A beginner's guide to (GNU) Emacs 24

Matthew Chan, themattchan.com, last updated November 2014

Part 0: Getting started

Why Emacs?

Emacs is the One True Editor. Emacs can do anything you can possibly think of to any file you can possibly think of. Emacs is infinitely extensible with a variant of Lisp, the most beautiful programming language ever devised. Emacs is a user-friendly interface to every tool in the GNU/Linux command line toolchain. Emacs is the only piece of software you need to master.

Getting Emacs

Linux: 1) Install `emacs` from your package manager. Note that when you start it from the command line, it will launch the GUI version. To use it in the terminal, run `emacs -nw`

Mac OS X: 1) Get MacPorts or Homebrew. 2) Emacs is hosted as two separate packages: emacs-carbon for the GUI version (emacs.app), and emacs for the CLI version. Install both.[1]

Windows: 1) Download a Linux distro. 2) Install Linux on your computer. 3) See above (Linux emacs installation)

Starting Emacs

Emacs startup gets horribly slow when you install more packages and configurations. Setup the following aliases in your bashrc/zshrc/:

```
em=`emacs -nw'
qem=`emacs -quick -nw'
```

To start terminal emacs, run `em` for the emacs with your configs, and `qem` without configs (for quick edits).

To close emacs, do `C-x C-c` (see key notation below)

Setting your shell variables

Once you learn emacs, you'll want to use it everywhere. Add this to your shell config to prevent command line utilities from opening things in anything but emacs, and to enable emacs commands globally.

```
#### emacs ####
export EDITOR="emacs"
export ALTERNATE_EDITOR=""
export GIT_EDITOR="emacs"
export VISUAL="emacs"

# set emacs keybindings
bindkey -e
```

[1] Mac OS X comes with command line emacs, but its usually an old version, and it doesn't have GUI support.

Starter config file

To smooth out the rough edges of vanilla emacs, here's a barebones init.el file for you to use: https://gist.github.com/themattchan/112f4a71294aec281fa9.

Place it in `~/.emacs.d/init.el` and start emacs.

I'm against using large opaque packages like the Emacs Starter Kit or Emacs Prelude, as fantastic as they might be. They change so much of vanilla emacs it becomes hard to separate the base system from the packages. Learn the editor first, then go ahead and move onto a package after you understand the organization of emacs lisp. The init file is different - there aren't any functions or complicated things, and you can generally understand what it's doing even if you've never seen an S-expr based language before. Plus, you get to learn some lisp by reading it.

p # Part 1: Basics

These are the commands and functions that you absolutely must know to navigate and use Emacs productively. You can probably learn them all by using emacs for a day.

WARNING to learn the commands properly, it's important that you force yourself to use them. Edit with text emacs exclusively for a week or two and they will become second nature.

Key notation

The most puzzling part about emacs documentation is the notation for key commands. Learn this, and you're halfway there (you can just google the rest :))

- `C-` Control. It is recommended that you remap Control to Caps Lock — a) Caps Lock is useless, and b) the placement of the Control key causes "emacs pinky"[2].
- `M-` Meta (Alt)[3]
- `s-` Super (unbound modifier by default)
- `H-` Hyper (unbound modifier by default)
- `S-` Shift
- `RET` Return/enter
- `<character>` type the character

e.g. `C-x r t` means press `x` while holding down Control, then release both and type `r` followed by `t`.

Get me out of here!

You quit emacs with `C-x C-c`.

The Emacs interface — buffers, windows, and frames

Emacs is organized by *buffers*, similar to individual open files. Everything is opened in a buffer, and changes are only recorded in the buffer until you write to disk. There are two types of buffers: buffers, in which text is edited and program outputs are displayed, and the *minibuffer*, the space along the bottom where you enter extended commands in (and where command status is shown).

There is only one minibuffer (which behaves like a split, see below).

Every buffer has a *modeline*, displaying meta information about that buffer — the filename, *editing mode*, cursor location, line ending, etc.

You can only be in one buffer at a time.

The main "window" you are in is called a frame. A frame can be split into different *windows* or *split-panes*, each displaying a different buffer. To split a frame and manage splits, do the following prefixed by `C-x`:

- `0` close current split

[2] The old IBM keyboards with Control where Caps is had it right, then someone thought it would be a bright idea to add Caps Lock.

[3] On some terminal emulators you have to set this properly — iTerm (Mac) set Option as +Esc, Terminal (Mac) set Option as Meta.

- `1` close all other splits
- `2` new horizontal split
- `3` new vertical split
- `o` "other split" (cycle through splits)

e.g. `C-x 3` makes a new split

You can have multiple frames open; in GUI emacs each frame is a window (in the OS sense), in text emacs each frame is like a tab.

Frame commands are all prefixed with `C-x 5`, followed by the same arguments as the window commands:

- `0` close current frame
- `1` close all other frames
- `2` new frame
- `o` "other frame" (cycle through frames)

e.g. `C-x 5 2` to make a new frame.

The buffer list is independent of how buffers are displayed: you can have 10 frames, each with 5 split panes, all displaying and editing the same buffer. Splits and frames are only ways to access and display buffers.

Manipulating frames and splits do not change the buffers that are open, they have to be closed separately. You manipulate buffers like so:

- `C-x k` "kill" buffer (close it)
- `C-x b` show buffer list in minibuffer. You can then go to a buffer by typing its name
- `C-x C-b` show a list of buffers in a new buffer
- `C-x <L/R arrow>` cycle through buffers
- `C-x C-s` save current buffer to disk
- `C-x s` save all open buffers to disk (prompt for each one)
- `C-x w` write the current buffer as a new file to disk

Buffers with name *name* are special.

