択

領域に対するスピードコマンドですね．普通にシングルキーを割り当てると，日本語 IME が有効な時に上手くいかないので，activate-mark-hook と deactivate-mark-hook に細工しています．

- [Kungsgeten/selected.el: Keymap for when region is active](#)
- [selected.el - Keymap for when region is active : emacs](#)

postpone の後に呼んでいるのは，Emacs 起動時の単純な高速化対策です．

```
(with-eval-after-load "postpone"
  (when (require 'selected nil t)
    (define-key selected-keymap (kbd ";") #'comment-dwim)
    (define-key selected-keymap (kbd "e") #'my-eval-region-echo)
    ;; (define-key selected-keymap (kbd "E") #'my-eval-region-echo-as-function)
    (define-key selected-keymap (kbd "=") #'count-words-region)
    (when (require 'helpful nil t)
      (define-key selected-keymap (kbd "m") #'helpful-macro))
    (define-key selected-keymap (kbd "f")
      (if (fboundp 'helpful-function) #'helpful-function #'describe-function))
    (define-key selected-keymap (kbd "v")
      (if (fboundp 'helpful-variable) #'helpful-variable #'describe-variable))
    (define-key selected-keymap (kbd "w") #'osx-dictionary-search-pointer)
    (define-key selected-keymap (kbd "5") #'query-replace-from-region)
    (define-key selected-keymap (kbd "g") #'my-google-this)
    (define-key selected-keymap (kbd "s") #'osx-lib-say-region)
    (define-key selected-keymap (kbd "q") #'selected-off)
    (define-key selected-keymap (kbd "x") #'my-hex-to-decimal)
    (define-key selected-keymap (kbd "X") #'my-decimal-to-hex)
    (defun my-eval-region-echo ()
      (interactive)
      (when mark-active
        (eval-region (region-beginning) (region-end) t)))
    (setq selected-org-mode-map (make-sparse-keymap))
    (define-key selected-org-mode-map (kbd "t") #'org-table-convert-region)

    (when (autoload-if-found
          '(my-org-list-insert-items my-org-list-insert-checkbox-into-items)
          "utility" nil t)

      (define-key selected-keymap (kbd "-") #'my-org-list-insert-items)
      (define-key selected-keymap (kbd "_")
        #'my-org-list-insert-checkbox-into-items))

    (when (autoload-if-found
          '(helm-selected)
          "helm-selected" nil t)
      (define-key selected-keymap (kbd "h") #'helm-selected))

    (when (require 'help-fns+ nil t)
      (defun my-describe-selected-keymap ()
        (interactive)
        (describe-keymap 'selected-keymap))
      (define-key selected-keymap (kbd "H") #'my-describe-selected-keymap))
    (unless noninteractive
      (postpone-message "selected"))
```

```
(selected-global-mode 1))))
```

#### 5.40. [helm-selected.el] selecte.el のアクション候補を絞り込み

- [記事](#)にしました .

```
(with-eval-after-load "selected"
  (when (autoload-if-found '(helm-selected) "helm-selected" nil t)
    (define-key selected-keymap (kbd "h") 'helm-selected)))
```

#### 5.41. TODO [isolate.el] ブラケット等の入力をアシスト

- [casouri/isolate: Surrounding magics, extensible](#)

```
(autoload-if-found
  '(isolate-quick-add
    isolate-long-add isolate-quick-delete
    isolate-quick-chnge isolate-long-change)
  "isolate" nil t)
```

#### 5.42. TODO [git-complete.el] GIT grep を使う補完エンジン

- [zk-phi/git-complete: {Emacs} Yet another completion engine powered by "git grep"](#)

```
(when (autoload-if-found
  '(git-complete)
  "git-complete" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-c f <tab>") 'git-complete)))
```

#### 5.43. TODO [tiny.el] 連番入力をサポート

- [abo-abo/tiny: Quickly generate linear ranges in Emacs](#)

```
(when (require 'tiny nil t)
  (tiny-setup-default))
```

#### 5.44. TODO [bratex.el] LaTeX 数式のブラケット入力をサポート

- [sbrisard/bratex: Emacs package for manipulation of brackets in LaTeX mode](#)

```
(when (autoload-if-found
  '(bratex-config)
  "bratex" nil t)

  (with-eval-after-load "postpone"
    (add-hook 'yatex-mode-hook #'bratex-config)))
```

## 5.45. **DONE [iedit.el] バッファ内の同じ文字列を一度に編集する [OBSOLETE]**

replace-from-region.el を selected.el で呼び出すので満足しています。表示色の点では、highlight-symbol.el があるので、今からどれが編集されるのかもわかりやすいです。

[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)
```

## 5.46. **DONE [lookup.el] 辞書**

**[OBSOLETE]**

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

;;; 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/text/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 requires "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

```

## 5.47. **DONE** [cacoo] Cacao で描く

[OBSOLETE]

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)

  (with-eval-after-load "cacoo"
    (require 'cacoo-plugins))

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

```

## 5.48. **DONE** [zencoding-mode] HTML 編集の高速化

[OBSOLETE]

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 (autoload-if-found
      '(zencoding-mode zencoding-expand-line)
      "zencoding-mode" "Zen-coding" t)

  (with-eval-after-load "zencoding-mode"
    (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))
```

## 5.49. **DONE** [sdic.el] 英辞郎で英単語を調べる

[OBSOLETE]

- osx-directory.el に移行しました .

<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)

  (with-eval-after-load "sdic"
    (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"))))

  ;; 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))
```

## 5.50. **DONE** [dictionary.app] macOS の dictionary.app で COBUILD5 をひく

[OBSOLETE]

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)
    (begin)))
```

```

(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 COBUILD 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．

### 5.50.1. 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 .

### 5.50.2. 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 (autoload-if-found
      '(dict-app-search)
      "dict-app" nil t)

  ;; 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))
```

## 6. 表示サポート

### 6.1. キーコマンド入力中に入力過程をミニバッファに反映する

標準値は 1 です . 例えば , C-c f r で発動する関数があるとき , C-c を入力するとその直後にはミニバッファに何も表示されませんが , echo-keystrokes だけ経過すると , C-c が表示されます . 0 に設定すると , いくら経過しても何も表示しません . which-key.el の設定で表示を 1.0 にすると , 時系列的に , 0.5 秒でキー入力の状態を表示し , 1.0 秒で続くキーで入力可能なコマンドがリストアップ表示されます .

```
(setq echo-keystrokes 0.5)
```

### 6.2. [delight.el] モードラインのモード名を短縮する

- [GNU ELPA - delight](#)

以前は diminish.el を使用していましたが , ELPA 配布のパッケージに移行しました . メジャーモードの短縮表示もマイナーモードの場合と同様に設定できます . この点で , 設定がスッキリしました .

postpone の後に呼んでいるのは , Emacs 起動時の単純な高速化対策です .

```
(with-eval-after-load "postpone"
  (when (require 'delight nil t)
    (unless noninteractive
      (postpone-message "delight")))
  (delight
   ';; Major modes
   (c-mode "C" :major)
   (c++mode "C++" :major))
```

```

(js2-mode "JS" :major)
(csharp-mode "C#" :major)
(prog-mode "Pr" :major)
(emacs-lisp-mode "El" :major)
(python-mode "Py" :major)
(perl-mode "Pl" :major)
(web-mode "W" :major)
(change-log-mode "ChangeLog" :major)
(lisp-interaction-mode "Lisp" :major)

;; Shorten for minor modes
(ggtags-mode " G" "ggtags")
;; (orgstruct-mode " OrgS" "org")
(orgalist-mode " ol" "orgalist")

;; No display for minor modes
(org-fancy-priorities-mode nil "org-fancy-priorities")
(smooth-scroll-mode nil "smooth-scroll")
(eldoc-mode nil "eldoc")
(centered-cursor-mode nil "centered-cursor-mode")
(volatile-highlights-mode nil "volatile-highlights")
(aggressive-indent-mode nil "aggressive-indent")
(all-the-icons-dired-mode nil "all-the-icons-dired")
(yas-minor-mode nil "yasnipet")
(auto-complete-mode nil "auto-complete")
(ws-butler-mode nil "ws-butler")
(isearch-mode nil "isearch")
(auto-revert-mode nil "autorevert")
(global-whitespace-mode nil "whitespace")
(emmet-mode nil "emmet-mode")
(helm-mode nil "helm-mode")
(abbrev-mode nil "abbrev")
(doxymacs-mode nil "doxymacs")
(editorconfig-mode nil "editorconfig")
(rainbow-mode nil "rainbow-mode")
(highlight-symbol-mode nil "highlight-symbol")
(which-key-mode nil "which-key")
(fancy-narrow-mode nil "fancy-narrow")
(smartparens-mode nil "smartparens")
(projectile-mode nil "projectile")
(selected-minor-mode nil "selected"))))

```

### 6.3. 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。

```

(with-eval-after-load "org"
  (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))) ""
                        " N"))
          entry))
      mode-line-modes)))

```

## 6.4. 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)))))
```

## 6.5. モードラインの色をカスタマイズする

配色は定期的に変えています .

```
(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")
```

### 6.5.1. 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

## 6.6. 行番号の表示制限を拡張する

デフォルトだと 200 桁までしか把握しないので，いつも表示されるように拡張します．

```
(setq line-number-display-limit-width 100000)
```

## 6.7. *visible-bell* のカスタマイズ

最近はや陶しくなってしまう，ビープ音も無しかつ視覚効果も無しにしています．

```
;; (setq visible-bell nil) ;; default=nil  
(setq ring-bell-function 'ignore)
```

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

以前は，<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．

```
(when (autoload-if-found  
      '(echo-area-bell)  
      "echo-area-bell" nil t)  
  
  (with-eval-after-load "echo-area-bell"  
    (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))
```

## 6.8. 常に *scratch* を表示して起動する

最近では，起動用にメジャーモードを書いて対応しています．詳しくはリンク先にて．

- [高速起動用ミニマム\\*scratch\\*バッファ - Qiita](#)

```
(when (require 'empty-booting nil t)  
  ;; (setq initial-buffer-choice t) ;; 引数付き起動すると画面分割される  
  (setq initial-scratch-message nil)  
  (setq initial-major-mode 'empty-booting-mode)  
  ;; :underline "#203e6f"
```

```
(set-face-foreground 'header-line "#FFFFFF") ;; "#203e6f" #333333 "#FFFFFF"
(set-face-background 'header-line "#5F7DB7") ;; "#ffb08c" "#7e59b5"
(set-face-attribute 'header-line nil
                    :inherit nil
                    :overline nil
                    :underline nil)
(setq header-line-format
      (concat
        " GNU Emacs"
        (format-time-string "W%W: %Y-%m-%d %a.")))

(with-eval-after-load "postpone"
  (defun ad:split-window-below (&optional _size)
    "An extention to switch to \*scratch\* buffer after splitting window."
    (my-open-scratch))
  ;; (advice-add 'split-window-below :after #'ad:split-window-below)

  (defun my-open-scratch ()
    "Switch the current buffer to \*scratch\* buffer."
    (interactive)
    (switch-to-buffer "*scratch*"))
  (global-set-key (kbd "C-M-s") #'my-open-scratch))
```

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．

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．

```
;; Start Emacs with scratch buffer even though it call session.el/desktop.el
(add-hook 'emacs-startup-hook (lambda () (switch-to-buffer "*scratch*")))
(setq initial-major-mode 'text-mode)
(setq initial-scratch-message
      (concat "
                (format-time-string "%Y-%m-%d (%a.)") "\n"
                "-----"
                "-----\n"))
```

## 6.9. 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．

スクロールバーの非表示は起動後に実施されるため，フレームがチラつきます．それを解消するために，現在は，ソースコードレベルでオフにしています．パッチに興味がある場合は，[こちら](#)．以下は，MBP ビルド向けに設定しています．

```
;; Show scroll bar or not
(when (and (display-graphic-p)
           (eq window-system 'mac))
  (set-scroll-bar-mode nil)) ; 'right
```

## 6.10. 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.
(when (display-graphic-p)
  (tool-bar-mode -1))
```

## 6.11. 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)
```

## 6.12. 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.
(with-eval-after-load "postpone"
  (unless noninteractive
    (postpone-message "line-number-mode")
    (line-number-mode 1)))
```

## 6.13. 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)
```

## 6.14. 行番号をバッファに表示する

Emacs Version 26.1 からネイティブ実装に切り替わった `display-line-numbers.el` を使います . 普段は使わないので , 必要に応じてグローバルにトグルできるようにしてあります .

```
(when (autoload-if-found
      '(my-toggle-display-line-numbers-mode)
      "display-line-numbers" nil t)

  (with-eval-after-load "display-line-numbers"
    (custom-set-variables
      '(display-line-numbers-width-start t))

    (defun my-toggle-display-line-numbers-mode ()
      "Toggle variable `global-display-line-numbers-mode'."
      (interactive)
      (if (fboundp 'global-display-line-numbers-mode) ;; 26.1 or later
          (let ((flag (if global-display-line-numbers-mode -1 1)))
            (global-display-line-numbers-mode flag))
          (user-error "The display-line-numbers is NOT supported")))))
```

## 6.15. 時刻をモードラインに表示する

`postpone` の後に呼んでいるのは , Emacs 起動時の単純な高速化対策です .

```
;; Show clock in in the mode line
```

```
(with-eval-after-load "postpone"
  (setq display-time-format "%H%M.%S") ;; %y%m%d.
  (setq display-time-interval 1)
  (setq display-time-default-load-average nil)
  (unless noninteractive
    (postpone-message "display-time-mode")
    (display-time-mode 1)))
```

## 6.16. 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](http://mic-paren.el) があり，現在はこれを利用している．

postpone の後に呼んでいるのは，Emacs 起動時の単純な高速化対策です．

```
(with-eval-after-load "postpone"
  (when (require 'mic-paren nil t)
    (setq paren-sexp-mode nil)
    (set-face-foreground 'paren-face-match "#FFFFFF")
    ;; Deep blue: #6666CC, orange: #FFCC66
    (set-face-background 'paren-face-match "#66CC66")
    (unless noninteractive
      (postpone-message "mic-paren")
      (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)
```

## 6.17. 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](http://ubulog.blogspot.jp/2007/09/emacs_09.html)
- [emacs でタブと EOF と全角空白の表示を。](#)
- [whitespace-mode を使って、ファイルの保存時に行末のスペースや末尾の改行を削除する - Qiita](#)

英語で原稿を書く時に全角スペースが入っているを苦勞するので，強調表示して編集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．パッケージを使うと too much かなという印象があったので，個別の設定だけを使わせてもらっています．

```
(with-eval-after-load "postpone"
;; 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))

;;show EOF
;; (defun set-buffer-end-mark()
;;   (let ((overlay (make-overlay (point-max) (point-max))))
;;     (overlay-put overlay 'before-string #("[EOF]" 0 5 (face highlight)))
;;     (overlay-put overlay 'insert-behind-hooks
;;       '((lambda (overlay after beg end &optional len)
;;         (when after
;;           (move-overlay overlay (point-max) (point-max)))))))
;;   (add-hook 'find-file-hooks #'set-buffer-end-mark))
```

## 6.18. バッファの終わりをフリンジで明示

以下の設定では，ウィンドウ以下にバッファが続いているかを表す矢印と，続いていないことを示すカギカッコをフリンジに表示します．

```
(setq-default indicate-buffer-boundaries
  '((top . nil) (bottom . right) (down . right)))
```

## 6.19. [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．正直なところ，それほどメリットを感じていません．

```
(if (executable-find "cmigemo")
  (when (autoload-if-found
```



```

      '(migemo-init)
      "migemo" nil t)

(add-hook 'isearch-mode-hook #'migemo-init)

(with-eval-after-load "migemo"
  (custom-set-variables
    '(completion-ignore-case t) ;; case-independent
    '(migemo-command "cmigemo")
    '(migemo-options '("-q" "--emacs" "-i" "\a"))
    '(migemo-dictionary "/usr/local/share/migemo/utf-8/migemo-dict")
    '(migemo-user-dictionary nil)
    '(migemo-regex-dictionary nil)
    '(migemo-use-pattern-alist t)
    '(migemo-use-frequent-pattern-alist t)
    '(migemo-pattern-alist-length 1024)
    '(migemo-coding-system 'utf-8-unix))))
(message "--- cmigemo is NOT installed.))

```

## 6.20. [helm.el] 続・何でも絞り込みインターフェイス

M-s を helm-swoop にあてていましたが，M-s を入り口にたくさんの検索系コマンドが割り振られているため M-s M-s に変えました．

以下の設定では，helm に関連するモジュールも多数含まれます．helm がキックされるタイミングで有効化させます．

- helm-google
- helm-swoop
- helm-projectile
- helm-bm

```

(when (autoload-if-found
      '(helm-google)
      "helm-google" nil t)

  (with-eval-after-load "helm-google"
    (custom-set-variables
      '(helm-google-tld "co.jp"))))

(when (autoload-if-found
      '(helm-buffers-list helm-recentf)
      "helm" nil t)

  (global-set-key (kbd "C-M-r") 'helm-recentf)
  (global-set-key (kbd "C-x C-b") 'helm-buffers-list)

  (with-eval-after-load "helm"
    (require 'helm-config nil t)))

```

```

(when (autoload-if-found
      '(helm-M-x
        helm-locate helm-descbinds
        helm-occur helm-swoop helm-flycheck helm-bookmarks)
      "helm-config" nil t)

(global-set-key (kbd "M-x") 'helm-M-x)

(with-eval-after-load "postpone"

  ;; (defun my-helm-swoop ()
  ;;   (interactive)
  ;;   (when (and (fboundp 'dimmer-mode)
  ;;             (eq dimmer-mode t))
  ;;     (setq dimmer-mode -1))
  ;;   (helm-swoop))

  (global-set-key (kbd "C-M-l") 'helm-locate)
  (global-set-key (kbd "C-c f b") 'helm-bookmarks)
  (global-set-key (kbd "M-s M-s") 'helm-swoop)
  (global-set-key (kbd "C-c o") 'helm-occur)
  (global-set-key (kbd "C-h d") 'helm-descbinds))

(with-eval-after-load "helm-config"
  (helm-mode 1)

  ;; (when (require 'diminish nil t)
  ;;   (diminish 'helm-mode)) ;; " H"

  (when (require 'helm-swoop nil t)
    ;; カーソルの単語が org の見出し ( *の集まり ) なら検索対象にしない .
    (setq helm-swoop-pre-input-function
      (lambda ()
        (unless (thing-at-point-looking-at "^\\\\"*+")
          (thing-at-point 'symbol))))
    ;; 配色設定
    (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-files nil t)
    (define-key helm-find-files-map
      (kbd "<tab>") 'helm-execute-persistent-action)
    (define-key helm-read-file-map
      (kbd "<tab>") 'helm-execute-persistent-action))

  ;; (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 (memq window-system '(mac ns))
  (setq helm-locate-command "mdfind -name %s %s"))

;; (require 'helm-css-scss nil t)
;; (require 'helm-emmet nil t)
;; (require 'helm-descbinds nil t)

(when (require 'helm-projectile nil t)
  ;; M-x helm-projectile-find-file (C-c p f)
  (setq projectile-completion-system 'helm)
  ;; projectile.el のキーバインドをオーバーライド
  (helm-projectile-toggle 1))

(require 'helm-bm nil t)))

;; 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)))

```

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

編集差分の視覚化は，元々 git-gutter が提供している機能です．しかし有効にするとフレームの幅が若干広がってしまうので，気になる人は git-gutter-fringe を使えばよいです．

```

(when (autoload-if-found
      '(git-gutter-mode)
      "git-gutter" nil t)

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

  (with-eval-after-load "git-gutter"
    (custom-set-variables
      '(git-gutter:lighter ""))

    (when (require 'git-gutter-fringe nil t)
      (custom-set-variables
        '(git-gutter-fr:side 'left-fringe)))

```

```

;; (require 'fringe-helper nil t) ;; byte-compile 時に明示的に指定が必要 .
;; "!"
(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..."
  ".XXXXXXX."
  ".XXXXXXX."
  "...XX..."
  "...XX..."
  ".....")
;; "-"
(fringe-helper-define 'git-gutter-fr:deleted nil
  "....."
  "....."
  "....."
  ".XXXXXXX."
  ".XXXXXXX."
  "....."
  "....."
  ".....")
(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"))))

```

## 6.22. [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)))

```

## 6.23. [calendar.el] カレンダーで週番号を表示する

ビルドインの calendar.el にある calendar-intermonth-text をカスタマイズすると、カレンダーに週番号を表示させることが可能です。ただ、calendar.el に記載されている例だと、calendar-week-start-day が 1 以外の時に計算結果がおかしくなるので、次のように calendar-absolute-from-gregorian に渡す値を補正する必要があります。

```
(with-eval-after-load "postpone"
  (global-set-key (kbd "C-c f c c") 'calendar))

(with-eval-after-load "calendar"
  (setq calendar-week-start-day 1)
  (copy-face 'default 'calendar-iso-week-header-face)
  (set-face-attribute 'calendar-iso-week-header-face nil
    :height 1.0 :foreground "#1010FF"
    :background (face-background 'default))
  (setq calendar-intermonth-header
    (propertize " w"
      'font-lock-face 'calendar-iso-week-header-face))

  (copy-face font-lock-constant-face 'calendar-iso-week-face)
  (set-face-attribute 'calendar-iso-week-face nil
    :height 1.0 :foreground "orange"
    :background (face-background 'default))

  (setq calendar-intermonth-text
    '(propertize
      (format "%02d"
        (car
          (calendar-iso-from-absolute
            (calendar-absolute-from-gregorian
              (list month
                (- day (1- calendar-week-start-day)) year))))
      'font-lock-face 'calendar-iso-week-face)))
```

## 6.24. [which-key] キーバインドの選択肢をポップアップする

guide-key.el の後発。guide-key.el の改良でもあり、デイスパッチャが見やすく、直感的でとても使いやすい。

起動時間を短縮するため、設定の読み込みは org-mode か helm 起動時に行うように遅延設定している。

```
(when (autoload-if-found
  ' (which-key-mode)
  "which-key" nil t)

  (with-eval-after-load "postpone"
    (unless noninteractive
      (postpone-message "which-key")
      (which-key-mode 1))))
```

```
(with-eval-after-load "which-key"
  (custom-set-variables
    '(which-key-idle-delay 1.0))))
```

## 6.25. [highlight-symbol] 同じ名前のシンボルをハイライトする

一定時間が過ぎると，カーソル下のワードをバッファ内で検索し，ハイライトしてくれる．殺風景なバッファに動きが出て良い．また，highlight-symbol-nav-mode を使うと，シンボル間を M-n/M-p で移動できるので，毎度検索しなくてよい．

```
(when (autoload-if-found
      '(highlight-symbol-mode highlight-symbol-nav-mode)
      "highlight-symbol" nil t)

  (dolist (hook '(emacs-lisp-mode-hook c-mode-common-hook prog-mode-hook))
    (add-hook hook #'highlight-symbol-mode))

  (with-eval-after-load "highlight-symbol"
    (custom-set-variables
      '(highlight-symbol-idle-delay 0.5))))
```

## 6.26. [all-the-icons-dired] フォントを使ったアイコンを表示

dired で，ファイルのアイコンを表示します．all-the-icons をインストール後に，M-x all-the-icons-install-fonts を忘れずに実行する必要があります．neotree の設定は別セクションに記載しています．

```
(when (autoload-if-found
      '(all-the-icons-dired-mode)
      "all-the-icons-dired" nil t)

  (add-hook 'dired-mode-hook #'all-the-icons-dired-mode))
```

## 6.27. [eldoc.el] コンテキストに応じてヘルプを表示

postpone の後に呼んでいるのは，Emacs 起動時の単純な高速化対策です．

```
(when (autoload-if-found
      '(turn-on-eldoc-mode)
      "eldoc" nil t)

  (dolist (hook '(emacs-lisp-mode-hook org-mode-hook c-mode-common-hook))
    (add-hook hook #'turn-on-eldoc-mode))

  (with-eval-after-load "eldoc"
    (custom-set-variables
      '(eldoc-idle-delay 1.))))
```

## 6.28. [keycast.el] 入力しているキーとコマンドをモードラインに表示

- 入力した文字や発行したコマンドキーをモードラインに表示．コマンドの名前もリ

アルタイムに表示される .

