Emacs Cheatsheat

Joshua Branson

June 3, 2016

1 Why use Emacs?

For starters, Emacs is freedom respecting software. This is often shortened to "free software", or "libre software". This means that one has the right to:

- 0 run the program for any reason.
- 1 the right to help yourself. This is the right to study the source code and change it.
- 2 The right to help your neighbor. This is the right to distribute unmodified copies of Emacs to your neighbor. You could also even sell this program to your neighbor!
- 3 The right to help your community. This is the right to distribute your modified version of Emacs. You could also sell your modified version!

  So Emacs grants you, the user, many freedoms that other programs lack. Please notice that this is computing freedom. Free software is liberty software, not necessarily gratis software. However, Emacs is Free software, and it can usually be obtained as gratis software.

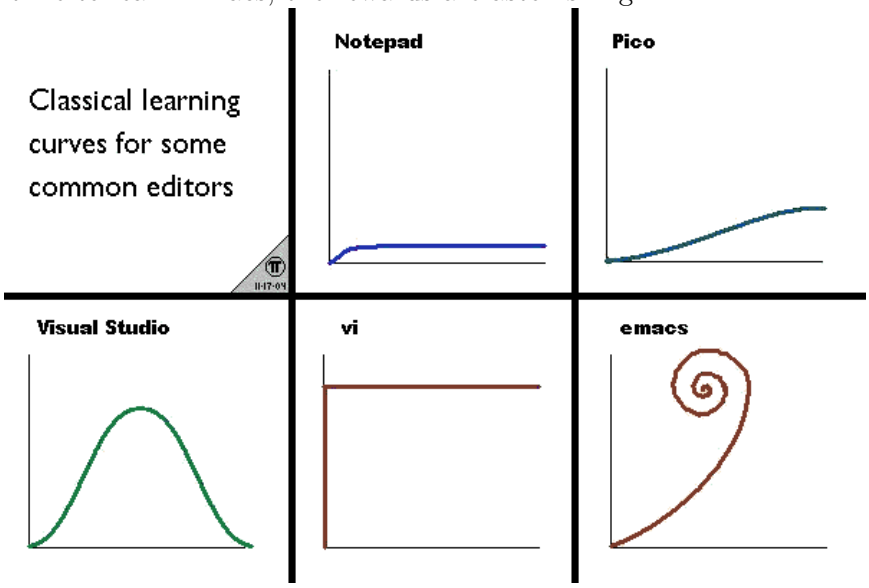
There are also some practical reasons to use Emacs. Plenty of programmers these days use drastically simple programming editors like notepad, notepad++, nano. These editors are "What you see is what you get" type editors, and they are frustratingly lacking in good features, and one cannot easily configure these editors. Emacs gives you the ability to install hundreds, if not thousands of add-ons, and configure **everything** that your text editor does.

So what can Emacs do? You can use emacs to use commands to manipulate and transpose regions of text, write in nearly every programming

language imaginable, read your email, browse the internet, play games, browse local files, look at a photo gallery, stream internet music, watch online movies (coming soon), create text expandable abbreviations, clock your working hours, invoice clients, schedule your weekly agenda, syntax checking, browse documentation, commit your development changes to version control, emulate vim, all of which can be configured and tweaked emacs in numerous ways. If that doesn't satisfy you, then you can programmatically change emacs' behavior to better suit your development workflow.

2 Emacs Introduction

Emacs is a incredibly old software. It's been around since the late 80s and early 90s, and its amazing flexibility is probably what has kept it alive this long. Learning Emacs can be a huge challenge of its own, but if you take the time to learn Emacs, the rewards are astonishing.



(Original source appears to be Steve Rowe's blog. "A friend of mine put this together" Source).

