Core Emacs Customizations everyone should have

Arne Babenhauserheide

April 3, 2013

Contents

1 Intro

I have been tweaking my emacs configuration for years, now, and I added quite some cruft. But while searching for the right way to work, I also found some gems which I direly miss in pristine emacs.

This file is about those gems.

Babcore is strongly related to Prelude. Actually it is just like prelude, but with the stuff *I* consider essential.

But before we start, there is one crucial piece of advice which everyone who uses Emacs should know:

```
C-g: abort
```

Hold control and hit g.

That gets you out of almost any situation. If anything goes wrong, just hit C-g repeatedly till the problem is gone - or you cooled off far enough to realize that a no-op is the best way to react.

To repeat: If anything goes wrong, just hit **C-g**.

2 Package Header

As Emacs package, babcore needs a proper header.

```
;; Copyright (C) 2013 Arne Babenhauserheide

;; Author: Arne Babenhauserheide (and various others in Emacswiki and elsewhere).
;; Maintainer: Arne Babenhauserheide
;; Created 03 April 2013
;; Version: 0.0.2
;; Version Keywords: core configuration

;; This program is free software; you can redistribute it and/or
;; modify it under the terms of the GNU General Public License
;; as published by the Free Software Foundation; either version 3
;; of the License, or (at your option) any later version.

;; This program is distributed in the hope that it will be useful,
;; but WITHOUT ANY WARRANTY; without even the implied warranty of
;; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;; GNU General Public License for more details.

;; You should have received a copy of the GNU General Public License
;; along with this program. If not, see <http://www.gnu.org/licenses/>.

;;; Commentary:
;; Quick Start / installation:
;; 1. Download this file and put it next to other files Emacs includes
;; 2. Add this to you .emacs file and restart emacs:
;;      (require 'babcore)
;;
;; Use Case: Use a common core configuration so you can avoid the
;;   tedious act of gathering all the basic stuff over the years and
;;   can instead concentrate on the really cool new stuff Emacs offers
;;   you.
;;
;; Todo:
;;

;;; Change Log:
;; 2013-04-03 - Minor adjustments
;; 2013-02-29 - Initial release

;;; Code:
```

Additionally it needs the proper last line. See finish up for details.

3 Feature Gems

3.1 package.el, full setup

The first thing you need in emacs 24. This gives you a convenient way to install just about anything, so you really should use it.

Also I hope that it will help consolidate the various emacs tips which float around into polished packages by virtue of giving people ways to actually get the package by name -

and keep it updated almost automatically.

```
;; Convenient package handling in emacs

(require 'package)
;; use packages from marmalade
(add-to-list 'package-archives '("marmalade" . "http://marmalade-repo.org/packages/"))
;; and the old elpa repo
(add-to-list 'package-archives '("elpa-old" . "http://tromey.com/elpa/"))
;; and automatically parsed versiontracking repositories.
(add-to-list 'package-archives '("melpa" . "http://melpa.milkbox.net/packages/"))

;; Make sure a package is installed
(defun package-require (package)
  "Install a PACKAGE unless it is already installed
or a feature with the same name is already active.

Usage: (package-require 'package)"
  ; try to activate the package with at least version 0.
  (package-activate package '(0))
  ; try to just require the package. Maybe the user has it in his local config
  (condition-case nil
      (require package)
    ; if we cannot require it, it does not exist, yet. So install it.
    (error (package-install package))))

;; Initialize installed packages
(package-initialize)
;; package init not needed, since it is done anyway in emacs 24 after reading the init
;; but we have to load the list of available packages
(package-refresh-contents)
```

3.2 Flymake

Flymake is an example of a quite complex feature which really everyone should have.

It can check any kind of code, and actually anything which can be verified with a program which gives line numbers.