- [tarsius/keycast: Show current command and its key in the mode line](#)

```
(autoload-if-found '(keycast-mode) "keycast" nil t)
```

## 6.29. TODO [rainbow-delimiters.el] 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なります .  
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))))
```

## 6.30. TODO [yascroll.el] フリンジにスクロールバーを出す

yascroll を使います .

- <http://d.hatena.ne.jp/m2ym/20110401/1301617991>
- 2015-03-15: smooth-scroll との組み合わせで重いのを確認 .

postpone の後に呼んでいるのは , Emacs 起動時の単純な高速化対策です .

```
(with-eval-after-load "postpone"
  (when (require 'yascroll nil t)
    (setq yascroll:delay-to-hide 2)
    (setq yascroll:disabled-modes '(org-mode))
    (set-face-foreground 'yascroll:thumb-fringe "#b2cefb")
    (set-face-background 'yascroll:thumb-fringe "#b2cefb")
    (unless noninteractive
      (postpone-message "yascroll")
      (global-yascroll-bar-mode 1))))
```

### 6.31. *TODO* [dimmer.el] 現在のバッファ以外の輝度を落とす

```
(when (autoload-if-found
      '(dimmer-mode dimmer-process-all dimmer-off dimmer-on)
      "dimmer" nil t)

  (with-eval-after-load "postpone"
    (unless noninteractive
      (postpone-message "dimmer-mode")
      (custom-set-variables
        '(dimmer-exclusion-regexp
          "^\\*[Hh]elm\\|^ \\*Minibuf\\|^ \\*Echo\\|^\\*Calendar\\|^\\*Org")
        '(dimmer-fraction 0.6))
      (dimmer-mode 1)))

  (with-eval-after-load "dimmer"
    (defun dimmer-off ()
      (dimmer-mode -1)
      (dimmer-process-all))
    (defun dimmer-on ()
      (dimmer-mode 1)
      (dimmer-process-all))
    (add-hook 'focus-out-hook #'dimmer-off)
    (add-hook 'focus-in-hook #'dimmer-on)))
```

### 6.32. *DONE* バッテリー情報をモードラインに表示する

[OBSOLETE]

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

### 6.33. *DONE* [mode-icons] 使用中のモード表示をアイコンで代替 [OBSOLETE]

しばらく使ってみたが，統一感が失われるので使用停止中．

```
(when (require 'mode-icons nil t)
  ;; アイコンを保存しているディレクトリを指定
  (setq mode-icons--directory
        (expand-file-name "~/ .emacs.d/.cask/package/icons"))
  (mode-icons-mode 1))
```

### 6.34. *DONE* [stock-ticker] 株価をモードラインに表示

[OBSOLETE]

季節が過ぎ去りました．不使用です．

日経平均やダウ平均の状況をモードラインに表示します．表示が長くなる傾向があるので，`stock-ticker--parse` を再定義して，銘柄（3桁のみ）と変動率だけを表示しています．

起動時には不要なので，ウィンドウにフォーカスが移った時に開始して，さらに1分でモードラインから消えるようにしています．



色々と不安定になってきたので、最近は使っていません。

```
(when (autoload-if-found
      '(my-activate-stock-ticker stock-ticker-global-mode)
      "stock-ticker" nil t)
  (with-eval-after-load "stock-ticker"
    (defun stock-ticker--parse (data)
      "Parse financial DATA into list of display strings."
      (let ((qs (assoc-default
                  'quote (assoc-default
                          'results (assoc-default 'query data)))))
        (mapcar
         (lambda (q)
           (let ((percent (assoc-default 'PercentChange q))
                 (name (assoc-default 'Name q))
                 (symbol (assoc-default 'Symbol q))
                 (change (assoc-default 'Change q)))
             (format " %s:%s"
                     (substring
                      (if (or
                          (string-match "=" symbol)
                          (string-match "\\^" symbol))
                          name symbol) 0 3)
                     (if percent percent ""))))
          qs)))
    (setq stock-ticker-display-interval 5)
    (setq stock-ticker-symbols '("^N225" "DOW"))
    (setq stock-ticker-update-interval 300)

    (defun my-activate-stock-ticker (&optional duration)
      "Activate stock-ticker within the given duration."
      (stock-ticker-global-mode 1)
      (unless (numberp duration)
        (setq duration 90))
      (run-with-timer duration nil 'stock-ticker-global-mode -1)))

  (add-hook 'focus-in-hook #'my-activate-stock-ticker))
```

## 6.35. **DONE** [zlc.el] find-file バッファを zsh ライクにする [OBSOLETE]

ファイル選択を zsh ライクに変更できます。しっくりこないので不使用。

```
(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 (map minibuffer-local-map)))
    ;; like menu select
    (define-key map (kbd "C-n") 'zlc-select-next-vertical)
    (define-key map (kbd "C-p") 'zlc-select-previous-vertical)
    (define-key map (kbd "C-f") 'zlc-select-next)
    (define-key map (kbd "C-b") 'zlc-select-previous)
    ;; reset selection
    (define-key map (kbd "C-c") 'zlc-reset)))
```

### 6.36. **DONE** [stripe-buffer.el] テーブルの色をストライプにする [OBSOLETE]

[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 (autoload-if-found
      '(turn-on-stripe-table-mode)
      "stripe-buffer" nil t)

  (add-hook 'org-mode-hook #'turn-on-stripe-table-mode))
```

### 6.37. **DONE** [guide-key] キーバインドの選択肢をポップアップする [OBSOLETE]

`which-key` に移行しました。

自分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て有効化し、`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 (autoload-if-found
      '(guide-key-mode)
      "guide-key" nil t)

  (with-eval-after-load "guide-key"
    (setq guide-key/guide-key-sequence '("C-c f" "C-c f c"))
    (setq guide-key/popup-window-position 'bottom)
    (setq guide-key/idle-delay 0.5)
    (setq guide-key/highlight-command-regexp
          '(("my-" . "red")
            ("takaxp:" . "blue"))))

  (with-eval-after-load "org"
    (guide-key-mode 1))

  (with-eval-after-load "postpone"
    (unless noninteractive
      (postpone-message "guide-key-mode")
      (guide-key-mode 1))))
```

### 6.38. **DONE** [anything.el] 何でも絞り込みインターフェイス [OBSOLETE]

- `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 (autoload-if-found
```

```

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

(with-eval-after-load "anything-startup"
  (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 (memq window-system '(mac 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

;; 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))

```

## 6.39. DONE [diminish.el] モードラインのモード名を短くする [OBSOLETE]

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。

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。代替パッケージに、[rich-minority-mode](#) があります。

メジャーモードの短縮表示は diminish に頼らず、単純に各モードの hook で対処します。

postpone の後に呼んでいるのは、Emacs 起動時の単純な高速化対策です。

```
(with-eval-after-load "postpone"
  (when (require 'diminish nil t)
    (unless noninteractive
      (postpone-message "diminish"))
    (with-eval-after-load "ggtags" (diminish 'ggtags-mode " G"))
    (with-eval-after-load "org" (diminish 'orgstruct-mode " OrgS"))
    (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))
    (with-eval-after-load "all-the-icons-dired"
      (diminish 'all-the-icons-dired-mode))
    (with-eval-after-load "yasnipet" (diminish 'yas-minor-mode))
    (with-eval-after-load "auto-complete" (diminish 'auto-complete-mode))
    (with-eval-after-load "ws-butler" (diminish 'ws-butler-mode))
    (with-eval-after-load "isearch" (diminish 'isearch-mode))
    (with-eval-after-load "autorevert" (diminish 'auto-revert-mode))
    (with-eval-after-load "smooth-scroll" (diminish 'smooth-scroll-mode))
    (with-eval-after-load "whitespace" (diminish 'global-whitespace-mode))
    (with-eval-after-load "emmet-mode" (diminish 'emmet-mode))
    (with-eval-after-load "abbrev" (diminish 'abbrev-mode))
    (with-eval-after-load "doxymacs" (diminish 'doxymacs-mode))
    (with-eval-after-load "editorconfig" (diminish 'editorconfig-mode))
    (with-eval-after-load "rainbow-mode" (diminish 'rainbow-mode))
    (with-eval-after-load "guide-key" (diminish 'guide-key-mode))
    (with-eval-after-load "highlight-symbol" (diminish 'highlight-symbol-mode))
    (with-eval-after-load "which-key" (diminish 'which-key-mode))
    (with-eval-after-load "fancy-narrow" (diminish 'fancy-narrow-mode))
    (with-eval-after-load "smartparens" (diminish 'smartparens-mode))
    (with-eval-after-load "selected" (diminish 'selected-minor-mode))
    ;; (with-eval-after-load "org-autolist" (diminish 'org-autolist-mode))
    ;;; (with-eval-after-load "helm" (diminish 'helm-mode " H"))
  ))
```

;; メジャーモードの短縮

```
(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 "Pr")))
(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 'change-log-mode-hook (lambda () (setq mode-name "ChangeLog")))
(add-hook 'lisp-interaction-mode-hook (lambda () (setq mode-name "Lisp")))
```

## 7. メディアサポート

### 7.1. [emms.el] メディアプレーヤー

mpv を使い , emacs からメディアファイルを再生します . Bongo.el よりも使いやすい印象 .

```
(when (autoload-if-found
      '(emms-play-file
        emms-play-playlist emms-play-directory my-play-bgm
        emms-next emms-previous emms-stop emms-pause)
      "emms" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-c e b") 'my-play-bgm)
    (let ((base "C-c e "))
      (global-set-key (kbd (concat base "n")) 'emms-next)
      (global-set-key (kbd (concat base "p")) 'emms-previous)
      (global-set-key (kbd (concat base "s")) 'emms-stop)
      (global-set-key (kbd (concat base "SPC")) 'emms-pause)))

  (with-eval-after-load "emms-mode-line"
    (defun ad:emms-mode-line-playlist-current ()
      "Display filename in mode-line, not full-path."
      (format emms-mode-line-format
        (file-name-nondirectory
          (emms-track-description
            (emms-playlist-current-selected-track)))))
    (advice-add 'emms-mode-line-playlist-current :override
      #'ad:emms-mode-line-playlist-current))

  (with-eval-after-load "emms-mode-line-icon"
    (setq emms-mode-line-format "%s")
    (defun ad:emms-mode-line-icon-function ()
      "Replace the default musical note icon with a Unicode character."
      (concat " "
        emms-mode-line-icon-before-format
        "♪"
        (emms-mode-line-playlist-current)))
    (advice-add 'emms-mode-line-icon-function :override
      #'ad:emms-mode-line-icon-function))

  (with-eval-after-load "emms"
    (when (require 'emms-setup nil t)
      (emms-standard)
      (emms-default-players))
```

```

(require 'helm-emms nil t)

(defun my-play-bgm ()
  (interactive)
  (let ((file "~/Dropbox/12-audio/ffximusic104.m4a"))
    (when (file-exists-p file)
      (emms-play-file file)
      (emms-toggle-repeat-track))))

;; setup an additional player
(if (executable-find "mpv")
    (when (require 'emms-player-mpv nil t)
      (add-to-list 'emms-player-list 'emms-player-mpv)

      ;; (defvar emms-player-mpv-ontop nil)
      ;; (defun emms-player-mpv-toggle-ontop ()
      ;;   "Toggle float on top."
      ;;   (interactive)
      ;;   (if emms-player-mpv-ontop
      ;;       (emms-player-mpv-disable-ontop)
      ;;       (emms-player-mpv-enable-ontop)))

      ;; (defun emms-player-mpv-enable-ontop ()
      ;;   "Enable float on top."
      ;;   (let ((cmd (emms-player-mpv--format-command "set ontop yes")))
      ;;     (call-process-shell-command cmd nil nil nil))
      ;;   (setq emms-player-mpv-ontop t))

      ;; (defun emms-player-mpv-disable-ontop ()
      ;;   "Disable float on top."
      ;;   (let ((cmd (emms-player-mpv--format-command "set ontop no")))
      ;;     (call-process-shell-command cmd nil nil nil))
      ;;   (setq emms-player-mpv-ontop nil))

      ;; (global-set-key (kbd "C-c e t") 'emms-player-mpv-toggle-ontop)
    )

    (message "--- mpv is NOT installed.))))))

```

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

- ネタとして最高の出来

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

M-x google-maps で起動します .

```

(when (autoload-if-found
      '(google-maps)
      "google-maps" nil t)

  (with-eval-after-load "google-maps"
    (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. [japanlaw.el] Emacs 電子六法

Emacs で総務省の「[法令データ提供システム](#)」に登録された法令データを閲覧します。  
`w3m` が必要です。

- [Emacs 電子六法](#)
- [mhayashi1120/japanlaw.el](http://mhayashi1120/japanlaw.el): Emacs 電子六法

```
(if (executable-find "w3m")
    (autoload-if-found
      '(japanlaw)
      "japanlaw" nil t)

    (message "--- w3m is NOT installed."))
```

### 7.4. [sunshine.el] 天気を知る

- <https://openweathermap.org/> にアカウントを作り，API を呼び出すための専用 ID を発行する必要があります。取得した id を，`sunshine-appid` に格納し，`sunshine-location` で対象地域を設定すれば，`M-x sunshine-forecast` で天気が表示されます。`M-x sunshine-quick-format` を使うと，結果がミニバッファに表示されます。

```
(when (autoload-if-found
      '(sunshine-forecast sunshine-quick-forecast)
      "sunshine" nil t)

    (with-eval-after-load "sunshine"
      ;; (setq sunshine-location "Tokyo, Japan")
      ;; (setq sunshine-appid ".....")
      (custom-set-variables
        '(sunshine-show-icons t)
        '(sunshine-units 'metric))))
```

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

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))
```

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

### [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。

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

  (with-eval-after-load "bongo-mplayer"
    (defun init-bongo ()
      (interactive)
      (bongo)
      (find-file "~/Desktop/next/Tidy/hoge.bongo-playlist")))

  ;; 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)))
```

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

```
(with-eval-after-load "org"
  (require 'org-player nil t))
```

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が保留中

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



## 8. 履歴 / ファイル管理

### 8.1. 履歴サイズを大きくする

`t` で無限大に指定できる .

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

### 8.2. Undo バッファを無限に取る

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

### 8.3. 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>

`recentf-auto-cleanup` を `'mode` などにと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 (autoload-if-found
      '(recentf-mode
        my-recentf-save-list-silence
        my-recentf-cleanup-silence
        recentf-open-files)
      "recentf" nil t)

  (with-eval-after-load "postpone"
    (unless noninteractive
      (postpone-message "recentf-mode")
      (recentf-mode 1)))

  (with-eval-after-load "recentf"

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

    (defun my-recentf-save-list-silence ()
      (interactive)
      (let ((message-log-max nil))
        (if (fboundp 'shut-up)
            (shut-up (recentf-save-list))
            (recentf-save-list)))
      (message ""))
```

```
(defun my-recentf-cleanup-silence ()
  (interactive)
  (let ((message-log-max nil))
    (if shut-up-p
      (shut-up (recentf-cleanup))
      (recentf-cleanup)))
  (message ""))

(add-hook 'focus-out-hook #'my-recentf-save-list-silence)
(add-hook 'focus-out-hook #'my-recentf-cleanup-silence)))

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

## 8.4. バッファ保存時にバックアップを生成させない

[BACKUP]

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

## 8.5. 特定のファイルを Dropbox 以下にバックアップする

[BACKUP]

複数の端末で Emacs を使うと，稀に端末の環境に依存した設定ファイルが必要になります．それを共有してそのまま使うことはできないので，所定の場所での同期を避ける必要があります．私の場合は，`recentf` がそれに該当します．とは言えバックアップしていないのは不安なので，なんとかします．引数にバックアップ対象のファイルリストを渡せます．

事前に `~/Dropbox/backup` の下に，`system-name` で得られる値のディレクトリを作成する必要があります．`my-backup` は [./utility.el](#) で実装しています．

`postpone` の後に呼んでいるのは，Emacs 起動時の単純な高速化対策です．

```
(when (autoload-if-found
      '(my-backup)
      "utility" nil t)

  (with-eval-after-load "postpone"
    (defun my-backup-recentf ()
      (my-backup "~/emacs.d/recentf"))
    (run-with-idle-timer 180 t 'my-backup-recentf)))
```

## 8.6. [backup-each-save.el] クラッシュに備える

[BACKUP]

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，バッファ編集時に `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，`emacs` 起動後に `postpone` モードが有効になる時点で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。

postpone の後に呼んでいるのは、Emacs 起動時の単純な高速化対策です。

```
(when (autoload-if-found
      '(backup-each-save my-auto-backup)
      "backup-each-save" nil t)

  (add-hook 'after-save-hook #'my-auto-backup)

  (with-eval-after-load "backup-each-save"
    (defun my-auto-backup ()
      (unless (equal (buffer-name) "recentf")
        (backup-each-save)))
    (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))

  ;; なぜか (backup-each-save) の直接呼び出しだとだめ
  (with-eval-after-load "postpone"
    (when (require 'backup-each-save nil t)
      (unless noninteractive
        (postpone-message "backup-each-save"))))

  ;; %y-%m-%d_%M:%S で終わるファイルを本来のメジャーモードで開く
  (add-to-list 'auto-mode-alist '("-[0-9-]\\{8\\}_[0-9:]\\{5\\}$" nil t)))

(when (autoload-if-found
      '(recursive-delete-backup-files delete-backup-files)
      "utility" nil t)

  (with-eval-after-load "postpone"
    (recursive-delete-backup-files 7)))
;; (add-hook 'kill-emacs-hook #'my-delete-backup-7days))
```

## 8.7. [dired] ファイラのサポートツール

[DIRE]

dired.el をリッチに使うツール群です。

- gited
- dired-x
- dired-narrow
- dired-du
- helm-dired-history

```
(with-eval-after-load "dired"
  (when (require 'gited nil t)
```

```

(define-key dired-mode-map (kbd "C-x C-g") 'gited-list-branches))
;; https://github.com/Fuco1/dired-hacks

(when (require 'dired-narrow nil t)
  (define-key dired-mode-map (kbd "/") 'dired-narrow))

(require 'dired-du nil t)

