Intro to Vim

Hackers @ Berkeley

Pandu Rendradjaja
pandu@berkeley.edu

About the workshop

- Part I: Interactive intro like ‘vimtutor’
  - Learn some basics and get a feel for the main ideas of Vim
- Part II: Some cool/useful things, resources

About the gray slides

- The white slides were for the workshop, but they’re not completely self-explanatory.
- These gray slides (not used in the workshop) are supposed to fill in the blanks and mention some things that I didn’t write in the white slides.

http://tinyurl.com/hellovim

What is Vim?

- Obviously, a text editor. What makes it different?
  - A second-to-none keyboard-based *language* for manipulating and navigating text.
    - *Very fast.*
    - Mouse optional.
  - Extremely extensible and configurable; has a large community with plugins, etc. for every use imaginable. (See also: Emacs.)
  - A practical note: Vi(m) is everywhere. A bit of familiarity comes in handy surprisingly often.

Vi(m) is OLD!

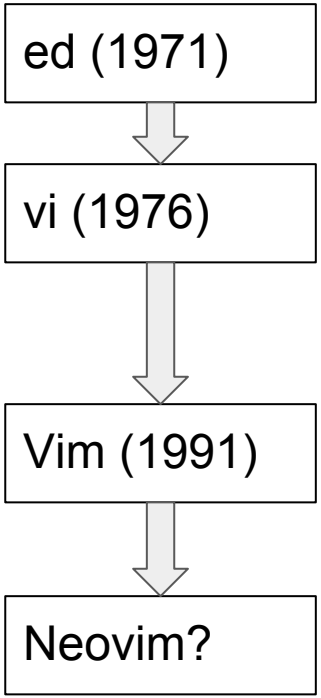
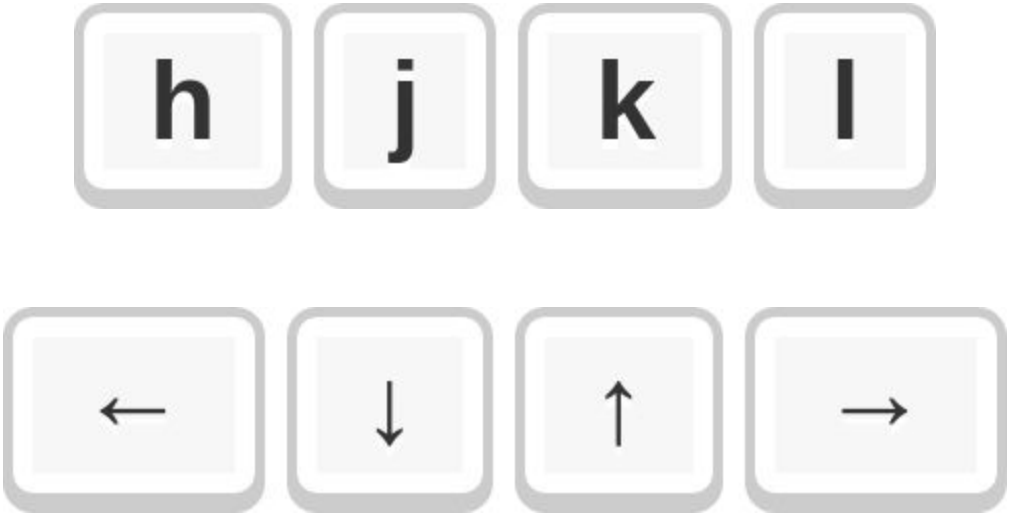


Image shamelessly lifted from: http://hackaday.com/2014/02/08/raspi-powered-adm-3a-dumb-terminal/

http://tinyurl.com/hellovim

Getting started

- Go to http://tinyurl.com/hellovim. Install Vim if you need to, and download the text file there as well.
- This text file will contain some interactive exercises.
  - If you've done the `vimtutor`, it's in much the same style, (and does overlap in content a bit) but, unlike the `vimtutor`, it's not self-explanatory.
  - Refer to these slides for instructions on the exercises.
- Open it up in Vim, then go on.