As alternative you might want to look into flycheck. It looks really cool, but I don't yet have experience with it, so I cannot recommend it, yet.

```
;; Flymake: On the fly syntax checking

; stronger error display
(defface flymake-message-face
  '((((class color) (background light)) (:foreground "#b2dfff"))
    (((class color) (background dark))  (:foreground "#b2dfff")))
  "Flymake message face")

; show the flymake errors in the minibuffer
(package-require 'flymake-cursor)
```

3.3 auto-complete

This gives you inline auto-completion preview with an overlay window - even in the text-console. Partially this goes as far as API-hints (for example for elisp code). Absolutely essential.

```
;; Inline auto completion and suggestions
(package-require 'auto-complete)
```

3.4 ido

To select a file in a huge directory, just type a few letters from that file in the correct order, leaving out the non-identifying ones. Darn cool!

```
; use ido mode for file and buffer Completion when switching buffers
(require 'ido)
(ido-mode t)
```

3.5 printing

Printing in pristine emacs is woefully inadequate, even though it is a standard function in almost all other current programs.

It can be easy, though:

```
;; Convenient printing
(require 'printing)
(pr-update-menus t)
; make sure we use localhost as cups server
(setenv "CUPS_SERVER" "localhost")
(package-require 'cups)
```

3.6 outlining everywhere

Code folding is pretty cool to get an overview of a complex structure. So why shouldn't you be able to do that with any kind of structured data?

```
; use allout minor mode to have outlining everywhere.
(allout-mode)
```

3.7 Syntax highlighting

Font-lock is the emacs name for syntax highlighting - in just about anything.

```
; syntax highlighting everywhere
(global-font-lock-mode 1)
```

3.8 org and babel

Org-mode is that kind of simple thing which evolves to a way of life when you realize that most of your needs actually are simple - and that the complex things can be done in simple ways, too.

It provides simple todo-lists, inline-code evaluation (as in this file) and a full-blown literate programming, reproducible research publishing platform. All from the same simple basic structure.

It might change your life... and it is the only planning solution which ever prevailed against my way of life and organization.

```
; Activate org-mode
(require 'org)
; and some more org stuff

; http://orgmode.org/guide/Activation.html#Activation

; The following lines are always needed.  Choose your own keys.
(add-to-list 'auto-mode-alist '("\\.org\\'" . org-mode))
```

```
; And add babel inline code execution
; babel, for executing code in org-mode.
(org-babel-do-load-languages
 'org-babel-load-languages
 ; load all language marked with (lang . t).
 '((C . t)
   (R . t)
   (asymptote)
   (awk)
   (calc)
   (clojure)
   (comint)
   (css)
   (ditaa . t)
   (dot . t)
   (emacs-lisp . t)
   (fortran)
   (gnuplot . t)
   (haskell)
   (io)
   (java)
   (js)
   (latex)
   (ledger)
   (lilypond)
   (lisp)
   (matlab)
   (maxima)
   (mscgen)
   (ocaml)
   (octave)
   (org . t)
   (perl)
   (picolisp)
   (plantuml)
   (python . t)
   (ref)
   (ruby)
   (sass)
   (scala)
   (scheme)
   (screen)
   (sh . t)
   (shen)
   (sql)
   (sqlite)))
```

3.9 Nice line wrapping

If you're used to other editors, you'll want to see lines wrapped nicely at the word-border instead of lines which either get cut at the end or in the middle of a word.

global-visual-line-mode gives you that.

```
; Add proper word wrapping
(global-visual-line-mode t)
```

3.10 goto-chg

This is the kind of feature which looks tiny: Go to the place where you last changed something.

And then you get used to it and it becomes absolutely indispensable.

```
; go to the last change
(package-require 'goto-chg)
(global-set-key [(control .)] 'goto-last-change)
; M-. can conflict with etags tag search. But C-. can get overwritten
; by flyspell-auto-correct-word. And goto-last-change needs a really
; fast key.
(global-set-key [(meta .)] 'goto-last-change)
```

3.11 flyspell

Whenever you write prosa, a spellchecker is worth a lot, but it should not unnerve you.

Install aspell, then activate flyspell-mode whenever you need it.

