Vim 101

August 2020

Former Emacs user here. Trying to uncover the basics of Vim keybindings, especially for selection, movement and action on text objects, when using Doom Emacs (yes, just cheating, I'm still an Emacs guy).

Overview

Hereafter, I will consider only three kinds of operation in text editing related tasks: motion, selection and transformation. Motion or movement refers to moving around in the buffer. Transformations are applied to text objects, whether it is a single object (e.g., adjacent word or line) or a visual selection. Finally, selection refers to the restriction of the whole buffer to a particular region of text, using either character, word, line, sentence or paragraph as the unit of interest. I will not discuss keybindings specific of a particular mode (e.g., VCS, Markdown, Org) although I may record some shortcuts specific to `prog-mode` in the last section.

There are many good references on Vim itself, or Vim best practices.[1] In the meantime, here's a nice overview of why Vim may be a good companion for programming stuff, written by Steve Losh: Coming Home to Vim. The document Emacs/Evil for Vim Users provides an excellent introduction to the difference between Emacs and Vim terminolgy (e.g., what does 'yanking' means in Emacs or Vim), and how Evil compares to standard Vim. Finally, I recommend the Vim quick reference card for a handy cheat sheet.

[1] Neil 2018; Robbins and Hannah 2008.

Although this document focus on Vim (or Neovim) itself, it is intended as a cheat sheet for Emacs users like me. I have had a hard time switching to modal editing, and I still need some Emacs keybindings, which I also use in other TUI settings, like the terminal (e.g., `C-a` and `C-e` to navigate to the begining or end of the line, `C-w` to delete the previous word).

> I've been able to get by without almost ever using Emacs' default keybindings. The exceptions for me are `C-g` and `C-h`. Even if you don't plan on learning emacs' keybindings in full, I recommend learning these when starting out. — Fox Kiester

Indeed, the `C-g` shortcut remain important as it allows to break almost everything in Emacs, but the help system is easily accessible using Doom Emacs using the leader key (`SPC`) only. Likewise, `C-c C-c` remains ubiquitous in various modes, both in Doom and vanilla Emacs.

So far the best implementation of modal editing à la Vim in Emacs is provided by the Evil package. While the defaults are good, you will likely benefit from its companion packages by using either Spacemacs or Doom Emacs, the later being probably closer to Emacs core functionalities. If you ever need to go back to the classical Emacs editing mode, you'll just have to press `C-z` and you'll get your familiar Emacs keybindings (minus those remapped by the starter kit you choose, if any).

Unless otherwise stated, the following settings apply to both Vim (and mostly Vi) and Emacs (with Evil mode).

Basic stuff

Modes

We can distinguish two main modes in Vim: Normal mode and Insert mode. Insert mode is for writing text *per se* while Normal mode is reserved for moving around or altering text under the cursor or nearby. In addition to Normal mode, there is also a Visual mode, which can operate from point under cursor, line or block. Normal and Visual modes are discussed in the next section. The following paragraphs summarize the main commands that can be used to enter Insert mode.[2]

[2] See also Vim Crash Course, which largely inspired this section.

Whenever we want to write down something, we press `i` or `a` to enter Insert mode before or after the current character. To go back to Normal mode, we just have to press `<Esc>`. There are, however, many other ways to switch to Insert mode, as detailed in Table 1.

Keys	Description
I	Switch to Insert mode at the beginning of the current line
A	Switch to Insert mode After the end of the current line
o	Add a new line after the current line and switch to Insert mode
O	Add a new line before the current line and switch to Insert mode
c<char>	Change text from cursor until occurrence of <char>
cc	Change the whole line
C	Changes from here to end of line

Table 1: Switching between Normal and Insert modes

Normal mode is a bit specific to Vim or other modal editors like Kakoune or Evil mode in Emacs. Insert mode is what we usually expect from the state of a classical text editor: characters are printed as you write them. However, there is an additional mode that bear similarities with Insert mode: Replacement mode. In this case, `r` means to replace the current character with the next character typed, while `R` is used to replace characters until going back to Normal mode (`<Esc>`). Other variants for in-place replacement are detailed in Table 1. Note that some of these key combinations appear as

shorthand for more general key compositions. For instance, `C` may be seen as `Di`, and `o` as `$a<Enter>` (or `A<Enter>`). This is where Vim really rocks: You don't need to learn a lot of keybindings but how to best combine them.