h
j
k
l
←
↓
↑
→

Instructions for exercise 1

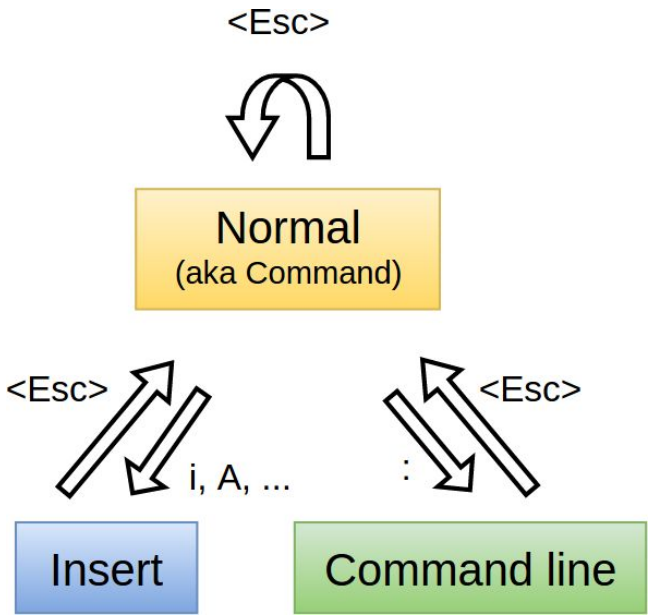
- Use the arrow keys or hjkl to move your cursor to below the “Welcome to Intro to Vim!” line.
- Then press i, type “Hello world!”, and then press the Escape key.

Hello, world!

What was that all about?

What was that all about?

- Vim is a **modal** editor.
  - Three main modes.
  - Not as confusing as it seems!
  - Save yourself some headaches: “default” to Normal.



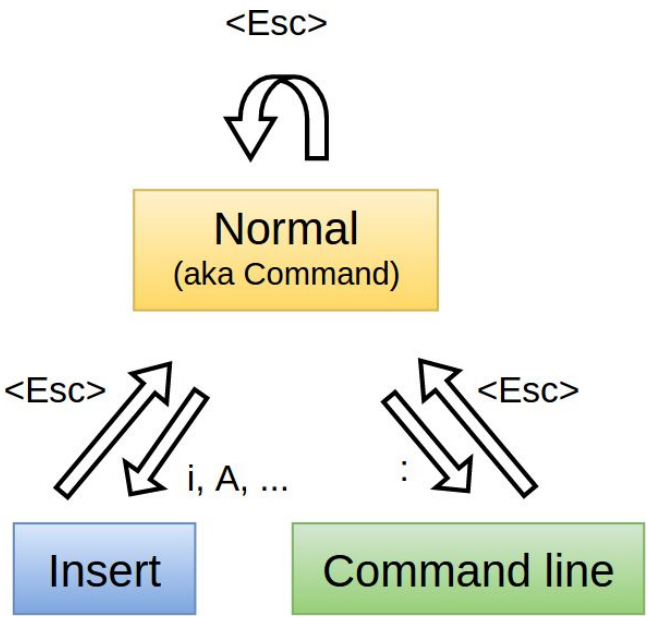
set nocompatible

Colon commands

```
:set nocompatible
```

- Change settings
- Save, open and close files
- Compile
- Search and replace
- Access the help
- And so on...

- Turns off backwards-compatibility with vi: saner defaults
- Affects many other settings, e.g.:
  - Showing current mode
  - Multiple undo!
  - And others...


<Esc>
Normal
(aka Command)
<Esc>
i, A, ...
<Esc>
:
Insert
Command line


A


X

Commands for exercises 2-5

- `i` Enter Insert mode before the cursor
- `<Esc>` Return to Normal mode (<Esc> is Vim notation for the Escape key.)
- `A` Enter Insert mode at the end of the current line
- `u` Undo
- `x` Delete the character at the cursor
- `dw` Delete a word
- `d%` Delete a pair of parens/brackets/etc. (and what's in between)

Instructions for exercises 2-5

- #1: Use the arrow keys or hjkl to move your cursor to below the “Welcome to Intro to Vim!” line. Then press i, type “Hello world!”, and press Escape.
- #2: Add the missing punctuation! Notice that you’ll need to use A instead of i in order to get a question mark at the end of the line.
- #3: Delete the extra characters with x.
- #4: Delete the extra word with dw.
- #5: Delete the parenthesized text by moving your cursor to one of the parentheses and pressing d%.


d
w


d
%

w

to next word

delete to next word

dw

delete to next word

operator motion

d%

delete to matching paren

operator motion

Notes on the previous slides

- **This is probably the most important part of the workshop.**
- The “language” of Vim consists primarily of *operators* and *motions*.
- Motions are commands by themselves, and can be used to navigate around a file.
- Operators must be used together with motions (or, as we’ll see later, text objects) to form “sentences.”
- Think of operators as “verbs,” and motions as… well, not nouns, but “destinations.” (If you want, you can call them prepositional phrases. “To the next word,” etc.)