It needs some dabbling, though, to make it work nicely with non-english text.

```
; Make german umlauts work.
(setq locale-coding-system 'utf-8)
(set-terminal-coding-system 'utf-8)
(set-keyboard-coding-system 'utf-8)
(set-selection-coding-system 'utf-8)
(prefer-coding-system 'utf-8)

;aspell und flyspell
(setq-default ispell-program-name "aspell")

;make aspell faster but less correctly
(setq ispell-extra-args '("--sug-mode=ultra" "-w" "T1\ss" ))
(setq ispell-list-command "list")
```

3.12 control-lock

If you have to do the same action repeatedly, for example with flyspell hitting next-error and next-correction hundreds of times, the need to press control can really be a strain for your fingers.

Sure, you can use viper-mode and retrain your hands for the completely alien command set of vim.

A simpler solution is adding a sticky control key - and that's what control-lock does: You get modal editing with your standard emacs commands.

Since I am a german, I simply use the german umlauts to toggle the control-lock. You will likely want to choose your own commands here.

```
; control-lock-mode, so we can enter a vi style command-mode with standard emacs keys.
(package-require 'control-lock)
; also bind M- and M-o toggling control lock.
(global-set-key (kbd "M-") 'control-lock-toggle)
(global-set-key (kbd "C-") 'control-lock-toggle)
(global-set-key (kbd "M-) 'control-lock-toggle)
(global-set-key (kbd "C-) 'control-lock-toggle)
(global-set-key (kbd "C-z") 'control-lock-toggle)
```

3.13 Basic key chords

This is the second strike for saving your pinky. Yes, Emacs is hard on the pinky. Even if it were completely designed to avoid strain on the pinky, it would still be hard, because any system in which you do not have to reach for the mouse is hard on the pinky.

But it also provides some of the neatest tricks to reduce that strain, so you can make Emacs your pinky saviour.

The key chord mode allows you to hit any two keys at (almost) the same time to invoke commands. Since this can interfere with normal typing, I would only use it for letters which are rarely typed after each other.

These default chords have proven themselves to be useful in years of working with Emacs.

```
; use key chords invoke commands
(package-require 'key-chord)
(key-chord-mode 1)
; buffer actions
(key-chord-define-global "vg"     'eval-region)
(key-chord-define-global "vb"     'eval-buffer)
(key-chord-define-global "cy"     'yank-pop)
(key-chord-define-global "cg"     "\C-c\C-c")
; frame actions
(key-chord-define-global "xo"     'other-window);
(key-chord-define-global "x1"     'delete-other-windows)
(key-chord-define-global "x0"     'delete-window)
(defun kill-this-buffer-if-not-modified ()
  (interactive)
  ; taken from menu-bar.el
  (if (menu-bar-non-minibuffer-window-p)
      (kill-buffer-if-not-modified (current-buffer))
    (abort-recursive-edit)))
(key-chord-define-global "xk"     'kill-this-buffer-if-not-modified)
; file actions
(key-chord-define-global "bf"     'ido-switch-buffer)
(key-chord-define-global "cf"     'ido-find-file)
(key-chord-define-global "vc"     'vc-next-action)
```

To complement these tricks, you should also install and use workrave or at least type-

break-mode.

3.14 X11 tricks

These are ways to improve the integration of Emacs in a graphical environment.

We have this cool editor. But it **is** from the 90s, and some of the more modern concepts of graphical programs have not yet been integrated into its core. Maybe because everyone just adds them to the custom setup :)

On the other hand, Emacs always provided split windows and many of the "new" window handling functions in dwm and similar - along with a level of integration with which normal graphical desktops still have to catch up. Open a file, edit it as text, quickly switch to org-mode to be able to edit an ascii table more efficiently, then switch to html mode to add some custom structure - and all that with a consistent set of key bindings.

But enough with the glorification, let's get to the integration of stuff where Emacs arguably still has weaknesses.

