Editing Text With Vim

Hassan Shabbir

February 26, 2019

Contents

1 The main text editors

There are two main (extensible) text editors for Unix and GNU/Linux[1] 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, so much so that since their inception many programmers have been in a "war", arguing over which text editor is better.[2]

1.1 VI

VI (pronounced "vee-eye"), and other text editors based off of it are great at creating and editing text, so much so that you will hear VI users call it a "language" for 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.[3] The main feature of VI is the cohesive "language" it provides for editing text.

1.2 GNU Emacs

GNU[4] Emacs (pronounced "g'noo ee-maks") is a text editor that is extremely configurable and allows for a lot more IDE-like features. This is due to the fact that Emacs is a text editor written and extensible in the general purpose functional programming language Lisp. Emacs can also emulate VI keybindings using EVIL mode (EVIL stands for Extensible VI Layer). It is also interesting to note that Richard Stallman,[5] the creator of GNU Emacs, also created the first alternative to Unix, which was GNU, the Free Software Foundation, as well as the GNU General Public Licence.

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.

[1] I'd just like to interject for a moment. What you usually refer to as Linux, is in fact, GNU/Linux, or as I've recently taken to calling it, GNU plus Linux. Linux is not an operating system unto itself, but rather another free component of a fully functioning GNU system made useful by the GNU corelibs, shell utilities and vital system components comprising a full OS as defined by POSIX. (See GNU Linux copypasta.)

[2] See The Editor Wars.

[3] This is usually not encouraged, especially at the beginning, since having plugins hinder your ability to understand Vim, and are nice to have and not necessarily mandatory for the functioning of Vim. In fact, for GNU/Linux users, the IDE is the operating system. See Unix as IDE.

[4] GNU stands for "GNU's Not Unix", a recursive acronym.

[5] Richard Stallman.

2 Which VI version should I use?

The only differences between the different versions of VI are stated below. There may be more differences, but those won't be important to a beginner.

I will also assume that you will be using Vim or NeoVim for the rest of this file, but will generally refer to them as Vim.

The three different VI versions you should be familiar with are VI, Vim, and NeoVim.

2.1 VI

VI[6] is the oldest text editor of the three[7], which was created in 1976.[8] VI is short for Visual, differentiating it from line editors. 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[9] 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[10] is the newest version of vim, first released in 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.

---

[6] VI pronounced "vee-eye", also pronounced "vy" but that is an unofficial pronunciation.

[7] Technically, the "ed" and "ex" editors are even older, but are quite cumbersome to use. For example, they require you to print a range of lines to be able to see them.

[8] This is where the command mode in VI comes from, see below. Also, see Actually Using Ed for some extreme masochism.

[9] Vim pronounced "vim".

[10] NeoVim pronounced "neo-vim".

3 Installing Vim/NeoVim

Packages (programs) can easily be installed in any Linux distribution using the command line. For example, to manage packages on Ubuntu Linux, use the package manager `apt`. The full command to install Vim would then be `sudo apt install vim`, and the command to install NeoVim would be `sudo apt install nvim`.[11]

4 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 changes what the keys on your keyboard will do.

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.[12]

4.1 Normal Mode (Modification; Movement)

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 to move 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. There are three different ways to get to Normal mode: by pressing the `Esc` key, `ctrl-c`, or `ctrl-[`.[13]

The two main things you can do from within Normal Mode are either modifying the text, or moving the cursor around. However, as stated before, Vim gives you an easy language to talk about changes by combining the modification command with a motion command. An optional number prefix allows you to repeat the complete command that many number of times.

[11] The command to both run and install it is `nvim` NOT `neovim`.

[12] For more on how vim works see this awesome answer on StackOverflow Your problem with Vim is that you don't grok vi and Why, oh WHY, do those #?@! nutheads use vi?.

[13] Most Vim users prefer to remap the Caps Lock key to Escape, for convenience.

Therefore, the template of a complete Vim command is `<count> <operator> <motion/text-object>`[14].

Some examples of the Vim text editing language are listed below, in English:

- Delete the default amount (uses <operator>)
- Move to the end of the line (uses <motion>)
- Delete to the end of the line (uses both <operator> and <motion>)
- Delete the current paragraph (uses both <operator> and <text-object)
- Paste the default amount 10 times (uses both <count> and <operator>)
- Paste the current paragraph 10 times (uses <count>, <operator> and <text-object>)

4.2 Insert Mode (Add Text)

When opening a document with Vim, you will be in Normal Mode. To get into Insert Mode, for example, 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.[15] 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 commands should be done using Normal Mode, not the arrow keys or the mouse, allowing your hand to stay on home row[16].) You will notice that the cursor will become a block again in NeoVim.