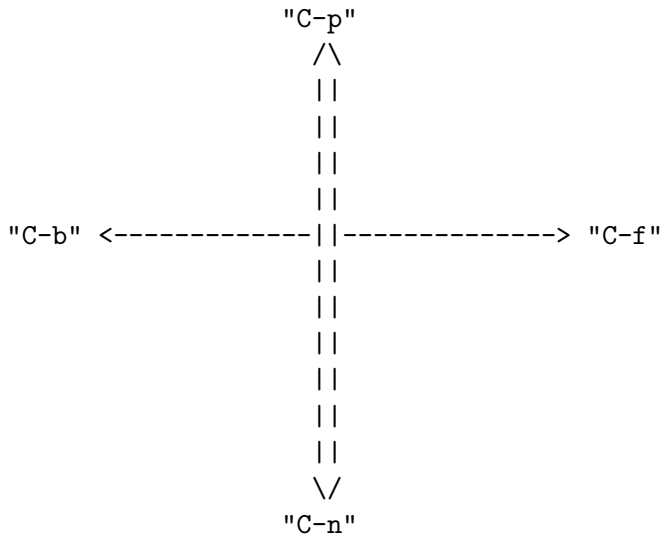
Since Emacs is so old, it created some terminology that modern text editors do not use. For example Emacs has "Frames" and "Windows", "yanking" and "killing" and no other text editor uses those terms, but the best way to learn Emacs, is to dive into the terminology. So put on your big programming pants, and follow me down the Emacs golden brick road of confusion!

So go ahead and launch Emacs. When you do, you should see a blinking rectangle. When you start to type this rectangle moves and letters appear. That blinking rectangle is called *point*. When *point* moves around the text, we say that point moves in the *buffer*. The *buffer* then is a region in Emacs that contains text. Since you've just started Emacs, the whole Emacs graphical display is one *frame*. You can also launch Emacs from the terminal and that is still just one frame of Emacs. If you open a second instance of Emacs, you now have two frames. Emacs also has windows. An Emacs *window* is the portion of Emacs that contains the buffer, or the text. It's also possible to for a frame to have multiple windows. You can read more about creating and deleting windows and frames in Frame and Window Commands.

3 Built in Emacs Features

3.1 Movement

One can always use the arrow keys to move point, but Emacs provides some nifty ways of moving point as well. Pressing and holding the control key whilst hitting "f" will move point forward one character. In Emacs lingo we call that a keychord, and it is denoted like this "C-f". Similarly, holding and pressing the control key, then hitting "b" will move point backward one character. That keychord is denoted as "C-b". "C-p" will move point back to the up previous line, and "C-n" will move point down to the next line. So the basic movement keys look like this:



3.2 Yanking and Killing

Most text editors use the words "cutting", "pasting", and "coping" for the commands `C-x`, `C-v`, and `C-c`. Emacs used different words and different commands. What is normally called cutting is *killing* in Emacs. In this way, one kills to the kill region, a section where emacs stores killed text. Pasting is called *yanking*. One yanks from the kill region to past into the buffer. Copying in Emacs is also called copying.

To kill or copy a region of text, you must first *mark* that region or highlight it. To begin marking a region of text one presses `C-<space>` or `C-SPC`. `<space>` and `SPC` means hitting the spacebar key. Then you can move around point until you have marked the entire region. To kill the region type `C-w`. To copy the region type `M-w`. M stands for the meta key, which on today's keyboard usually means the alt key. So `M-w` means to press and hold the alt key, then hit the w key. Alternatively hitting the ESC key is the same as pressing and holding the alt key. So `M-w` is the same thing as `ESC-w`.

To yank the killed or copied text back into the buffer press the `C-y` keychord.

3.3 Frame and Window Commands

Emacs has windows and frames, and they are probably not what you think they are. A Frame is probably what you would think a window is. So when you follow a link in your browser in a new window, that is a window in browser terminology. In Emacs you can do the same thing, but it is called opening a new frame. Go ahead and open a new frame now by typing `C-x 5 2`. To get back to one Emacs frame, you can either close the additional frame normally, or you can press `C-x 5 0`.

An Emacs window then is the buffer portion of the frame that displays text. One frame can have multiple windows. If you type `C-x 1`, then Emacs is only displaying one window. Then, if you type `C-x 2` or `C-x 3`, then Emacs is displaying 2 windows. So a window then is a section of the Emacs Frame that contains text from a buffer.