3.14.1 frame-to-front

Get the current Emacs frame to the front. You can for example call this via emacsclient and set it as a keyboard shortcut in your desktop (for me it is F12):

```
emacsclient -e "(show-frame)"
```

This sounds much easier than it proves to be in the end... but luckily you only have to solve it once, then you can google it anywhere...

```
(defun show-frame (&optional frame)
  "Show the current Emacs frame or the FRAME given as argument.

And make sure that it really shows up!"
  (raise-frame)
  ; yes, you have to call this twice. Don\T1\textquoteright t ask me why.\kern \fontdimen 3\font .\kern \f
  ; select-frame-set-input-focus calls x-focus-frame and does a bit of
  ; additional magic.
  (select-frame-set-input-focus (selected-frame))
  (select-frame-set-input-focus (selected-frame)))
```

3.14.2 urgency hint

Make Emacs announce itself in the tray.

```
;; let emacs blink when something interesting happens.
;; in KDE this marks the active Emacs icon in the tray.
(defun x-urgency-hint (frame arg &optional source)
  "Set the x-urgency hint for the frame to arg:

- If arg is nil, unset the urgency.
- If arg is any other value, set the urgency.

If you unset the urgency, you still have to visit the frame to make the urgency setting disappear (at leas
  (let* ((wm-hints (append (x-window-property
                            "WM_HINTS" frame "WM_HINTS"
                            source nil t) nil))
         (flags (car wm-hints)))
    ; (message flags)
    (setcar wm-hints
            (if arg
                (logior flags #x00000100)
              (logand flags #x1ffffeff)))
    (x-change-window-property "WM_HINTS" wm-hints frame "WM_HINTS" 32 t)))

(defun x-urgent (&optional arg)
  "Mark the current emacs frame as requiring urgent attention.

With a prefix argument which does not equal a boolean value of nil, remove the urgency flag (which might o
  (interactive "P")
  (let (frame (car (car (cdr (current-frame-configuration)))))
  (x-urgency-hint frame (not arg))))
```

3.14.3 fullscreen mode

Hit X11 to enter fullscreen mode. Any self-respecting program should have that... and now Emacs does, too.

```
; fullscreen, taken from http://www.emacswiki.org/emacs/FullScreen#toc26
; should work for X und OSX with emacs 23.x (TODO find minimum version).
; for windows it uses (w32-send-sys-command #xf030) (#xf030 == 61488)
(defvar babcore-fullscreen-p t "Check if fullscreen is on or off")
(setq babcore-stored-frame-width nil)
(setq babcore-stored-frame-height nil)

(defun babcore-non-fullscreen ()
  (interactive)
  (if (fboundp 'w32-send-sys-command)
      ;; WM_SYSCOMMAND restore #xf120
      (w32-send-sys-command 61728)
    (progn (set-frame-parameter nil 'width
                                (if babcore-stored-frame-width
                                    babcore-stored-frame-width 82))
           (set-frame-parameter nil 'height
                                (if babcore-stored-frame-height
                                    babcore-stored-frame-height 42))
           (set-frame-parameter nil 'fullscreen nil))))

(defun babcore-fullscreen ()
  (interactive)
  (setq babcore-stored-frame-width (frame-width))
  (setq babcore-stored-frame-height (frame-height))
  (if (fboundp 'w32-send-sys-command)
      ;; WM_SYSCOMMAND maximaze #xf030
      (w32-send-sys-command 61488)
    (set-frame-parameter nil 'fullscreen 'fullboth)))

(defun toggle-fullscreen ()
  (interactive)
  (setq babcore-fullscreen-p (not babcore-fullscreen-p))
  (if babcore-fullscreen-p
      (babcore-non-fullscreen)
    (babcore-fullscreen)))

(global-set-key [f11] 'toggle-fullscreen)
```

3.14.4 default key bindings

I always hate it when some usage pattern which is consistent almost everywhere fails with some program. Especially if that is easily avoidable.

