Emacs – Beating the Learning Curve

From Zero to Lightspeed.

Timo Bingmann

May 26, 2016

Contents

1 Overview

This is a quick and dirty emacs tutorial. It does not advertise the most flashy features or goodies. Instead, it focuses on solid day-to-day efficiency features. It is probably best to print out this tutorial, grab a version of emacs, and start reading and trying out each command in this manual from top to bottom.

Do not start using emacs with the default config. Clone this repository as `.emacs.d` by running

`git clone https://github.com/bingmann/dot-emacs.git .emacs.d`

in your home directory. Then start emacs, which will download lots of additional packages. If that fails, check that you have at least emacs 24.5 by running `emacs --version`, and download a packaged tarball snapshot of the repository from

`https://github.com/bingmann/dot-emacs/releases/download/20160527/dot-emacs-snapshot-20160527.tar.bz2`

Emacs commands are **sequences of key strokes**, also called **key chords**. You must get every stroke right, otherwise a different command is called. Key modifiers are written as

- `S-` for Shift and/or implicitly notated by a capital or symbol,
- `C-` for Control, and
- `M-` for Alt, called Meta in the emacs world.

I prefer to disable the menubar in emacs, since it takes up screen space and distracts from the text. But for learning emacs it is a **good idea to enable it**: edit `.emacs.d/init.el`, find `(menu-bar-mode -1)`, and replace it with `(menu-bar-mode 1)` to enable the menubar after the next restart.

emacs has an **invisible interface**; even with the menu bar, you get no buttons to click on, and only the most necessary output about what it happening. Interaction and status information is shown in the **mode line** and the **mini-buffer** at the bottom.

2 Most Important Key Chords

2.1 When Things Go Wrong

These are maybe the **most important** key strokes: **cancel and undo** when things go wrong. If some command does not work as expected: press these and try again; emacs may have been in some state you did not know about.

Cancel Command	`C-g`
Cancel Minibuffer	`C-]` (try if `C-g` did not work)
Undo Typing	`C-/` or `C-S-_` (underscore)

Some buffers can be closed with `q`, usually those which are interactive buffers and not plain text editing ones.

2.2 Meta-X

Nearly any Emacs command can be called using `M-x` + long function description. All key chords are bindings to these long functions, and yes, you can change the key bindings some day.

See this xkcd comic for on what `M-x` is for: `https://xkcd.com/378/`. Then try `M-x butterfly`.

2.3 Move Around in Text

Move	`arrow` keys, `PageUp/Down`, etc
Typing	`[keys]`, `Delete`, `Backspace`
Move Faster (jump words)	`C-arrows`
Beginning and End	`C-Home` / `C-End`
Move Along Brace Tree	`C-M-S-arrows`
Center Line in Window	`C-l`
Move to Last Edit	`C-M-l` (repeatable)
Goto Line	`M-g`

2.4 Mark, Copy, and Paste Text

Copy and Paste are **totally different** from the usual `C-c`, `C-x`, and `C-v` sequences. **Get used to it.** `C-c` and `C-x` are used for more powerful things in emacs than mere copying/cutting.

Start Marking Region	`C-space`
Copy	`M-w`
Cut (called Kill)	`C-w`
Paste (called Yank)	`C-y`
Yank Previous Kill	`M-y` (repeatable)
Kill Line	`C-k` (repeatable)
Delete Word Forward/Backward	`C-Delete` / `C-Backspace`
Cut/Kill Word Forward/Backward	`M-Delete` / `M-Backspace`

2.5 Open File into Buffers

Open File	`C-x C-f`
Save Buffer	`C-x C-s`
Save All Buffers	`C-x s`
Save As ...	`C-x C-w`
Close (Kill) Buffer	`C-x k`
Revert Buffer	`M-x revert-buffer`

You don't need to enter the full path. Just open a directory and use the directory listing to navigate.

In my config there is a special mode activated which makes opening **existing** files faster (you see the available completions), but creating new ones from scratch more difficult. To create/write a **new file** press `C-z` while in `C-x C-f` mode. It stops the automatic fuzzy searches for existing files.

2.6 Switching Buffers (Open Files)

Cycle Buffer	`M-S-left` / `M-S-right`
Buffer List	`M-S-up` (originally: `C-x b`)

2.7 Organizing Buffers into Frames and Windows

There are frames and windows in emacs: frames are independent windows as seen by their border. Frames can internally be split into windows (horizontally or vertically). This often happens automatically, but you can do it manually to edit files at multiple places.