(when (require 'dired-x nil t)
  (defun my-reveal-in-finder ()
    "Reveal the current buffer in Finder."
    (interactive)
    (shell-command-to-string "open ."))
  ;; dired-x を読み込んだあとじゃないとだめ
  (define-key dired-mode-map (kbd "M-f") 'my-reveal-in-finder)
  (dired-extra-startup))

;; https://github.com/xuchunyang/emacs.d
;; type "!" or "X" in dired
(when (eq system-type 'darwin)
  (setq dired-guess-shell-alist-user
    (list
      (list (rx (and "."
                    (or
                     ;; Videos
                     "mp4" "avi" "mkv" "rmvb"
                     ;; Torrent
                     "torrent"
                     ;; PDF
                     "pdf"
                     ;; Image
                     "gif" "png" "jpg" "jpeg")
                string-end)) "open")))))

(with-eval-after-load "helm-config"
  (require 'helm-dired-history nil t))

```

## 8.8. [dired-recent.el] 訪問したディレクトリの履歴を取る

[DIREN]

```

(when (autoload-if-found
      '(dired-recent-open dired-recent-mode)
      "dired-recent" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-x C-d") 'dired-recent-open))

  (with-eval-after-load "dired-recent"
    (require 'helm-config nil t)
    (dired-recent-mode 1)))

```

## 8.9. [osx-trash] system-move-file-to-trash を有効にする

[osx-trash](#) は、OSX で `system-move-file-to-trash` を使えるようにする。単独で使うのはあまり考えられないので、読み込みを `dired` に紐付けます。

```

(with-eval-after-load "dired"
  (setq dired-use-ls-dired nil))

```

```
(when (require 'osx-trash nil t)
  (setq delete-by-moving-to-trash t)
  (osx-trash-setup)))
```

## 8.10. [undo-tree] 編集履歴をわかりやすくだる

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 (autoload-if-found
  ' (my-undo-tree-visualize)
  "undo-tree" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-x u") 'my-undo-tree-visualize))

  (with-eval-after-load "undo-tree"
    (global-undo-tree-mode)
    (setq undo-tree-mode-lighter nil) ;; モードライン領域を節約

    (defvar my-undo-tree-active nil)
    (defvar my-undo-tree-width 90)

    (defun my-undo-tree-visualize ()
      (interactive)
      (if (require 'moom nil t)
        (moom-change-frame-width-double)
        (when (and (not my-undo-tree-active)
          (not (eq buffer-undo-list t)))
          (set-frame-width nil (+ (frame-width) my-undo-tree-width))
          (setq my-undo-tree-active t)))
        (undo-tree-visualize)))

    (define-key undo-tree-map (kbd "C-x u") 'my-undo-tree-visualize)

    (defun my-undo-tree-visualizer-quit ()
      (interactive)
      (undo-tree-visualizer-quit)
      (if (require 'moom nil t)
        (moom-change-frame-width-single)
        (delete-window)
        (when my-undo-tree-active
          (set-frame-width nil (- (frame-width) my-undo-tree-width))
          (setq my-undo-tree-active nil)))
        (when (< (frame-width) 80)
          (set-frame-width nil 80)))

    (define-key undo-tree-visualizer-mode-map (kbd "q")
      'my-undo-tree-visualizer-quit)))
```

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

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

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。

`postpone` の後に呼んでいるのは、Emacs 起動時の単純な高速化対策です。

```
(with-eval-after-load "postpone"
  (when (require 'auto-save-buffers nil t)
    (unless noninteractive
      (postpone-message "auto-save-buffers"))

    (defun my-ox-hugo-auto-saving-p ()
      (when (equal major-mode 'org-mode)
        (or (and (boundp 'org-capture-mode) ;; when activating org-capture
                  org-capture-mode)
            (and (fboundp 'org-entry-get)
                  (equal "" (org-entry-get (point) "EXPORT_FILE_NAME"))))))

    (defun my-auto-save-buffers ()
      (cond ((equal major-mode 'undo-tree-visualizer-mode) nil)
            ((equal major-mode 'diff-mode) nil)
            ((string-match "Org Src" (buffer-name)) nil)
            ((my-ox-hugo-auto-saving-p) nil)
            (t
             (auto-save-buffers))))

    (run-with-idle-timer 1.6 t #'my-auto-save-buffers)))
```

## 8.12. [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
- `session-initialize` の呼び出しタイミングに注意．前回終了時の状況を保存しているため，他のパッケージの初期化が終わった後に呼び出すと，パッケージが想定している初期化を上書きしてしまい，不安定になる．`after-init-hook` 推奨．
- [ ] 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 (autoload-if-found
      '(session-initialize)
      "session" nil t)

  (unless noninteractive
    (add-hook 'after-init-hook #'session-initialize))

  (with-eval-after-load "session"
    (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)))

;; 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-momminor-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)))

```

### 8.13. [neotree.el] ディレクトリ情報をツリー表示

```

(when (autoload-if-found
      '(neotree neotree-toggle)
      "neotree" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-c n") #'neotree-toggle))

  (with-eval-after-load "neotree"
    (custom-set-variables
      '(neo-show-hidden-files t)
      '(neo-theme 'arrow)
      '(neo-smart-open t)
      '(neo-window-width 25)
      '(neo-show-hidden-files nil)
      '(neo-window-position 'left))
    ;; (setq neo-vc-integration '(face char)) ;; It's heavy at 2017-08-31

    (when (require 'all-the-icons-dired nil t)
      (setq neo-theme (if (display-graphic-p) 'icons 'arrow)))

    (defvar my-neo-adjusted-window-width (+ 3 neo-window-width))
    (defun ad:neotree-show ()
      "Extension to support change frame width when opening neotree."
      (unless (neo-global--window-exists-p)
        (when (require 'moom nil t)
          (setq moom-frame-width-single
                (+ moom-frame-width-single my-neo-adjusted-window-width))
          (setq moom-frame-width-double
                (+ moom-frame-width-double my-neo-adjusted-window-width)))
        (set-frame-width nil (+ (frame-width) my-neo-adjusted-window-width)))
      (advice-add 'neotree-show :before #'ad:neotree-show))

    (defun ad:neotree-hide ()
      "Extension to support change frame width when closing neotree."
      (when (neo-global--window-exists-p)
        (when (require 'moom nil t)
          (setq moom-frame-width-single
                (- moom-frame-width-single my-neo-adjusted-window-width))
          (setq moom-frame-width-double
                (- moom-frame-width-double my-neo-adjusted-window-width)))
        (set-frame-width nil (- (frame-width) my-neo-adjusted-window-width))
        (when (> 80 (frame-width)) ;; fail safe
          (set-frame-width nil 80)))
      (advice-add 'neotree-hide :before #'ad:neotree-hide)))

```



## 8.14. [helpful.el] リッチなヘルプページ

ヘルプの内容を見やすく構造化して表示してくれます。関数や変数のヘルプだけでなく、キーやマクロなども見やすく表示してくれます。

```
(when (autoload-if-found
      '(helpful-key helpful-function helpful-variable)
      "helpful" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-h k") 'helpful-key)
    (global-set-key (kbd "C-h f") 'helpful-function)
    (global-set-key (kbd "C-h m") 'helpful-macro)
    (global-set-key (kbd "C-h v") 'helpful-variable)))
```

## 8.15. [keyfreq.el] コマンドログ

発行しているコマンドの使用頻度を記録し確認できます。デフォルトで、`~/.emacs.keyfreq` に情報が記録されます。

```
(when (autoload-if-found
      '(keyfreq-mode keyfreq-autosave-mode ad:keyfreq-show)
      "keyfreq" nil t)

  (with-eval-after-load "postpone"
    (unless noninteractive
      (postpone-message "keyfreq-mode")
      (keyfreq-mode 1)))

  (with-eval-after-load "keyfreq"
    (defun ad:keyfreq-show ()
      "Extension to make the buffer view-only."
      (interactive)
      (if shutup-p
          (shut-up (view-buffer keyfreq-buffer))
          (view-buffer keyfreq-buffer)))
    (advice-add 'keyfreq-show :after #'ad:keyfreq-show)
    ;; (define-key keyfreq-mode-map (kbd "q")
    ;;   (lambda () (interactive)
    ;;     (when (string= (buffer-name) keyfreq-buffer)
    ;;       (kill-buffer-and-window))))
    (setq keyfreq-file
          (expand-file-name "~/Dropbox/emacs.d/.keyfreq"))
    (keyfreq-autosave-mode 1)))
```

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

[OBSOLETE]

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

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

[OBSOLETE]

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>

```
(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)))
```

## 8.18. **DONE** バッファ保存時にバックアップファイルを生成する [OBSOLETE]

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)))
```

## 8.19. DONE ミニバッファの履歴を保存しリストアする

[OBSOLETE]

```
(when (require 'savehist nil t)
  ;; ヒストリファイルを明示的に指定
  (setq savehist-file "~/Dropbox/emacs.d/.history")
  (savehist-mode 1))
```

## 8.20. DONE Emacs 終了時に開いていたバッファを起動時に復元する

[OBSOLETE]

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 .

```
(with-eval-after-load "postpone"
  (when (require 'desktop nil t)
    (setq desktop-files-not-to-save "\\(^/tmp\\|\\(^/var\\|\\(^/ssh:\\|\\)")
    (unless noninteractive
      (postpone-message "desktop")
      (desktop-save-mode 1))))
```

## 8.21. DONE 深夜にバッファを自動整理する

[OBSOLETE]

私は頻繁に再起動する派なので , 使っていません .

- <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))
```

# 9. 開発サポート

## 9.1. 便利キーバインド

コメントアウトとコンパイルをすぐ呼べるようにします .

```
(with-eval-after-load "postpone"
  (global-set-key (kbd "C-;") 'comment-dwim) ;; M-; is the default
  (global-set-key (kbd "C-c c") 'compile))
```

## 9.2. [gist.el] Gist インターフェイス

- [defunkt/gist.el: Yet another Emacs paste mode, this one for Gist.](https://github.com/defunkt/gist.el)

事前に `github.user` と `github.oauth-token` を設定します .

```
(autoload-if-found '(gist-mode) "gist" nil t)
```

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

- <http://doxymacs.sourceforge.net/>

```
(when (autoload-if-found
      '(doxymacs-mode)
      "doxymacs" nil t)

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

  (with-eval-after-load "doxymacs"
    (setq doxymacs-doxxygen-style "JavaDoc")
    (add-hook 'font-lock-mode-hook
      (lambda ()
        (when (memq major-mode '(c-mode c++-mode))
          (doxymacs-font-lock))))
    (define-key doxymacs-mode-map (kbd "C-c C-s") 'ff-find-other-file)))
```

## 9.4. [matlab.el] Matlab 用の設定

```
(when (and (memq window-system '(mac ns))
          (> emacs-major-version 23))
  (when (autoload-if-found
        '(matlab-mode matlab-shell)
        "matlab" nil t)
    (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 (autoload-if-found
      '(flycheck-mode)
      "flycheck" nil t)

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

  (with-eval-after-load "flycheck"
    (require 'helm-flycheck nil t)
    (setq flycheck-gcc-language-standard "c++14")
    (setq flycheck-clang-language-standard "c++14")
    ;; TODO: really needed?
    ;; (when (require 'flycheck-clang-tidy nil t)
```

```
;; (add-hook 'flycheck-mode-hook #'flycheck-clang-tidy-setup))
;; http://qiita.com/sendakiha/items/cddb02cfdbc0c8c7bc2b
;; (when (require 'flycheck-pos-tip nil t)
;;   '(custom-set-variables
;;     '(flycheck-display-errors-function
;;       #'flycheck-pos-tip-error-messages)))
;; )

;; (flycheck-add-next-checker 'javascript-jshint
;;   'javascript-gjslint)
```

## 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 .

auto-complete が正しく効いているかは , バッファ内で適当にパスを打ち込んで , 補完候補が表示されるかで判定判定できると思います ( /home を入力とか )

```
(when (autoload-if-found
      '(ac-default-setup ac-org-mode-setup)
      "auto-complete" nil t)

  (with-eval-after-load "postpone"
    (dolist (hook
              (list 'org-mode-hook '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 "*" (buffer-name))
```

```

                                (ac-default-setup)))
  (add-hook 'org-mode-hook #'ac-org-mode-setup))

(with-eval-after-load "auto-complete"
  (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 2.0)
  ;; 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")
    ;; ac-source-words-in-same-mode-buffers
    (setq ac-sources '(ac-source-filename
      ac-source-abbrev
      ac-source-dictionary

```

## 9.7. [auto-complete-clang.el] オム二補完

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る，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 -std=c++14 -stdlib=libstdc++ -emit-pch ~/ .emacs.d/stdafx.h -o ~/ .emacs.d/stdafx.pch -I/opt/local/include -I/opt/local/include/netpbm -I/Users/taka/devel/icp/lib
```

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 (autoload-if-found
      '(auto-complete ac-cc-mode-setup)
      "auto-complete" nil t)

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

  (with-eval-after-load "auto-complete"
    (when (require 'auto-complete-clang nil t)
      (setq ac-clang-executable (executable-find "clang"))
      ;; 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 '("-std=c++14" "-w" "-ferror-limit" "1"
                               "-fcxx-exceptions"))
        (setq ac-sources '(ac-source-clang
                           ac-source-yasnipet
                           ac-source-gtags))))))
```

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. References

- <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-completeel Emacs.html>
- <http://www.pluginmasters.com.br/pluginfeed/post/73768/awesome-cc-autocompletion-in-emacs>

### 9.8. [origami.el] 関数の折りたたみ

- `hideshowvis.el` よりも軽い印象なので、現在は `origami.el` を使用中。
- [gregsexton/origami.el: A folding minor mode for Emacs](#)
- [zenozeng/yafolding.el: Yet another folding extension for Emacs](#)
- [outline-magic/outline-magic.el at master · tj64/outline-magic](#)
- 後発として [bicycle.el](#) があります。

```
(when (autoload-if-found
      '(origami-mode origami-toggle-node)
      "origami" nil t)

  (dolist (hook '(emacs-lisp-mode-hook c-mode-common-hook yatex-mode-hook))
    (add-hook hook #'origami-mode))

  (with-eval-after-load "origami"
    (define-key origami-mode-map (kbd "C-t") #'origami-toggle-node)
    (define-key origami-mode-map (kbd "C-u C-t")
      #'origami-toggle-all-nodes)))
```

### 9.9. [quickrun.el] お手軽ビルド

カレントバッファで編集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 (autoload-if-found
      '(quickrun)
      "quickrun" nil t)

  (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)))
(add-hook 'gnuplot-mode-hook
  (lambda () (define-key gnuplot-mode-map (kbd "<f5>") 'quickrun)))
```

## 9.10. [ggtags.el] タグジャンプ

- [Emacs における C/C++ の環境を整える - tasuwo blog](#)

```
(if (not (executable-find "gtags"))
  (message "--- gtags is NOT installed in this system.")

  (when (autoload-if-found
    '(ggtags-mode)
    "ggtags" nil t)

    (with-eval-after-load "ggtags"
      (setq ggtags-completing-read-function t)
      (define-key ggtags-mode-map (kbd "M-]") nil))

    (dolist (hook (list 'c-mode-common-hook))
      (add-hook hook (lambda () (ggtags-mode 1))))))

(when (autoload-if-found
  '(helm-gtags-mode)
  "helm-gtags" nil t)

  (add-hook 'c-mode-common-hook #'helm-gtags-mode)

  (with-eval-after-load "helm-gtags"
    (custom-set-variables
      '(helm-gtags-mode-name ""))))))
```

## 9.11. [0xc.el] N 進数変換

- 実施頻度の高い 16 進数と 10 進数の相互変換に重宝します .

```
(when (autoload-if-found
  '(0xc-convert 0xc-convert-point my-decimal-to-hex my-hex-to-decimal)
  "0xc" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-c f h") '0xc-convert))

  (with-eval-after-load "0xc"
    (defun my-decimal-to-hex ()
      (interactive)
      (0xc-convert 16 (word-at-point)))
    (defun my-hex-to-decimal ()
      (interactive)
      (0xc-convert 10 (word-at-point))))))
```

## 9.12. [hexl.el] バイナリファイルを開く

ビルトインの `hexl-mode` を使います .

- [GNU Emacs Manual\(Japanese Translation\): Editing Binary Files](#)

```
(with-eval-after-load "hexl"
  (custom-set-variables
    '(hexl-bits 8)))
```

## 9.13. [uuid.el] UUID の生成

- [nicferrier/emacs-uuid: The UUID module from the EmacsWiki, initially by someone called James Mastros.](#)

新しい UUID を生成してカーソル位置に書き出すように `my-uuid-string` を定義しています .

```
(when (autoload-if-found
      '(uuid-string my-uuid-string)
      "uuid" nil t)

  (with-eval-after-load "uuid"
    (defun my-uuid-string ()
      (interactive)
      (insert (uuid-string)))))
```

## 9.14. [package-lint.el] MEPLA 登録用 Lint

MELPA への登録を目指す emacs-lisp パッケージの実装で , 所定の書式で記述されているかを確認できます . `docstring` のチェックは , `M-x checkdoc` でできます .

- <https://github.com/purcell/package-lint>

```
(autoload-if-found '(package-lint-current-buffer) "package-lint" nil t)
```

## 9.15. [projectile.el] ディレクトリ単位でファイル群をプロジェクト扱いする

- プロジェクト内に限定して検索をかける時に重宝する .
- カレントバッファがプロジェクト内のファイルの場合は , タイトルバーにプロジェクト名を出すように設定しています .

```
(when (autoload-if-found
      '(projectile-mode)
      "projectile" nil t)
```

```
(with-eval-after-load "postpone"
  (unless noninteractive
    (postpone-message "projectile-mode")
    (setq projectile-keymap-prefix (kbd "C-c p"))
    (projectile-mode 1)))

(with-eval-after-load "neotree"
  ;; M-x helm-projectile-switch-project (C-c p p)
  (setq projectile-switch-project-action 'neotree-projectile-action)

  (defun ad:neotree-dir (path)
    "Extension to change the frame width automatically."
    (interactive "DDirectory: ")
    (unless (neo-global--window-exists-p)
      (neotree-show))
    (neo-global--open-dir path)
    (neo-global--select-window))
  (advice-add 'neotree-dir :override #'ad:neotree-dir))

(with-eval-after-load "projectile"
  (defun ad:projectile-visit-project-tags-table ()
    "Extensions to skip calling `visit-tags-table'."
    nil)
  (advice-add 'projectile-visit-project-tags-table :override
    #'ad:projectile-visit-project-tags-table)

  (setq projectile-mode-line-lighter "")
  (setq projectile-dynamic-mode-line nil)
  (setq projectile-tags-command "gtags")
  (setq projectile-tags-backend 'ggtags)
  (setq projectile-tags-file-name "GTAGS")

  (setq projectile-use-git-grep t)
  ;; (setq projectile-mode-line
  ;;   '(:eval (format " P:%s" (projectile-project-name))))
  (setq projectile-mode-line "")

  (setq icon-title-format
    (setq frame-title-format
      '(:eval
        (let ((project-name (projectile-project-name)))
          (unless (string= "-" project-name)
            (format "(%s) - " project-name))))
        "%b")))))
```

## 9.16. TODO [EditorConfig] コードスタイルの強制

- [EditorConfig](#)
- [editorconfig/editorconfig-emacs: EditorConfig plugin for emacs](#)
- [10sr/editorconfig-charset-extras-el: Extra EditorConfig Charset Support](#)

postpone の後に呼んでいるのは , Emacs 起動時の単純な高速化対策です .

```
(with-eval-after-load "postpone"
```

```
(if (executable-find "editorconfig")
    (when (require 'editorconfig nil t)
        (unless noninteractive
            (postpone-message "editorconfig")
            ;; (add-to-list 'editorconfig-exclude-modes 'org-mode)
            ;; (when (require 'editorconfig-charset-extras nil t)
            ;;     (add-hook 'editorconfig-custom-hooks
            ;;               #'editorconfig-charset-extras))
            (editorconfig-mode 1)))
    (message "editorconfig is NOT installed.")))
```

例えば，次のようなファイルを共有プロジェクトに保存しておきます．`.editorconfig` として配置します．

```
root = true

[*]
charset = utf-8
end_of_line = lf
insert_final_newline = true
indent_style = space
indent_size = 2
trim_trailing_whitespace = true
```

## 9.17. TODO [cov] カバレッジの状態をフリンジで確認

gcov の結果をフリンジに表示します．

- <https://github.com/adamniederer/cov>

```
(autoload-if-found '(cov-mode) "cov" nil t)
```

## 9.18. TODO [magit.el] Git クライアント

```
(when (autoload-if-found
      '(magit-status)
      "magit" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-c m") 'magit-status)))
```

## 9.19. TODO [format-all.el] コード整形

- [format-all - MELPA](#)

様々なプログラミング言語に対して，共通のコード整形のトリガー ( `M-x format-all-buffer` ) を提供します．整形するために使うフォーマッタは，原則的に言語ごとに異なる外部ツールを呼び出すので，必要に応じてそれらのツールを別途インストールします．

```
(with-eval-after-load "postpone"
```

```
(require 'format-all nil t))
```

## 9.20. TODO [emr.el] リファクタリング

```
(with-eval-after-load "postpone"
  (when (require 'emr nil t)
    (define-key prog-mode-map (kbd "M-RET") 'emr-show-refactor-menu)
    (emr-initialize)))
```

