# Editing Text With Vim

Hassan Shabbir

February 11, 2019

## Contents

# 1 The main text editors

There are two main (extensible) text editors for Unix and Gnu/Linux operating systems. These are the VI and Emacs lines of text editors.

Both of these text editors are highly configurable and therefore are highly recommended by many programmers.

## 1.1 VI

VI (pronounced "ve-eye"), and other text editors based off of it are great at creating and editing text. It is (usually) a basic text editor, however newer versions of it allow a user to customize it and get IDE-like features within it. The main feature of VI is the way in which it allows a user to edit text.

## 1.2 Emacs

Emacs (pronounced "ee-maks") is a text editor that is extremly configurable, and allows for a lot more IDE-like features. This is due to the fact that Emacs is a text editor based upon the functional programming language Lisp. This means that you are able to use a general purpose programming language to modify the text editor. Emacs can also emulate VI keybindings using EVIL mode (EVIL stands for Extensible VI Layer).

## 1.3 Emacs and VI Lore

**Lore:** a body of traditions and knowledge on a subject or held by a particular group, typically passed from person to person by word of mouth.

Both of these text editors are really amazing, so much so that since their inception many programmers have been arguing over which text editor is better, see The Editor Wars (clickable).

VI is the editor of the beast since VI in roman numerals is 6, so VI VI VI would be 666.

From here on out we will be talking about the VI line of editors. The curious reader is encouraged to read more about Emacs, if interested.

# 2 Which Vim version should I use?

NOTE: There is almost no difference between these different text editors except for the information written below.

NOTE: I will assume that Vim/NeoVim is used for the rest of this file, but will refer to it as Vim, since I am lazy.

There are three different VI versions you should be familiar with.

### 2.1 VI

VI[1] is the oldest text editor we will be looking at, which was created in 1976. VI is short for Visual, differentiating it from line editors.[2] If you plan to work with many servers, you should expect literally expect every server to have it. It is very minimalistic so it won't tell you when you are in insert mode in any way (see below), for example. This makes it harder to understand for beginners, and doesn't have all the features of Vim.

### 2.2 Vim

Vim[3] is the newer version of VI, first released in 1991. Due to it being very popular for a long time, and still is (last stable release: 17 May 2018), it has many resources available on the internet for it. It is also recently modern, so it shouldn't be too difficult to use. Therefore, whenever searching for general vim resources use vim in the search terms.

### 2.3 NeoVim

NeoVim[4] is the newest version of vim, first released in November 1, 2015. This has the most interactive features and therefore is the one that I would recommend to new users. For example, when you switch between different modes, the cursor changes, helping you to remember which mode you are in.

## 3 Modal editing with Vim

The first thing you need to know about Vim is that there are four main modes in which you operate. Each of these modes change what the keys on your keyboard will do.

[1] VI pronounced "ve-eye", also pronounced "vy" but that is an unofficial pronounciation

[2] Technically, the "ed" and "ex" editors are even older, but they literally show you nothing (you have to manually print lines), so it is like editing text in the dark. This is where the command mode in VI comes from, see below. Also see Actually Using Ed (clickable) for some extreme masochism.

[3] VIM pronounced "vim"

[4] NeoVim pronounced "neo-vim". The command to run it is `nvim` NOT `neovim`, and it can be installed using `sudo apt install nvim`.