After a few days of use, you should realize that Normal mode should be your default "home mode." Insert mode is dedicated to editing text, and you should refrain from trying to move around or jump to another location in your buffer using this mode. Instead, switch back to Normal mode as soon as you're done with your editing task, and then stick to this mode while you don't need to write down some other text: Use motion to jump around, or select text or region using visual mode or shorthand commands (see next section).

Motion

Normal mode is the main mode, together with Visual mode, we use to jump to different locations in a buffer or to select part of it. We can consider two (complementary) approaches for moving around in a buffer: moving by entity (e.g., character, word, matching brackets, line, or block), or by matching pattern after a search query, based on regular expression or plain text. Most of the time, this is followed by an edit operation (in Insert mode), and then we go back to Normal mode for further operations, as recommended above.

The quadriumvirat for Vim motion by character are the `hjkl` keys. No matter what others say, the arrow keys are also a good fit in many cases. Combined with the `<fn>` key on a Mac, for example, this allows for both moving in all directions, and scrolling page up or down. However, the `hjkl` combo remains useful if you want to combine them with a quantifier, e.g. `5j` to jump to the fifth line after the current one, or `>5j` to indent the current line and the next 5 — or if efficient touch typing and the home row keys work for you.

Basically, what I usually want to do is: move at the beginning or end of current (physical) line – in Insert mode I use standard Emacs keybindings `C-a` and `C-e`,[3] jump to the previous or next word (but this is generally to yank or delete this word), or to the previous or next blank line (useful in `text-mode` and `prog-mode`). I also like moving between opening and closing brackets or parenthesis. Finally, I find it useful to quickly jump to a specific line number.

So basically, given the next snippet of text, I want to be able to jump quickly to position A, B, C and D, or to select region delimited by either two of these anchors. This basically amounts to jumping to a specific word in a sentence, or the beginning or end of the sentence itself. Let's call it a line in this context.

[3] To use standard Emacs movements to jump to the beginning or end of line in Insert mode, one can use (in Vim): `inoremap <C-e> <C-o>$` and `inoremap <C-a> <C-o>0`. With Evil, this becomes (see Evil Mode best practice on Stack Overflow): `(define-key evil-normal-state-map "\C-e" 'evil-end-of-line)`, with similar instructions for `evil-motion-state-map` and `evil-visual-state-map`. Note that the `evil-insert-state-map` is not necessary with Doom Emacs.

```
The quick brown fox jumps over the lazy dog.
^   ^                           ^           ^
A   B                           C           D
```

There are more complex scenarios but I feel like these are the most common jumps: A is the beginning of a sentence (or line), A–B corresponds to moving to the beginning of the next word, A–C is a jump to next word that start with a 't', and D is the end of the sentence (or line), which means that A–D is akin to jumping from the beginning to the end of the sentence. Common keybindings appear in Table 2.[4] Of note, Emacs offers a visual-line-mode which allows to wrap words at the right edge of the window while redefining simple editing commands to act on visual lines, not logical lines. This may be confusing at first since `$` will jump to the end of the current line, but then `j` or `k` will jump to the beginning or end of the visual block, not the previous or next visual line.

[4] Note that `[` and `]` are prefix operators, much like `z` or `g`, in Doom Emacs. They are generally used to navigate in the buffer list, or to jump to the next or previous error or Git hunk. See also Table 3.

