A Quick-Start Guide to Using Vi(m) Effectively

Jamie Tanna, https://jvt.me

May 16, 2017

Outline I

Preface

The Structure of this Talk

- This talk doesn't expect any Vi(m) knowledge.
- Feel free to ask questions as and when you want!
- This talk is expected to be given in an interactive form, swapping between slides and editor regularly.
- This talk is released as Free Software under the GNU General Public License version 3, and can be found at https://gitlab.com/jamietanna/talks.

How I use Vim

I'm a very heavy Vim user - prioritising Vim for all my development where possible. I have my dotfiles available on GitLab.

I started off by originally using Sublime Text as my main editor, but one day I was watching the brilliant *Destroy All Software* talks, and was absolutely amazed and inspired at the speed Gary was navigating source files at. It was at this point I wanted to learn an editor very well[1].

However, I had heard that Vim had a steep learning curve, and wasn't quite ready for that level of involvement. So instead, I looked into Emacs (controversial, I know!) and for a good couple of months I used it as my daily driver.

---

[1] As well as following the *Pragmatic Programmer*'s advice of "Use a Single Editor Well"

How I use Vim

But there was a big issue - SSHing into servers and working on machines I didn't own/use regularly meant that I had to pull around 160MB of dependencies to get basic text editing. That wasn't ideal!

Therefore I looked into using Vim, and haven't really looked back. I've gotten my configs to a level where I can be productive, and am still constantly learning, a couple of years on.

What is Vi(m)?

Vi

- First released in 1976, by Bill Joy, as a `visual` editor for the line editor `ex`
- Lots of different versions around, but standardised in the Single UNIX Spec and POSIX.

Vim

- Created by Bram Moolenaar (BDFL), first public release in 1991
- `VI iMproved`
- Both CLI and GUI versions

Why Should I Use It?

Modal Editing

Modal editing is the ability to switch between different modes:

- `insert` mode – allow entering of text (`i` at cursor, `a` after cursor)
- `normal` mode – used for navigation and manipulation (default, `ESC`)
- `command` mode – usually denoted by a colon then the command, i.e. `:quit`

Now, this doesn't necessarily sound better, but when you get used to it, it makes a lot more sense, especially as it means that you can keep using your keyboard for many different things.

Reduction of Keystrokes

As a person who constantly wants to improve my workflow, and reduce the number of tasks I have to complete a given project, Vim is great for me. I'm constantly learning new ways to optimise the number of keystrokes that are required to perform given steps.

Availability

As mentioned, Vi is going to be available on any POSIX system you can touch – so if you are ever going to be working on a POSIX machine, you can rest assured you'll be able to run Vi.

This means that you should be prepared to make simple edits for when you need to, as well as potentially even more involved changes and development work.

How Do I Perform Basic Tasks?

Movement

- `h` – ←
- `j` – ↓
- `k` – ↑
- `l` – →
- Why not just use arrow keys?
  - "It's not the Vim way"
  - Movement away from the "home row"[2] is the exact opposite of what you want to be doing in Vim

[2] i.e. the middle row, starting `asdf`

Searching for Content

- In order to search for a given string, you can use `/term` from `normal` mode
- To search backwards, you can use `?term` from `normal` mode
- Once searching, you may want to use `n` to go to the next result and `N` to go to the previous result

Changing Text

- change `<motion>`
- delete `<motion>`
- replace `<motion>`

Copy + Paste

- yank
- paste after the cursor
- Paste before the cursor

Undo/Redo

Sometimes you'll make changes you didn't mean to. You can use `u` to undo the change, and `C-r` to redo it.

Quitting/Saving Files

- `:q(uit)` - but stop us if we have unsaved content
- `:w(rite)`
- `:wq` – combination of write and quit
- `:q(uit)!` – quit, don't care about any changes we have
- `:w(rite)!` – write, useful to overwrite a file if it's set as readonly

Wait, Why is Everything Done in a Really Weird Way?

It's a Grammar and Vocabulary

Vim's actions and keys are like a language – for instance, you can `change a word` (which is done by the keypresses `caw`) or `delete inside the (...)`.

Operators

For instance, there are many actions that can occur within this vocabulary, such as:

Operator	Meaning
`c`	Change
`d`	Delete
`r`	Replace
`y`	Yank
`~`	Swap Case
`gq`	Perform Text Formatting

Motions

Motion	Meaning
`h`	move $\leftarrow$[3]
`j`	move $\downarrow$
`k`	move $\uparrow$
`l`	move $\rightarrow$
`w`	move to the next word
`b`	move to the beginning of the current word
`e`	move to the end of the current word
`4j`	four lines down

[3] Note that these movements are inclusive; this will i.e. delete the character under the cursor, and the character to the left

Motions

Additionally, some Vim-only text objects:

Motion	Meaning
`aw`	“a word”
`iw`	“inside word”
`as`	“a sentence”
`ap`	“a paragraph”

Putting it Together

Keypress	Action
`caw`	change “a word”
`diw`	delete “inside a word”
`yas`	yank/copy “a sentence”
`bw`	backwards a word

Conventions

Some conventions:

- Repeated actions, such as deleting a line, can be done with `dd`.
- Capitalisation often denotes “until the end of line”; Deleting to the end of a line can be done with `D` (or normally as `d$`

Composability

As mentioned above, it's all about having a language you can use. The composability comes when you can do things like:

- `dap`
- `ci"`
- `ct>`
- `=%`
- `5dw`

Tips and Tricks

Write to a File As Root

- Have you ever forgotten to `sudo vim ...`?
- You can avoid this by running `:w sudo tee % >/dev/null`
  - `:write`
  - into a process call
  - `sudo tee` run the `tee` command as root
  - `%` with the current filename as an argument
  - `>/dev/null` redirect output (via `tee`) away, so we don't have to see it

Word Wrapping

If you wish to have your text formatted within the right character limit, for instance, in a commit message, you can use gq.
It converts:

```
Make sure that we rewrite any Git URLs such that we're always going to be able
     to ‘git push‘ via SSH. This means that we can clone over HTTPS, but push
     through SSH.
```

To:

```
Make sure that we rewrite any Git URLs such that we're always going to
be able to ‘git push‘ via SSH. This means that we can clone over HTTPS,
but push through SSH.
```

How Can I Best Get Used To It?

Phasing it in

In order to get some exposure to something like Vi(m), it's easiest by actually using it. However, by switching to solely Vim, you'll hurt your own productivity.
The best way to get going is to start slowly introducing it into your workflow with little bits here and there.

Use it for Writing Commit Messages

By default, your commit message is through Vi(m). Therefore, instead of running `git commit -m "..."`, just run `git commit`. This will get you using it for basic text editing, and will start you off in a nice way, without having to take the plunge to learning it for your main editor.

`vimtutor`

`vimtutor` is a great tutorial that runs you through all the basics for Vim, such as those listed here, in a much more interactive way.

Learn X In Y Minutes

A great resource that can be used as a quick reference/cheat sheet - https://learnxinyminutes.com/docs/vim/

Installing in your Preferred Editor

As mentioned, there are plugins that can be installed in your existing editor - some of which can be found at:

- IntelliJ IdeaVim
- Atom vim-mode
- Emacs Evil mode