Full Frame	`M-~` (remap this if you use a German keyboard) or `C-x 1`
Split Frame	`M-2` or `C-x 5`
Kill Frame	`M-3`
Split Window	`C-x 2` horizontally, `C-x 3` vertically
Movement among Frames	`M-left` / `M-right`
Switch to other Frame	`C-x o`

2.8 Search and Replace

Incremental Search	`C-s <text>`
Search Backwards	`C-r <text>`
Search/Replace	`M-S-%` (follow prompt, keys: `y`, `n`, `^`, `!`)
Regex Search/Replace	`C-M-S-%`

2.9 Change Font Size

Larger Font	`C-mouse wheel up` or `C-x +` (repeat `+`)
Smaller Font	`C-mouse wheel down` or `C-x -` (repeat `-`)
Zero Font Size	`C-x 0`

3 Advanced Text Editing

Undo, redo, search, and replace can be constrained using the mark region. Undo only within a marked region is very powerful for coding!

Further simple commands are:

Comment or Uncomment Region	(mark region) `C-space`
Wrap (Fill) Paragraph	`M-q`
Fix Indentation of Region	`M-C-q` (depends on mode)
Upper/Lowercase Words	`M-u` / `M-l`

3.1 Rectangle Copy/Paste

Mark Rectangle Region	`C-space` (same as usual mark region, ignore blue area)
Insert Text in Each Line	`C-x r t`
Open Area (insert with spaces)	`C-x r o`
Cancel Area (overwrite with spaces)	`C-x r c`
Kill (Cut) Area	`C-x r k`
Yank (Paste) Area (inserts space)	`C-x r y`

3.2 Keyboard Macros

Record Macro	`C-x (`
End Recording	`C-x )`
Execute Macro	`C-x e` (then repeat `e` for more)
Repeated Execution	`ESC <num> C-x e`
Execute Macro and Append More Keys	`C-u C-x (`
Append More Keys (without execute)	`C-u C-u C-x (`

3.3 Run Shell Commands

Run Single Command	`M-S-!`
Pipe Marked Area to Command	`M-S-(pipe)`

3.4 Multiple Cursors

Make More Cursors	`C-S-click`
Exit Multi-Cursors	`C-g`
Mark All Matches with Cursors	`C-c !`

Multi cursors is a hack, but works reasonably well for simple things. Keyboard macros are more powerful.

4 Directory Listings (dired)

The directory listing, called dired, can be used to navigate the file system, perform copy/move operations, and more.

Show Dir/File	`Enter`
Go Up To Parent Directory	`Backspace`
Refresh	`g`
Mark File	`m`
Unmark File	`u`
Unmark all	`S-U`
Delete File or Marked Files	`D`
Copy File or Marked Files	`C`
Rename File or Marked Files	`R`
Chmod a File	`M`
Copy Marked Filelist to Clipboard	`M-w` (Copy)
Cut Marked Filelist to Clipboard	`C-w` (Cut)
Paste Filelist from Clipboard	`C-y` (Paste)
Make Directory	`+`
Search and Replace in Marked Files	`Q`
Mark File for Deletion	`d`
Execute Deletion for Marks	`x`

Note: with my config, PDFs open with evince.

4.1 Give Me a Terminal, NOW!

If dired is not good enough, and you need a terminal. The `F4` key opens an (external) terminal in the current directory. It works *everywhere*, also if you are editing a file.

Open Terminal, HERE.	`F4`

4.2 Tramp Mode

Emacs can also edit files on a remote system, controlling it via ssh. For this "open" a remote directory using `/[user@]ssh-host:path`. You can also just use `/ssh-host:` if you configure ssh host aliases.

Most (really: almost all) operations are fully transparent, e.g., dired works on the remote system just as well. Yes, copy and paste works across machines. Yes, open a terminal if you need it. Yes, remote compilation works! Yes, magit works (see below).

5 Customization

Emacs has a myriad of customizable configuration variables, and every package can add its own set. But customizing all this complexity is actually easy, because the configuration browser is very good.

Launch Customization Browser	`M-x customize`
Search Variables, Funtions, etc	`M-x apropos`
Change Face (i.e. Color and Font) under Cursor	`M-x customize-face`

Note that you must load the package you want to customize before calling customize, otherwise the options may not appear. (Kind of obvious, otherwise the options of all possible packages would need to be in the menu).

