Useful Emacs Keybindings

Jul 22, 2018 • emacs productivity • 0 comments

[Table of Contents]

I don't mean to brag, but I'm one of the best Emacs users on the planet. What's my secret? Well, apart from having a genius-level IQ, there is no secret, really. However, it helps that I customized Emacs to make me as productive as possible, and part of doing that has been setting up custom keybindings. In fact, many of my keybindings are so brilliant that you may at first sight not understand them. I do implore you to persevere. This article may just change your life. At the least, you'll have a few more keybindings in your arsenal.

save-all

Whenever you try to save all your buffers at once you have to confirm that you want to save for *every* buffer. That's why I wrote the following simple function. I find this so useful that I run it every time I tab out of Emacs using `focus-out-hook`.

elisp

```elisp
;; Automatically save on loss of focus.
(defun save-all ()
  "Save all file-visiting buffers without prompting."
  (interactive)
  (save-some-buffers t) ;; Do not prompt for confirmation.
  )

(global-set-key (kbd "C-x s") 'save-all)

;; Automatically save all file-visiting buffers when Emacs loses focus.
(add-hook 'focus-out-hook 'save-all)
```

Never lose your work again!

Easy Window Management

I shortened the window management keys, like `C-x 0`, since I use them so often.

```elisp
(global-set-key (kbd "C-0") 'delete-window-balance)
(global-set-key (kbd "C-1") 'delete-other-windows)
(global-set-key (kbd "C-2") 'split-window-below-focus)
(global-set-key (kbd "C-3") 'split-window-right-focus)
```

You lose the ability to specify prefix arguments by holding Control and pressing numbers, but I don't use prefix args that often, and there is always `C-u x`.

Wait, `split-window-right-focus`? Yeah, I forgot to tell you: I made my own functions for splitting windows. Am I awesome or what? Basically they just move focus to the new window, which is what I want almost always. They also balance the windows, because why not?

```elisp
(defun delete-window-balance ()
  "Delete window and rebalance the remaining ones."
  (interactive)
  (delete-window)
  (balance-windows))

(defun split-window-below-focus ()
  "Split window horizontally and move focus to other window."
  (interactive)
  (split-window-below)
  (balance-windows)
  (other-window 1))

(defun split-window-right-focus ()
  "Split window vertically and move focus to other window."
  (interactive)
  (split-window-right)
  (balance-windows)
  (other-window 1))
```

Being Super

You can get a lot more mileage out of Emacs by using the Super key.

You can get a lot more mileage out of Emacs by using the Super key.

“The what now?”

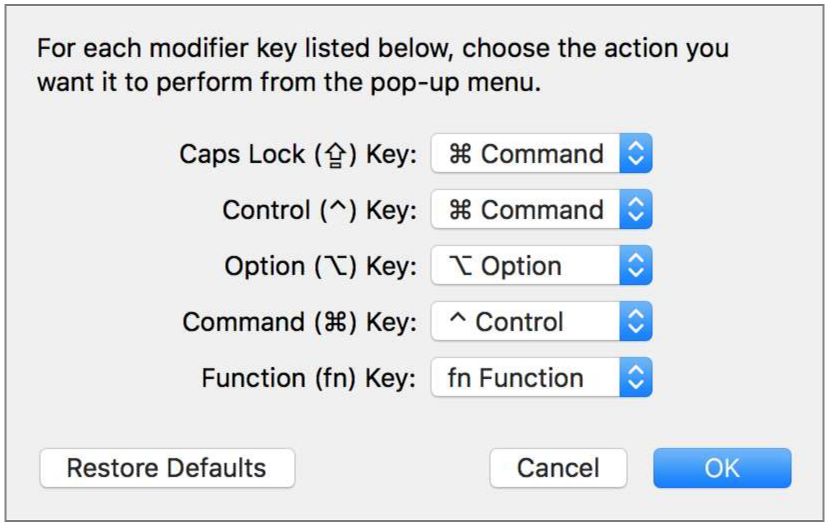
The Super key. Super is the Windows key on Windows and the Command key on OSX. Since it’s there, why not use it?

Caps Lock

I recommend binding your machine’s Caps Lock key to Super. Caps Lock is not the best location for a common modifier key[1] but it’s perfect for handy yet not too frequently-used commands. In other words, I wouldn’t use Super keybindings for something like editing text, but they’re great for navigating buffers and workspaces.

Note for OSX

On OSX, these are the key mappings I recommend. Basically, I just set the Caps Lock key to Command and the Command keys to Control.



For each modifier key listed below, choose the action you want it to perform from the pop-up menu.