This is REALLY cool.

Easy-to-remember commands

Other editors

Home
End
Ctrl-Left
Ctrl-Right
Ctrl-Shift-K
Ctrl-T
Ctrl-Del
Ctrl-Backspace
Ctrl-K K
Ctrl-K Backspace
Ctrl-Shift-D

Easy-to-remember commands

Other editors

Home
End
Ctrl-Left
Ctrl-Right
Ctrl-Shift-K
Ctrl-T
Ctrl-Del
Ctrl-Backspace
Ctrl-K K
Ctrl-K Backspace
Ctrl-Shift-D

Vim

	d	y	<	>
w				
b				
0				
$				
/				
{				
}				
%				

Easy-to-remember commands

Other editors

Home
End
Ctrl-Left
Ctrl-Right
Ctrl-Shift-K
Ctrl-T
Ctrl-Del
Ctrl-Backspace
Ctrl-K K
Ctrl-K Backspace
Ctrl-Shift-D

Vim

	d	y	<	>
w	**dw**	**yw**	**<w**	**>w**
b	**db**	**yb**	**<b**	**>b**
0	**d0**	**y0**	**<0**	**>0**
$	**d$**	**y$**	**<$**	**>$**
/	**d/**	**y/**	**</**	**>/**
{	**d{**	**y{**	**<{**	**>{**
}	**d}**	**y}**	**<}**	**>}**
%	**d%**	**y%**	**<%**	**>%**

Easy-to-remember commands

Other editors

Home
End
Ctrl-Left
Ctrl-Right
Ctrl-Shift-K
Ctrl-T
Ctrl-Del
Ctrl-Backspace
Ctrl-K K
Ctrl-K Backspace
Ctrl-Shift-D

Vim

	d	y	<	>
w	**dw**	**yw**	**<w**	**>w**
b	**db**	**yb**	**<b**	**>b**
0	**d0**	**y0**	**<0**	**>0**
$	**d$**	**y$**	**<$**	**>$**
/	**d/**	**y/**	**</**	**>/**
{	**d{**	**y{**	**<{**	**>{**
}	**d}**	**y}**	**<}**	**>}**
%	**d%**	**y%**	**<%**	**>%**

Notes on the previous slides

- You might have heard that Vim requires a lot of memorization.
- That’s probably true, but in order to use any other editor effectively, you’ll need to memorize some stuff, too.
- One of the cool things about Vim, though, is it makes your memorization more efficient:
  - In another text editor, **if you memorize 11 things, you get 11 commands** you can use.
  - In Vim, if you memorize 11 things, (say, 4 operators and 7 motions) you get 28 sentences on top of the 7 motions by themselves: **35 commands!**

Instructions for exercises 6-7

- #6: Experiment with the operators and motions from the next two slides! Don’t feel like you need to memorize these right away. Vim has a lot of “vocabulary,” and you’ll learn it as you go.
- #7: Change the word “because” to “therefore,” fill in the missing code, and copy and paste the long word.

Operators for exercises 6-7

- `d` Delete
- `c` Change (= delete + insert)
- `y` Yank

Yank is called “copy” in modern editors. Use p to “put,” which is Vim’s term for “paste.”

This is not to be confused with “yank” in Emacs, which is like “put”/”paste.”

By the way, “delete” actually corresponds better to modern “cut,” because you can also use p to paste deleted text.

Motions for exercises 6-7

- `w e b` Move by words
- `W E B` Move by WORDs (sequences of non-whitespace characters)
- `( )` Move by sentences
- `{ }` Move by paragraphs
- `0 $` To beginning/end of line
- `%` To matching paren, etc.


d
i
'