## 9.21. DONE [hideshowvis.el] 関数の表示 / 非表示

[OBSOLETE]

- <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 (memq window-system '(mac ns))
  (> emacs-major-version 23))
  (when (autoload-if-found
    '(hideshowvis-enable hideshowvis-minor-mode)
    "hideshowvis" nil t)

    (dolist (hook (list 'perl-mode-hook 'c-mode-common-hook))
      (add-hook hook #'hideshowvis-enable))

    (add-hook 'emacs-lisp-mode-hook
      (lambda () (unless (equal "*scratch*" (buffer-name))
        (hideshowvis-enable))))

    (with-eval-after-load "hideshowvis"
      (define-key hideshowvis-mode-map (kbd "C-(") 'hs-hide-block)
      (define-key hideshowvis-mode-map (kbd "C-)" 'hs-show-block))))
```

# 10. Org Mode

## 10.1. 基本設定

Org Mode は巨大なパッケージなので、設定項目が必然的に増えます。一度に全部を設定しようとせず、必須機能と見た目の設定から入り、興味があり使いたい機能の設定へと徐々に拡張するのが近道です。

```
(when (autoload-if-found
  '(org-mode)
  "org" "Org Mode" t)

  ;; テキストファイルを Org Mode で開きます。
  (push '("\\.txt$" . org-mode) auto-mode-alist)

  ;; Font lock を使う
  (add-hook 'org-mode-hook #'turn-on-font-lock)

  ;; ホームポジション的な Org ファイルを一発で開きます。
  (global-set-key (kbd "C-M-o"))
```

```

(lambda () (interactive)
  (my-show-org-buffer "next.org"))

(with-eval-after-load "postpone"
  (global-set-key (kbd "C-c r") 'org-capture)
  (global-set-key (kbd "C-c l") 'org-store-link)
  (global-set-key (kbd "C-c a") 'org-agenda))

(with-eval-after-load "org"
  ;; 関連モジュールの読み込み
  (require 'org-habit nil t)
  (require 'org-mobile nil t)
  (require 'org-eldoc nil t) ;; org-eldoc を読み込む
  (unless noninteractive
    (require 'org-emms nil t)) ;; emms のリンクに対応させる

  (add-to-list 'org-modules 'org-id)
  (add-to-list 'org-modules 'ox-odt)
  (when (version< "9.1.4" (org-version))
    (add-to-list 'org-modules 'org-tempo))

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

  ;; 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 'time) ; 'time 以外に , '(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)

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

  ;; 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)

  ;; 数式をハイライト
  (setq org-highlight-latex-and-related '(latex entities))

  ;; 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 "");

;; org-clock の計測時間をモードラインではなくタイトルに表示する
(setq org-clock-clocked-in-display 'frame-title)

;; 1 分未満は記録しない
(setq org-clock-out-remove-zero-time-clocks t)

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

;; 非表示状態の領域への書き込みを防ぐ
;; "Editing in invisible areas is prohibited, make them visible first"
(setq org-catch-invisible-edits 'show-and-error)
(defun ad:org-return (f &optional arg)
  "An extension for checking invisible editing when you hit the enter."
  (interactive "P")
  (org-check-before-invisible-edit 'insert)
  (apply f arg))
(advice-add 'org-return :around #'ad:org-return)

;; - 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." . "-")) )

;; 完了したタスクの配色を変える
;; https://fucol.github.io/2017-05-25-Fontify-done-checkbox-items-in-org-mode.html
(font-lock-add-keywords
 'org-mode
  `((("^[ \t]*\\(?:[-+*]\\| [0-9]+[)]\\.\\| [ \t]+\\(\\(?:\\[ @\\(?:start:\\|)?[0-9]+\\| [ \t]*\\)?\\[\\(?:X\\|\\([0-9]+\\)/\\2\\)\\][^\\n]*\\n\\)"
    1 'org-headline-done prepend))
    'append))

(defun my-do-org-update-statistics-cookies ()
  (interactive)
  (message "Update statistics...")
  (do-org-update-statistics-cookies)
  (message "Update statistics...done"))
(define-key org-mode-map (kbd "C-c f 2")
  'my-do-org-update-statistics-cookies)

;; C-c & が yasnipet 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)

;; (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))

(with-eval-after-load "org-clock"
  (defun my-org-clock-out-and-save-when-exit ()
    "Save buffers and stop clocking when kill emacs."
    (when (org-clocking-p)
      (org-clock-out)
      (save-some-buffers t)))
  (add-hook 'kill-emacs-hook #'my-org-clock-out-and-save-when-exit)))
```

## 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

```
;; ~/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
(when (autoload-if-found
      ' (my-ox-icalendar my-ox-icalendar-cleanup)
      "ox-icalendar" nil t)

  (with-eval-after-load "org"
    (define-key org-mode-map (kbd "C-c f l") 'my-ox-icalendar))

  (with-eval-after-load "ox-icalendar"
    ;; 生成するカレンダーファイルを指定
    ;; 以下の設定では、このファイルを一時ファイルとして使う (削除する)
    (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))

    (defun my-ox-icalendar ()
      (interactive)
      (let ((message-log-max nil)
            (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))

;; 自サーバにアップロード
(message "Uploading...")
(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))))))

```

## 10.4. スピードコマンド

慣れてくると Org Mode の通常キーバインドでは満足できなくなります。コンテキストに応じて適切なキーバインドを設定するのが Emacs の醍醐味ですから、そういう欲求が出てきたらスピーコマンドの出番です。シングルキーで各ツリーのステータスを変更したり、任意のコマンドを呼び出せます。

```

(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
               '("D" my-org-todo-complete-no-repeat "DONE"))
  ;; (add-to-list 'org-speed-commands-user '("N" org-shiftmetadown))
  ;; (add-to-list 'org-speed-commands-user '("P" org-shiftmetaup))
  (add-to-list 'org-speed-commands-user '("! " my-org-set-created-property))
  (add-to-list 'org-speed-commands-user
               '("$ " call-interactively 'org-archive-subtree))

  ;; 周期タクスを終了させます。
  (defun my-org-todo-complete-no-repeat (&optional ARG)
    (interactive "P")
    (when (org-get-repeat)
      (org-cancel-repeater))
    (org-todo ARG)))

```

## 10.5. face 関連

Org バッファは日々のタスク管理で閲覧する頻度が高くと期間が長いです。好みの状態にカスタマイズして使いましょう。

```

(with-eval-after-load "org"

```

```

;; Font lock を使う
(global-font-lock-mode 1)

;; ウィンドウの端で折り返す
(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")
    ("CHECK" :foreground "#FF9900" :background "#FFF0F0" :underline t)
    ("ICAL" :foreground "#33CC66")
    ("APPROVED" :foreground "#66CC66")
    ("QUESTION" :foreground "#FF0000")
    ("WAIT" :foreground "#CCCCCC" :background "#666666")
    ("MAIL" :foreground "#CC3300" :background "#FFEE99")
    ("PLAN" :foreground "#FF6600")
    ("PLAN2" :foreground "#FFFFFF" :background "#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")
    ("Bug" :foreground "#FF0000")
    ("Report" :foreground "#66CC66")
    ("Background" :foreground "#66CC99")
    ("Chore" :foreground "#6699CC")
    ("Domestic" :foreground "#6666CC")
    ("BeMerged" :foreground "#6666CC")
    ("Doing" :foreground "#FF0000")
    ("Draft" :foreground "#9933CC") ; Draft(r1,r2,r3) -> Review(1,2)
    ("Review" :foreground "#6633CC")
    ("Revisit" :foreground "#6633CC")
    ("Redmine" :foreground "#CC6666")
    ("Ongoing" :foreground "#CC6666") ; for non scheduled/reminder
    ("Repeat" :foreground "#CC9999") ; for interval tasks
    ("Mag" :foreground "#9966CC")
    ("buy" :foreground "#9966CC")
    ("pay" :foreground "#CC6699")
    ("try" :foreground "#FF3366")
    ("secret" :foreground "#FF0000")
    ("emacs" :foreground "#6633CC")
    ("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 "#9966CC")
("story"      :foreground "#FF7D7D")
("Test"       :foreground "#FF0000" :weight bold)
("Attach"     :foreground "#FF0000" :underline t :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"))

```

## 10.6. 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)" "PLAN (p)" "PLAN2 (P)" "|" "DONE (d) ")
      (sequence "FOCUS (f)" "CHECK (C)" "ICAL (c)" "|" "DONE (d) ")
      (sequence "WAIT (w)" "SLEEP (s)" "QUESTION (q)" "|" "DONE (d) ")
      (sequence "REV1 (1)" "REV2 (2)" "REV3 (3)" "|" "APPROVED (a@/!) ") ))

;; 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)))

```

## 10.7. 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しますが，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)
```

## 10.8. README を常に org-mode で開く

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

  (push '("[rR][eE][aA][dD][mM][eE]" . org-mode) auto-mode-alist))
```

## 10.9. ソースブロック等の大文字記述を一括で小文字に変える

<https://scripter.co/org-keywords-lower-case/> で紹介されている関数を，ログ出力部分だけ加筆して使っています．

```
(with-eval-after-load "org"
  (defun my-lowercase-org-keywords ()
    "Lower case Org keywords and block identifiers."
    (interactive)
    (save-excursion
      (goto-char (point-min))
      (let ((case-fold-search nil)
            (count 0))
        (while (re-search-forward
                  "\\(?:1:#[\\+][A-Z_]+\\(?:_[:alpha:]]+\\)*\\)\\(?:[:[ :=~'"]\\|
$\\)\""
                  nil :noerror)
          (setq count (1+ count))
          (let* ((prev (match-string-no-properties 1))
                 (new (downcase prev)))
            (replace-match new :fixedcase nil nil 1)
            (message "Updated(%d): %s => %s" count prev new)))
          (message "Lower-cased %d matches" count))))
```

## 10.10. イベント通知

ここでは，`terminal-notifier` ではなく，事実上，後継アプリと言える `alerter` を使う設定です．

macOS 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．`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．appt-disp-window をカスタマイズすれば，appt を経由して TODO のアイテムも通知されます．

ns-notification-sound に OS で定められたアラーム音を設定できます．音楽を流したい時は，org-show-notification-handler の引数で指定すると設定できます．

	sticky	sound	Function
pomodoro	nil	Glass	my-pomodoro-notify
org-notify	t	default	my-desktop-notification-handler
trigger.org	select	default	my-desktop-notify
appt	select	Glass/Pop	ad:appt-disp-window
<b>Note</b>			
pomodoro		pomodoro loop	
org-notify		org からの通知	
trigger.org		Alarms on a table	
appt		TODO items with deadline	

```
(with-eval-after-load "postpone"
  ;; Select from Preferences: { Funk | Glass | ... | Purr | Pop ... }
  (defvar ns-default-notification-sound "Pop")

  (defvar ns-alerter-command
    (executable-find "/Users/taka/Dropbox/bin/alerter")
    "Path to alerter command. see https://github.com/vjeantet/alerter")

  (defun my-desktop-notification (title message &optional sticky sound timeout)
    "Show a message by `alerter' command."
    (start-process
      "notification" "*notification*"
      ns-alerter-command
      "-title" title
      "-message" message
      "-sender" "org.gnu.Emacs"
      "-timeout" (format "%s" (if sticky 0 (or timeout 7)))
      "-sound" (or sound ns-default-notification-sound)))

  ;; eval (org-notify "hoge") to test this setting
  (defun my-desktop-notification-handler (message)
    (my-desktop-notification "Message from org-mode" message t))
```

```
(with-eval-after-load "org"
  (when ns-alerter-command
    (setq org-show-notification-handler #'my-desktop-notification-handler))))
```

## 10.11. 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。

[2018-03-19 Mon]: alerter に統一しました。

通知アプリと org-mode のバッファを組み合わせで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通知アプリがそのアラームをデスクトップに表示してくれます。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、通知アプリが 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。

my-set-alarms-from-file は、[utility.el](#) に記述した関数です。

```
(with-eval-after-load "org"
  (when (require 'utility nil t)
    (let ((trigger-file "~/Dropbox/org/db/trigger.org"))
      (when (file-exists-p trigger-file)
        (my-set-alarms-from-file "~/Dropbox/org/db/trigger.org") ;; init
        (add-hook 'after-save-hook #'my-update-alarms-from-file)))) ;; update
```

## 10.12. [org-capture] 高速にメモを取る

Emacs を起動している限り、いつでもどこでもメモを記録できます。

```
(when (autoload-if-found
  '(org-capture)
  "org-capture" nil t)

  (with-eval-after-load "org"
    ;; キャプチャ時に作成日時をプロパティに入れる
    ;; Thanks to https://emacs.stackexchange.com/questions/21291/add-created-timestamp-to-logbook
```

```

(defvar my-org-created-property-name "CREATED"
  "The name of the org-mode property.
This user property stores the creation date of the entry")
(defun my-org-set-created-property (&optional active NAME)
  "Set a property on the entry giving the creation time.

By default the property is called CREATED. If given the `NAME'
argument will be used instead. If the property already exists, it
will not be modified."
  (interactive)
  (let* ((created (or NAME my-org-created-property-name))
        (fmt (if active "<%s>" "[%s]")))
    (now (format fmt (format-time-string "%Y-%m-%d %a %H:%M"))))
    (unless (org-entry-get (point) created nil)
      (org-set-property created now))))

(with-eval-after-load "org-capture"
  (defun my-org-toggle-block-visibility ()
    "Testing..."
    (interactive)
    (when (looking-at org-drawer-regexp)
      (org-flag-drawer ; toggle block visibility
        (not (get-char-property (match-end 0) 'invisible)))))

  (add-hook 'org-capture-before-finalize-hook #'my-org-set-created-property)

;; 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-academic-file (concat org-directory "academic.org"))
(defvar org-capture-ical-file (concat org-directory "org-ical.org"))
(defvar org-capture-buffer-file (concat org-directory "db/buffer.org"))
(defvar org-capture-notes-file (concat org-directory "db/note.org"))
(defvar org-capture-english-file (concat org-directory "db/english.org"))
(defvar org-capture-diary-file (concat org-directory "log/diary.org"))
(defvar org-capture-article-file (concat org-directory "db/article.org"))
(defvar org-capture-blog-file
  (concat org-directory "blog/entries/imadenale.org"))

;; see org.pdf:p73
(setq org-capture-templates
  `(("t" "TODO 項目を INBOX に貼り付ける" entry
    (file+headline ,org-default-notes-file "INBOX") "*** TODO %?\n\t")
    ("a" "記事リストにエントリー" entry
    (file+headline ,org-capture-article-file "INBOX")
    "*** READ %?\n\t")
    ("c" "同期カレンダーにエントリー" entry
    (file+headline ,org-capture-ical-file "Scheduled")
    "*** TODO %?\n\t")
    ("d" "Doing タグ付きのタスクを Inbox に投げる" entry
    (file+headline ,org-default-notes-file "INBOX")
    "*** TODO %? :Doing:\n - \n")
    ("l" "本日のチェックリスト" entry
    (file+headline ,org-capture-diary-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" "アイデアを書き込む" entry (file+headline ,org-default-notes-
    file "INBOX"))

```

```

    *** %?\n - \n\t%U")
    ("b" "Create new post for imadenale blog" entry
      (file+headline ,org-capture-blog-file ,(format-time-string "%Y"))
      *** TODO \n:PROPERTIES:\n:EXPORT_FILE_NAME: %?\n:EXPORT_HUGO_TAGS:
\n:END:\n")
    ("B" "Create new post for imadenale blog (UUID)" entry
      (file+headline ,org-capture-blog-file ,(format-time-string "%Y"))
      *** TODO %?\n:PROPERTIES:\n:EXPORT_FILE_NAME: %(uuid-string)\
\n:EXPORT_HUGO_TAGS: \n:END:\n")
    ;; ("b" "Bug タグ付きの TODO 項目を貼り付ける" entry
    ;;   (file+headline ,org-default-notes-file "INBOX")
    ;;   *** TODO %? :bug:\n %i\n %a %t")
    ("T" "時間付きエントリー" entry (file+headline ,org-default-notes-file
"INBOX")
      *** %? %T--\n")
    ("n" "ノートとして INBOX に貼り付ける" entry
      (file+headline ,org-default-notes-file "INBOX")
      *** %? :note:\n#\n - \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-academic-file
      " に書き込む")
      entry (file+headline ,org-capture-academic-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---")
    ("w" ,(concat "英単語を " org-capture-english-file
      " に書き込む") entry
      (file+headline ,org-capture-english-file "WORDS")
      *** %? :%(get-current-date-tags):\n「」\n - ")
    ("g" ,(concat "英語ノートを " org-capture-english-file
      " に書き込む")
      entry (file+headline ,org-capture-english-file "GRAMMER")
      *** %? :%(get-current-date-tags):\n\n%U")
    ))))

```

### 10.13. [org-agenda] タスク / 予定管理

入門者にとって org-agenda は簡単ではないので、無理に使わなくて良いと思います。  
Org Mode の動作に十分慣れたら戻ってくる程度で十分です。

```

(with-eval-after-load "org-agenda"
  ;; アジェンダ作成対象 (指定しないと agenda が生成されない)
  ;; ここを間違えると, MobileOrg, iCal export もうまくいらない
  (setq org-agenda-files
    '("~/Dropbox/org/org-ical.org" "~/Dropbox/org/next.org" "~/Dropbox/org/
db/cooking.org"
      "~/Dropbox/org/db/daily.org" "~/Dropbox/org/minutes/wgl.org"
      "~/Dropbox/org/tr/work.org" "~/Dropbox/org/academic.org"
      "~/Dropbox/org/db/trigger.org" "~/Dropbox/org/org2ja.org"))

```



```

;; 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 0)

;; 時間幅が明示的に指定されない場合のデフォルト値 (分指定)
(setq org-agenda-default-appointment-duration 60)

;; アジェンダビューで FOLLOW を設定 (自動的に別バッファに当該タスクを表示)
(setq org-agenda-start-with-follow-mode t)

;; Customized Time Grid
(setq org-agenda-time-grid ;; Format is changed from 9.1
  '((daily today require-timed)
    (0800 1000 1200 1400 1600 1800 2000 2200 2400)
    "....."
    "-----"
  ))

;; (setq org-agenda-current-time-string "< d(' - ' ｲﾚｯ")
(setq org-agenda-current-time-string "< < < < < < < < < < < < < ｲﾚｯ")
(setq org-agenda-timegrid-use-ampm t)

(with-eval-after-load "moom"
  ;; Expand the frame width temporarily during org-agenda is activated.
  (defun my-agenda-frame-width ()
    (moom-change-frame-width
      (floor (* 1.2 moom-frame-width-single))))
  (add-hook 'org-agenda-mode-hook #'my-agenda-frame-width)

  (defun ad:org-agenda--quit (&optional _bury)
    (moom-change-frame-width))
  (advice-add 'org-agenda--quit :after #'ad:org-agenda--quit))

(add-hook 'org-finalize-agenda-hook #'my-org-agenda-to-appt)

;; 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))))

(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))))

```

```

)

;; 所定の時刻に強制的に Agenda を表示
(defvar my-org-agenda-auto-popup-list
  '("01:00" "11:00" "14:00" "17:00" "20:00" "23:00"))
(defun my-popup-agenda ()
  (interactive)
  (let ((status use-dialog-box))
    (setq use-dialog-box nil)
    (when (y-or-n-p-with-timeout "Popup agenda now?" 10 nil)
      (org-agenda-list))
    (message "")
    (setq use-dialog-box status)))
(defun my-popup-agenda-set-timers ()
  (interactive)
  (cancel-function-timers 'my-popup-agenda)
  (dolist (trigger my-org-agenda-auto-popup-list)
    (when (future-time-p trigger)
      (run-at-time trigger nil 'my-popup-agenda))))
(my-popup-agenda-set-timers)
(run-at-time "24:00" nil 'my-popup-agenda-set-timers)) ;; for next day

;; Doing 管理
(with-eval-after-load "org"
  (define-key org-mode-map (kbd "<f11>") 'my-toggle-doing-tag)
  (define-key org-mode-map (kbd "C-<f11>") 'my-sparse-doing-tree)

  ;; 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
          (org-back-to-heading t)
          ;; before 9
          ;; (unless (org-at-heading-p)
          ;;   (outline-previous-heading))
          (org-toggle-tag my-doing-tag
            (if (string-match
                  (concat ":" my-doing-tag ":")
                  (org-get-tags-string))
                'off 'on))))
      (org-reveal)))

  ;; ついでに calendar.app も定期的に強制起動する
  (defun my-popup-calendar ()
    (interactive)
    (if (and (eq system-type 'darwin)
              (window-focus-p))
        (shell-command-to-string "open -a calendar.app")
        (message "--- input focus is currently OUT.)))

  (defun my-popup-calendar-set-timers ()
    (interactive)

```

```
(cancel-function-timers 'my-popup-calendar)
(dolist (trigger my-org-agenda-auto-popup-list)
  (when (future-time-p trigger)
    (run-at-time trigger nil 'my-popup-calendar))))

(when (memq window-system '(mac ns))
  (my-popup-calendar-set-timers)
  (run-at-time "24:00" nil 'my-popup-calendar-set-timers))) ;; for next day
```