Caps Lock (⇪) Key: ⌘ Command
Control (^) Key: ⌘ Command
Option (⌥) Key: ⌥ Option
Command (⌘) Key: ^ Control
Function (fn) Key: fn Function

You can get to this menu by going to **System Preferences > Keyboard > Modifier Keys**.

Default bindings

Emacs has some default bindings using Super which are consistent with common actions. On OSX, for example, `Command + C` copies text in practically every application, and Emacs is no different.

The default bindings which I left alone are `s-a`, `s-s`, `s-x`, `s-c`, and `s-v`, which correspond to `mark-whole-buffer`, `save-buffer`, `kill-region`, `kill-ring-save`, and `yank`.

All of these commands already have canoninal keybindings in Emacs, like `C-w` for `kill-region`, but the super-bindings are still convenient to have. For example, copy/pasting some text from a different application into Emacs requires no mental context switch and can be done with one hand. Besides, since pressing `Command + C` followed by `Command + V` is so universal and ingrained into muscle memory, making an exception for Emacs would be weird. So, I avoid overwriting these keybindings in the following sections.

Buffer Navigation

These are my buffer navigation keybindings.

elisp

```elisp
;; Or switch-to-buffer if you don't use helm.
(global-set-key (kbd "s-j") 'helm-mini)
(global-set-key (kbd "s-p") 'previous-buffer)
(global-set-key (kbd "s-n") 'next-buffer)
(global-set-key (kbd "s-k") 'kill-this-buffer)
```

I chose `s-j` for `helm-mini` / `switch-to-buffer` since I perform this action very frequently and `J` is the dominant homerow key for the right hand (my left hand presses Super, which I bound to the Caps Lock key).

Finding Text

For this section, you'll need ag installed and the helm-ag, projectile and helm-projectile packages.

Note:

The commands in this section make use of the helm interface. If you're not a `helm` guy (e.g. you use ivy), then you're on your own.

`ag` is a grep-like tool, but faster, while `projectile` allows you to run ag on your project to search for text. You'll need to be inside a “project” for this to work, and the easiest way to do that is to make the current directory version-controlled. It’s a good habit to get into!

I have two commands here: `helm-projectile-ag-inexact`, which searches the current project for some case-insensitive text (and ignores word boundaries), and `helm-projectile-ag-exact`, which is case-sensitive and only returns matches along word boundaries. I employ the first one when I want a fuzzier search and the second when I know exactly what I’m looking for. `helm-projectile-ag-exact` will fill in the search term from the symbol at the point (cursor), so it can be called by putting the point over a function call and searching for its definition, or the other way around.

elisp

```elisp
(defun helm-projectile-ag-inexact ()
  "Run helm-projectile-ag case-insensitive and without word boundaries."
  (interactive)
  (setq helm-ag-base-command "ag --nocolor --nogroup --ignore-case")
  (setq helm-ag-insert-at-point nil)
  (helm-projectile-ag)
  )

(defun helm-projectile-ag-exact ()
  "Run helm-projectile-ag case-sensitive and with word boundaries."
  (interactive)
  (setq helm-ag-base-command
        "ag --nocolor --nogroup --word-regexp --case-sensitive")
  (setq helm-ag-insert-at-point 'symbol)
  (helm-projectile-ag)
  )

(global-set-key (kbd "s-i") 'helm-projectile-ag-inexact)
(global-set-key (kbd "s-u") 'helm-projectile-ag-exact)
(global-set-key (kbd "s-o") 'helm-ag-pop-stack)
```

The mnemonic I use to remember these is that `s-i` is **i**nexact while `s-o` p**o**ps the stack.

Note:

There are also packages that integrate helm with rg, such as helm-rg, but I haven’t used them myself. **Update:** I’ve tried to replace `ag` with `rg` but wasn’t able to get the same “exact” and “inexact” behavior I specified above.

dumb-jump

For immediately jumping to a function definition in an intelligent way, there's dumb-jump. It works well for most languages and it runs `ag` (or `rg`) behind the scenes, so you need one of those installed.

elisp

```elisp
;; Jump to definitions using ag
(use-package dumb-jump
  :bind (
         ("M-g o" . dumb-jump-go-other-window)
         ("M-g j" . dumb-jump-go)
         ("M-g q" . dumb-jump-quick-look)
         )
  :config
  (setq dumb-jump-selector 'helm)
  (setq dumb-jump-prefer-searcher 'ag)
  )
```

Update:

I've since replaced this with smart-jump, which uses `dumb-jump` as a fallback.

goto-line-show

I got this tip from Reddit of all places. It's an enhancement to `goto-line`, which I had bound to `s-l`. Since I have line numbers turned off (they're not necessary most of the time), it's nice to temporarily turn them on while selecting a line to jump to.

elisp

```elisp
(defun goto-line-show ()
  "Show line numbers temporarily, while prompting for the line number input."
  (interactive)
  (unwind-protect
      (progn
        (linum-mode 1)
        (call-interactively #'goto-line))
    (linum-mode -1)))
(global-set-key (kbd "s-l") 'goto-line-show)
```

You can also use this to temporarily view line numbers and cancel with `C-g`. Is it *that* useful? Probably not. But it's cool.