Text objects

Instructions for exercises 8-10

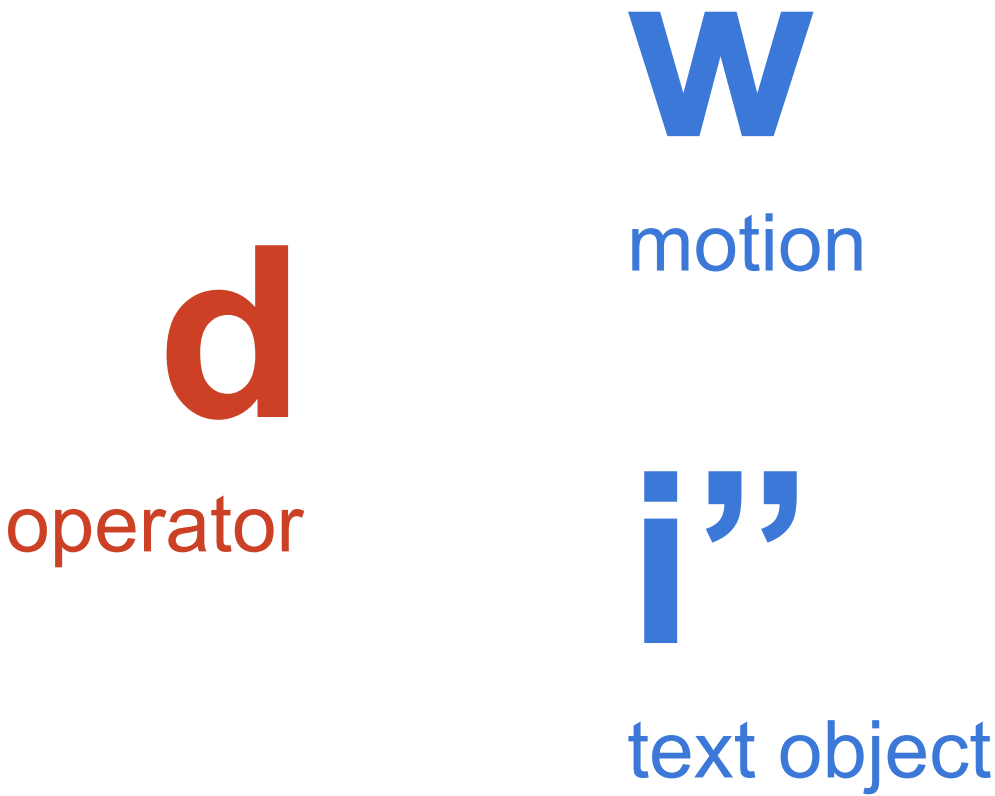
- #8
  - Move your cursor inside the single quotes, then type: `di'`
  - This command means “**delete** whatever is **inside** these **quotes**.”
  - Experiment with other operators, as well as with `(`, `[`, and `{`!
  - **This general concept is called a “text object.” Just like we can follow an operator with a motion, we can also follow it with a text object. (illustrated in the next slides)**
- #9
  - Try the commands indicated. Think of the “**a** versions” as **around**.
- #10
  - `dap`, `das`, and `daw` can delete “around” paragraphs, sentences, and words.

w

motion

d

operator


w
motion
d
operator
i”
text object

d

operator

w

motion (“where to”)

i”

text object (“what”)

Macros

start recording

(do any Vim commands here)

stop recording

play macro

Instructions for exercise 11

- As illustrated in the previous slides, you can use `q` and `@` to record and play back a macro. (The letter `x` can be replaced with any letter. This is the “name” of the macro.)
- This exercise contains some ASCII art, but it’s a little messed up. If you make the dots on the left line up, the picture will become clear!
- Go to the first line of the picture. Make sure your cursor is on the far left -- the beginning of the line.
    - You can press `0` to move to the beginning of the line.

Instructions for exercise 11

- Start recording: `qx`
- Delete up to the the dot: `dt.`
- Go to the beginning of the next line:`j0`
- Stop recording: `q`
- Play back the macro (do this multiple times as necessary): `@q`

Macros are really handy. Usually, I don't care about the name of the macro, so I'll just use `qq` because it's easy to type.

And since I'm really lazy, I've got a mapping for playing back the `q` macro just by pressing the Delete key:

```
:nnoremap <Del> @q
```

Part II

The rest of the workshop

- Practical things:
  - Save and quit
  - Configuration
- Demo of some other features
- Where to go from here

Summary of the rest of the workshop

- Save and quit:
  - :w saves the current file
  - :w FILENAME saves with a specific filename
  - :q quits
  - :q! force-quits, discarding unsaved changes
- Settings (like :set nocompatible) are lost when you quit Vim. To make them permanent, use a vimrc file.
- See “vimrc things” under http://tinyurl.com/hellovim for a bit of information about this.
- **See “Learning resources” under http://tinyurl.com/hellovim to find out where you can go to learn more Vim!**

Save and quit

vimrc (configuration)

(demo)

What now?