## 10.14. [orgbox.el] スケジュール追加のわかりやすい入力

C-c C-s をオーバーライドして orgbox-schedule を実行する .

```
(with-eval-after-load "org"
  (require 'orgbox nil t))
```

## 10.15. [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 .

```
(when (autoload-if-found
      '(appt my-org-agenda-to-appt ad:appt-display-message
            ad:appt-disp-window)
      "appt" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-c f 3") #'my-org-agenda-to-appt))

  (with-eval-after-load "appt"
    ;; window を フレーム内に表示する
    (setq appt-display-format 'echo)

    ;; 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 20)

    ;; 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 1)

    ;; appt-display-format が 'echo でも appt-disp-window-function を呼ぶ
    (defun ad:appt-display-message (string mins)
      "Display a reminder about an appointment.
The string STRING describes the appointment, due in integer MINS minutes.
The arguments may also be lists, where each element relates to a
separate appointment. The variable `appt-display-format' controls
the format of the visible reminder. If `appt-audible' is non-nil,
also calls `beep' for an audible reminder."
      (if appt-audible (beep 1))
      ;; Backwards compatibility: avoid passing lists to a-d-w-f if not
```

```

necessary.
  (and (listp mins)
        (= (length mins) 1)
        (setq mins (car mins)
                  string (car string)))
  (when (memq appt-display-format '(window echo))
    (let ((time (format-time-string "%a %b %e "))
          err)
      (condition-case err
        (funcall appt-disp-window-function
                  (if (listp mins)
                      (mapcar 'number-to-string mins)
                      (number-to-string mins))
                  time string)
        (wrong-type-argument
         (if (not (listp mins))
             (signal (car err) (cdr err))
             (message "Argtype error in `appt-disp-window-function' - \
update it for multiple appts?")
             ;; Fallback to just displaying the first appt, as we used to.
             (funcall appt-disp-window-function
                      (number-to-string (car mins)) time
                      (car string))))))
      err))
  (cond ((eq appt-display-format 'window)
        ;; TODO use calendar-month-abbrev-array rather than %b?
        (run-at-time (format "%d sec" appt-display-duration)
                     nil
                     appt-delete-window-function))
        ((eq appt-display-format 'echo)
         (message "%s" (if (listp string)
                           (mapconcat 'identity string "\n")
                           string))))))
(advice-add 'appt-display-message :override #'ad:appt-display-message)

(defun ad:appt-disp-window (min-to-app _new-time appt-msg)
  "Extension to support appt-disp-window."
  (if (string= min-to-app "0")
      (my-desktop-notification "### Expired! ###" appt-msg t "Glass")
      (my-desktop-notification
       (concat "in " min-to-app " min.") appt-msg nil "Tink")))
(cond
 ((eq appt-display-format 'echo)
  (setq appt-disp-window-function 'ad:appt-disp-window))
 ((eq appt-display-format 'window)
  (advice-add 'appt-disp-window :before #'ad:appt-disp-window))))

(with-eval-after-load "org"
  ;; アラーム表示を有効にする
  (unless noninteractive
    (add-hook 'org-agenda-mode-hook #'my-org-agenda-to-appt) ;; init
    (appt-activate 1))

  ;; org-agenda の内容をアラームに登録する
  ;; 定期的に更新する
  (defun my-org-agenda-to-appt ()
    (interactive)
    (org-agenda-to-appt t '(headline "TODO"))))
  (run-with-idle-timer 500 t 'my-org-agenda-to-appt)
  (add-hook 'org-capture-before-finalize-hook #'my-org-agenda-to-appt)

```

```
))
```

## 10.16. [org-refile] org ツリーの高速移動

```
(with-eval-after-load "org"
  ;; 履歴が生成されるのを抑制 .
  ;; [2/3] のような完了数が見出しにある時に転送先候補が重複表示されるため .

  ;; リファイル先でサブディレクトリを指定するために一部フルパス化
  (let ((dir (expand-file-name org-directory)))
    (setq org-refile-targets
      `(("next.org" :level . 1)
        ("org-ical.org" :level . 1)
        ("academic.org" :level . 1)
        (, (concat dir "tr/work.org") :level . 1)
        (, (concat dir "minutes/wgl.org") :level . 1)
        (, (concat dir "db/article.org") :level . 1)
        (, (concat dir "db/maybe.org") :level . 1)
        (, (concat dir "db/english.org") :level . 1)
        (, (concat dir "db/money.org") :level . 1))))

  (defun ad:org-refile (f &optional arg default-buffer rfloc msg)
    "Extension to support keeping org-refile-history empty."
    (let ((l (org-outline-level))
          (b (buffer-name)))
      (apply f arg default-buffer rfloc msg)
      (save-excursion
        (save-restriction
          (if (> l (org-outline-level))
              (outline-backward-same-level 1)
              (outline-up-heading 1))
          (org-update-statistics-cookies nil) ;; Update in source
          (org-sort-entries nil ?O)
          (org-refile-goto-last-stored)
          (org-update-parent-todo-statistics) ;; Update in destination
          (outline-up-heading 1)
          (org-sort-entries nil ?O)
          (unless (equal b (buffer-name))
              (switch-to-buffer b))))))
    (setq org-refile-history nil)
    (org-refile-cache-clear))
  (advice-add 'org-refile :around #'ad:org-refile)

  (defun ad:org-sort-entries (&optional _with-case _sorting-type
                                _getkey-func _compare-func
                                _property _interactive?)
    (outline-hide-subtree)
    (org-show-hidden-entry)
    (org-show-children)
    (advice-add 'org-sort-entries :after #'ad:org-sort-entries))
```

## 10.17. [org-babel] Org バッファでソースコードを扱う

```
(with-eval-after-load "ob-core"
  (setq org-edit-src-content-indentation 0)
  (setq org-src-fontify-natively t)
  (setq org-src-tab-acts-natively t)
  (setq org-confirm-babel-evaluate nil))
```

```
(setq org-src-window-setup 'current-window)
;; org-src-window-setup (current-window, other-window, other-frame)
(require 'ob-http nil t)
(require 'ob-gnuplot nil t)

(unless (executable-find "ditaa")
  (message "--- ditaa is NOT installed.))

;; Add ":results output" after program name
(org-babel-do-load-languages
 'org-babel-load-languages
 '( (dot . t)
    (C . t)
    (ditaa . t)
    (gnuplot . t)
    (perl . t)
    (shell . t)
    (latex . t)
    (sqlite . 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))
```

## 10.18. [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
    (if (version< "9.1.4" (org-version))
        '("S" . "src emacs-lisp")
        '("S" "#+begin_src emacs-lisp\n?\n\n#+END_SRC" "<src
lang=\"emacs-lisp\">\n\n</src>"))))
```

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

パワーポイントはもう使わなくてよいのです．

- <https://pxaka.tokyo/wiki/doku.php?id=emacs:org-tree-slide>

```
(when (autoload-if-found
  ' (org-tree-slide-mode)
  "org-tree-slide" nil t)
```

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

(with-eval-after-load "org-tree-slide"
  ;; <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)
  (unless noninteractive
    (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)))
```

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．

## 10.20. [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-tree-slide-autoclockin-p ()
    (save-excursion
      (save-restriction
        (widen)
        (goto-char 1)
        (let ((keyword "TREE_SLIDE:")
              (value "autoclockin")
              (result nil))
          (while
            (and (re-search-forward (concat "^#\\" keyword "[ \t]*") nil t)
                 (re-search-forward value (point-at-eol) t))
              (setq result t))
            result))))))

(when (require 'org-clock nil t)
  (defun my-org-clock-in ()
    (setq vc-display-status nil) ;; モードライン節約
    (when (and (my-tree-slide-autoclockin-p)
                (looking-at (concat "^#\\" org-not-done-regexp))
                (memq (org-outline-level) '(1 2 3 4)))
      (org-clock-in)))

  (defun my-org-clock-out ()
    (setq vc-display-status t) ;; モードライン節約解除
    (when (org-clocking-p)
      (org-clock-out)))

  (add-hook 'org-tree-slide-before-move-next-hook #'my-org-clock-out))
```

```
(add-hook 'org-tree-slide-before-move-previous-hook #'my-org-clock-out)
;; (add-hook 'org-tree-slide-before-content-view-hook #'my-org-clock-out)
(add-hook 'org-tree-slide-mode-stop-hook #'my-org-clock-out)
(add-hook 'org-tree-slide-after-narrow-hook #'my-org-clock-in)))
```

## 10.21. [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 (autoload-if-found
      '(org-tree-slide-mode my-proportional-font-toggle)
      "org-tree-slide" nil t)

  (with-eval-after-load "org-tree-slide"
    (defcustom use-proportional-font nil
      "The status of FONT property"
      :type 'boolean
      :group 'org-mode)

    (set-face-attribute 'variable-pitch nil
      :family "Verdana"
      ;; :family "Comic Sans MS"
      :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)))))
```

## 10.22. [org-tree-slide] ヘッドラインをリッチにする

org-tree-slide が有効な時だけ org-bullets を有効にして，ヘッドラインをリッチにします．元ネタは，org-beautify-theme.el です．

```
(when (autoload-if-found
      '(org-bullets-mode)
      "org-bullets" nil t)
```



```
(add-hook 'org-tree-slide-play-hook (lambda () (org-bullets-mode 1)))
(add-hook 'org-tree-slide-stop-hook (lambda () (org-bullets-mode -1)))
```

### 10.23. [calfw-org.el] 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 (autoload-if-found
      ' (my-cfw-open-org-calendar cfw:open-org-calendar)
      "calfw-org" "Rich calendar for org-mode" t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-c f c w") 'my-cfw-open-org-calendar))

  (with-eval-after-load "calfw-org"
    ;; icalendar との連結
    (custom-set-variables
      ' (cfw:org-icalendars ' ("~/Dropbox/org/org-ical.org"))
      ' (cfw:fchar-junction ?+) ;; org で使う表にフェイスを統一
      ' (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)
      (moom-change-frame-width-double)
      (cfw:open-org-calendar))

    (defun my-cfw-burry-buffer ()
      (interactive)
      (bury-buffer)
      (moom-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)))

;; (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"))))
```

## 10.24. [org-odt] ODT 形式に出力

```
(when (autoload-if-found
      ' (ox-odt)
      "ox-odt" nil t)

  (with-eval-after-load "ox-odt"
    ;; (add-to-list 'org-odt-data-dir
    ;;   (concat (getenv "HOME") "/Dropbox/emacs.d/config/"))
    (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"))))
```

## 10.25. [ox-twbs] Twitter Bootstrap 互換の HTML 出力

- [marsmining/ox-twbs: Export org-mode docs as HTML compatible with Twitter Bootstrap.](https://marsmining.github.io/ox-twbs/)

```
(with-eval-after-load "ox"
  (require 'ox-twbs nil t))
```

次のようなファイルを準備して org ファイルのヘッダで指定 ( `#+setupfile: theme-readtheorg.setup` ) するだけで, <https://takaxp.github.io/> のような HTML 出力もできます .

- [{O} Org HTML export \(ReadTheOrg\)](#)

*Listing 10.25.1:*

```
# -*- mode: org; -*-

#+html_head: <link rel="stylesheet" type="text/css"
href="https://www.pirilampo.org/styles/readtheorg/css/htmlize.css"/>
#+html_head: <link rel="stylesheet" type="text/css"
href="https://www.pirilampo.org/styles/readtheorg/css/readtheorg.css"/>

#+html_head: <script
```

```
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
#+html_head: <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/js/bootstrap.min.js"></script>
#+html_head: <script type="text/javascript"
src="https://www.pirilampo.org/styles/lib/js/jquery.stickytableheaders.min.js">
</script>
#+html_head: <script type="text/javascript"
src="https://www.pirilampo.org/styles/readtheorg/js/readtheorg.js"></script>
```

## 10.26. [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。

```
(with-eval-after-load "org"
  (when (require 'org-crypt nil t)
    (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)
    (define-key org-mode-map (kbd "C-c f c e") 'org-encrypt-entry)
    (define-key org-mode-map (kbd "C-c f c d") 'org-decrypt-entry)))
```

## 10.27. [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
  (when (and (require 'org-mac-iCal nil t)
              (require 'org-mac-link nil t))
    (define-key org-mode-map (kbd "C-c c") 'org-mac-grab-link)))
```

## 10.28. [org-download] org バッファに D&D でファイルを保存

org-attach とペアで使うのが良いでしょう．

```
(with-eval-after-load "org"
  (when (require 'org-download nil t)
    (setq org-download-screenshot-method 'screencapture)
    (setq org-download-method 'attach)))
```

## 10.29. [org-grep] org ファイルを grep する

```
(when (autoload-if-found
      ' (org-grep)
      "org-grep" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-M-g") 'org-grep))

  (with-eval-after-load "org-grep"
    (setq org-grep-extensions '(".org" ".org_archive"))
    (add-to-list 'org-grep-directories "~/emacs.d")
    (add-to-list 'org-grep-directories "~/emacs.d/.cask/package"))

  ;; "q"押下の挙動を調整
  (defun ad:org-grep-quit ()
    (interactive)
    (delete-window))
  (advice-add 'org-grep-quit :override #'ad:org-grep-quit)

  ;; for macOS
  (when (memq window-system '(mac ns))
    (defun org-grep-from-org-shell-command (regexp)
      (if org-grep-directories
        (concat "find -E "
          (if org-grep-directories
            (mapconcat #'identity org-grep-directories " ")
            org-directory)
          (and org-grep-extensions
            (concat " -regex '.*("
              (mapconcat #'regexp-quote org-grep-extensions
                "|")
              ")$'"))
            " -print0 | xargs -0 grep " org-grep-grep-options
            " -n -- " (shell-quote-argument regexp))
        "")))
```

### 10.30. [ox-reveal] ナイスなHTML5プレゼンテーション出力

C-c C-e R エクスポートを呼び出せます。続けて B を押せば、ブラウザで出力後の見た目を確認できます。ただし別途 [reveal.js](#) が使える状態にないとダメです。org-reveal-root を設定すれば、clone した reveal.js の場所を指定できます。

```
(with-eval-after-load "ox"
  (require 'ox-reveal nil t)
  (when (version< "9.1.4" (org-version))
    (setq org-reveal-note-key-char ?n)))
```

それ以外にも [ox-s5](#) , [org-ioslide](#) があります。

### 10.31. [org-dashboard] 進捗をプログレスバーで確認

- <https://github.com/bard/org-dashboard>
- 日本語だとバーの表示位置がずれるので修正。PR 反映済み。

```
(when (autoload-if-found
  ' (org-dashboard-display)
  "org-dashboard" nil t)

(with-eval-after-load "org"
  (define-key org-mode-map (kbd "C-c f y") 'org-dashboard-display)))
```

### 10.32. [org-clock-today] 今日の総作業時間をモードラインに表示

- CLOCK\_MODELINE\_TOTAL を設定するのとは別なアプローチ
- clock が動いている時だけモードラインに合計値が表示される
- [mallt/org-clock-today-mode: Emacs minor mode to show the total clocked time of the current day in the mode line](#)

```
(with-eval-after-load "org"
  (defun ad:org-clock-sum-today (&optional headline-filter)
    "Sum the times for each subtree for today."
    (let ((range (org-clock-special-range 'today nil t))) ;; TZ 考慮
      (org-clock-sum (car range) (cadr range)
        headline-filter :org-clock-minutes-today)))
  (advice-add 'org-clock-sum-today :override #'ad:org-clock-sum-today)

  (when (require 'org-clock-today nil t)
    (defun ad:org-clock-today-update-mode-line ()
      "Calculate the total clocked time of today and update the mode line."
      (setq org-clock-today-string
        (if (org-clock-is-active)
          ;; ナローイングの影響を排除し、subtree に限定しない。
          (org-clock-sum-today)
          0))))
```

```

        (save-excursion
          (save-restriction
            (with-current-buffer (org-clock-is-active)
              (widen)
              (let* ((current-sum (org-clock-sum-today))
                     (open-time-difference (time-subtract
                                              (float-time)
                                              (float-time
                                               org-clock-start-time)))
                     (open-seconds (time-to-seconds open-time-
difference)))
                (open-minutes (/ open-seconds 60))
                (total-minutes (+ current-sum
                                   open-minutes)))
              (concat " " (org-minutes-to-clocksum-string
                           total-minutes))))))
      "")
    (force-mode-line-update))
  (advice-add 'org-clock-today-update-mode-line
    :override #'ad:org-clock-today-update-mode-line)

  (unless noninteractive
    (org-clock-today-mode 1)))

```

### 10.33. [org-random-todo] ランダムにタスクを選ぶ

- デフォルトで org-agenda-files に登録されたファイルからランダムにタスクを選ぶ
- org-random-todo はミニバッファに選択したタスクを表示するだけ .
- org-random-todo-goto-current 最後に表示したタスクに移動する .

```

(autoload-if-found
  '(org-random-todo org-random-todo-goto-current)
  "org-random-todo" nil t)

```

### 10.34. [ox-hugo] Org ファイルから Hugo の記事をエクスポートする

- Org Mode でブログを書いて , [Hugo](https://gohugo.io/) で静的サイトに出力します .

```

(with-eval-after-load "org"
  ;; Require ox-hugo-auto-export.el explicitly before loading ox-hugo.el
  (require 'ox-hugo-auto-export nil t))

(with-eval-after-load "ox"
  (when (require 'ox-hugo nil t)
    (setq org-hugo-auto-set-lastmod t)
    (setq org-hugo-suppress-lastmod-period 86400.0) ;; 1 day
    ;; never copy files to under /static/ directory
    (setq org-hugo-external-file-extensions-allowed-for-copying nil)

    ;; see https://pxaka.tokyo/blog/2018/a-link-to-the-original-org-source-file
    (defun org-hugo-get-link-to-orgfile (uri alt)

```

```
"Return a formatted link to the original Org file.
To insert the formatted into an org buffer for Hugo, use an appropriate
macro, e.g. {{{srclink}}}.
```

```
Note that this mechanism is still under consideration."
```

```
(let ((line (save-excursion
              (save-restriction
                (unless (org-at-heading-p)
                  (org-previous-visible-heading 1))
                (line-number-at-pos))))))
  (concat "[[" uri (file-name-nondirectory (buffer-file-name))
          "#L" (format "%d" line) "]" alt "]]"))))
```

### 10.35. [ox-html] HTML 見出しへのリンクを固定する

標準の設定で HTML を出力すると、各見出しへのリンクとして `org0123456` のような ID が振られる。しかしそれらの ID は、HTML を出力するたびに更新されるため、出力した HTML の見出しをパーマリンクとして使用できない。これを防ぐためには、プロパティで `CUSTOM_ID` を指定する必要がある。

これを簡単にするための補助関数が公開されている。この[元ネタ](#)を参考に、少し改良して使用している。なお元ネタでは標準の `org-id-new` を直接利用しているが、以下の実装では、`org` の後ろに UUID の先頭 8 桁が続くように、ID の内容に制限をかけている。

`my-org-add-ids-to-headlines-in-file` は、バッファの保存時点、或いは、HTML エクスポートが走る直前のタイミングで自動実行させるのがスマートだが、当面は手動で運用する。

```
(with-eval-after-load "org"
  (defun my-add-custom-id ()
    "Add \"CUSTOM_ID\" to the current tree if not assigned yet."
    (interactive)
    (my-org-custom-id-get nil t))

  (defun my-get-custom-id ()
    "Return a part of UUID with an \"org\" prefix.
e.g. \"org3ca6ef0c\"."
    (let* ((id (org-id-new "")))
      (when (org-uuidgen-p id)
        (downcase (concat "org" (substring (org-id-new "") 0 8))))))

  (defun my-org-custom-id-get (&optional pom create)
    "Get the CUSTOM_ID property of the entry at point-or-marker POM.
If POM is nil, refer to the entry at point. If the entry does
not have an CUSTOM_ID, the function returns nil. However, when
CREATE is non nil, create a CUSTOM_ID if none is present
already. In any case, the CUSTOM_ID of the entry is returned.

See https://writequit.org/articles/emacs-org-mode-generate-ids.html"
    (interactive)
    (org-with-point-at pom
      (let ((id (org-entry-get nil "CUSTOM_ID")))
        (cond
```

```

    ((and id (stringp id) (string-match "\\S-" id))
     id)
    (create
     (setq id (my-get-custom-id))
     (unless id
      (error "Invalid ID")))
    (org-entry-put pom "CUSTOM_ID" id)
    (message "--- CUSTOM_ID assigned: %s" id)
    (org-id-add-location_id (buffer-file-name (buffer-base-buffer)))
    id))))

(defun my-org-add-ids-to-headlines-in-file ()
  "Add CUSTOM_ID properties to all headlines in the current file.
Which do not already have one. Only adds ids if the
`auto-id' option is set to `t' in the file somewhere. ie,
#+options: auto-id:t

See https://writequit.org/articles/emacs-org-mode-generate-ids.html"
  (interactive)
  (save-excursion
   (widen)
   (goto-char (point-min))
   (when (re-search-forward "^#\\+OPTIONS:.*auto-id:t" (point-max) t)
    (org-map-entries
     (lambda () (my-org-custom-id-get (point) 'create)))))))

