Vim Lab

Ethan Wong

Linux Users Group @ UIC

October 13, 2023

Table of Contents

Why Vim?

- Simple and ubiquitous text editor
- Usable in a *terminal*
- Keyboard-centric, meaning that a mouse is optional
- Vim's version of modal input is popular and commonly emulated
- Part of the POSIX standard!

Table of Contents

History

- Vi(no -m) was created by Bill Joy in **1976** for the Second BSD.[1]
- It is the visual mode for the `ex` command line text editor.
- Vi was created for remote editing over a *300* baud modem, which transmitted text slower than you can read!

Figure: Bill Joy, the creator of `vi`

[1] `https://web.archive.org/web/20060701083055/http://web.cecs.pdx.edu/~kirkenda/joy84.html`

History

- Vi iMproved (`vim`) was created by Bram Moolenaar in 1988 as a *clone* of `vi`.
- `vim` is a superset of `vi`, introducing new features such as syntax highlighting, undo/redo, screen splitting, and plugin support.
- Like `vi`, `vim` has also been ported to a wide range of OS's.
- Today, `vim` and its derivatives make up one of the most most popular text editor families.[2]

[2] `https://survey.stackoverflow.co/2023/#section-most-popular-technologies-integrated-development-environment`

Table of Contents

Goals

We will try to learn:

- How to switch between the different *modes* of `vim`
- How to navigate around a text document using **modal** keybindings
- Methods for remembering the the mnemonics
- How to select, manipulate, and insert text into files
- How to quit `vim` without restarting your computer...[3]

[3] a.k.a. commands

Modes

`vim` is known for a **modal** editing scheme.
What is modal editing?
In `vim`, various things that you typically do while editing text is split into different *modes*.

Mode	Purpose	How to Enter
Normal	Move around and manipulate *existing* text	Default, can return to by pressing Esc in other modes
Insert	Allows you to type text like in any standard editor	i, a, x, c, o, among others in Normal mode
Visual	Allows you to select regions of text as if you were using a mouse	v, ⇧+v, ctrl+v in Normal mode
Command	Allows you to execute commands like save and exit	: in Normal mode

There are other additional modes that `vim` has out-of-the-box, but these are the basics.
Why should editing be split into modes?

Keybinds

Modes allow for `vim` perform actions using simple and easy to remember keystroke sequences! These are known as motions!

Example

Navigating can be done with [h], [j], [k], [l] in Normal mode. These corrispond to [←], [↓], [↑], [→] respectively.[a]
This allows you to keep your hands on the home-row of your keyboard (especially good if you are a touch-typist)!

[a] People consider using the arrow keys in `vim` a cardinal sin.

Example 2

Additionally, an arbitrary number can be prefixed to modify motions.
One usecase is [5]+[j], which moves down 5 lines in one input sequence.

Keybinds

How can you easily remember these keybinds?
Most keybinds are **mnemonics** for their corrisponding action.

- [i]nsert enters Insert mode
- [a]ppend is the same as insert, but starts inserting at the character after the cursor
- [x]-out deletes a single character
- [d]elete takes your selection and deletes it
- [c]hange deletes your selection and enters Insert mode
- etc.

There are a billion keybinds in `vim`, so don't fret about learning all of them at once. Learning as you go is the method here!

Keybinds

`vim` motions are extremely powerful. They let you condense the work of selecting text and changing it into easy to remember input sequences!

Example 1

cc lets you delete a line and start retyping text in one shot (c hange a line)! Double pressing most actions will allow you to apply it to the line your cursor is on.

Before

```
This is a sentence!
The cursor is below this line.
_Emacs users are stinky :(
another wacky line!
```

After

```
This is a sentence!
The cursor is below this line.
|
another wacky line!
```

Keybinds

Example 2

dw Lets you d elete a w ord.
The w works to specify your selection after any action! So, this will work with c too!

Before

```
This is a sentence!
The cursor is below this line.
_Emacs users are stinky :(
another wacky line!
```

After

```
This is a sentence!
The cursor is below this line.
_users are stinky :(
another wacky line!
```

Keybinds

Example 3

Commands can be chained too! `2k` and `dip` Lets you move `2` up (`k`) and `d`elete the `i`nner `p`aragraph.[4]

Before

```
This is a sentence!
The cursor is below this line.
_Emacs users are stinky :(
another wacky line!
```

After

```
_
```

[4] inner meaning excluding any spacing surrounding the paragraph

Keybinds

Other useful actions include (but not all)...

