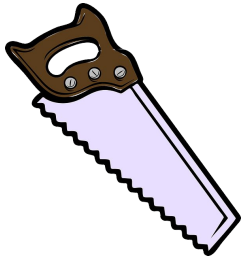
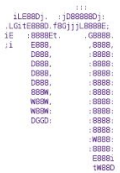
Vim

Adarsh Pyarelal

May 28, 2020

IVILab Summer Bootcamp 2020

Motivation

	Regular tools	Power tools
Saws		
Text editors	  	 

Motivation

- A capable text editor might mean different things in different contexts. E.g.
  - Emacs for LISP devs
  - IntelliJ for Java and Scala devs
- Reasons for learning vi:
  - Available everywhere, no need for GUI (I was forced to learn Vim to work with UA HPC)
  - Lightweight compared to IDEs
  - The vi philosophy of modal text editing transcends editors - vi keybindings are available for Emacs (evil-mode) and IDEs (from experience, IntelliJ IDEs and Jupyter both support vim bindings), so an investment in learning vi will pay off no matter what.
  - You will become *way* faster at writing and editing code (this includes LaTeX) - at least 10X as fast, if not more.
- Initial learning curve is steep, but pays off handsomely in the long run

Outline

- Motivation
- Modal editing
- Vim grammar
- Navigation
- Editing & working with text objects
- Customizing Vim (.vimrc)
- Plugins

For the next three weeks, you will use no other editor but Vim*

Modal editing

Vim has different ‘modes’:

- Normal (you’ll spend most of your time in this one)
- Visual
- Insert
- Replace
- Command

Vim grammar

- Instead of memorizing individual commands, learn the 'grammar' of vim, i.e. how to 'speak' to it.
- Vim 'sentences' generally follow the pattern below:

```
Editing: [N] [Verb] [Modifier] [Motion/Character/Object]
```

- `N` : A positive integer.
- The square brackets are there because:
  - Not all sentences will have all the components, and
  - Some values for the components are incompatible with other components. E.g. Some verbs ignore multipliers.

Reference: https://benmccormick.org/2014/07/02/learning-vim-in-2014-vim-as-language

Navigation | [N] [Verb] [Modifier] [Motion/Character/Object]

- Motion (these motions ignore multipliers)
  - 0: Go to the beginning of the line
  - $: Go to the end of the line
  - gg: Go to the top of the document
  - G: Go to the end of the document
- [Multiplier] Motion
  - h/j/k/l: Go one character left/down/up/right
  - w: Go to the beginning of the next word
  - e/ge: Go to the next/previous character that ends a word
  - b: Go to the previous character that begins a word
  - W, E/gE, B: Same as their lowercase counterparts, but only treats whitespace as word separators (the others treat '.', '-', ')', etc. as word separators)
  - Example:
    - 3h: move three characters to the left

Navigation II `[N] [Verb] [Modifier] [Motion/Character/Object]`

- `[Multiplier] Modifier Character`
  - The modifiers that can be used with this type of navigation pattern are:
    - `f/F` (find) - Go to the first occurrence to the right/left of the given character on the current line to the
    - `t/T` (till) - Move the cursor just left of the first occurrence to the right/left of the given character
  - Examples:
    - `3fx` - Move cursor to the 3rd occurrence of the letter 'x' to the right on the current line.
- `[Line number] Motion`
  - `Ngg/NG` - all these commands move the cursor to the *N*th line in the file (as does the command `:N`)

Editing I `[N] [Verb] [Modifier] [Motion/Character/Object]`

- `Verb` (these ones ignore multipliers)
  - Insert
    - `i`: Enter insert mode one character to the left.
    - `I`: Enter insert mode at the beginning of the current line (equivalent to `0i`)
  - Append
    - `a`: Enter insert mode one character to the right.
    - `A`: Enter insert mode at the end of the line (equivalent to `$a`)
  - Open
    - `o/O`: Add a blank line below/above the current one and enter insert mode
  - `R`: enter replace mode
  - `D`: Delete everything to the right of the cursor on the current line (equivalent to `d$`)
  - `C`: Delete everything to the right of the cursor on the current line and enter insert mode (equivalent to `c$`)
- `[Multiplier] Verb`
  - `p/P`: paste text before/after the cursor
  - `x/X`: delete the current character/the character just to the left
  - `r`: Enter replace mode for a single character (multiple if a multiplier is given)
  - `u`: Undo last action
  - `.` : repeat last action

