

# Vim Workshop

## Callum Howard

```
1 // transform.h
2 // Callum Howard 2015
3
4 #define FALSE 0
5 #define TRUE 1
6
7 void swap (char *a, char *b);
8 void transpose (char a[], int size);
9 int arrayCompare (char arrayA[], char arrayB[], int size);
10 int getStrictRoot (int input);
11 int coord (int x, int y, int size);
12
13 void flipH (char a[], int size) {
14     int dim = getStrictRoot(size);
15     int i = 0;
16     int j = 0;
17
18     // starting from outermost columns and moving in...
19     while (i < j) {
20         // swap columns
21         for (int k=0; k<dim; k++) {
22             swap(&a[coord(k, i, size)], &a[coord(k, j, size)]);
23         }
24         i++;
25         j--;
26     }
27 }
28
29 int coord (int y, int x, int size) {
30     int width = getStrictRoot(size);
31     return ((y * width) + x);
32 }
33
34 // gets square root and checks input is a square number
35 int getStrictRoot (int input) {
36     int root = sqrt(input);
37     // check if output is integer
38     assert(root*root == input);
39     return root;
40 }
```

```
1 // testTransform.c
2 // Callum Howard 2015
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <math.h>
7 #include <assert.h>
8 #include "transform.h"
9
10 void testSwap ();
11 void testTranspose ();
12 void testTranspose2 ();
13 void testTranspose3 ();
14 void testGetStrictRoot ();
15 void testCoord ();
16
17 int main (int argc, char *argv[]) {
18     testSwap();
19     testTranspose();
20     testTranspose2();
21     testTranspose3();
22     testGetStrictRoot();
23     testCoord();
24
25     return EXIT_SUCCESS;
26 }
27
28 void testSwap () {
29 }
30
31 void testTranspose () {
32 }
33
34 void testTranspose2 () {
35 }
36
37 void testTranspose3 () {
38 }
39
40 void testGetStrictRoot () {
41 }
42
43 void testCoord () {
44     assert(coord(2, 2, 9) == 8);
45     assert(coord(1, 2, 9) == 5);
46 }
```

NERD 1:testTransform.c 3:transform.c 4:transform.h cpp 100% : 11: 35 NORMAL +0 ~4 -0 master <rm.c 4:transform.h flipH() < c 71% : 50: 53 1:testTransform.c 3:transform.c 4:transform.h c 11% : 15: 0 Tagbar Name transform.h

```
# callumhoward at MacBook-Pro in ~/UNSW3969/COMP3411/naughtsAndCrosses on git:master x
$ gcc -Wall -Werror -O0 -o testTransform transform.c testTransform.c

# callumhoward at MacBook-Pro in ~/UNSW3969/COMP3411/naughtsAndCrosses on git:master x
$ ./testTransform
1 4 7 2 5 8 3 6 9
0 1 4 7 2 5 8 3 6 9
0 1 1 1 0 1 1 0 0 1

# callumhoward at MacBook-Pro in ~/UNSW3969/COMP3411/naughtsAndCrosses on git:master x
$
```



# What is a Text Editor?

# What is a Text Editor

On lab computers  
(available in exams)



Nano



Gedit



Kwrite



Vim



Emacs

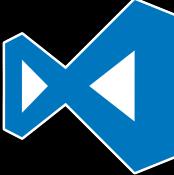
Other popular editors



Sublime Text 3



Atom.io



VS Code



Notepad++



(Notepad)



(TextEdit)

Integrated Development  
Environment (IDE)