This code fixes that for Emacs in KDE.

```
; Default KDE keybindings to make emacs nicer integrated into KDE.

; can treat C-m as its own mapping.
; (define-key input-decode-map "\C-m" [?\C-1])

(defun revert-buffer-preserve-modes ()
  (interactive)
  (revert-buffer t nil t))

; C-m shows/hides the menu bar - thanks to http://stackoverflow.com/questions/2298811/how-to-turn-off-alte
; f5 reloads
(defconst kde-default-keys-minor-mode-map
  (let ((map (make-sparse-keymap)))
    (set-keymap-parent map text-mode-map)
    (define-key map [f5] 'revert-buffer-preserve-modes)
    (define-key map [?\C-1] 'menu-bar-mode)
    (define-key map [?\C-+] 'text-scale-increase)
    (define-key map [?\C--] 'text-scale-decrease) ; shadows 'negative-argument which is also available via
    (define-key map [C-kp-add] 'text-scale-increase)
    (define-key map [C-kp-subtract] 'text-scale-decrease)
    map)
  "Keymap for `kde-default-keys-minor-mode'.")

;; Minor mode for keypad control
(define-minor-mode kde-default-keys-minor-mode
  "Adds some default KDE keybindings"
  :global t
  :init-value t
  :lighter ""
  :keymap 'kde-default-keys-minor-mode-map
  )
```

3.15 Insert unicode characters

Actually you do not need any configuration here. Just use

```
M-x ucs-insert
```

To insert any unicode character. If you want to see them while selecting, have a look at xub-mode from Ergo Emacs.

3.16 Highlight TODO and FIXME in comments

This is a default feature in most IDEs. Since Emacs allows you to build your own IDE, it does not offer it by default... but it should, since that does not disturb anything. So we add it.

fic-ext-mode highlight TODO and FIXME in comments for common programming languages.

```
;; Highlight TODO and FIXME in comments
(package-require 'fic-ext-mode)
(defun add-something-to-mode-hooks (mode-list something)
  "helper function to add a callback to multiple hooks"
  (dolist (mode mode-list)
    (add-hook (intern (concat (symbol-name mode) "-mode-hook")) something)))

(add-something-to-mode-hooks '(c++ tcl emacs-lisp python text markdown latex) 'fic-ext-mode)
```

3.17 Save macros as functions

Now for something which should really be provided by default: You just wrote a cool emacs macro, and you are sure that you will need that again a few times.

Well, then save it!

In standard emacs that needs multiple steps. And I hate that. Something as basic as saving a macro should only need one single step. It does now (and Emacs is great, because it allows me to do this!).

This bridges the gap between function definitions and keyboard macros, making keyboard macros something like first class citizens in your Emacs.

```
; save the current macro as reusable function.
(defun save-current-kbd-macro-to-dot-emacs (name)
  "Save the current macro as named function definition inside
your initialization file so you can reuse it anytime in the
future."
  (interactive "SSave Macro as: ")
  (name-last-kbd-macro name)
  (save-excursion
    (find-file-literally user-init-file)
    (goto-char (point-max))
    (insert "\n\n;; Saved macro\n")
    (insert-kbd-macro name)
    (insert "\n")))
```

3.18 Transparent GnuPG encryption

If you have a diary or similar, you should really use this. It only takes a few lines of code, but these few lines are the difference between encryption for those who know they need it and encryption for everyone.

```
; Activate transparent GnuPG encryption.
(require 'epa-file)
(epa-file-enable)
```

3.19 Colored shell commands

A shell without colors is really hard to read. Let's make that easier.

```
; colored shell commands via C-!
(add-hook 'shell-mode-hook 'ansi-color-for-comint-mode-on)
(defun babcore-shell-execute(cmd)
  "Execute a shell command in an interactive shell buffer."
   (interactive "sShell command: ")
   (shell (get-buffer-create "*shell-commands-buf*"))
   (process-send-string (get-buffer-process "*shell-commands-buf*") (concat cmd "\n")))
(global-set-key (kbd "C-!") 'babcore-shell-execute)
```

3.20 Save backups in ~/.local/share/emacs-saves

This is just an aestetic value: Use the directories from the freedesktop specification for save files.