In emacs the entire emacs program takes up a **frame**. But emacs allows you to view two different files in the same frame, by splitting the frame in half, or in two **windows**.

- `C-x o` Delete the selected window
- `C-x 1` Delete all the windows except the one that currently has point

- `C-x ^` make the selected window taller
- `C-x` make the selected window narrower
- `C-x` make the selected window wider

3.4 Bookmarks

Emacs bookmarks are nifty ways of saving your place in a file. If you wish to return to some obscure local or remote file, you can easily save point's current position, and return to it later.

- `C-x r m` sets a bookmark for you at point and it prompts you to name it
- `C-x r b` jumps you to a bookmark

3.5 Learning About Emacs

Emacs comes with some amazing documentation. `M-x info` will show you a list of the available documentation that comes with Emacs. This documentation is contained in info files, which can be easily navigated via Emacs.

Emacs also has a pretty powerful help command: `C-h`. `C-h v` will let you learn some documentation about a particular Emacs variable, while `C-h f` will show you documentation for an Emacs elisp function. Every time that you hit a keychord, an equivalent Emacs function is called. In fact `C-h k KEYCHORD`, will tell you the documentation for the a function based on its keychord. For example, `C-h k C-n` will display documentation for the function "next-line".

3.6 Configuring Emacs with Lisp

Emacs' flexibility stems from the ability to program-matically change it via Emacs lisp. If you dislike a particular aspect of Emacs, then you can easily change it by putting in some Emacs lisp code into your init file. This is the file that Emacs loads every time on startup. Any Emacs lisp functionality that you code in your init file, will be available every time you start emacs. You can find your init file by checking out the variable `user-init-file`. To do this press the keychord `C-h v RET user-init-file RET`.

3.7 Dired

Dired is the emacs file manager. It opens a buffer displaying all your files in the specified directory. With it you can perform numerous commands on marked files, like deleting, copying, moving, or even creating your own command.

3.7.1 Commands

- `n` next line
- `p` previous line
- `m` mark the current file under point
- ~% m REGEXP ¡RET¿ mark files based on a regular expression

3.7.2 Image Dired

You can also view images inside dired! Mark the images you wish to view, then press C-t d (image-dired-display-thumbs). Alternatively, you could also just run the command M-x image-dired.

3.8 Macros

A Macro is a remembered sequence of Emacs keychords that can be repeated. This is useful to easily repeat complicated commands. For example, I can write out the numbers 1 through a 100, if I hit 30 keys. I could write the numbers 1 through 1,000 by typing 31 keys!

- `C-x (` begin recording a keyboard macro
- `C-x )` end recording a keyboard macro
- `C-x e` performs the last created keyboard macro

3.9 Narrowing

Narrowing commands make Emacs only display portions of the buffer, whilst hidding all other regions. While Emacs is narrowed, all entered commands only affected the displayed regions. This means any hidden area cannot be modified while Emacs is narrowed. This is useful if you only want a macro to execute within a specific function. Here are the relevant narrowing commands:

`C-x n <letter>`

- `d` narrow to defun
- `r` widen to region
- `s` narrow to a org subtree
- `w` widden to the whole buffer

A much better way to use the narrowing commands is just to make emacs guess what you want whenever you press "C-x n", and that's what the following snippet does. I recommend that you put it in your .emacs file. The code works by figuring out which narrowing command you want to use. If point is currently in a definition, then the buffer will be narrowed to that definition. If point is in an org-subtree, then the buffer will be narrowed to that subtree.