Buffers and files

In order to edit files, you'll have to open files. If you run emacs from the command line, you can do `emacs <file1 file2 ... filen>` to open one or more files, each as a buffer. Inside Emacs, you *find* (open) files with `C-x C-f` then typing in the path to the file, which opens it in a buffer. Default filesearch has tab completion, and if you hit enter before you open a file then you open up the directory in the a buffer (and can select files from there).

- `C-x C-w` writes to a new file (save as)
- `C-x C-s` saves the current buffer

- `C-x s` saves all open buffers (will prompt yes/no on each buffer)

To make a new buffer, do `C-x b` and give it a name. Searching for a buffer with a name that isn't matched with an open buffer will create it.

Emacs automatically detects the file type and opens in the corresponding *mode* (see part 2), with editing commands specific to that filetype.

Cursor navigation

Learning how to navigate a document is arguably the most important part of learning emacs.

The Emacs navigation key bindings are structured logically. Modifier keys correspond to the size of the thing being manipulated: `C-` deals with small units: characters, lines. `M-` deals with large units: words, paragraphs, the whole file.

- **Words and characters**
  - `C-f` and `C-b` to move forward and back by one character (remember C-: character, f: forward, b: backward)
  - `M-f` and `M-b` to move forward and back by one word
- **Lines:** *remember that lines are always navigated by `C-`*
  - `C-a` and `C-e` to jump to beginning and end of line (remember a: beginning, e: end)
  - `C-n` and `C-p` to move forward and back by one line (remember n: next-line, p: previous-line)
- **Blocks of text** *remember that blocks are mostly navigated by `M-`*
  - `M-a` and `M-e` to jump to beginning and end of paragraph
  - `M-[` and `M-]` to move up and down paragraphs/curly-braced blocks of code
  - `M-v` and `C-v` to page up and page down
  - `M-<` and `M->` to jump to beginning and end of file
- `C-s` and `C-r` to search for a string forwards and backwards of the cursor
- `M-g g` to jump to a line directly

The great thing is, that these commands all work in your terminal (because you set the editor to be emacs, didn't you?). C-r to search for a previous command you entered, C-a to navigate the prompt, C-p to cycle through previous commands, etc.

Also, most of these commands are natively supported across the whole system in Mac OS X.

Emacs key bindings are the only key bindings that are truly ubiquitous. Learn them well.

Undo and redo

- `C-/` undo

There's no redo by default, but `redo.el` from the package archives will enable it. Install that, then add this to your init.el (not in the starter file).

```
(when (require 'redo nil 'noerror) (bind-key "C-?" 'redo))
```

This binds undo to `C-?`, or `C-S-/` (undo command while pressing shift)

Killing and yanking — aka cut and paste

Emacs doesn't have a clipboard in the sense that you're used to — instead, it has a superior alternative known as the kill ring. The kill ring is a ring buffer[4] where all your copied or "killed" (deleted) text goes. Text is never deleted, it's always cut into the ring buffer. To paste, you "yank" the text from the buffer and plop it at the desired point (where the cursor is). If you want something you killed earlier, you *rotate* the ring buffer and cycle through until you find it.

Ok, you know how it works. Here are the commands:

- `C-w` kills (cuts) a region
- `M-w` copies a region
- `C-y` yanks the top item from the kill ring (paste most recent)
- `M-y` rotates the kill ring (cycles what you've pasted) after a yank operation (always use `C-y` before `M-y`)

That's it! You now know all the commands to edit text comfortably in emacs. Next up, even more powerful text editing commands.

[4] http://en.wikipedia.org/wiki/Circular_buffer

Part 2: Power editing

Slightly more advanced features of Emacs: you'll want to start customising the editor after getting comfortable with the basics ## Tweaking split-panes

Resize pane `C-x ^` ++height `C-x {` ++width to left `C-x }` ++ width to right

`C-u <+- number>` repeat command, e.g. `C-u -6 C-x ^` shrink height by 6 units

`M-x command`

Emacs modes — major modes and minor modes

Rectangle editing

Dealing with prose

Tabs, spaces, and indentation

Set tab width on the fly `M-x set-variable <RET> tab-width <number>`

Indent region `C-M-\`

Tabs -> spaces and v.v. `M-x replace-string <RET> <TAB> <RET> <n no.  of spaces> <RET>`

Selecting

Entire buffer: `C-x h`

Text editing: Hard word wrap the buffer (fill)

C-x f set-fill-column (how wide should it get) To set to 80: C-u 80 C-x f M-x fill-region Wraps the entire buffer M-x fill-region-as-paragraph Wraps the entire buffer treating it as one paragraph M-q Wraps the current paragraph M-s Centers a line in however many columns

Directory editing:

`g` refresh current buffer (and dired listing)

Refresh init.el after modifying

`M-x load-file <RET> ./init.el <RET>`

prepend: C-x r t M-x string-insert-rectangle

Part 3: Customize!

Emacs customisations, and a bit of programming in Emacs Lisp. Any further and you'll have to learn Lisp, functional programming...

The .emacs.d directory and the init.el file

M-x customize

MELPA and packages

Fixing annoyances with included packages

Instant startup == never quit. The emacs server.

Appendix: Recommended packages