4.3 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 Vim beginners that this StackOverflow answer has 4,000 upvotes, and around 1.8 million views. First enter normal mode by pressing the escape key, then press `:`. You will now see a colon on the last line of the

[14] Text objects will be covered much later.

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

[16] See Why Vim doesn't need the mouse for more.

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!` instead and then press enter.

To get a deeper understanding of how Vim deals with files and why it has many different commands for exiting Vim, let's understand how Vim edits files. When you are in the terminal, the file you wish to edit resides in your hard drive. Then when you open the file in Vim, a copy of the file is put into memory for easy access. You then modify the file using Vim. If you try to close Vim now, it will be unsure if you were sure you wished to discard the changes or if you forgot you made changes. At this point, you can either tell it that you wish to discard the changes (`:q!`), or you forgot to save the changes (`:wq`). Once you choose to save the file the copy of the file that was in memory will then replace the version of the file on the hard drive.

4.4 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 allow you to refer to regions of text, such as "in braces", "in tag", "all paragraph", etc. Text objects will replace most simple uses of Visual Mode.

5 Vim editing commands

5.1 Before learning more Vim commands

Before continuing to learn more Vim commands, there are a few general things to cover.

Firstly, there will be a few different modes used below. If you are ever unsure of which mode you're in, you can always press the escape key to get you back into Normal Mode, from which you can run another command or go into another mode.

Secondly, commands in Vim are quite mnemonic and cohesive. This means that you can easily learn the key associated with a command by remembering what it stands for. An example of this is the `d` command which stands for delete. Also, commands that are related, but do something different are capitalized. For example, `D` deletes to the end of the line. Finally, the default action is defined by repeating the key for the command. For example, `dd` deletes the line, which is the default action for the delete command.

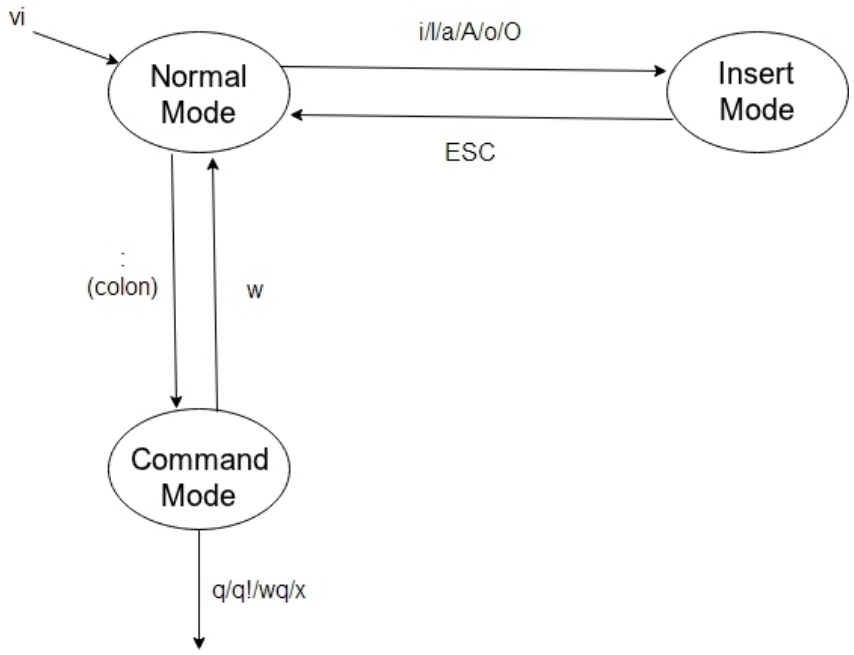

Figure 1: General overview of Vim Modes. Will be covered in depth later.

Third, instead of making your own notes of this document, you can use VI/Vim cheat sheets, such as the following cheat sheet.