You can see the blog post where I found this code snippet here.

```
;; Also set up narrow dwin
(defun narrow-or-widen-dwim (p)
  "Widen if buffer is narrowed, narrow-dwim otherwise.
Dwim means: region, org-src-block, org-subtree, or defun,
whichever applies first. Narrowing to org-src-block actually
calls ‘org-edit-src-code’.

With prefix P, don’t widen, just narrow even if buffer is
already narrowed."
  (interactive "P")
  (declare (interactive-only))
  (cond ((and (buffer-narrowed-p) (not p)) (widen))
        ((region-active-p)
         (narrow-to-region (region-beginning) (region-end)))
        ((derived-mode-p ’org-mode)
         ;; ‘org-edit-src-code’ is not a real narrowing
         ;; command. Remove this first conditional if you
         ;; don’t want it.
         (cond ((ignore-errors (org-edit-src-code))
                (delete-other-windows))
               ((ignore-errors (org-narrow-to-block) t))
               (t (org-narrow-to-subtree))))
        ((derived-mode-p ’latex-mode)
```

```
         (LaTeX-narrow-to-environment))
        (t (narrow-to-defun))))

;; This line actually replaces Emacs' entire narrowing
;; keymap, that's how much I like this command. Only copy it
;; if that's what you want.
(define-key ctl-x-map "n" #'narrow-or-widen-dwim)
```

3.10 org-mode

Emacs org-mode really deserves its own cheatsheat, so I won't go into much detail here, but I'll start you off with the basics. Org-mode is Emacs' organizational mode, and it's pure gold! With Org-mode I organize my daily agenda, todo lists. Parts of my emacs init files are written in it. I use it to keep track of my working hours, with which I then invoice clients. I use its markup to write MIME emails. I wrote all of my documentation in it, and I keep track of my finances with it! It truly is a remarkable emacs mode!

3.10.1 Org-mode's hierarchical structure

Org-mode lets you easily insert headings and sub headings with "C-RET". If you press it many times, you'll have something like this:

```
*
*
*
*
```

A line with just one "*" is a top level heading. If it has a two "**" below it, then it now has a sub-heading. Just like the following:

```
* I am a top level heading
** I am a sub-heading.
```

You can use the tab key to show/hide any sub level headings.

Org-mode is also great for todo lists. Pressing C-c C-t lets you mark an item as TODO or DONE.

Todo lists One can easily create simple todo lists with org-mode. In any org file press "C-RET". A "*" will have inserted itself into your buffer. Pressing "C-c C-t" will add the words "TODO". It'll look like:

```
* TODO
```

Pressing "C-c C-t" again, will change the status to DONE. You will end up with something looking like:

```
* DONE
```

3.10.2 org-babel

Org babel is a the best approach towards literate programming ever attempted, and it works! Almost all programming languages treat code as the first order citizen and hides comments behind a simple syntax. It could probably be moved into its own cheatsheet, but I will describe the basics for you here. For example here is some javascript:

```
console.log("hello world")
```

Let's write a trivial js function the literate way

```
#+BEGIN_SRC js :exports code
  var i = 5;
  if (i < 6) {
    i++;
  }
  console.log (i);
#+END_SRC
```

Literate programming might not be the best method of coding large projects, but it incredibly useful for writing your Emacs config files. You can see an example of my org more custom-izations here.

Specific header arguments `http://orgmode.org/manual/Specific-header-arguments.html org#Specific` header arguments

- :results syntax: :results [raw — silent — value — output] value is function mode. It means that org-mode will use the last executed command as the value of the output. ie:

  ```
  import time
  print("Hello, today's date is %s" % time.ctime())
  print('Two plus two is')
  return 2 + 2
  ```

```
echo "hello world"
echo "big cat"
ls -lh | grep emacs.org
```

- :exports [code — results — node — both]
- :dir Specify a default directory that the code is to be run in :dir ¡dir¿

```
ls
```

:dir can also specify remote directories to run code!

```
ls
```

4 Helpful Emacs modes

4.1 Bug Hunter

If you fairly regularly change your init file, then you will at some point open your init file with a broken emacs. Bug hunter helps you quickly narrow down the cause of the error. You want to make sure that bug-hunter is loaded early in your init, but then anytime that you find another issue, just run. `M-x bug-hunter-init-file e RET`

`https://github.com/Malabarba/elisp-bug-hunter`

4.2 Helm Mode

Helm mode is an interactive completetion framework that is much better than ido mode. C-c C-f helm-find-files

In this mode typing "~/ manage js$" will display a list of files in my home directory that contain the word "manage" and end with "js"

Typing C-l will display the files is the parent directory.