- o and O which creates a new line below/above the current line respectively
- gg and G which puts the cursor at the top/bottom of the file
- zz which centers the editing window
- b and e goes to the b eginning and e nd of the word your cursor is currently on.
- y to copy (d cuts) and p to paste
- u and ctrl + r to undo and redo.
- ⟨⟨ and ⟩⟩ to change indentation levels in code.

Table of Contents

Commands

There are lots of commands in `vim`, just like actions, but we'll just go over a few. To enter Command mode, just prepend a [:] before typing it!

- `:write` or `:w` saves your file[5]
- `:quit` or `:q` quits Vim[6]
- `:%s/<regex>/<regex>/gc` lets you find and replace
- `:help <command>` brings up the help-page for any Vim command

You can also chain commands, so `:wq` for example saves your file and exits in one shot.

Also, [/] and [?] let you search within your document.[7] [n] and [N] let you go backwards/forwards through your search results.

---

[5] you can specify a filename afterwards to emulate [File] [Save As]

[6] add an `!` to quit and ignore unsaved changes

[7] use `:noh` to clear highlighted results after

Table of Contents

Lets try a hands-on demo!

Firstly, open a terminal!

- Windows
  Open one of the following:
  - Powershell
  - PuTTy

  You'll need to `ssh` into `malware.cs.uic.edu`.
  Syntax: `ssh userXX@malware.cs.uic.edu`, where *XX* is a number between 1-25.
- MacOS
  Simply go to Applications > Utilities > Terminal and type `vim`, its included by default!
- Linux
  Open your terminal of choice, install `vim` if you need to (tho it should be included by default...), and run `vim`.

Crash Course

Challenge!

I want you to create a simple C program using `vim` and sucessfully compile it! It can be as simple as "Hello, World!" if you'd like!

```c
#include <stdio.h>

int main(int argc, char **argv) {
        printf("Hello World!\n");
}
```

If thats too easy for you, try these extra challenges:

- Don't use the arrow keys at all
- Write your program in Rust instead
- Do not leave `vim` at any point when writing the program

Crash Course

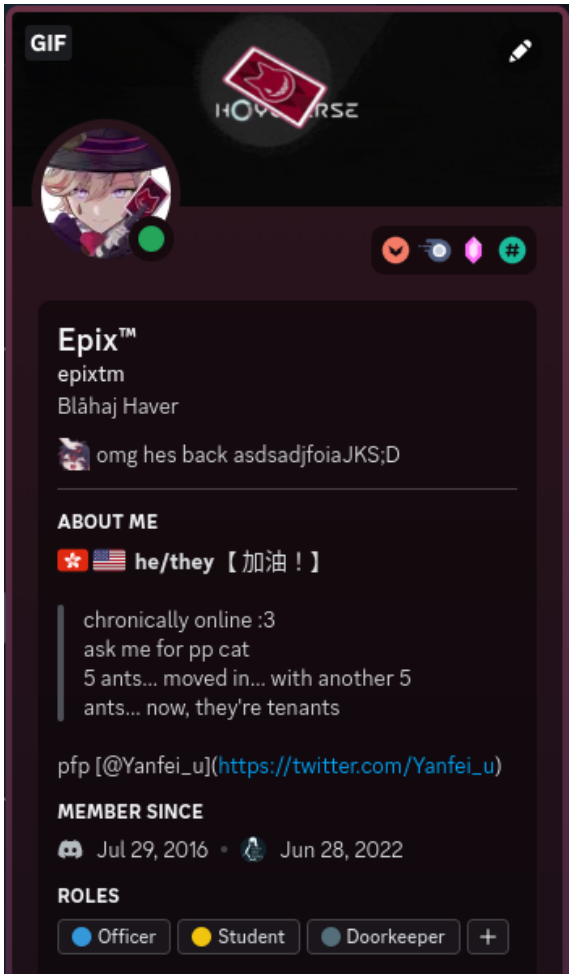
Closing Remarks

- This was a very surface-level introduction to `vim`.
  Some features not covered include:
  - Vim Modelines
  - Multiplexing buffers
  - Macros
  - Plugins
- Don't fret if this is alot of information at once! It's important to start small and learn what you need as you go.
- If you want to practice more with `vim`, try out `vimtutor` which is a guided tutorial to learn Vim.
- Don't be afraid to reference a cheatsheet like `https://vim.rtorr.com/` when learning!
- As always, practice makes perfect.

Thank you!

Crash Course

Me :3



The information in this presentation will be made available[8] on our website!
`https://lug.cs.uic.edu`

Join our Discord!

Figure:
`https://discord.gg/NgxTR7PX5e`

[8] sooner or later