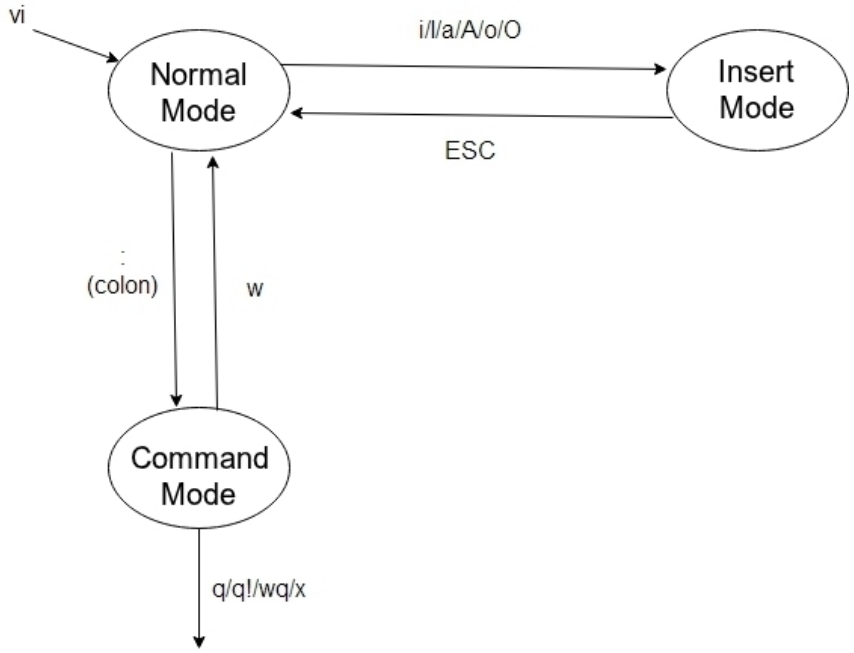


Figure 1: Vim Modes, and the way to access different Modes (Modified horribly by me).

In general, when editing text you will mostly be making small changes, and very rarely do you create whole documents without mistakes from start to finish (`cat > foo.txt` anyone?). For this reason, Vim is optimized for modifying text. Understanding modal editing (along with composability, repeatability, and text objects) is the key to understanding Vim[5].

### 3.1 Normal Mode (Movement; Modification)

Normal mode allows you to issue commands to Vim, which will then do something, other than putting the key that you pressed into the file. When you first open a file you will be in Normal Mode. Pressing keys such as `j` or `l` will move the cursor rather than adding those letters to the file. You should almost always be in Normal Mode, since editing text requires the ability of moving around the document, and deleting, replacing, copying text, etc. which are all possible in Normal mode. Thus this mode is called "normal" since it is the default mode when using Vim. To get back into Normal Mode use the escape key, or control and the left-squrebracket key.

### 3.2 Insert Mode (Add Text)

When opening a document with Vim, you will be in Normal Mode. To get into Insert Mode, you can press keys such as `i` or `a` and then you will be in Insert Mode. If you are using NeoVim, you will see the cursor become thin, and in both Vim and NeoVim you will see `--INSERT--` at the bottom of the terminal.[6] You can then use the arrow keys to get to the location, and press the keys to add them to the document. To get back to Normal Mode press escape. (This is not recommended, but can help you get used to Vim. Movement should be done using Normal Mode, not the arrow keys, allowing your hand to stay on home row) You will notice that the cursor will become a block again in NeoVim.

### 3.3 Command Mode (System Commands; Ed commands)

For now, the most important command mode you need to know will be the commands to exit Vim (which is accessible from Command Mode). This is such a problem for beginners that this stackoverflow answer (clickable) has 4,000 upvotes, and over 1 Million views. The first step is to press `:` from

[5]For more on how vim works see this awesome answer on Stackoverflow Your problem with Vim is that you don't grok vi (clickable).

[6]In VI you will neither see the cursor change nor the `--INSERT--` at the bottom

Normal Mode. If you are not in Normal Mode, get to it using the escape key. You will now see a colon on the last line of the terminal. If you wish to save your changes type `wq`, and then press enter. This command stands for write (save) the file then quit Vim. If you wish to throw away your changes type `q!` and then press enter.

### 3.4 Visual Mode (Select Text)

Visual Mode is used for performing an operation over all of the characters in the text. This can be useful when you don't know how to run operations using text objects. Text objects will replace most simple uses of Visual Mode.

## 4 Basic Vim editing commands

NOTE: Only learn the commands that you want. Then when you get annoyed by inefficiency come back to learn more.

NOTE: Pressing the Escape key will return you back to Normal Mode from any mode.

NOTE: Vim uses mnemonic devices (ie. `d` stands for delete) to help you remember what command does what. Use this to remember what each command does. Also, commands that are related, but do something different are capitalized, and the default action is defined by the repeated letter (such as `dd` for delete with default action).

Sections will be in the form: CommandName (from StartingMode)

Commands will be in the form:

- `COMMAND`: (mnemonic device) Description of command

### 4.1 Entering Vim (from bash prompt)

You can enter Vim from the commandline (not to be confused with Vim's Command Mode) by typing `vim file.txt`, replacing `file.txt` for the file you want to edit. If the file doesn't exist, it will be created. You will now be in Vim.

### 4.2 Movement Commands (from Normal Mode)

#### 4.2.1 Character Movement

- `h`: Move cursor left
- `j`: Move cursor down

- `k`: Move cursor up
- `l`: Move cursor right

The way to remember this is that the `h` key is on the left of the four keys, `l` is on the right, `j` is written with the hook below the line, and `k` has the vertical line above the line.