```

### 10.36. TODO [org-tree-slide] BEGIN\_SRC と END\_SRC を消して背景色を変える

hide-lines.el を使うことで、プレゼン時に BEGIN\_SRC と END\_SRC を非表示にしてすっきりさせます。さらに、ソースブロックの背景色を変えます（以下の例では、emacs-lisp を言語で指定している場合に限定）。

```

(when (autoload-if-found
      '(org-tree-slide-mode)
      "org-tree-slide" nil t)

  (with-eval-after-load "org-tree-slide"
    ;; FIXME 複数のバッファで並行動作させるとおかしくなる .hide-lines の問題？
    (when (and nil (require 'hide-lines nil t))
      (defvar my-org-src-block-faces nil)
      (defun my-show-headers ()
        (setq org-src-block-faces 'my-org-src-block-faces)
        (hide-lines-show-all))
      (defun my-hide-headers ()
        (setq my-org-src-block-faces 'org-src-block-faces)
        (setq org-src-block-faces
          '(("emacs-lisp" (:background "cornsilk"))))
        (hide-lines-matching "#\\+BEGIN_SRC")
        (hide-lines-matching "#\\+END_SRC"))
      (add-hook 'org-tree-slide-play-hook #'my-hide-headers)
      (add-hook 'org-tree-slide-stop-hook #'my-show-headers)

      ;; (defun ad:org-edit-src-code (&optional code edit-buffer-name)
      (defun ad:org-edit-src-code ()
        (interactive)
        (my-show-headers))

```



```
(advice-add 'org-edit-src-code :before #'ad:org-edit-src-code)
;; Block 外で呼ばれると, my-show-headers が呼ばれてしまう
(defun ad:org-edit-src-exit ()
  (interactive)
  (my-hide-headers))
(advice-add 'org-edit-src-exit :after #'ad:org-edit-src-exit)))
```

### 10.37. TODO [org-fstree] ディレクトリ構造を読み取る

```
(with-eval-after-load "org"
  (require 'org-fstree nil t))
```

### 10.38. TODO [ox] 出力形式の拡張

```
(with-eval-after-load "ox"
  ;; (setq org-export-default-language "ja")
  (require 'ox-pandoc nil t)
  (require 'ox-gmd nil t) ;; Quita-style
  (require 'ox-gfm nil t)) ;; Github-style
```

### 10.39. TODO Parers3.app からリンクを取得する

```
(with-eval-after-load "org-mac-link"
  (defcustom org-mac-grab-Papers-app-p t
    "Add menu option [P]apers to grab links from the Papers.app."
    :tag "Grab Papers.app links"
    :group 'org-mac-link
    :type 'boolean)

  (defun org-mac-papers-insert-frontmost-paper-link ()
    (interactive)
    (let ((result (org-mac-papers-get-frontmost-paper-link)))
      (if result
        (insert result)
        (message "Please open Papers.app and select a paper."))))

  (defun org-mac-papers-get-frontmost-paper-link ()
    (interactive)
    (message "Applescript: Getting Papers link...")
    (let ((result (org-as-mac-papers-get-paper-link)))
      (if (or (eq result nil) (string= result ""))
        nil
        (org-mac-paste-applescript-links result))))

  (defun org-as-mac-papers-get-paper-link ()
    (do-applescript
      (concat
        "if application \"Papers\" is running then\n"
        "  tell application \"Papers\" to activate\n"
        "  delay 0.3\n"
        "  set the clipboard to \"\"\n"
        "  tell application \"System Events\" to tell process \"Papers\"\n"
        "    keystroke \"l\" using {command down, shift down}\n"
        "  end tell\n"
        "  delay 0.2\n"
        "  set aLink to the clipboard\n"
        "  tell application \"System Events\" to tell process \"Papers\"
```

```

;; "      keystroke \"c\" using {command down, alt down}\n"
"      keystroke \"m\" using {command down, option down}\n"
"      end tell\n"
"      delay 0.2\n"
"      set aName to the clipboard\n"
"      tell application \"Emacs\" to activate\n"
"      return (get aLink) & \":split:~\" & (get aName) as string\n"
"else\n"
"      return\n"
"end if\n"))

(add-to-list 'org-mac-link-descriptors
  `("P" "apers" org-mac-papers-insert-frontmost-paper-link
    ,org-mac-grab-Papers-app-p) t))

```

## 10.40. TODO Papers3.app のリンクを開けるようにする

Papers3.app は、各文献に `papers3://` で始まる URI を割り当てています。このリンクを org バッファにペーストし、`org-open-at-point` (`C-c C-o`) で開けるようにします。

```

(with-eval-after-load "org"
  (when (eq system-type 'darwin)
    ;; Open `papers3://' link by C-c C-o.
    ;; (org-add-link-type will be obsoleted from Org 9.
    (org-link-set-parameters
      "papers3"
      :follow (lambda (path)
        (let ((cmd (concat "open papers3:" path)))
          (shell-command-to-string cmd)
          (message "%s" cmd))))))

```

## 10.41. TODO [org-attach] 外部ファイルを紐付ける

```

(when (autoload-if-found
  ' (org-attach du-org-attachments)
  "org-attach" nil t)

  (with-eval-after-load "org-attach"
    ;; org-insert-link で添付ファイルへのリンクをペーストできるようにする
    (setq org-attach-store-link-p t)
    ;; 自動付与されるタグの名前を変更
    (setq org-attach-auto-tag "Attach")
    ;; git-annex を使用しない
    (setq org-attach-git-annex-cutoff nil)

    (defvar org-attach-directory-absolute
      (concat (getenv "HOME")
        "/Dropbox/org/"
        (if (boundp 'org-attach-directory)
          "data/"))))

    (defun du-org-attachments ()
      "Show directory size for org-attachments."
      (interactive)
      (message "--- %s"
        (chomp (shell-command-to-string

```

```
(concat "/usr/bin/du -sh "
        org-attach-directory-absolute))))))
```

## 10.42. TODO [org-recent-headings] 訪問したツリーを記録し簡単に再訪問可能にする

```
(when (autoload-if-found
      '(org-recent-headings-helm)
      "org-recent-headings" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-c f r") 'org-recent-headings-helm))

  (with-eval-after-load "org-recent-headings"
    (custom-set-variables
      '(org-recent-headings-save-file "~/.emacs.d/org-recent-headings.dat"))
    (if shutup-p
      (shut-up (org-recent-headings-mode 1))
      (org-recent-headings-mode 1))))
```

## 10.43. TODO [orgnav] ツリーの検索インタフェース

```
(when (autoload-if-found
      '(orgnav-search-root)
      "orgnav" nil t)

  (with-eval-after-load "org"
    (define-key org-mode-map (kbd "C-c f n")
      (lambda () (interactive)
        (orgnav-search-root 3 'orgnav--goto-action))))
```

## 10.44. TODO [toc-org] 目次の挿入

- ツリーに TOC タグを付けて、`toc-org-insert-toc'を実行すれば OK

```
(autoload-if-found '(toc-org-insert-toc) "toc-org" nil t)
```

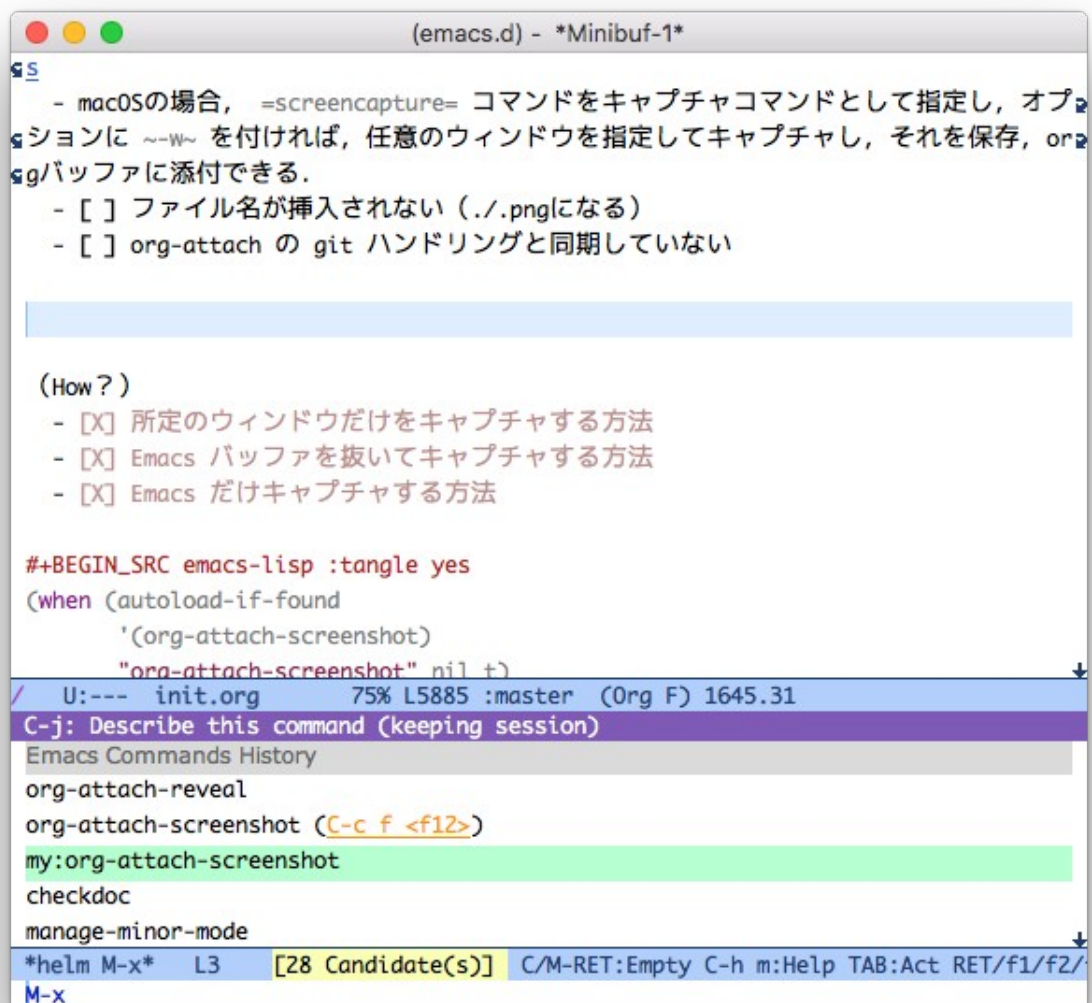
## 10.45. TODO [org-review.el] レビューフローのサポート

- <https://github.com/brabalan/org-review>
- 以下の設定では、org-agenda 使用時に C-c C-r r で発動

```
(with-eval-after-load "org-agenda"
  (when (require 'org-review nil t)
    (add-to-list 'org-agenda-custom-commands
      ("r" "Review projects" tags-todo "-CANCELLED/"
        (org-agenda-overriding-header "Reviews Scheduled")
        (org-agenda-skip-function 'org-review-agenda-skip)
        (org-agenda-cmp-user-defined 'org-review-compare)
        (org-agenda-sorting-strategy '(user-defined-down))))
    (org-defkey org-agenda-mode-map "\C-c\C-r"
      'org-review-insert-last-review)))
```

## 10.46. REV2 [org-screenshot] スクリーンショットを貼り付ける

- [dfeich/org-screenshot: screenshots integrated with emacs org mode attachments](https://dfeich.org/org-screenshot: screenshots integrated with emacs org mode attachments)
- macOS の場合， `screencapture` コマンドをキャプチャコマンドとして指定し，オプションに `-w` を付ければ，任意のウィンドウを指定してキャプチャし，それを保存，org バッファに添付できる．
- ☒ ファイル名が挿入されない ( `./png` になる )
- ☐ org-attach の git ハンドリングと同期していない



The screenshot shows an Emacs window titled "(emacs.d) - \*Minibuf-1\*". The main buffer contains Japanese text and a list of items with checkboxes. Below this is a light blue horizontal bar. The minibuffer at the bottom shows the command history, with "my:org-attach-screenshot" highlighted in green. The status bar at the very bottom shows "\*helm M-x\* L3 [28 Candidate(s)] C/M-RET:Empty C-h m:Help TAB:Act RET/f1/f2/" and "M-x" in the input field.

```
(emacs.d) - *Minibuf-1*
S
- macOSの場合， =screencapture= コマンドをキャプチャコマンドとして指定し，オプションに ~-w~ を付ければ，任意のウィンドウを指定してキャプチャし，それを保存，org バッファに添付できる．
- ☒ ファイル名が挿入されない ( ./png になる )
- ☐ org-attach の git ハンドリングと同期していない

(How ?)
- ☒ 所定のウィンドウだけをキャプチャする方法
- ☒ Emacs バッファを抜いてキャプチャする方法
- ☒ Emacs だけキャプチャする方法

#+BEGIN_SRC emacs-lisp :tangle yes
(when (autoload-if-found
      '(org-attach-screenshot)
      "org-attach-screenshot" nil t))
/ U:--- init.org 75% L5885 :master (Org F) 1645.31
C-j: Describe this command (keeping session)
Emacs Commands History
org-attach-reveal
org-attach-screenshot (C-c f <f12>)
my:org-attach-screenshot
checkdoc
manage-minor-mode
*helm M-x* L3 [28 Candidate(s)] C/M-RET:Empty C-h m:Help TAB:Act RET/f1/f2/
M-x
```

( How ? )

- [✓] 所定のウィンドウだけをキャプチャする方法
- [✓] Emacs バッファを抜いてキャプチャする方法
- [✓] Emacs だけキャプチャする方法

```
(when (autoload-if-found
      '(org-attach-screenshot)
      "org-attach-screenshot" nil t)

  (with-eval-after-load "org-attach-screenshot"
    (when (executable-find "screencapture")
      (setq org-attach-screenshot-command-line "screencapture -w %f"))
    (defun my-org-attach-screenshot ()
      (interactive)
      (org-attach-screenshot t (format-time-string
                                "screenshot-%Y%m%d-%H%M%S.png")))))
```

#### 10.47. **DONE** [org-autolist] ブリッツの入力を簡単に

**[OBSOLETE]**

ブリッツの入力や削除を Microsoft Word 的にします．意外と馴染まなかったので不使用です．

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

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

#### 10.48. **DONE** [MobileOrg] iOS との連携

**[OBSOLETE]**

- <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))))
```

## 10.49. **DONE** [org-export-generic] エクスポート機能を拡張する[OBSOLETE]

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 .

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

[OBSOLETE]

```
;;; 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)))

```

## 11. フレーム / ウィンドウ制御

### 11.1. 起動時の設定

```

(cond
  ((memq window-system '(mac ns)) ;; for Macintosh
   (setq initial-frame-alist
         (append
          '(top . 23)
          (left . 0)
          (alpha . (100 90))
          ;; (vertical-scroll-bars . nil)
          ;; (internal-border-width . 20)

```



```

;; (ns-appearance . nil) ;; 26.1 {light, dark}
(ns-transparent-titlebar . t)) ;; 26.1
initial-frame-alist)))

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

;; for Windows
(t (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)
(set-face-foreground 'vertical-border (face-background 'default))
(set-face-background 'vertical-border (face-background 'default))
;; (set-face-background 'fringe (face-background 'default))

;; カーソルの色
(defconst my-cursor-color-ime-on "#FF9300")
(defconst my-cursor-color-ime-off "#91C3FF") ;; #FF9300, #999999, #749CCC
(defconst my-cursor-type-ime-on '(bar . 2)) ;; '(hbar . 10)
(defconst my-cursor-type-ime-off '(bar . 2)) ;; '(hbar . 10)
(defvar my-ime-last nil)

(cond
  ((or (eq window-system 'ns)
    (fboundp 'mac-get-current-input-source))
    (when (fboundp 'mac-set-input-method-parameter)
      (mac-set-input-method-parameter
        "com.google.inputmethod.Japanese.base" 'title "あ")))

  (when (fboundp 'mac-get-current-input-source)
    (declare-function ad:mac-toggle-input-method "init" nil)
    (declare-function my-apply-cursor-config "init" nil)
    (declare-function my-ime-on "init" nil)
    (declare-function my-ime-off "init" nil)
    (declare-function my-ime-active-p "init" nil)

    (defun my-ime-active-p ()
      (not (string-match "\\..Roman$" (mac-get-current-input-source))))

    (setq my-ime-last (my-ime-active-p))
    (defvar my-ime-on-hook nil)
    (defvar my-ime-off-hook nil)

    (defun my-ime-on ()

```

```

(interactive)
(when (fboundp 'mac-toggle-input-method)
  (mac-toggle-input-method t))
(setq cursor-type my-cursor-type-ime-on)
(set-cursor-color my-cursor-color-ime-on)
(setq my-ime-last t)
(run-hooks 'my-ime-on-hook))

(defun my-ime-off ()
  (interactive)
  (when (fboundp 'mac-toggle-input-method)
    (mac-toggle-input-method nil))
  (setq cursor-type my-cursor-type-ime-off)
  (set-cursor-color my-cursor-color-ime-off)
  (setq my-ime-last nil)
  (run-hooks 'my-ime-off-hook))

(defvar my-ime-flag nil)
(add-hook 'activate-mark-hook
  (lambda ()
    (when (setq my-ime-flag (my-ime-active-p))
      (my-ime-off))))
(add-hook 'deactivate-mark-hook
  (lambda ()
    (when my-ime-flag
      (my-ime-on))))

(defun ad:mac-toggle-input-method (&optional arg)
  "Run hooks when IME changes."
  (interactive)
  (if arg
    (progn
      (setq cursor-type my-cursor-type-ime-on)
      (set-cursor-color my-cursor-color-ime-on)
      (run-hooks 'my-ime-on-hook))
    (progn
      (setq cursor-type my-cursor-type-ime-off)
      (set-cursor-color my-cursor-color-ime-off)
      (run-hooks 'my-ime-off-hook))))
(advice-add 'mac-toggle-input-method
  :before #'ad:mac-toggle-input-method)

;; for init setup
(setq-default cursor-type my-cursor-type-ime-on)

(when (boundp 'mac-ime-cursor-type) ;; private patch
  (setq mac-ime-cursor-type my-cursor-type-ime-on))

(defun my-apply-cursor-config ()
  (interactive)
  (if (my-ime-active-p)
    (progn
      (setq cursor-type my-cursor-type-ime-on)
      (set-cursor-color my-cursor-color-ime-on))
    (setq cursor-type my-cursor-type-ime-off)
    (set-cursor-color my-cursor-color-ime-off)))
;; (my-apply-cursor-config) will be called later in this file.

(with-eval-after-load "postpone"
  (run-with-idle-timer 3 t #'my-apply-cursor-config))

```

```

;; Enter minibuffer with IME-off, and restore the latest IME
(add-hook 'minibuffer-setup-hook
  (lambda ()
    (if (not (my-ime-active-p))
        (setq my-ime-flag nil)
        (setq my-ime-flag t)
        (my-ime-off))))
(add-hook 'minibuffer-exit-hook
  (lambda ()
    (when my-ime-flag
      (my-ime-on))))

;; (defun ad:find-file (FILENAME &optional WILDCARDS)
;;   "Extension to find-file as before-find-file-hook."
;;   (message "--- ad:findfile")
;;   (apply FILENAME WILDCARDS))
;; (advice-add #'find-file :around #'ad:find-file)

;; http://tezfm.blogspot.jp/2009/11/cocoa-emacs.html
;; バッファ切替時に input method を切り替える
;; (with-eval-after-load "postpone"
;;   (when (and (fboundp 'mac-handle-input-method-change)
;;             (require 'cl nil t))
;;     (add-hook
;;      'post-command-hook
;;      (lexical-let ((previous-buffer nil))
;;        (message "Change IM %S -> %S" previous-buffer (current-buffer))
;;        (lambda ()
;;          (unless (eq (current-buffer) previous-buffer)
;;            (when (bufferp previous-buffer)
;;              (mac-handle-input-method-change))
;;            (setq previous-buffer (current-buffer))))))))
;; )

((eq window-system 'mac) ;; EMP: Emacs Mac Port
  (when (fboundp 'mac-input-source)
    (defun my-mac-keyboard-input-source ()
      (if (string-match "\\\\.Roman$" (mac-input-source))
          (progn
            (setq cursor-type my-cursor-type-ime-off)
            (add-to-list 'default-frame-alist
                        `(cursor-type . ,my-cursor-type-ime-off))
            (set-cursor-color my-cursor-color-ime-off))
          (progn
            (setq cursor-type my-cursor-type-ime-on)
            (add-to-list 'default-frame-alist
                        `(cursor-type . ,my-cursor-type-ime-on))
            (set-cursor-color my-cursor-color-ime-on))))

    (when (fboundp 'mac-auto-ascii-mode)
      ;; (mac-auto-ascii-mode 1)
      ;; IME ON/OFF でカーソルの種別や色を替える
      (add-hook 'mac-selected-keyboard-input-source-change-hook
        #'my-mac-keyboard-input-source)
      ;; IME ON の英語入力 + 決定後でもカーソルの種別や色を替える
      ;; (add-hook 'mac-enabled-keyboard-input-sources-change-hook
      ;;   #'my-mac-keyboard-input-source)

      (declare-function my-mac-keyboard-input-source "init" nil)
      (my-mac-keyboard-input-source))))

```

```
(t nil))
```

## 11.2. 複数フレーム対応

フレームを複数使う場合の設定です .