Xcode



Visual Studio



Eclipse



Netbeans



Geany

OSX Windows

Windows

# What is a Text Editor

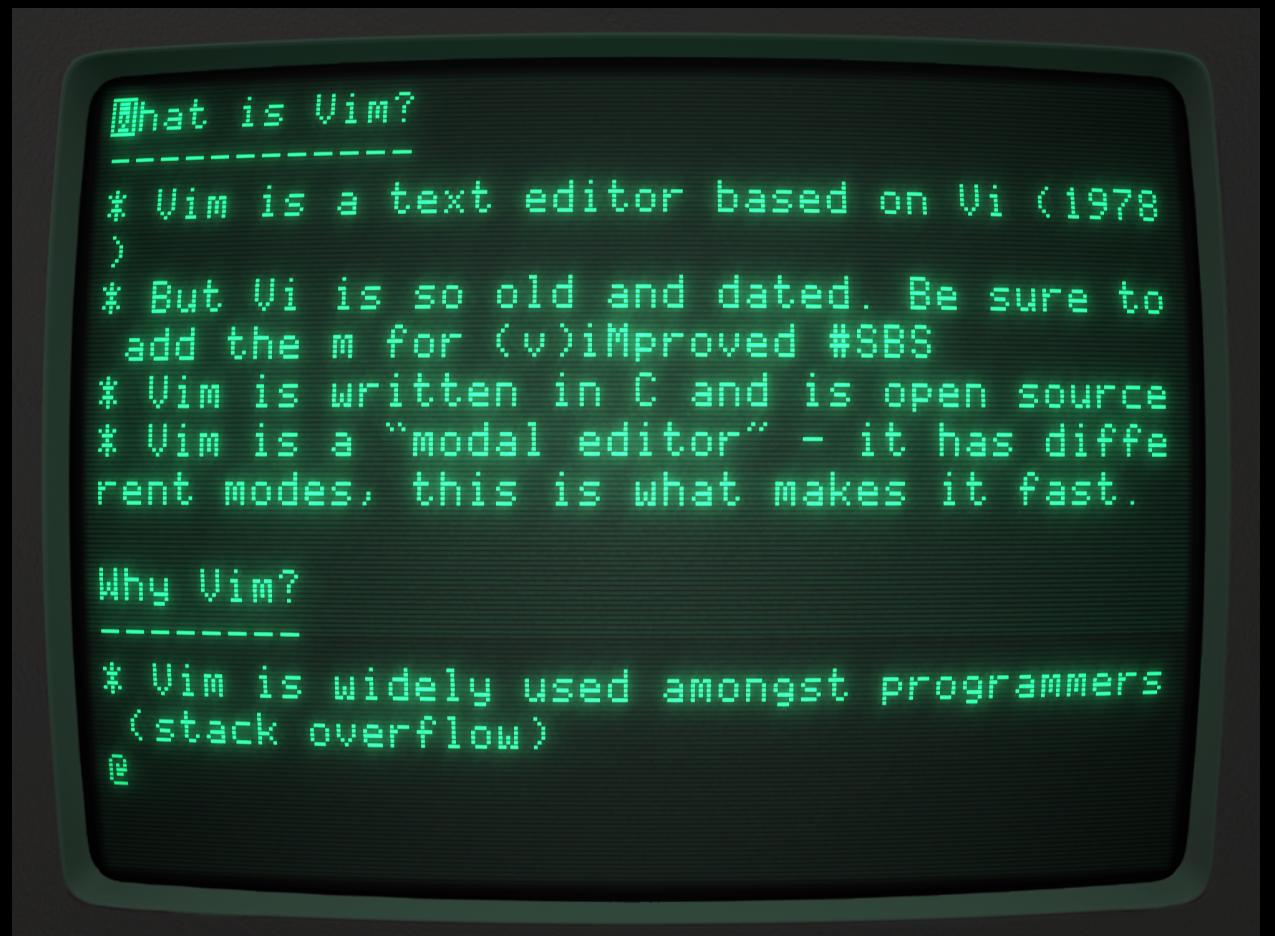
Not Text Editors:





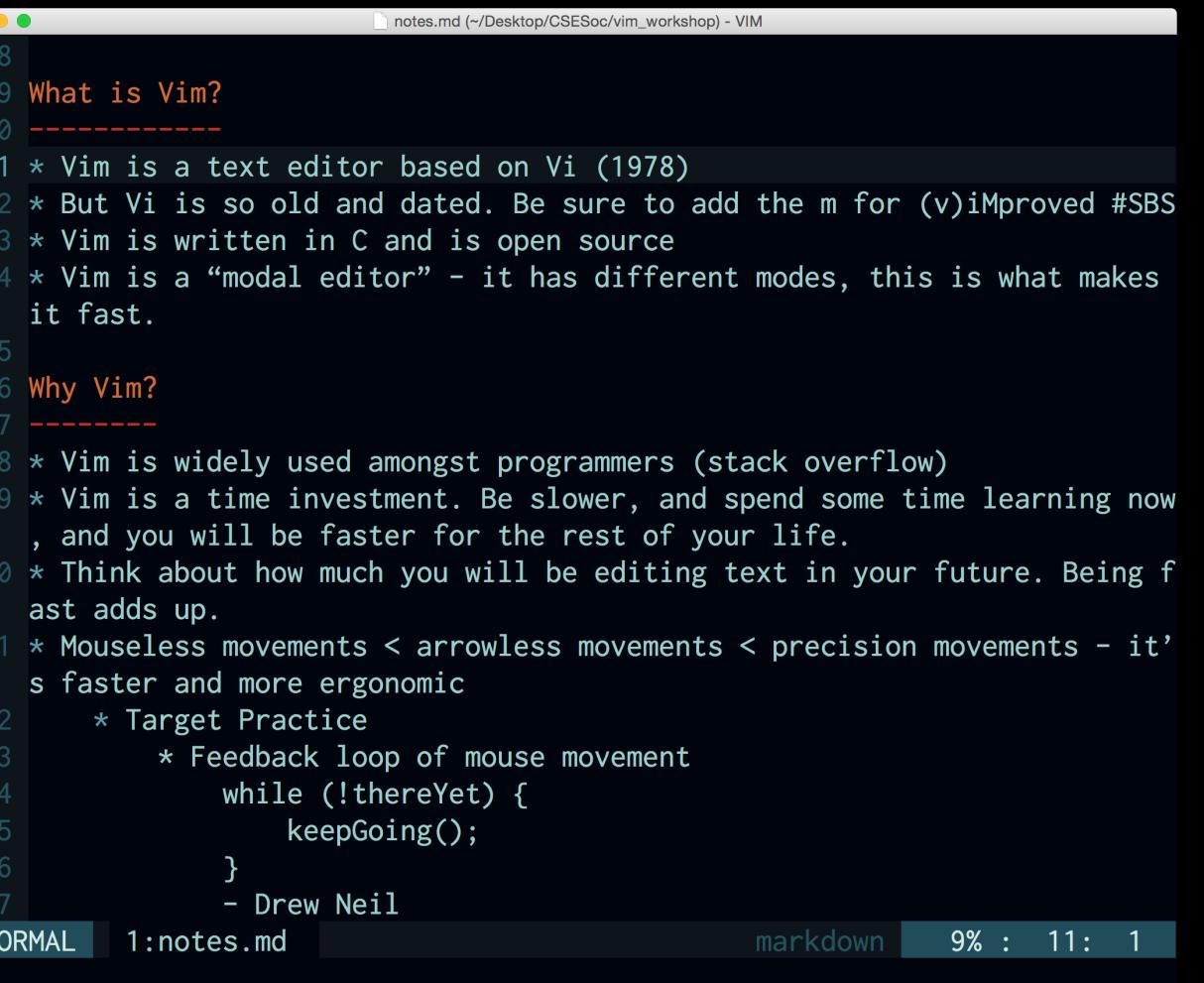
# Vi vs. Vim vs. NeoVim

Vi (1978)