Keys	Description
b	move cursor to previous word
w	move cursor to next word
0	Go to beginning of line
$	Go to end of line
gg	Go to beginning of buffer
G	Go to end of buffer
:X	Go to line number X
f<char>	Go to next occurrence of char on line
F<char>	Go to previous occurrence of char on line
C-u	Scroll to previous screen
C-d	Scroll to next screen
o	Toggle between beginning and ending of selection
%	Toggle between beginning and ending of matching delimiters
(	Jump to beginning of paragraph
)	Jump to end of paragraph
{	Jump to next empty line
}	Jump to previous empty line

Table 2: Basic shortcuts for motion in visual mode

Selection

As seen above, motion does constitute the most important aspect of Vim actions in Normal mode. Motion can also be used in conjunction with specific action, e.g. delete all the characters until the next comma, or copy the text within the matching parenthesis. Let us consider the same example as before:

```
The quick brown fox jumps over the lazy dog.
```

In Normal mode, to select the first word in the above example, `yw` can be used when the cursor is set to the first character. If this is not

the case, moving at the beginning of the visual line (`0`) will suffice (see next section). To select the second word ('quick'), we would first need to move the cursor to the first character, and then select the word. We can use either one of these two instructions: `wyw` or `fqyw`. To select 'jumps' instead, we would use `4wyw` or `fjyw`. When using the `f` key to jump to the next occurrence of a `<char>`, don't forget that we can repeat such a jump by pressing `;` as many times as needed.

Finally, the evil-easymotion package (in Emacs) or vim-easymotion plugin (in Vim) will simplify a lot of this "jump-around" tasks, since it will provide you with visual marks (alphabetical characters, or ordered bigrams) that you just need to select using the keyboard: For instance, if you're looking for the next word that starts with the "a" letter, instead of typing `fa`, you can just use the corresponding `easymotion` shortcut which will highlight all "a" next to the cursor position, and replace them with alphabetical letters; the first match will be labelled "a", the second one "b", and so on. Then you just have to type the letter you want to jump at its location.

To select the whole line, `yy` or `y$` could be used instead if we are at the beginning of the line. The instruction for yanking, `y`, can be replaced with `d`, to delete, or `c`, to edit in place. On any given character, `r` allow to switch to replace mode and to edit the character under the cursor, as discussed above. Capital letter variants for `y`, `c`, `d` and `r` do exist: They are used as a shorthand for `y$` (not in all config), `c$` (see table 1), `d$` and `r$`.

Most importantly, recall that unlike Emacs, the cursor is always positioned on a character, and not in between. Also, this is often the case that when we are at the end of a line in Insert mode (e.g., just after the comma ending a sentence), switching back to Visual mode brings us one character backward (i.e., on the comma itself).[5]

A simple alternative to `yw` is `yiw` (or `yaw`): these commands allow to yank the current word excluding (or including) the surrounding whitespaces. Also, `yf<char>` will yank from the current cursor position up to and including the character `<char>`. See next section about the `f` (or `F`) instruction. Technically, those kind of instructions imply a motion (`y{motion}`), like `iw` (inside word). Unlike `yw`, `yW` will select the whole object, including special character like `-`, `#` or `::`. In the following Racket snippet, the cursor is on the first letter of the function name, `largest-prime-factor`:

```
(require math)

(define (largest-prime-factor x)
        ^
(apply max (map car (factorize x))))
```

[5] In Vim, the cursor go back one character back when exiting Insert mode. If you don't like it, in Emacs you can add the following instruction in your config file: `(setq evil-move-cursor-back nil)`.

In this case, `yw` will only yank `largest` while `yW` will yank the full function name. Using motion (see next section), we could also use `y%` to yank the function name and its parameter, that is all text objects occurring between the parenthesis. A similar effect can be obtained with `yi(` (yank inside parenthesis), of course. Strictly speaking, `%` is used for motion: inside the brackets it will move the cursor to the first brackets, while % on any of the two matching delimiters will move the cursor from one to the other.