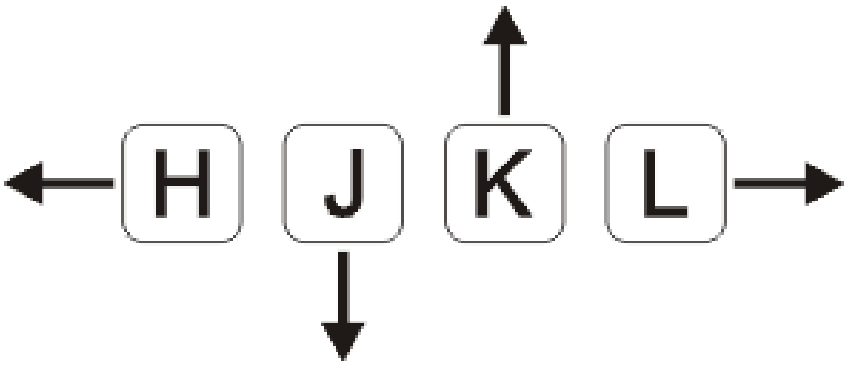


Figure 2: A graphical depiction of h, j, k, l

#### 4.2.2 Word Movement

- `w`: (Word) Go forward by one word
- `b`: (Back) Go back by one word
- `e`: (End) Go to the next end of word

#### 4.2.3 Line Movement

- `^`: (This is from Regexes) Go to start of line
- `$`: (This is from Regexes) Go to end of line

#### 4.2.4 File Movement

- `gg`: Go to start of file
- `G`: Go to end of file

#### 4.2.5 Find Char Movement

- `f`: (Find Char) Find character forward
- `F`: (Find Char) Find character backward
- `;`: Run `f` / `F` again
- `,`: Run `f` / `F` again in opposite direction

#### 4.2.6 Search Term Movement

- `/`: Enter search term, then press enter
- `n`: (Next) Go to next location matching search term
- `N`: (Previous/Backwards Next) Go to previous location matching search term

### 4.3 Insert Commands (from Normal Mode)

These commands will change you automatically from Normal Mode to Insert Mode.

- `i`: (Insert) Enter Insert Mode before current character
- `I`: (Insert) Enter Insert Mode at the beginning of the line
- `a`: (Append) Enter Insert Mode after current character
- `A`: (Append) Enter Insert Mode at the end of the line

### 4.4 Deletion Commands (from Normal Mode)

NOTE: The composable nature of Vim should be apparent in this section.

- `x`: Delete character under cursor
- `dd`: (Delete, Default) Delete current line
- `dw`: (Delete Word) Delete until the end of the word
- `d3w`: (Delete Word) Delete 3 number of words, etc.
- `diw`: (Delete In Word) Delete the whole word
- `diw`: (Delete In Word) Delete the whole word
- `diW`: (Delete In Word) Delete the whole space delimited word

### 4.5 Deletion Commands (from Visual Mode)

- `d`: (Delete) Delete current visual selection
- `x`: (Delete) Delete current visual selection

### 4.6 Change Commands (from Normal Mode)

Change deletes something then puts you in Insert Mode to add text.

- `cc`: (Change, Default) Delete line, then go into Insert Mode
- `cw`: (Change Word) Delete until the end of the word, then go into Insert Mode
- `c3w`: (Change Word) Delete 3 number of words, etc., then go into Insert Mode
- `ciw`: (Change In Word) Delete the whole word, then go into Insert Mode
- `ciw`: (Change In Word) Delete the whole word, then go into Insert Mode
- `ciW`: (Change In Word) Delete the whole space delimited word, then go into Insert Mode

### 4.7 Yank (Copy) Commands (from Normal Mode)

NOTE: To copy text to use in other applications, use the `"+` prefix, see registers below.

- `yy`: (Yank, Default) Yank (copy) the current line, for Vim use only
- `yiw`: (Yank) Yank (copy) the current line, for Vim use only
- `"+yy`: (Yank, Default) Yank (copy) the current line, for any application
- `"+yiw`: (Yank) Yank (copy) the current line, for any application

### 4.8 Yank (Copy) Commands (from Visual Mode)

- `y`: (Yank) Yank (copy) current visual selection

### 4.9 Paste Commands (from Normal Mode)

- `p`: (Paste) Paste the last deletion/yank

### 4.10 Paste Commands (from Visual Mode)

- `p`: (Paste) Paste, replacing current visual selection

### 4.11 Undo Command (from Normal Mode)

- `u`: (Undo) Undo last change

### 4.12 Visual Mode Commands (from Normal Mode)

First enter Visual Mode using any of the below, then make the selection using the movement commands as you would from Normal Mode. Then run the command on the selection, such as yank, delete, etc.