6 Source code editing

Buffers in emacs have so called "modes" active, which enable various features for the particular buffer. E.g., modes add color highlighting, keybindings, extra macros and functions, etc. Modes can be activated and deactivated, usually done using `M-x blah-mode`. E.g. `M-x orgtbl-mode` activates/deactives the org-table-mode in the buffer.

Emacs knows most programming languages, even hip new ones. If not out-of-the-box, then there is an addon mode for it. They are usually automatically activated by the file extension.

TAB and **automatic indentation** is highly sophisticated in emacs. Do not fight it, emacs will win. Instead either customized it (to the peril that others will use the defaults), or adapt to it.

7 C/C++ Projects

7.1 Identifier Expansion

emacs has the cedet package for parsing C/C++ files. It is not as good as one wants and it is slow. There are some newer alternatives that use clang, but I have not been happy with them. Everything breaks down once there are few dependent or complicated templates around.

Hence, I only use a dumb, very fast word expander: you write a prefix and it will look backwards in the text for words that starts with it. This often turns out to be more powerful, since it also expands words from strings, and from comments, from other open buffers, and then looks into the current directory for matching file names.

hippie-expand `M-/`

7.2 Snippets

I use yasnippets to expand some very frequently used code blocks. There are not that many, otherwise you should use the language and write a function for it. Expansion is trigged by snippet `keyword + TAB`.

`main`	`int main(int argc...)  ...`
`for`	a for loop with iterators/ranges
`fori`	a for loop with index
`inc`	`#include <...>`
`noncopy`	copy-construct + operator= set to "delete"
`noncopymove`	noncopy + move constructor / operator= set to "default"
`op«`	`friend std::ostream operator « (...)`
`copy`	`Copyright (C) <year> Timo Bingmann <tb...`
`cout`	`std::cout « ...  « std::endl;`
`hr`	`/***************/` (80 cols) horizontal rule
`lock`	`std::unique_lock<std::mutex> lock(mutex_);`
`debug`	`static const bool debug = true;`
`doc`	`/*!` doxygen block
`dog`	`//!  \name ...  \{` (doxygen group block)
`try`	`try { ...  } catch (...)  { ...  }`

7.3 Tag/Symbol and Semantic Jumps

Instead of complicated code parsing, I use `ggtags` and the GNU "`global`" source code indexer for function lookups and jumps. The problem is that it is made for plain C, and has only limited support for C++. This means it will lookup functions and symbols without the class context.

To lookup and jump to a symbol, position the cursor on it and press:

Jump to Symbol	`M-.`
Jump to Next Occurrence	`M-,`

When run the first time, ggtags asks where to find or create the `GTAGS` index. This is usually the root of your source tree. Press 'n' when it asks whether to use "ctags", it then uses GNU global.

emacs more advanced IDE-like features for C/C++ are provided by the `cedet` package. It is far from perfect, but does provide nice highlighting and useful basic features: switching between headers and source code Renaming variable within the function scope works sometimes (but not always).

Switch hpp/cpp Files	`F3`
Cedet Rename Variable	`C-c C-r`

Pressing `F3` on a `#include` line will try to open the header.

7.4 Ede (Cedet) Projects

Emacs is not very intelligent in detecting the root of a C/C++ project. It must be told where projects start. This also makes cedet work better.

ede projects work great with CMake. The additional include paths are also used during keyword expansion.

For emacs to know about a project, you must put the following into your `.emacs.d`:

```
(ede-cpp-root-project "thrill"
    :file "~/thrill/CMakeLists.txt"
    :include-path '("/extlib/gtest/")
    :compile-command "cd build && make -j4 && ctest -V && cd .. && doxygen"
    ))
```

The compile-command can be used to set a default (magic) compile command line to run within the project directory. Yes, this gives you click-able error messages. It also saves all buffers for you.

Auto-Magic Compile or Recompile	`F5`
Custom Compile Command	`M-x compile`

7.5 Grep and Ag

For searching a source tree I currently use `ag`, which is a `grep` replacement for source files (construct matching automaton, mmap files, etc goodies, read: FAST).

Search for Words (ag or grep)	`C-c C-s`

7.6 Bookmarks

Set/Unset Bookmark on Line	`M-F2`
Jump to Next Bookmarked Line	`F2`
Jump to Previous Bookmark	`S-F2`

7.7 ecb - Emacs Code Browser

Some people like it. It gives you a directory listing, and a class and method listing for navigation.

Start ECB `M-x ecb-activate`

7.8 gdb inside emacs.

Possible, but wicked complex. I have not mastered it.