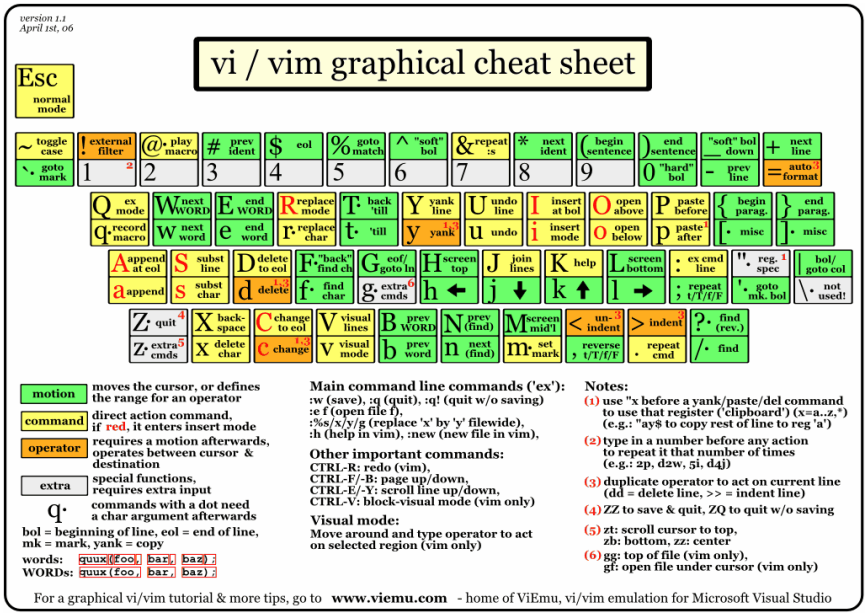

Figure 2: A cheat sheet for many Vim commands explained below. (Don't get overwhelmed by all of the commands!)

Finally, the headings will help you to organize the information you learn into compartments. The headings will be in the form `CommandName (from StartingMode)`, with subheadings dividing the topic into `beginner`[17] and `intermediate` commands. The commands themselves will be in the form `COMMAND: (mnemonic device) Description of command`.

5.2 Entering NeoVim (from bash prompt)

You can enter NeoVim from the command line (not to be confused with Vim's Command Mode) by typing `nvim file.txt`[18], replacing `file.txt`

[17] The 'Beginner' subheading will show the commands that I personally think a new Vim user should learn. Also, you only need to learn the commands that you think are necessary for yourself, coming back later to deepen your knowledge of Vim.

[18] You can also type multiple files after the command and switch between them using `:bn` (buffer next) and `:bp` (buffer previous). A buffer can be thought of as a file that Vim has open for you, that you are currently not editing.

for the file you want to edit. If the file doesn't exist, it will be created. You will now be in NeoVim.

If you wish to use Vim, replace `nvim` in the command above with `vim`.

5.3 Movement Commands (from Normal Mode)

- Why Vim doesn't need the mouse

5.3.1 Character Movement

1. Beginner

    - `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.

    Character movement can also be prefixed with a number such as `5l`, to go 5 characters right.

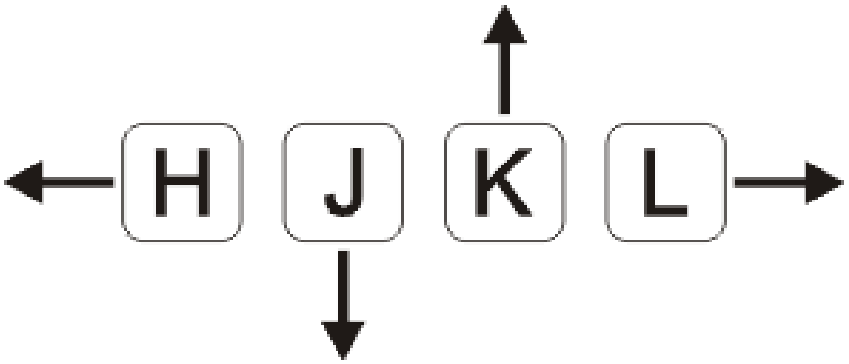

Figure 3: A graphical depiction of h, j, k, l

5.3.2 Line Movement

1. Beginner

- `^`: (This is from Regexes[19]) Go to start of line
- `$`: (This is from Regexes) Go to end of line

5.3.3 File Movement

1. Beginner
   - `gg`: (Go) Go to start of file
   - `G`: (Go) Go to end of file
2. Intermediate
   - `50gg`: (Go) Go to line 50

5.3.4 Word Movement

1. Intermediate

   Frankly, I used to just spam `h` and `l` for quite a while, so these commands aren't strictly necessary.