Typing C-z when point is on a directory, will show the files in that directory.

Helm has nth commands. Instead of typing tab to get to the action menu just press C-e for the 2nd action and C-j for the 3rd action. You can also bind a key to an action menu (define-key helm-map (kbd "¡C-tab¿") 'helm-select-4th-action)

4.2.1 commands

You can learn how to write your own helm commands here: write your own helm extentions `C-c h m` open helm-man-woman `C-c h h g` open helm info gnus `C-c h h r` open the helm-emacs-info `C-c h b` is helm-resume which opens up the last helm instance. `M-<space>` mark candidate `C-h m` inside a helm window will show you all of helm's keybindings

4.3 El-doc

El-doc shows you a function's documentation in the mini-bar as you write it. By default it works for emacs lisp extremely well. functions. You can add this to your init file if you'd like to try it for emacs lisp.

```
(add-hook 'emacs-lisp-mode-hook 'eldoc-mode)
```

4.4 Yasnippet

Yasnippet is Emacs non-official but pretty much most commonly used snippet system. You can define an abbreviation, then when expanded does what the snippet file says to do. `http://ergoemacs.org/emacs/yasnippet_templates_howto.html`

4.4.1 Important Characters

- `$&` indents the line according to the major mode
- `‘(some-lisp-code)‘` embods lisp code
- `$0` where point will be when the snippet ends
- `$n` where n is a number ie: `$1`, `$2`, etc. If you have multiple $3, then typing some text in one $3 will also be put in the other $3.
- =${n:¡placeholder text¿}

4.5 Undo Tree

Undo tree is a mode that lets you visually step through the changes that you have done to the buffer. You can step backwards and forwards through time. In normal creation of a documentent, a user typically creates several changes that the emacs undo command is not sufficient to solve. A document's historical content is not always linear. Instead, during normal editing, a

user can write content that the normal emacs undo command forgets about. This is where undo tree is helpful. Invoking `M-x undo-tree` shows the user a visual representation of the buffer in time. Using the arrow keys (or conventional emacs replacements), one can step through a documentent's progression.

4.6 Ediff

Ediff is emacs's cool way of comparing two files and merging them into one. `M-x ediff` starts the process. Emacs will prompt you to ediff two files, and then you can begin merging the files together.

4.6.1 Commands

- `a` copies buffer a diff to buffer b
- `b` copies buffer b diff to buffer a
- `A` toggles readonly mode of buffer a
- `B` toggles readonly mode of buffer b
- `wa` save buffer a
- `wb` save buffer b
- `!` update the differance regions. If you press `a` and `b` multiple times, you should probably do a `!`
- `*` highlights the words in the diff region that differ
- `ra` restore the diff region in buffer a
- `rb` restore the diff region in buffer b
- `z` suspend the ediff session
- `s` make the merge buffer as small as possible

When you specify files, you can edit the files as root using tramp's syntax like this.

/su::/path/to/file

4.7 Tramp

Tramp is an emacs extension that lets you edit remote files. To use tramp, just begin by opening a file via `C-x C-f` (find-file) then typing one of the following special syntaxes:

/HOST:FILENAME /USER@HOST:FILENAME /USER@HOST#PORT:FILENAME /METHOD:USER@HOST:FILENAME /METHOD:USER@HOST#PORT:FILENAME

5 Regexp

Regular expressions are nifty ways of searching/replacing regions of text.

Consider this example

```
if (isadmin() || ismanager ()) {
    //some code here
}
```

Suppose that you want to add a space between both "is" in the functions. The following would do this:

`M-x dired-do-query-replace-regexp is(admin|manager) RET is 1 RET`

But let's get a basic understanding of regexps.

6 Useful Elisp Libraries

- ctable `https://github.com/kiwanami/emacs-ctable`
- s `https://github.com/magnars/s.el`
- f `https://github.com/rejeep/f.el`
- dash `https://github.com/magnars/dash.el`