A Few More

Here are a few more keybindings I have for you.

```elisp
(global-set-key (kbd "s-g") 'magit-status)
(global-set-key (kbd "s-y") 'helm-show-kill-ring)
(global-set-key (kbd "s-h") 'helm-mark-ring)
```

`magit-status` I call all the time – it opens the main Magit buffer for the current project.

`helm-show-kill-ring` is very handy for finding something you killed a while ago or for searching for some specific killed text. I even sometimes open the dialog just to see what I last killed, without having to yank and undo. `helm-mark-ring` shows you last-visited locations in the buffers, with the option of being able to jump to them.

Misc

revert-buffer

I have auto-revert turned on – `(global-auto-revert-mode 1)` – but it's still necessary to manually revert the buffer sometimes, e.g. when I want to re-run some hooks:

```elisp
(global-set-key [f5] 'revert-buffer)
```

It's kind of like a refresh, so I bound it to f5.

cycle-spacing

I have `cycle-spacing` as an alternative to `just-one-space`. From the help page:

> The first call in a sequence acts like 'just-one-space'. It deletes all spaces and tabs around point, leaving one space. The second call in a sequence deletes all spaces. The third call in a sequence restores the original whitespace (and point).

elisp

```elisp
(global-set-key (kbd "M-SPC") 'cycle-spacing)
```

I actually forgot about this keybinding until now. Oops. Guess I should start using it.

zap-up-to-char

This is an occasionally helpful command, as in, I use it maybe once a month. It's an improvement on `zap-to-char`, where you specify a character and all characters from the point up to *and including* the specified character are deleted. I've found it's more practical in most situations *not* to kill the specified character, hence the following:

elisp

```elisp
(autoload 'zap-up-to-char "misc"
  "Kill up to, but not including ARGth occurrence of CHAR." t)
(global-set-key (kbd "M-z") 'zap-up-to-char)
```

indent-buffer

Oh, here's a good one! This lets you indent all the lines in a buffer without having to select the whole buffer, which would lose the current point position.

elisp

```elisp
(defun indent-buffer ()
  "Indent the whole buffer."
  (interactive)
  (indent-region (point-min) (point-max))
  )

(global-set-key (kbd "C-c n") 'indent-buffer)
```

If you use a code formatter like gofmt or rustfmt, you can also bind this key to the format command in the respective modes. For example:

elisp

```elisp
(use-package rust-mode
  :mode "\\.rs\\'"
  :init
  (setq rust-format-on-save nil)
  :config
  (define-key rust-mode-map (kbd "C-c n") #'rust-format-buffer)
  )
```

Don’t You Use evil-mode?

No. Modal editing Vim-style may be popular, and it’s supposedly helped people with RSI issues, but that’s just because no one knows how to properly set up modifier keys for Emacs. For example, I’m pretty sure that using Caps Lock as Control has a high chance of *causing* RSI.

If you can find a suitable keyboard/OS and configure the Alt and Control keys to be next to the spacebar, Emacs bindings are even better than Vi modal editing, in my opinion. Pressing/lifting a thumb is easier than changing editing modes. To be fair, I am a world-class user of the Emacs style and not of the Vim style.

Conclusion

Well, now you know pretty much all of my secrets. Although, I guess I never had secrets, since my `init.el` file has been on GitHub for over a year now.

What are *your* favorite keybindings? Leave them in the comments below!

Footnotes

[1]: Telling people to set Caps Lock to Control is terrible advice and causes unnecessary pain. The pinky is the weakest finger, stretching sideways is an awkward movement that should be minimized, and common modifier keys (Shift, Control, Alt) should be present on *both* sides of the keyboard. ↩

Filed under emacs productivity

top ⇡

⇠ Jekyll: Syntax Highlighting And Line Numbers
Memory Management In Unsafe Rust ⇢

Note: anonymous comments require approval.

Login

Add a comment

M↓ MARKDOWN

COMMENT ANONYMOUSLY

ADD COMMENT