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

5.3.5 Find Char Movement

1. Beginner
   - `fx`: (Find Char) Find character 'x' forwards
   - `;`: Run `f` / `F` again
2. Intermediate
   - `Fx`: (Find Char) Find character 'x' backwards
   - `,`: Run `f` / `F` again in opposite direction
   - `tx`: ('Til/Until) Go up until character 'x', forwards
   - `Tx`: ('Til/Until) Go up until character 'x', backwards

[19] Regexes, or regular expressions, are a way of doing things like parsing and substituting in a file. The regex `^hi` says to match the line starting with `hi` and the regex `^$` says match the empty line (ie. the line that starts and ends with nothing in between).

5.3.6 Search Term Movement

1. Beginner
    - `/`: (Search[20]) Input search term then press enter, searches forward
    - `n`: (Next, same direction) Go to next location matching search term
2. Intermediate
    - `N`: (Next, opposite direction) Go to previous location matching search term
    - `?`: Input search term then press enter, searches backward

5.4 Insert Commands (from Normal Mode)

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

5.4.1 Beginner

- `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

5.4.2 Intermediate

- `o`: (Open) Add new line below and go into insert mode
- `O`: (Open) Add new line above and go into insert mode

5.5 Deletion Commands (from Normal Mode)

NOTE: The composable nature of Vim should be apparent in this section.[21]

[20] Similar to regexes and its uses in awk commands, etc.
[21] See Mastering the Vim language for more.

5.5.1 Beginner

- `x`: Delete character under cursor
- `dd`: (Delete, Default) Delete current line
- `dw`: (Delete Word) Delete until the end of the word
- `dfc`: (Delete Find 'c') Delete including the first 'c' on the right of the cursor
- `diw`: (Delete In Word) Delete the whole word
- `diW`: (Delete In Word) Delete the whole space delimited word

5.5.2 Intermediate

I can't really be bothered to count how many words I want to delete. I prefer doing things like `dw..` instead, see below.

- `d3w`: (Delete Word) Delete 3 number of words, etc.

5.6 Deletion Commands (from Visual Mode)

5.6.1 Beginner

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

5.7 Change Commands (from Normal Mode)

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

5.7.1 Beginner

- `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
- `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

5.7.2 Intermediate

I can't really be bothered to count how many words I want to change. I prefer doing things like `cw..` instead, see below.

- `c3w`: (Change Word) Delete 3 number of words, etc., then go into Insert Mode

5.8 Misc. Normal Commands (from Normal Mode)

These commands change the file while keeping you in Normal Mode.

- `»`: (Indent) Indent the current line
- `>ip`: (Indent In Paragraph) Indent the current paragraph
- `«`: (Outdent) Outdent the current line
- `<ip`: (Outdent In Paragraph) Outdent the current paragraph

5.9 Yank (Copy) Commands (from Normal Mode)

NOTE: To copy text to use in other applications, use the `"+` prefix, which may not work in VI/Vim, also see registers below.

5.9.1 Beginner

- `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

5.10 Yank (Copy) Commands (from Visual Mode)

5.10.1 Beginner

- `y`: (Yank) Yank (copy) current visual selection, for Vim use only
- `"+y`: (Yank) Yank (copy) the current selection, for any application

5.11 Paste Commands (from Normal Mode)

5.11.1 Beginner

- `p`: (Paste) Paste the last deletion/yank, from Vim only
- `"+p`: (Paste) Paste the last copied text, from any application

5.12 Paste Commands (from Visual Mode)

5.12.1 Beginner

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

5.13 Undo Command (from Normal Mode)

5.13.1 Beginner

- `u`: (Undo) Undo last (atomic) change

5.14 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.

5.14.1 Beginner

- `v`: (Visual) Enter character-wise Visual Mode
- `V`: (Visual) Enter line-wise Visual Mode

5.14.2 Intermediate

- `ctrl-v`: (Visual) Enter block-wise Visual Mode

NOTE: To comment out lines, use block-wise selection with `ctrl-v`, then press `I`, and type the character comment (`//` for example), and hit escape. It can also be used as a poor man's version of a macro (see below). Another way would be to use the Vim Commentary plugin (see below), with the command `gc`.

5.15 Command Mode (from Normal Mode)

5.15.1 Beginner

All of the below can be simplified to just `:w` and `:q`[22], since Vim will warn you if you try to quit with unsaved changes.