Interlude: text objects

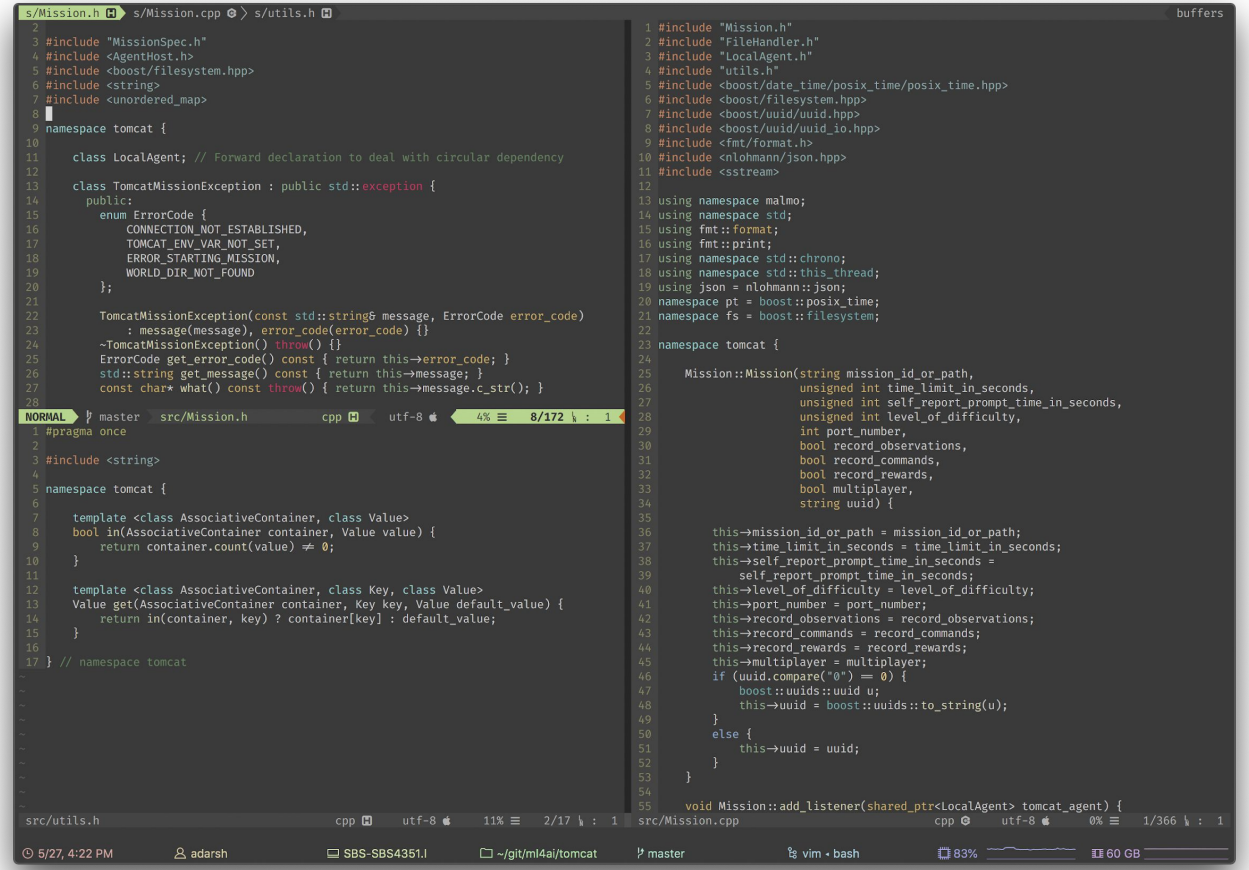
- IMHO the most ‘bang-for-the-buck’ thing to learn in Vim.
- There are a few built-in text objects:
  - Word (w)
  - Sentence (s)
  - Paragraph (p)
  - Selection inside/around parentheses, braces, quotes, etc.
  - Lines are treated specially - to apply an action to a line, typically you would just repeat the verb (e.g. yy, cc, dd)
- Additional ones are available via plug-ins (e.g. LaTeX environments)

Editing II `[N] [Verb] [Modifier] [Motion/Character/Object]`

- `[N] Verb [Modifier] Motion/Object`
  - The verbs that can be used with this pattern are:
    - `y`: (yank) - copy text
    - `d`: (delete) - cut text
    - `c`: (change) - cut text and enter insert mode.
    - `gq`: Format text
    - `v`: visual selection
  - A `Modifier` can be one of:
    - A positive integer N (for repeating)
    - `t/T/f/F` (see slide 9)
    - `i/a` - For use with text objects. You can think of them as 'in' and 'a' (or 'around')
  - Examples:
    - `2yw`: Yank all the text till the beginning of the next word, twice - i.e., yank the next two words.
    - `2y3w`: Yank three words twice - i.e. yank the next six words.
    - `2ctf`: Cut all text between the current cursor position and the second occurrence of the character 'f' to the right of the cursor.
    - `ci"`: Delete all text inside the double quotes (assuming the cursor is between the quotes) and enter insert mode
    - `ca(`: Delete all text inside the parentheses as well as the parentheses themselves, and enter insert mode

Splits

- Vertical split - `:vsp`
- Horizontal split - `:sp`
- Split navigation: `<Ctrl>-w + h/j/k/l`
- Making splits equally sized: `<Ctrl>-w=`



.vimrc

...or, why does my vim window not look like the screenshots here???!

The file ~/.vimrc holds the customization settings for vim. Here's mine:
https://github.com/adarshp/dotfiles/blob/master/.vimrc

Feel free to copy over the whole thing into your .vimrc or just the parts you like.

Eventually, you should probably know what the different lines in the vimrc mean :)

Plugins

- You'll probably want a plugin manager for Vim - plugins can provide powerful functionality.
- I use https://www.github.com/junegunn/vim-plug - the first few lines of my ~/.vimrc (see slide 15) - automatically install vim-plug if it's not already installed

Things you probably will want to know about later

- Screen navigation (`<Ctrl>-f, <Ctrl-b>`)
- Tabs (:tabe, gt)
- Macros
- Registers
- ctags