Thanks to the folks at CERN for this.

```
(setq backup-by-copying t      ; don't clobber symlinks
      backup-directory-alist
      '(("." . "~/.local/share/emacs-saves"))    ; don't litter my fs tree
      delete-old-versions t
      kept-new-versions 6
      kept-old-versions 2
      version-control t)       ; use versioned backups
```

3.21 Basic persistency

If I restart the computer I want my editor to make it easy for me to continue where I left off.

It's bad enough that most likely my brain buffers were emptied. At least my editor should remember how to go on.

3.21.1 saveplace

If I reopen a file, I want to start at the line at which I was when I closed it.

```
; save the place in files
(require 'saveplace)
(setq-default save-place t)
```

3.21.2 recentf

Also I want to be able to see the most recently opened files. Almost every single program on my computer has a "recently opened files" list, and now emacs does, too.

```
; show recent files
(package-require 'recentf)
(recentf-mode 1)
(setq recentf-max-menu-items 1000)
```

3.21.3 savehist

And I want to be able to call my recent commands in the minibuffer. I normally don't type the full command name anyway, but rather C-r followed by a small part of the command. Losing that on restart really hurts, so I want to avoid that loss.

```
; save minibuffer history
(require 'savehist)
(savehist-mode t)
```

3.21.4 desktop globals

This is the chainsaw of persistency. I commented it out, because it can be overkill and actually disturb more than it helps, when it recovers stuff I did not need.

```
;; save registers and open files over restarts,
;; thanks to http://www.xsteve.at/prg/emacs/power-user-tips.html
;; save a list of open files in ~/.emacs.desktop
;; save the desktop file automatically if it already exists
;(setq desktop-save 'if-exists)
;(desktop-save-mode 1)

;; ;; save a bunch of variables to the desktop file
;; ;; for lists specify the len of the maximal saved data also
;; (setq desktop-globals-to-save
;;       (append '((extended-command-history . 300)
;;                 (file-name-history        . 100)
;;                 (grep-history             . 30)
;;                 (compile-history          . 30)
;;                 (minibuffer-history       . 5000)
;;                 (query-replace-history    . 60)
;;                 (read-expression-history  . 60)
;;                 (regexp-history           . 60)
;;                 (regexp-search-ring       . 20)
;;                 (search-ring              . 2000)
;;                 (shell-command-history    . 50)
;;                 tags-file-name
;;                 register-alist)))

;; ;; restore only 5 buffers at once and the rest lazily
;; (setq desktop-restore-eager 5)

; maybe nicer: http://github.com/doomvox/desktop-recover
```

3.22 use the system clipboard

Finally one more minor adaption: Treat the clipboard gracefully. This is a tightrope stunt and getting it wrong can feel awkward.

This is the only setting for which I'm not sure that I got it right, but it's what I use...

```
; Use the system clipboard
(setq x-select-enable-clipboard t)
```

3.23 Add license headers automatically

In case you mostly write free software, you might be as weary of hunting for the license header and copy pasting it into new files as I am. Free licenses, and especially copyleft licenses, are one of the core safeguards of free culture, because they give free software developers an edge over proprietarizing folks. But they are a pain to add to every file...

Well: No more. We now have legalese mode to take care of the inconvenient legal details for us, so we can focus on the code we write. Just call M-x legalese to add a GPL header, or C-u M-x legalese to choose another license.

```
(package-require 'legalese)
```

3.24 finish up

Make it possible to just (require 'babcore) and add the proper package footer.

```
(provide 'babcore)
;;; babcore.el ends here
```

4 Summary

With the babcore you have a core setup which exposes some of the essential features of Emacs and adds basic integration with the system which is missing in pristine Emacs.

Now go and see the M-x package-list-packages to see where you can still go - or just use Emacs and add what you need along the way. The package list is your friend, as is Emacswiki.

Happy Hacking!