```
(declare-function my-apply-theme "init" nil)
(with-eval-after-load "postpone"
  (defun ad:make-frame ()
    (my-apply-theme)
    (when (require 'moom-font nil t)
      (moom-font-resize)))
  (advice-add 'make-frame :after #'ad:make-frame)
  (global-set-key (kbd "M-`") 'other-frame))
```

## 11.3. [moom.el] キーボードでフレームの場所を移す

(2017-07-30) frame-ctr から moom.el にリネームしました . Moom は , Many Tricks がリリースするソフトウェアで , 今後はこのソフトウェアのポーティングを目指します .

拙作の [moom.el](#) を使います .

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` をそのまま使っても , 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 .

近しいパッケージに , `WindMove` と `FrameMove` がありますが , これらは基本的にカーソルの移動を目的にしています .

- [{Home} Wind Move](#) (ビルトイン)
  - ウィンドウ間のカーソル移動を改善
- [{Home} Frame Move](#)
  - フレーム間のカーソル移動を改善

また類似のパッケージに `frame-cmds` があります .

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

moom-autoloads に必要なコマンドを羅列していますが，melpa でインストールする場合は関係ありません．パッケージ内で autoload 指定してあります．with-eval-after-load の中だけを設定すれば十分です．

```
(defconst moom-autoloads
  '(moom-cycle-frame-height
    moom-move-frame-to-edge-top moom-move-frame my-frame-reset
    moom-toggle-frame-maximized
    moom-move-frame-to-center moom-move-frame-right moom-move-frame-left
    moom-fill-display-band moom-move-frame-to-edge-right moom-fill-band
    moom-change-frame-width moom-change-frame-width-double
    moom-change-frame-width-single))

(when (autoload-if-found
      moom-autoloads
      "moom" nil t)

  (global-set-key (kbd "<f2>") 'moom-cycle-frame-height)

  (with-eval-after-load "postpone"
    (unless noninteractive
      (postpone-message "moom")))
    (when (and (require 'moom nil t)
              window-system)
      (setq moom-lighter "M")
      (setq moom-verbose t)
      (let ((moom-verbose nil))
        (moom-recommended-keybindings 'all))
      (moom-mode 1))))

(when (autoload-if-found
      '(moom-font-increase
        moom-font-decrease moom-font-size-reset moom-font-resize)
      "moom-font" nil t)

  (add-hook 'moom-font-after-resize-hook #'moom-expand-height)
  (add-hook 'moom-font-after-resize-hook #'moom-move-frame-to-edge-top)

  (with-eval-after-load "moom-font"
    (setq moom-scaling-gradient (/ (float 50) 30))
    (setq moom-font-table
      '((50 30) (49 29) (48 29) (47 28) (46 28) (45 27) (44 26) (43 26)
        (42 25) (41 25) (40 24) (39 23) (38 23) (37 22) (36 22) (35 21)
        (34 20) (33 20) (32 19) (31 19) (30 18) (29 17) (28 17) (27 16)
        (26 16) (25 15) (24 14) (23 14) (22 13) (21 13) (20 12) (19 11)
        (18 11) (17 10) (16 10) (15 9) (14 8) (13 8) (12 7) (11 7) (10 6)
        (9 5) (8 5) (7 4) (6 4) (5 3)))))
```

### 11.3.1. TODO 最大化時にモードラインを消す

バッファの切替時にモードラインの状態を引き継ぐこともできますが，とりあえず想定外にしています．代わりに，必要ならば M-x my-moom-toggle-mode-line で表示可能にしています．

マイナーモードとして扱えるようにするには，Bastien 氏の記事が参考になります．

- [Emacs mode for hiding the mode-line](#)

```
(with-eval-after-load "moom"
  (defvar my-moom-hide-mode-line t)
  (defun ad:moom-toggle-frame-maximized ()
    (when mode-line-format
      (setq my-mode-line-format mode-line-format))
    (when my-moom-hide-mode-line
      (setq mode-line-format
        (if moom--maximized nil my-mode-line-format)))
    (message " ")))
  (advice-add 'moom-toggle-frame-maximized
    :after #'ad:moom-toggle-frame-maximized))

;; (make-variable-buffer-local 'my-mode-line-format)
(defvar-local my-mode-line-format nil)
(with-eval-after-load "pomodoro"
  (set-default 'my-mode-line-format mode-line-format)
  (defvar my-mode-line-sticky t)
  (defun my-mode-line-toggle-sticky ()
    (interactive)
    (setq my-mode-line-sticky (not my-mode-line-sticky))
    (if my-mode-line-sticky
      (my-mode-line-on)
      (my-mode-line-off)))

  (defun my-mode-line-off ()
    "Turn off mode line."
    (when (fboundp 'dimmer-mode)
      (dimmer-mode 1))
    (when (fboundp 'pomodoro:visualize-stop)
      (pomodoro:visualize-stop))
    (when mode-line-format
      (setq my-mode-line-format mode-line-format))
    (setq mode-line-format nil))

  (defun my-mode-line-on ()
    "Turn on mode line."
    (when (fboundp 'dimmer-mode)
      (dimmer-mode -1))
    (when (fboundp 'pomodoro:visualize-start)
      (pomodoro:visualize-start))
    (unless my-mode-line-format
      (error "Invalid value: %s" my-mode-line-format))
    (setq mode-line-format my-mode-line-format)
    (redraw-frame))

  (defun my-toggle-mode-line ()
    "Toggle mode line."
    (interactive)
    (if mode-line-format
      (my-mode-line-off)
      (my-mode-line-on))
    (message "%s" (if mode-line-format "( ! ∪! ) b ON !" "( ! ^! ) p OFF!")))

  (unless noninteractive
    (if shut-up-p
      (shut-up (my-toggle-mode-line))
```

```
(my-toggle-mode-line)))
(define-key moom-mode-map (kbd "<f5>") 'my-toggle-mode-line)
(add-hook 'find-file-hook
  (lambda () (unless my-mode-line-sticky
    (my-mode-line-off)))))
```

## 11.4. [winner.el] ウィンドウ構成の履歴を辿る

- ビルトインの `winner.el` を使います .

ウィンドウ分割状況と各ウィンドウで表示していたバッファの履歴を辿ることができます .  
`winner-undo` で直前の状態に戻せます . 例えば , 誤って `C-x 0` で分割ウィンドウを閉じた時にも , 即座に元の状態に戻すことが可能です .

```
(with-eval-after-load "postpone"
  (unless noninteractive
    (postpone-message "winner-mode")
    (when (require 'winner nil t)
      (define-key winner-mode-map (kbd "C-x g") 'winner-undo)
      (define-key winner-mode-map (kbd "C-(") 'winner-undo)
      (define-key winner-mode-map (kbd "C-)") 'winner-redo)
      (winner-mode 1)))))
```

## 11.5. TODO [shackle.el] ポップアップウィンドウの制御

[ONGOING]

- `popwin` の後発パッケージ
- `popwin.el` にある `dedicated` が使えないので , 本当に移行するか検討中 .
  - ウィンドウからフォーカスが外れた時に自動的にウィンドウを消す機能
- 以下の設定では , パッケージ読み込みのトリガーを `postpone` にひも付け
- `align` がないと , `size` が効かない ?
- バッファ名に正規表現を使う時は , `regexp` が必要 .
- `org` 関連でデイスパッチャとそれを経由するバッファは制御が効かない ?

```
(with-eval-after-load "postpone"
  (when (require 'shackle nil t)
    (setq shackle-default-ratio 0.33)
    (setq shackle-rules
      '(("*osx-dictionary*" :align above :popup t)
        ("*wclock*" :align above :popup t :select t)
        ("*Checkdoc Status*" :align above :popup t :noselect t)
        ;; ("*Help*" :align t :select 'above :popup t :size 0.3)
```

```
;; ("^\\*Helm.+" :regexp t :align above :size 0.2)
))
(unless noninteractive
  (postpone-message "shackle")
  (shackle-mode 1)))
```

### 11.5.1. checkdoc.el 用の追加設定

M-x checkdoc 実行時に checkdoc-minor-mode-map に checkdoc のウィンドウを閉じるキーバインドを設定．q または C-g を押下すると，マイナーモードが終了し，設定したキーバインドが無効になる．マイナーモードが有効な間は，q が占有されてバッファに入力できなくなる．その場合は，必要に応じて M-x my-delete-checkdoc-window を実行する．

```
(with-eval-after-load "checkdoc"
  (defun my-delete-checkdoc-window ()
    (interactive)
    (let ((checkdoc-window (get-buffer-window "*Checkdoc Status*")))
      (when checkdoc-window
        (delete-window checkdoc-window)))
    (checkdoc-minor-mode -1))

  (defun ad:checkdoc ()
    (interactive)
    (checkdoc-minor-mode 1)
    (define-key checkdoc-minor-mode-map (kbd "q")
      'my-delete-checkdoc-window)
    (define-key checkdoc-minor-mode-map (kbd "C-g")
      'my-delete-checkdoc-window))

  (advice-add 'checkdoc :before #'ad:checkdoc))
```

## 11.6. [e2wm.el] 二画面表示

[OBSOLETE]

使用頻度が低くなりました．

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>

```
(when (autoload-if-found
  ' (my-e2wm:dp-two e2wm:dp-two e2wm:start-management)
  "e2wm" nil t)

  (with-eval-after-load "e2wm"
    (defun my-e2wm:dp-two ()
      (interactive)
      (e2wm:dp-two)
      (setq e2wm:c-two-recipe
        '(- (:lower-size 10)
          (| left right)
          sub))
      (setq e2wm:c-two-wininfo
```



```

      '(:name left )
      (:name right )
      (:name sub :default-hide t)))

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

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

;; 幅を画面の最大に矯正
(add-hook 'e2wm:pre-start-hook
  (lambda ()
    (set-frame-width
     (selected-frame)
     (/ (- (display-pixel-width) 30) (frame-char-width)))))

;; 幅をデフォルト値に戻す
(add-hook 'e2wm:post-stop-hook
  (lambda ()
    (set-frame-width (selected-frame)
                     moom--frame-width)))

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

;; setting for e2wm
(when (autoload-if-found
      '(change-frame-double-window
        change-frame-single-window)
      "frame-ctr-e2wm" nil t)
  ;; 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))

```

## 11.7. [tabbar-ruler] バッファをタブ切り替え可能に

[OBSOLETE]

- 不要かな

```

(when (require 'tabbar-ruler nil t)
  ;; (when (require 'tabbar-ruler nil t)
  (setq tabbar-ruler-global-tabbar t) ; If you want tabbar
  (setq tabbar-ruler-popup-menu t) ; If you want a popup menu.
  (setq tabbar-ruler-popup-toolbar t) ; If you want a popup toolbar
  ;; (setq tabbar-ruler-fancy-tab-separator 'round)
  ;; (setq tabbar-ruler-fancy-current-tab-separator 'round)
  (setq tabbar-ruler-invert-deselected nil)
  (setq tabbar-ruler-modified-symbol t))

```

## 11.8. [elscreen.el] Emacs バッファをタブ化

[OBSOLETE]

- cycle-buffer で十分かなと言う感じで不使用になりました .

```
;;; 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)
```

## 11.9. [popwin.el] ポップアップウィンドウの制御

[OBSOLETE]

- 現在は緩やかに shackle に移行中です .
- ただし , dedicated オプションが使えないので , ウィンドウからフォーカスが外れた時に自動的にウィンドウを消すことができません .
- それぞれに q を実装すれば , 近い動作にはできる .
- <https://github.com/m2ym/popwin-el/>
- popwin:display-buffer を autoload してもうまいかない .

postpone の後に呼んでいるのは , Emacs 起動時の単純な高速化対策です .

```
;; 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)  
  
(with-eval-after-load "postpone"  
  (when (require 'popwin nil t)  
    (unless noninteractive  
      (postpone-message "popwin-mode")  
      (popwin-mode 1))  
  
    ;; Performed in shackle  
    (push '("*Help*" :height 20 :position top :dedicated t)  
          popwin:special-display-config)  
    (push '("*osx-dictionary*" :height 20 :position top :dedicated t)  
          popwin:special-display-config)  
    (push '("*wclock*" :height 10 :position top :dedicated t)  
          popwin:special-display-config)  
  
    ;; Checking...  
    (push '("*Checkdoc Status*" :position top :dedicated t)  
          popwin:special-display-config)  
    (push '("*frequencies*" :height 20 :position bottom :dedicated t)
```

```

        popwin:special-display-config)
(push ' ("CAPTURE-next.org" :height 10 :position bottom :noselect t)
      popwin:special-display-config)
(push ' ("CAPTURE-org-ical.org":height 10 :position bottom :noselect t)
      popwin:special-display-config)
(push ' ("dict-app-result" :height 20 :position bottom)
      popwin:special-display-config)
(push ' ("*timer*" :height 20 :position bottom :dedicated t)
      popwin:special-display-config)
(push ' ("Calendar" :position top :dedicated t)
      popwin:special-display-config)
(push ' ("*Org Dashboard*" :position bottom)
      popwin:special-display-config)
(push ' ("*Org Select*" :height 10 :position bottom)
      popwin:special-display-config)
(push ' ("*Org Grep*" :height 20 :position bottom :dedicated t)
      popwin:special-display-config)
(push ' ("*Occur*" :height 10 :position bottom)
      popwin:special-display-config)
(push ' ("*Shell Command Output*" :height 10 :position bottom :dedicated t)
      popwin:special-display-config)
(push ' ("*eshell*" :height 10 :position bottom)
      popwin:special-display-config)
;;;      (undo-tree-visualizer-buffer-name :height 10 :position top)
;;;      (undo-tree-diff-buffer-name :height 20 :position bottom)

;; Not performed
(push ' ("*Org Agenda*" :height 10 :position top)
      popwin:special-display-config)
))

```

## 12. フォント / 配色関連

### 12.1. 正規表現を見やすくする

Emacs Lisp の正規表現を色付けします .

```

(with-eval-after-load "postpone"
  (set-face-foreground 'font-lock-regexp-grouping-backslash "#66CC99")
  (set-face-foreground 'font-lock-regexp-grouping-construct "#9966CC"))

```

Emacs Lisp で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](#)

### 12.2. 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 .

```
(with-eval-after-load "postpone"
  (unless noninteractive
    (postpone-message "generic-x")
    (require 'generic-x nil t)))
```

### 12.3. カーソル行に色をつける

- ビルトインの `hl-line` を使います .
- <http://murakan.cocolog-nifty.com/blog/2009/01/emacs-tips-1d45.html>
- <http://www.emacswiki.org/cgi-bin/emacs/highlight-current-line.el>

`postpone` の後に呼んでいるのは , Emacs 起動時の単純な高速化対策です .

```
(with-eval-after-load "postpone"
  (if (display-graphic-p)
    (unless noninteractive
      (postpone-message "global-hl-line-mode"))
    (setq hl-line-face 'underline))
  (global-hl-line-mode 1))
```

色の設定は `custom-set-faces` で .

```
(custom-set-faces
 ' (hl-line
   (((background dark)) :background "#484c5c")
   (t (:background "#DEEDFF")))))
```

色の変更ではなく , 下線に変更したい場合は , `(setq hl-line-face 'underline)` とする .

### 12.4. カーソルを点滅させない

以下の例では , 入力が止まってから 30 秒後に 0.2 秒間隔で点滅します . 次に入力が始まるまで , 点滅が続きます .

`postpone` の後に呼んでいるのは , Emacs 起動時の単純な高速化対策です .

```
(with-eval-after-load "postpone"
  (setq blink-cursor-blinks 0)
  (setq blink-cursor-interval 0.2)
  (setq blink-cursor-delay 30)
  (unless noninteractive
    (postpone-message "blink-cursor-mode")
    (blink-cursor-mode 1)))
```

## 12.5. カーソル位置のフォントを確認

M-x describe-char すると、カーソル位置のフォントの情報が別バッファに表示されます。

## 12.6. 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。最近では、[Migu 2M](#) を気に入って使っています。

Table 12.6.1: 環境別の使用フォント

	ASCII	日本語
Mac	Monaco	Migu 2M 又は ヒラギノ丸ゴ
Windows	Inconsolata	メイリオ
Linux	Inconsolata	MigMix

- <http://d.hatena.ne.jp/setoryohei/20110117/1295336454>
- [Unicode Fonts for Ancient Scripts](#)

```
; 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
(defconst my-font-size 12)
(defconst my-ja-font "Migu 2M") ;; "Hiragino Maru Gothic Pro"
(defconst my-ascii-font "Monaco") ;; "Inconsolata", Monaco
;; (defconst my-ja-font "Hiragino Maru Gothic Pro") ;; "Hiragino Maru Gothic Pro"
;; (defconst my-ascii-font "Inconsolata") ;; "Inconsolata", Menlo, "Ricty Diminished"
(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))
(defun my-unicode-font-setter (spec)
  (set-fontset-font nil 'unicode spec))
(defun my-font-config (&optional size ascii ja)
  "Font config.
- SIZE: font size for ASCII and Japanese (default: 12)
```

```

- ASCII: ascii font family (default: \"Monaco\")
- JA: Japanese font family (default: \"Migu 2M\")
"
  (when (memq window-system '(mac ns))
    (let ((font-size (or size my-font-size))
          ;; (unicode-font (or uc "FreeSerif"))
          (ascii-font (or ascii my-ascii-font))
          (ja-font (or ja my-ja-font)))
      ;; (my-unicode-font-setter
      ;; (font-spec :family unicode-font :size font-size))
      (my-ascii-font-setter (font-spec :family ascii-font :size font-size))
      (my-ja-font-setter (font-spec :family ja-font :size font-size))))))

(cond
  ;; CocoaEmacs
  ((memq window-system '(mac ns))
    (when (>= emacs-major-version 23)

      ;; Fix ratio provided by set-face-attribute for fonts display
      (setq face-font-rescale-alist
        '(("^-apple-hiragino.*" . 1.0) ; 1.2
          (".*Migu.*" . 1.2)
          (".*Ricty.*" . 1.0)
          (".*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
      ;; (my-font-config) ;; see `my-apply-theme'
    )

  ((eq window-system 'ns)
    ;; Anti aliasing with Quartz 2D
    (when (boundp 'mac-allow-anti-aliasing)
      (setq mac-allow-anti-aliasing t)))

  ((eq window-system 'w32) ; windows7
    (let ((font-size 14)
          (font-height 100)
          (ascii-font "Inconsolata")
          (ja-font "メイリオ")) ;; Meiryo UI
      (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

  ((eq window-system 'x) ; for SuSE Linux 12.1
    (let
      ((font-size 14)
       (font-height 100)
       (ascii-font "Inconsolata")
       ;; (ja-font "MigMix 1M")
       (ja-font "Migu 2M"))
      (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

```

### 12.6.1. 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 ．

### 12.6.2. 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．

## 12.7. 行間を制御する

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

## 12.8. 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．

- <http://d.hatena.ne.jp/syohex/20111228/1325086893>

```
(with-eval-after-load "diff-mode"
  (set-face-attribute 'diff-added nil
    :background nil :foreground "green"
    :weight 'normal)

  (set-face-attribute 'diff-removed nil
    :background nil :foreground "firebrick1"
    :weight 'normal)

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

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

## 12.9. 背景を黒系色にする

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

## 12.10. 日中と夜中でテーマを切り替える

私は使っていませんが，簡単にテーマを切り替えるためのパッケージ([heaven-and-hell](#))も存在します．

```
;;;###autoload
(defun my-daylight-theme ()
  (interactive)
  (when (require 'daylight-theme nil t)
    (mapc 'disable-theme custom-enabled-themes)
    (load-theme 'daylight t)
    (setq default-frame-alist
          (delete (assoc 'ns-appearance default-frame-alist)
                  default-frame-alist))
    (setq default-frame-alist
          (delete (assoc 'ns-transparent-titlebar default-frame-alist)
                  default-frame-alist))
    (add-to-list 'default-frame-alist '(ns-transparent-titlebar . t))
    (add-to-list 'default-frame-alist '(ns-appearance . light))
    ;; (redraw-frame)
  ))

;;;###autoload
(defun my-night-theme ()
  (interactive)
  (when (require 'night-theme nil t) ;; atom-one-dark-theme
    (mapc 'disable-theme custom-enabled-themes)
    (load-theme 'night t)
    (setq default-frame-alist
          (delete (assoc 'ns-appearance default-frame-alist)
                  default-frame-alist))
    (setq default-frame-alist
          (delete (assoc 'ns-transparent-titlebar default-frame-alist)
                  default-frame-alist))
    (add-to-list 'default-frame-alist '(ns-transparent-titlebar . t))
    (add-to-list 'default-frame-alist '(ns-appearance . dark))
    ;; (redraw-frame)
  ))
```

## 12.11. 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．

```
(when (display-graphic-p)
  (declare-function my-night-time-p "init" (begin end))
  (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 my-frame-appearance nil) ;; {nil, 'dark, 'light}
(defun my-apply-theme (&optional type)
  (interactive "MType (light or dark): ")
  (setq my-frame-appearance
    (cond ((equal "light" type)
           'light)
          ((equal "dark" type)
           'dark)
          (t
           my-frame-appearance)))
  (cond ((eq my-frame-appearance 'dark)
         (my-night-theme))
        ((eq my-frame-appearance 'light)
         (my-daylight-theme))
        (t
         (let ((night-time-in 21)
               (night-time-out 5))
           (if (my-night-time-p (* night-time-in 60) (* night-time-out 60))
               (my-night-theme)
               (my-daylight-theme))))))
  (when (fboundp 'mac-get-current-input-source)
    (my-apply-cursor-config))
  (my-font-config)) ;; apply font setting

(my-apply-theme)) ;; this may override or reset font setting

```

## 12.12. [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 (autoload-if-found
      '(rainbow-mode)
      "rainbow-mode" nil t)

  (dolist (hook '(emmet-mode-hook emacs-lisp-mode-hook org-mode-hook))
    (add-hook hook #'rainbow-mode)))

```

### 12.12.1. 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
```

## 12.13. [edit-color-stamp] Qt 経由でカラーピッカーを使う

- [sabof/edit-color-stamp: Edit hex color stamps using a QT, or the internal color picker](https://github.com/sabof/edit-color-stamp)
- Macintosh でもちゃんと使えます .

```
(when (and (executable-find "qt_color_picker")
  (autoload-if-found
    ' (edit-color-stamp)
    "edit-color-stamp" nil t))

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-c f c p") 'edit-color-stamp)))
```

( 設定手順 )

- QT をインストールして , qmake に PATH を通す ( 例 : /usr/local/opt/qt/bin )

- `edit-color-stamp` を入手してインストール先のフォルダに移動
- さらに , `qt_color_picker` ディレクトリに移動
- `qt_color_picker.pro` を編集して , "QT += " に "widgets" を追加して保存する .

```
QT += core gui widgets
```

- `qmake qt_color_picker.pro && make` を実行
- 出来上がった `qt_color_picker.app` を `/Applications/` にコピー ( 移動 )
- `exec-path` にパスを通す

```
(add-to-list 'exec-path
"/Applications/qt_color_picker.app/Contents/MacOS/")
```

- `(executable-find "qt_color_picker")` でプログラムを呼べることを確認
- `#FFFFFF` などの色の上で `M-x edit-color-stamp` を実行する
- カラーピッカーで色を選んで「OK」すれば , `#FFFFFF` が更新される .