8 Magit - Git Magic!

Magit is awesome. It is version controlling at a new level. It is also **dangerously magical.**

Launch magit `C-F12`

You see a listing of the current status, like `git status`. Navigate it like a directory listing. **Single keystrokes** do a lot of things in magit, beware! Keystrokes most often operate on the thing the **usual cursor** is on, beware where the cursor is!

Help!	`?`
Visit Thing under Cursor	`Enter`
Expand Thing under Cursor	`TAB`
Refresh (use often)	`g` (same as in dired)
Stage File or Diff Part	`s`
Unstage a Staged File or Diff	`u`
Kill Changes in File or Diff Part	`k`
Start Commit	`c c`
Amend Last Commit	`c a`
(in commit) Stop Commit	`C-c C-k` (almost usual "kill buffer")
(in commit) Save Commit	`C-c C-c`
Push Commits	`P` (read message) `u`
Pull Commits	`F u` or `F -r u` for `--rebase`
Show Branches	`y`
Checkout Branch	`b b`
Merge Branch	`m m`
Show Log	`l` (read 100 options) `l`
Stash	`z` (options) `z`
Add to .gitignore	`i`

Magit lets you do things like:

- kill only partial changes (use region marking too)
- commit only a part of the changes in the working directory

- stash everything not staged (check before committing parts)
- use emacs merge tools: do magit merge, press `e` on a conflict.
- apply patch chunks from other diffs `a`. also: revert *parts* of commits.
- kill local or remote branches: `k` in branch list.
- spell checking in commit messages.

9 Editing LaTeX

LaTeX editing with emacs works for me as follows. The AUCTeX package has a lot more commands than I use, but the following is all I need. It is activated automatically when opening a `.tex` file.

You usually want automatic spell checking when editing a LaTeX document, but emacs must be configured to activate `flyspell` and with the right **dictionary**. For this one puts the following line as the **first line** in the `.tex` file:

```
% -*- mode: latex; mode: flyspell; ispell-local-dictionary: "en_US"; coding: utf-8 -*-
```

This enables `latex-mode`, `flyspell-mode`, UTF-8 encoding, and sets the dictionary to `en_US`. See `/usr/share/hunspell/` for dictionaries available on your system. There is a snippet for this mode-line, so don't bother copying and pasting.

When editing LaTeX these are the most important key chords:

Compile with pdflatex	`C-c C-c`
View PDF File (once compiled)	`C-c C-v`

In my config the compile command runs a script `flymake-pdflatex`, which tries to be smart about when to run `bibtex` for bibliography and `makeindex` for symbol lists. It is not perfect, but adaptable.

Emacs and evince can synchronize! Pressing `C-c C-v` moves the PDF viewer to the current line in emacs. `C-click` in evince moves emacs to the clicked line (but only if compiled with `C-c C-c`). I practically **never** search for a section in emacs: just read the PDF, and click for editing!

Further key chords:

Show TOC for navigation	`M-S-down` (opposite to buffer menu)
Insert `\ref` from List	`M-C-r`
Insert `\cite` from Bibliography	`M-C-c`
Close Environment	`C-]`
Align `&` Columns in a Tabular	`M-x align-current`

Snippets while editing LaTeX files:

`begin`	`\begin{...} ... \end{...}`
`hr`	`% ---` (80 cols)
`itemize`	`\begin{itemize} \item ... \end{itemize}`
`enumerate`	`\begin{enumerate} \item ... \end{enumerate}`
`frame`	`% -- \begin{frame}{...} ... \end{frame}`
`modeline`	`-*- ... -*-` with dictionary selection

10 org-mode

Org mode is the ultimate text-based tool for organizing things. The great thing about org-mode files is, you can send them to people with no comment or additional program, and they can read it. Plain ASCII files will be readable as long as there are computers.

Learning org-mode is another presentation: see `http://orgmode.org/`

I use an org-mode file as my "welcome page" to emacs. It contains **everything**: TODOs, **short cuts to current and past projects**, links to remote configuration files, general notes, command line snippets, multi-year statistical information like electricity bills, etc.

This file is written in org-mode, which can export to HTML, Markdown, PDF, and probably a thousand other formats.

org-mode has automatic text-based tables, which can calculate and sort by columns.

11 evil-mode

There is evil in the world. If you want to turn from evil to good, emacs may help you change from your evil way **gradually**. For this there is `evil-mode`, which people say does evil things in emacs **better than evil itself**.

`http://www.emacswiki.org/emacs/Evil`