- `:w`: (Write) Write the file
- `:q`: (Quit) Quit Vim, without having modified the file
- `:q!`: (Quit!) Quit Vim, throwing away modifications
- `:wq`: (Write-Quit) Write the file, then quit Vim
- `:x`: (Exit) Shorthand for `:wq`

5.15.2 Intermediate

- `:! date`: (`!` is similar to `|`) Run bash command `date` and show the result without adding to file
- `:r! date`: (Read) Run bash command `date` and read in the result into the file
- `:s/foo/bar/g`[23]: (Substitute) Substitute 'foo' with 'bar', globally (ie. each occurrence)

5.16 Command Mode (from Visual Mode)

Visually select text then enter Command Mode using `:`. NOTE: you will see `:'<,'>`[24] instead. This just tells Vim to run the command over the whole selection.

5.16.1 Intermediate

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

[22] See the command mode section above for more detail.

[23] The text `foo` can be either a literal string or a regex, such as `^foo`.

[24] 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."

6 Composability and repeatability

This section should introduce you to even more advanced concepts.

6.1 Text Objects

NOTE: All text objects can be used with delete, yank, copy, etc. "In" deletes the text inside, while "All" deletes quotes, braces, and a single space (so the spaces around it end up balanced).

6.1.1 Beginner

- `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

6.2 Dot (`.`) command

6.2.1 Beginner

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 the dot command.

Expanding on the above, one way[25] to quickly rename variables would be to first search for a variable using `/`, then using `ciw` to change the variable to something else. Finally, repeat this change all throughout the document using `n` to go to the next instance, and `.` to apply the change.

6.3 Number Prefixes

6.3.1 Intermediate

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

6.4 Macros

6.4.1 Intermediate

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

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

Fun fact: you can also define recursive[26] macros[27]. This allows you to create a single macro that runs forever (of course, Vim will stop the macro at the end of the document, for example). An example of this is the following key sequences: `qaqqaV~j@aq@a`[28], which switches the case of every character until the end of the document.

[25] The other way would be to run a search and replace, such as `:s/foo/bar/g`, which would replace all occurrences of `foo` with `bar`.

[26] Here's a quick introduction to recursion. Recursion is defining something in terms of what you are defining. For example, a directory can contain multiple files and multiple directories. A math example would be an equation like `x = 1 + x`, and replacing x on the right with `1 + x` giving us `x = 1 + 1 + x`, and continuing to infinity would give us `x = 1 + 1 + 1 + 1 + ...`. A similar expansion can be carried out on the acronym `GNU`, left as an exercise for the reader.

[27] See Advanced Vim macros.

[28] Usually the `q` register is used for macros, but this can get confusing when first starting out. The command would then be `qqqqqV~j@qq@q`. Also, this macro can be simplified to `ggVG~`.

6.5 Registers

6.5.1 Intermediate

The most important part about registers is that the `"+`[29] prefix is used to store the global clipboard, which can be accessed by any program. Frankly, I don't use any register other than the global one.

Other actions, such as yanks and deletions can be prefixed with a register, for later retrieval.

A useful combination is using registers for editing a macro you wrote. To continue from the macros section, you can write an incorrect macro, paste it into the file, modify it, copy it back to the register, and then run that macro. This seems quite difficult, but there can be really long macros that you would rather go through the above to change a character than to remake the macro from the beginning.

7 Extending Vim for yourself

7.1 Configuration File

To change the default behaviour of Vim, and to keep it even after quitting, you must modify a configuration (also known as a dot file[30]) file for Vim. For GNU/Linux and NeoVim users this file is `~/.config/nvim/init.vim`, for Vim users the file is `~/.vimrc`. If you use Windows, the file will be `_vimrc`[31] (in the home directory in Windows).

For example, typing `:colorscheme elflord`[32] from Normal Mode, will change the colour scheme to elflord for the current Vim session. Once you close Vim this setting will be gone. To save this setting, save the following lines in the configuration file as:

```
" Set colour scheme to elflord
colorscheme elflord
```

Notice the lack of a colon at the beginning of the line. The `"` indicates a line comment.

[29] The `"` is used to retrieve registers, with `+` referring to the name of the register to be accessed, (in this case it is the special "global register").

[30] Because the file starts with a `.`, I know, so original.

[31] Locate Home in Windows.

[32] To find a colour scheme you like from the preinstalled colour schemes, go to List installed colorschemes.

Here are some other settings you may wish to add to your Vim configuration file.[33] In general, you should always copy the comments along with the actual code. (NOTE: always understand what every command does before adding it to your configuration file.)