## 12.14. *TODO [volatile-highlights]* コピペした領域を強調

コピペ直後の数秒に限定して , コピペした領域をハイライトします . さらに , 所定の間隔で表示色を変えることで , ハイライト色が徐々に消えていくようなエフェクトにしています .

```
(when (autoload-if-found
      '(volatile-highlights-mode my-vhl-change-color)
      "volatile-highlights" nil t)

  (dolist (hook '(org-mode-hook emacs-lisp-mode-hook emmet-mode-hook))
    (add-hook hook #'volatile-highlights-mode))

  (with-eval-after-load "volatile-highlights"
    (set-face-attribute
      'vhl/default-face nil :foreground "#FF3333" :background "#FFCDDC")
    (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 ()
    (interactive)
    (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 "#FFCDDC"))

(defun my-yank (&optional ARG)
  (interactive)
  (yank ARG)
  (when window-system
    (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") 'my-org-yank)
  (defun my-org-yank ()
    (interactive)
    (org-yank)
    (when window-system
      (my-vhl-change-color))))))

```

## 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。

なお、Org Mode と組み合わせたい場合は、通知を `org-clock-in` と `org-clock-out` に連動させる [org-pomodoro.el](#) があります。

`postpone` の後に呼んでいるのは、Emacs 起動時の単純な高速化対策です。

```

(when (autoload-if-found
      '(pomodoro:start)
      "pomodoro" nil t)

  (with-eval-after-load "postpone"
    (when (and (not noninteractive)
              (not (boundp 'pomodoro:timer)))
      ;; 重複起動を回避
      (pomodoro:start nil)))

  (with-eval-after-load "pomodoro"
    ;; 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 "<<")
    (setq pomodoro:mode-line-long-rest-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 4)
    ;; 作業時間関連
    (setq pomodoro:work-time 25      ; 作業時間
          pomodoro:rest-time 5      ; 休憩時間
          pomodoro:long-rest-time 30 ; 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
    (custom-set-faces
      '(pomodoro:work-face
        (((background dark)) :foreground "#DB4C46" :bold t)
        (t (:foreground "#A130C4" :bold t)))) ;; #8248c4, #956dc4, #9d64c4
      '(pomodoro:rest-face
        (((background dark)) :foreground "#3869FA" :bold t)
        (t (:foreground "#203e6f" :bold t))))
      '(pomodoro:long-rest-face
        (((background dark)) :foreground "#008890" :bold t)
        (t (:foreground "#1c9b08" :bold t)))) ;; 00B800

    (defun my-pomodoro-status ()
      "Show the current `pomodoro' status in minibuffer when focus-in."
      (interactive)
      (when pomodoro:timer
        (let ((message-log-max nil))
          (message
            (format "[%d/%s] %s to go "
                    pomodoro:work-count
                    pomodoro:max-iteration
                    (pomodoro:time-to-string pomodoro:remainder-seconds))))))

    (add-hook 'focus-in-hook #'my-pomodoro-status)

    (defvar my-pomodoro-speak nil)
    (defun my-pomodoro-speak-toggle ()
      (interactive)
      (setq my-pomodoro-speak (not my-pomodoro-speak)))

    (when (memq window-system '(mac ns))

```

```

;; 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))))))

(declare-function my-pomodoro-notify "init" nil)
(defun my-pomodoro-notify ()
  (my-desktop-notification
    "Pomodoro"
    (concat "三三^(*°▽°)/ Go #"
      (format "%s" (1+ pomodoro:work-count))) nil "Glass"))
(add-hook 'pomodoro:finish-work-hook #'my-pomodoro-notify)))

```

## 13.2. [pomodoro.el] 続・ポモドーロの実践

さらに，モードラインに出すスタータスを表すサインに動きを付けます．ただ頻繁にモードラインを更新して描画するので，環境によっては動作が重くなるかもしれません．

各 interval の数値を `setq` で後から変更する場合は，その直後に `pomodoro:activate-visual-work-sign` もしくは変更する interval に対応する同様の関数を実行してください．モードラインに変更が反映されます．

```

(with-eval-after-load "pomodoro"
  ;; 追加実装

```







```

        (pomodoro:activate-visual-long-rest-sign))
      (t
        (pomodoro:update-work-sign)
        (pomodoro:activate-visual-work-sign))))

(defun pomodoro:visualize-stop ()
  (setq my-pomodoro-visualize nil)
  (setq pomodoro:mode-line-work-sign "")
  (setq pomodoro:mode-line-rest-sign "")
  (setq pomodoro:mode-line-long-rest-sign "")
  (force-mode-line-update t)
  (when (timerp pomodoro:update-sign-timer)
    (cancel-timer pomodoro:update-sign-timer)))

(defun ad:pomodoro:start (f &rest minutes)
  "Extensions to stop pomodoro and timers"
  (interactive "P")
  (pomodoro:visualize-start)
  (apply f minutes))

(defun ad:pomodoro:stop (f &rest do-reset)
  "Extensions to stop pomodoro and timers"
  (interactive)
  (pomodoro:visualize-stop)
  (apply f do-reset))

(when my-pomodoro-visualize
  (advice-add 'pomodoro:start :around #'ad:pomodoro:start)
  (advice-add 'pomodoro:stop :around #'ad:pomodoro:stop))

;; work
(defun pomodoro:update-work-sign ()
  "Update pomodoro work-sign on modeline."
  (when my-pomodoro-visualize
    (setq pomodoro:mode-line-work-sign
      (car pomodoro:mode-line-work-sign-list))
    (setq pomodoro:mode-line-work-sign-list
      (pomodoro:list-rotate pomodoro:mode-line-work-sign-list))
    (force-mode-line-update t)))

(defun pomodoro:activate-visual-work-sign ()
  (pomodoro:activate-visual-sign
    'pomodoro:update-work-sign pomodoro:update-work-sign-interval))

;; rest
(defun pomodoro:update-rest-sign ()
  "Update pomodoro rest-sign on modeline."
  (when my-pomodoro-visualize
    (setq pomodoro:mode-line-rest-sign
      (car pomodoro:mode-line-rest-sign-list))
    (setq pomodoro:mode-line-rest-sign-list
      (pomodoro:list-rotate pomodoro:mode-line-rest-sign-list))
    (force-mode-line-update t)))

(defun pomodoro:activate-visual-rest-sign ()
  (pomodoro:activate-visual-sign
    'pomodoro:update-rest-sign pomodoro:update-rest-sign-interval))

;; long rest
(defun pomodoro:update-long-rest-sign ()
  "Update pomodoro long-rest-sign on modeline."

```

```

(when my-pomodoro-visualize
  (setq pomodoro:mode-line-long-rest-sign
    (car pomodoro:mode-line-long-rest-sign-list))
  (setq pomodoro:mode-line-long-rest-sign-list
    (pomodoro:list-rotate pomodoro:mode-line-long-rest-sign-list))
  (force-mode-line-update t)))

(defun pomodoro:activate-visual-long-rest-sign ()
  (pomodoro:activate-visual-sign
    'pomodoro:update-long-rest-sign pomodoro:update-long-rest-sign-interval))

;; ステータスが切り替わる時に表示を入れ替える
(when my-pomodoro-visualize
  (add-hook 'pomodoro:finish-rest-hook #'pomodoro:activate-visual-work-sign)
  (add-hook 'pomodoro:finish-work-hook #'pomodoro:activate-visual-rest-sign)
  (add-hook 'pomodoro:long-rest-hook
    #'pomodoro:activate-visual-long-rest-sign)))

```

### 13.3. [google-this.el] 単語をグーグル検索

カーソル下の単語を検索して，結果をブラウザで受け取ります．google-this を直接呼ぶと検索確認を聞かれるので，すぐに検索するようにカスタマイズします．M-x google-this-word を使うのも手ですが，ハイフンで連結された文字列を拾えないので好みがわかります．

```

(when (autoload-if-found
  ' (my-google-this google-this google-this-word)
  "google-this" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-c f g") 'my-google-this))

  (with-eval-after-load "google-this"
    (defun my-google-this ()
      (interactive)
      (google-this (current-word) t))))

```

### 13.4. [lingr.el] チャットルームに自動参加

init.el 読書会が毎週土曜日の 22 時に開催されています．作業に集中していると，つつい忘れてしまうので，その時間が来たら自動的にログインするように設定しています．ユーザ名とパスワードは，セキュリティを考慮して別なファイルに記載しています．

```

(when (autoload-if-found
  ' (lingr-login my-lingr-login)
  "lingr" nil t)

  (with-eval-after-load "lingr"
    (custom-set-variables
      ' (lingr-icon-mode t)
      ' (lingr-icon-fix-size 24)
      ' (lingr-image-convert-program (or (executable-find "convert")
        lingr-image-convert-program)))))

```

```
(when (autoload-if-found
      '(my-lingr-login)
      "utility" nil t)

      (with-eval-after-load "postpone"
        (when (future-time-p "23:00")
          ;; do not use `run-at-time' at booting since diary-lib.el
          ;; will be loaded. It requires loading cost.
          (run-at-time "23:00" nil 'my-lingr-login))))
```

## 13.5. [multi-term.el] ターミナル

たまに使いたくなるので一応 .

```
(when (autoload-if-found
      '(multi-term)
      "multi-term" nil t)

      (with-eval-after-load "multi-term"
        (setenv "HOSTTYPE" "intel-mac")
        (my-night-theme))))
```

## 13.6. [osx-lib.el] OSX 用ユーティリティ

[MACOS]

- [raghavgautam/osx-lib](https://github.com/raghavgautam/osx-lib)
- OSX の機能呼び出すためのツール集
- `osx-lib-say-region` を呼び出せば , 選択領域を機械音で発声させられます .
- `selected.el` と組み合わせると選択後にシングルキーで実施できるのでさらに便利です .

```
(when (autoload-if-found
      '(osx-lib-say osx-lib-say-region)
      "osx-lib" nil t)

      (with-eval-after-load "osx-lib"
        (custom-set-variables
          '(osx-lib-say-ratio 100)
          '(osx-lib-say-voice "Samantha"))))
```

例えば , カエルの歌の輪唱ができます . 以下のソースブロックで `C-c C-c` を適切なタイピングで押してみてください .

```
#+begin_src emacs-lisp :results silent
(setq osx-lib-say-voice "Kyoko") ;; Samantha
(setq osx-lib-say-rate 90) ;; min 90 for Samantha
(setq yoursript "かえるのうたが      きこえてくるよ      クア クア クア クア ケケケケ
ケケケケ クア クア クア")
(osx-lib-say yoursript)
```

```
#+end_src
```

### 13.7. *iterm2* を Emacs から呼び出したい

[MACOS]

C-M-i を *iterm2.app* の呼び出しに割り当てます。シェルでの作業はどうも *eshell* では満足できないので、なお *flyspell* を使っていると、遅延呼び出しした後にキーバインドを奪われるので、取り返します。

`cmd-to-open-iterm2` の実装は、[./utility.el](#) にあります。

```
(when (autoload-if-found
      '(my-cmd-to-open-iterm2)
      "utility" nil t)

  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-M-i") #'my-cmd-to-open-iterm2))

  (with-eval-after-load "flyspell"
    (define-key flyspell-mode-map (kbd "C-M-i") #'my-cmd-to-open-iterm2))

  (with-eval-after-load "org"
    (define-key org-mode-map (kbd "C-M-i") #'my-cmd-to-open-iterm2)))
```

### 13.8. カレントバッファのあるディレクトリをターミナルで表示

```
(with-eval-after-load "postpone"
  (global-set-key (kbd "C-c f t") 'my-open-current-directory-in-terminal))
```

### 13.9. [\[gif-screencast.el\]](#) ユーザアクションのたびにスクショを取る

- [Ambrevar/emacs-gif-screencast: One-frame-per-action GIF recording for optimal quality/size ratio](#)
- ~~色々 Linux 用に設計・前提とされていて、今のところ Mac で使うのはちょっと厳しい。~~
- macOS 用に追加実装しました。 [本家にマージ](#)されています。
- スクショ専門ツールもあります。こちらは `import` コマンドを利用します。
- [tarsius/frameshot: Take screenshots of a frame](#)

```
(when (autoload-if-found
      '(gif-screencast)
      "gif-screencast" nil t)

  (with-eval-after-load "gif-screencast"
    (setq gif-screencast-want-optimized nil))
```

```

(setq gif-screencast-args '("-x"))
(setq gif-screencast-capture-format "ppm")

;; Start... M-x gif-screencast
(define-key gif-screencast-mode-map (kbd "<f5>") 'gif-screencast-stop)
(define-key gif-screencast-mode-map (kbd "S-<f5>")
  'gif-screencast-toggle-pause)

;; 拡張
(defcustom gif-screencast-additional-normal-hooks '()
  "A list of hooks. These hooks activate `gif-screencast-capture'."
  :group 'gif-screencast
  :type '(repeat hook))

(add-to-list 'gif-screencast-additional-normal-hooks
  'window-size-change-functions) ;; for which-key.el, as of 26.1
(add-to-list 'gif-screencast-additional-normal-hooks
  'window-configuration-change-hook) ;; for which-key.el
(add-to-list 'gif-screencast-additional-normal-hooks 'focus-in-hook)
(add-to-list 'gif-screencast-additional-normal-hooks 'focus-out-hook)
(add-to-list 'gif-screencast-additional-normal-hooks 'minibuffer-setup-
hook)
(add-to-list 'gif-screencast-additional-normal-hooks 'minibuffer-exit-hook)
;; (add-to-list 'gif-screencast-additional-normal-hooks 'pre-redisplay-
functions)
;; pre-redisplay-functions はやばい .

;; modification-hooks
(defun ad:gif-screencast ()
  (dolist (hook gif-screencast-additional-normal-hooks)
    (add-hook hook #'gif-screencast-capture)))
(advice-add 'gif-screencast :after #'ad:gif-screencast)

(defun ad:gif-screencast-stop ()
  (dolist (hook gif-screencast-additional-normal-hooks)
    (remove-hook hook 'gif-screencast-capture)))
(advice-add 'gif-screencast-stop :after #'ad:gif-screencast-stop)

(defun ad:gif-screencast-toggle-pause ()
  (if (memq 'gif-screencast-capture (default-value 'pre-command-hook))
    (dolist (hook gif-screencast-additional-normal-hooks)
      (remove-hook hook 'gif-screencast-capture))
    (dolist (hook gif-screencast-additional-normal-hooks)
      (add-hook hook #'gif-screencast-capture))))
(advice-add 'gif-screencast-toggle-pause
  :before #'ad:gif-screencast-toggle-pause)

(defun ad:gif-screencast-opendir ()
  "Open the output directory when screencast is finished."
  (if (not (eq system-type 'darwin))
    (my-gif-screencast-opendir-dired)
    (shell-command-to-string
      (concat "open " gif-screencast-screenshot-directory))
    (shell-command-to-string
      (concat "open " gif-screencast-output-directory))))
(advice-add 'gif-screencast-stop :before #'ad:gif-screencast-opendir)

(defun my-gif-screencast-opendir-dired ()
  "Open directories for screenshots and generated GIFs by Dired."
  (interactive)

```

```
(dired gif-screencast-output-directory)
(dired gif-screencast-screenshot-directory)))
```

### 13.10. [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．

```
(defconst utility-autoloads
  '(my-date
    my-window-resizer my-cycle-bullet-at-heading my-open-file-ring
    ;; selected
    my-org-list-insert-items
    my-org-list-insert-checkbox-into-items
    ;; M-x
    my-eval-org-buffer
    my-org-list-delete-items
    my-org-list-delete-items-with-checkbox
    my-org-list-insert-items-with-checkbox
    my-org-list-delete-checkbox-from-items
    ;; others
    kyoko-mad-mode-toggle mac:delete-files-in-trash-bin library-p
    org2dokuwiki-cp-kill-ring open-current-directory
    reload-ical-export my-show-org-buffer my-get-random-string init-auto-
install
    add-itemize-head add-itemize-head-checkbox insert-formatted-current-date
    insert-formatted-current-time insert-formatted-signature
    export-timeline-business export-timeline-private chomp
    count-words-buffer do-test-applescript my-open-current-directory-in-
terminal
    delete-backup-files recursive-delete-backup-files describe-timer
    my-browse-url-chrome my-setup-package-el my-kill-emacs-hook-show))

(when (autoload-if-found
      utility-autoloads
      "utility" nil t)
  (with-eval-after-load "postpone"
    (global-set-key (kbd "C-M--") 'my-cycle-bullet-at-heading)
    (global-set-key (kbd "<f12>") 'my-open-file-ring)
    (global-set-key (kbd "C-c t") 'my-date)
    (global-set-key (kbd "C-c f 4") 'my-window-resizer)))
```

### 13.11. [manage-minor-mode.el] マイナーモードの視覚的な管理

マイナーモードの動作状況を視覚的に確認できます．さらに，モードの ON/OFF を指定できます．

- [minor-mode を簡単に切り替える manage-minor-mode.el Emacs Lisp - Web 学び](#)

```
(when (autoload-if-found
      '(manage-minor-mode)
      "manage-minor-mode" nil t)
  (with-eval-after-load "manage-minor-mode"
```

```
(define-key manage-minor-mode-map (kbd "q")
  (lambda () (interactive)
    (delete-window (get-buffer-window "*manage-minor-mode*")))))
```

### 13.12. TODO [password-store.el] パスワード管理

- <http://h12.me/article/password-management>
- <http://www.tricksofthetrades.net/2015/07/04/notes-pass-unix-password-manager/>
- Add no-tty into ~/.gnupg/gpg.conf if an error occurred.
- システムで pass コマンドを使えることが条件

```
(if (executable-find "pass")
  (when (autoload-if-found
        '(helm-pass)
        "helm-pass" nil t)

    (with-eval-after-load "postpone"
      (global-set-key (kbd "C-c f p") 'helm-pass)))
  (message "--- pass is NOT installed."))
```

もしくは, id-manager.el もパスワード管理に利用できる. こちらは特定のコマンドがシステムにインストールされていなくても動く.

- The encrypted file is stored as ~/.idm-db.gpg.
- <http://d.hatena.ne.jp/kiwanami/20110221/1298293727>

```
(when (autoload-if-found
      '(id-manager)
      "id-manager" nil t)

  (with-eval-after-load "id-manager"
    (setenv "GPG_AGENT_INFO" nil)))
```

### 13.13. TODO [network-watch.el] ネットワークインターフェイスの状態を監視

- プロキシ内での挙動は不明.

```
(when (autoload-if-found
      '(network-watch-mode
        network-watch-active-p ad:network-watch-update-lighter)
      "network-watch" nil t)

  (with-eval-after-load "postpone"
    (unless noninteractive
```

```
(postpone-message "network-watch-mode")
(if shutup-p
  (shut-up (network-watch-mode 1))
  (network-watch-mode 1)))

(with-eval-after-load "network-watch"
  (defun ad:network-watch-update-lighter ()
    "Return a mode lighter reflecting the current network state."
    (unless (network-watch-active-p) " ↓NW↓"))
  (advice-add 'network-watch-update-lighter
    :override #'ad:network-watch-update-lighter)))
```

## 14. おわりに

以上が、私の `init.el` とその説明です。