- `v`: Enter character-wise Visual Mode
- `V`: Enter line-wise Visual Mode
- `ctrl-v`: Enter block-wise Visual Mode

### 4.13 Command Mode (from Normal Mode)

- `:w`: Write the file
- `:q`: Quit Vim, without having modified the file
- `:q!`: Quit Vim, throwing away modifications
- `:wq`: Write the file, then quit Vim
- `:x`: Shorthand for `:wq`
- `:! date`: Run bash command `date` and show the result without adding to file
- `:r! date`: Run bash command `date` and read in the result into the file

### 4.14 Command Mode (from Visual Mode)

Visually select text then enter Command Mode using `:`. NOTE: you will see `:'<,'>` instead. This just tells Vim to run the command over the whole selection.[7]

- `:'<,'>! wc -l`: Run bash command on visually selected text

## 5 Composability and Repeatability

### 5.1 Text Objects

NOTE: All text objects can be used with delete, yank, copy, etc.

- `iw`: (In Word)
- `aw`: (All Word)
- `is`: (In Sentence)
- `as`: (All Sentence)
- `ip`: (In Paragraph)
- `ap`: (All Paragraph)
- `i"`: (In Quote)
- `a"`: (All Quote)
- `i}`: (In Brace)
- `a}`: (All Brace)
- `it`: (In Tag) Used in HTML
- `at`: (All Tag) Used in HTML

[7] So the command will run in the range `x,y`, and a `'a` refers to the mark a, with the `<` referring to the first and `>` referring to the last selection. So all together it says "run the command from the beginning of the selection, to the end of the selection."

### 5.2 Dot (.) command

The dot command repeats the last complete command that you ran. For example if you changed a word to "Hi" using `ciwHi` and then escape, you can change another word to "Hi" using dot.

This is one way of renaming variables. First search for a variable using `/`, then using `ciw` change the variable to something else. Finally, repeat this change all throughout the document using `n` to go to the next instance, and `.` to run the change.

### 5.3 Number Prefixes

Most commands can be prefixed, meaning you can run commands like `d5w` which will delete the next 5 words.

### 5.4 Macros

Macros can be used for creating groups of repeatable commands. In other words, start macro, run general commands (ie. `w` rather than `llllllll`), stop macro, run the macro previously defined on all of the remaining text.

- `qa`: Record Macro in register `a`, see below
- `q`: While recording, it will end the macro
- `@a`: Run Macro in register `a`

## 6 Registers

The most important part about registers is that the `"+` register is used to store the global clipboard, which can be accessed by any program. Frankly, I don't use registers at all otherwise.

## 7 Extending Vim for yourself

To change the default behaviour of Vim, you can modify a configuration file called `.vimrc` (in Gnu/Linux) or `_vimrc` (in Windows, I think).

This will allow you to use plugins, change colorshemes, map keys to commands, etc.

## 7.1 Plugins

These are a few plugins that I would consider quite useful. All links clickable.

- Vim Plug: Vim plugin manager

To be able to use the below plugins you need to install a plugin manager, this is the one I personally use.

- Vim Sensible: default settings for Vim

This is useful for starting off in Vim. (Not needed for NeoVim.)

- Vim Commentary: (un)comment lines of code
- Vim Surround: surround text objects with text
- Vim Vinegar: simple file browser in Vim
- Emmet Vim: create HTML easily
- Ctrlp Vim: fuzzy find files
- Targets Vim: add more text objects to Vim

More plugins for Vim can be found on `https://vimawesome.com`.

### 7.1.1 ColorSchemes

- Space Vim Dark
- Solarized

### 7.1.2 Vim in other places

- Bash: Bash has a Vim mode that can be enabled
- Athame: Full Vim in the terminal, ie. when writing bash commands
- Vimium: Vim in Chrome

There are also other applications that will use Vim-like keybindings by default, such as `man`.

# 8 Conclusion

Congratulations on finishing this whole document! You should now know enough to be able to use vim, and look up whatever you need on the internet. To become proficient with Vim, you should use it repeatedly, until the basic commands come to you without much thought.