The below file is also available as `configuration.vim` at Hassan-Shabbir/vim-introduction.

```
" Enable filetype plugins, such as syntax highlighting for files.
filetype plugin indent on

" Enable syntax highlighting.
syntax enable

" Set line numbering to change based on Mode.
" This is especially helpful when moving n lines up.
set number relativenumber

" Set autoread to true. When a file is changed from the outside,
" the file will be reloaded.
set autoread

" With a map leader it's possible to do extra key combinations
" like '<leader>w' saves the current file. 'mapleader' is
" usually the backslash key ('\'), however, below we set it
" to the ',' key, since it is easier to reach.
let mapleader = ","

" This is how you would define "in normal mode, if I press
" the leader key (see above), followed by the 'w' key,
" map it to be the same as writing the file".
nmap <leader>w :w!<cr>

" Set a space of 3 lines between the cursor and the top/bottom
" of the window, making it easier to get the context of the code.
set so=3

" Turn on a completion menu on the bottom. Used when you try to
" tab-complete something in command mode.
```

[33] This is mostly taken from Amix's Vimrc.

```
set wildmenu set wildmode=list:longest,full

" Configure backspace so it acts as it should act. Namely,
" allow backspace to delete new lines, delete past the start
" of insert mode, and delete autoindent.
set backspace=eol,start,indent

" Ignore case when searching, so '/hi' will match 'hi' in the
" text, along with 'Hi'.
set ignorecase

" If a case is used, however, search match using case. So
" '/Hi' will only match 'Hi', and not 'hi', (since we
" explicitly told the case).
set smartcase

" Highlight search results.
set hlsearch

" Make search act like search in modern browsers.
set incsearch

" Don't redraw while executing macros (for performance).
set lazyredraw

" Return to last edit position when opening files
" NOTE: Put this all on one line
au BufReadPost *
  if line("'\"") > 1 && line("'\"") <= line("$")
  | exe "normal! g'\"" | endif
```

7.2 Plugins

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

- 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, (no real reason).

- Vim Sensible: set default settings for Vim

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

- Numbers Vim: add relative line numbers to Vim (great for going n lines up or down)
- Vim Commentary: (un)comment lines of code with a text object
- 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`.

7.2.1 ColorSchemes

- Space Vim Dark
- Solarized

7.2.2 Vim in other places

- Bash/Zsh: Both Bash and Zsh have Vim modes that can be enabled in their respective dot-files
- 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 More Resources

8.1 Vimtutor

The `vimtutor` command can be used from your shell (ie. bash). Once run, `vimtutor` will guide you through some basic Vim commands.

8.2 Vim's `:help` command

A great way to learn more about a particular command in Vim is to use vim's amazing built-in help system using `:help`, followed by the command details (mode, modifier, key, etc.; see `:help`). This is usually recommended for understanding what a command does, since Google is sometimes less helpful. If you only use `:help` and press enter, the introduction to the help system will be shown.

Vim also has the `:helpgrep` command which searches the help system for the searchterm you entered.

8.3 VimCasts.org

There are 76 Vim casts in total. It is highly recommended to start from #1, since the Vim casts get more advanced later on. All Vim casts are highly recommended.

- Vim Casts

The below are paid books from the creator of the Vim casts:

- Practical Vim
- Modern Vim

8.4 Youtube Videos

These are some Youtube videos on Vim. They have been sorted from beginner to advanced. Videos by Luke Smith (in general) and ThoughtBot (on Vim) are recommended.

- Learning Vim in a week
- Why Vim doesn't need the mouse
- Mastering the Vim language
- Vimrc and Vim plug-in overview
- Vim macros: why and how to!
- How Vim makes my daily life easier
- How to do 90% of what plugins do (with just Vim)

8.5 Blog Posts

The following blog post helps to motivate the use of Vim.

- Why, oh WHY, do those #?@! nutheads use vi?

All arabesque posts are recommended, for example the following:

- Unix as IDE
- Advanced Vim macros

8.6 Miscellaneous

The following are fun stories written about a fictitious master and a desciple.

- Vim Koans
- The Dharma of Vi
- Emperor Sh and the traveller

The Vim cheat sheet given above:

- VI/Vim cheat sheet

9 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[34]. To become proficient with Vim, you should use it repeatedly, until the Beginner commands come to you without much thought.

Thanks for reading, and good luck on your journey to become a Vim master!

[34] Such as Vim Tips Wiki.