Copy, cut and paste do not involve the system clipboard (like `pbcopy` and `pbpaste` on macOS) but Vim default register which is denoted as a single double quote ("). In Insert mode, the last object copied can be pasted using `C-r "` (this is the default value). In Normal mode, we use `""p`.[6] As a final note, instead of pasting in place (i.e., without moving the cursor afterwards), one may use `gp` (paste after) or `gP` (paste before) to move the cursor after the pasted object.

[6] The first double quote allows to access register values, and the second double quote means that we want the default register. The more general syntax to yank text to a named register is a single double quote followed by `\<char>y`, where `<char>` is the name of the register, usually a single letter [a-z]. To put the text in the system clipboard, use the $*$ register.

Objects

We have seen how to operate basic movements and actions on selected regions. The last important concept in Vim is that of object. The most common objects are `w` (or `W`) for words, `s` for sentences, `p` for paragraphs, `b` for parenthesized blocks, `"` for quotes, as well as all variations of brackets (rounded, squared, etc.).

As seen above, `yw` amounts to an action on a specific object, which is a word, but could be anything else: a sentence, a paragraph or any number of objects in between square brackets.

Advanced stuff

Motion using prefix operators

Keys	Description
g;	Goto last change
gˆ	First non blank character
g0	Jump to beginning of visual line
gm	Jump to middle of visual line
g$	Jump to end of visual line
gd	Go to definition
gD	Go to references
zz	Scroll line to center (of screen)
zb	Scroll line to bottom
zt	Scroll line to top

Table 3: Motion shortcuts in visual mode for Doom Emacs

The shortcuts listed in Table 3 are specific to Doom Emacs and are usually mapped under the `g` or `z` prefix operator. There are also

keybindings specific to `prog-mode` like `gd` or `gD`: they are mapped to the corresponding `xref-find-*` functions unless the lsp package is installed in which case they are associated to `lsp-find-*` functions. Likewise, `[l` and `]l` in an Org buffer can be used to jump to the previous or next link (in the EWW browser, it is just `<tab>` and `s-<tab>`).[7]

The vim-wordmotion allows for further refinements in the case of word motions.

[7] Here, `s-<tab>` means `<shift> + <tab>`, not the hyper, meta, or alt key commonly refered to as `s` in Doom Emacs.

Shortcuts in insert mode

Keys	Description
C-h	Delete the character before the cursor during insert mode
C-w	Delete word before the cursor during insert mode
C-j	Add new line during insert mode
C-t	Indent line one shiftwidth during insert mode
C-d	De-indent line one shiftwidth during insert mode
C-n	Auto-complete next match before the cursor during insert mode
C-p	Auto-complete previous match before the cursor during insert mode

Table 4: Motion shortcuts in insert mode

Custom commands

The `z` prefix is used for folding: `za` and `zc` are used to open the fold at point and to close all opened folds.

The `z` prefix is also used for spelling tasks: `z=` corrects word at point, `zg` will add the word at point in the dictionary, and `zw` will mark it at incorrect. Note that navigation is handled by `]s` and `[s`, which are used to navigate to the next or previous mispelled word. Finally, don't forget that `C-p` allows to autocomplete the word at point, which may be handy in some case.

Navigating the file system

Vim has a built-in file browser (`netrw`), but several plugins are available to browse files and directory. The most popular ones are probably NERDTree, vim-dirvish and defx, although vim-vinegar is also a valid option since it provides some enhancements to netrw. It all depends whether you need a nice sidebar showing all files and directories of the current project, possibly with Git status and nice icons alongside. If this is not the case, then `vim-dirvish` or `vim-vinegar` are more interesting options since they allow to manipule file paths like in any Vim buffer.

When using `vim-vinegar`, the basic shortcuts are quite simple to memorize: `-` allows to open netrw in the current directory, `.` ap-

pend the file path to an ex command (:), and ~ go back to the home directory.

References

Neil, Drew (2018). *Modern Vim: Craft Your Development Environment with Vim 8 and Neovim*. The Pragmatic Bookshelf.

Robbins, Arnold and Linda Hannah Elbert adn Lamb (2008). *Learning the Vi and Vim editors*. O'Reilly.