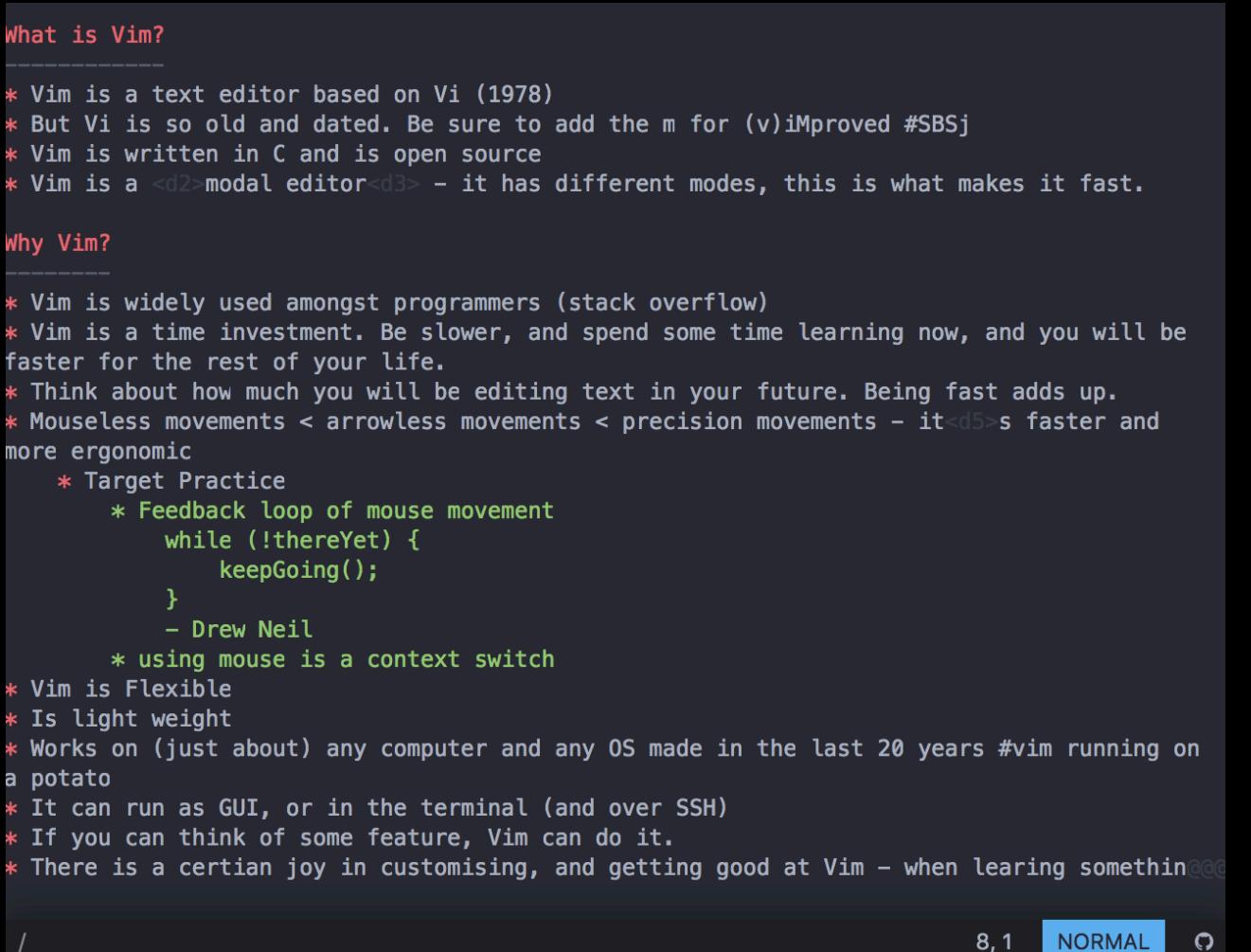
```
What is Vim?  
-----  
* Vim is a text editor based on Vi (1978)  
)  
* But Vi is so old and dated. Be sure to  
add the m for (v)improved #SBS  
* Vim is written in C and is open source  
* Vim is a "modal editor" - it has different modes, this is what makes it fast.  
  
Why Vim?  
-----  
* Vim is widely used amongst programmers  
(stack overflow)  
@
```

Vim



```
notes.md (~/Desktop/CSEsociety/vim_workshop) ~ VIM  
8  
9 What is Vim?  
10 -----  
11 * Vim is a text editor based on Vi (1978)  
12 * But Vi is so old and dated. Be sure to add the m for (v)improved #SBS  
13 * Vim is written in C and is open source  
14 * Vim is a "modal editor" - it has different modes, this is what makes it fast.  
15  
16 Why Vim?  
17 -----  
18 * Vim is widely used amongst programmers (stack overflow)  
19 * Vim is a time investment. Be slower, and spend some time learning now  
, and you will be faster for the rest of your life.  
20 * Think about how much you will be editing text in your future. Being fast adds up.  
21 * Mouseless movements < arrowless movements < precision movements - it's faster and more ergonomic  
22     * Target Practice  
23         * Feedback loop of mouse movement  
24             while (!thereYet) {  
25                 keepGoing();  
26             }  
27         - Drew Neil  
NORMAL 1:notes.md      markdown 9% : 11: 1
```

NeoVim



```
What is Vim?  
-----  
* Vim is a text editor based on Vi (1978)  
* But Vi is so old and dated. Be sure to add the m for (v)improved #SBS  
* Vim is written in C and is open source  
* Vim is a <d2> modal editor<d3> - it has different modes, this is what makes it fast.  
  
Why Vim?  
-----  
* Vim is widely used amongst programmers (stack overflow)  
* Vim is a time investment. Be slower, and spend some time learning now, and you will be faster for the rest of your life.  
* Think about how much you will be editing text in your future. Being fast adds up.  
* Mouseless movements < arrowless movements < precision movements - it's faster and more ergonomic  
    * Target Practice  
        * Feedback loop of mouse movement  
            while (!thereYet) {  
                keepGoing();  
            }  
        - Drew Neil  
        * using mouse is a context switch  
* Vim is Flexible  
* Is light weight  
* Works on (just about) any computer and any OS made in the last 20 years #vim running on a potato  
* It can run as GUI, or in the terminal (and over SSH)  
* If you can think of some feature, Vim can do it.  
* There is a certain joy in customising, and getting good at Vim - when learning something new  
/
```

# Vim is a ‘modal’ editor

NORMAL

Keys are used to move  
around and modify text

INSERT

Type like you are used to,  
just like nano

VISUAL

For selecting text

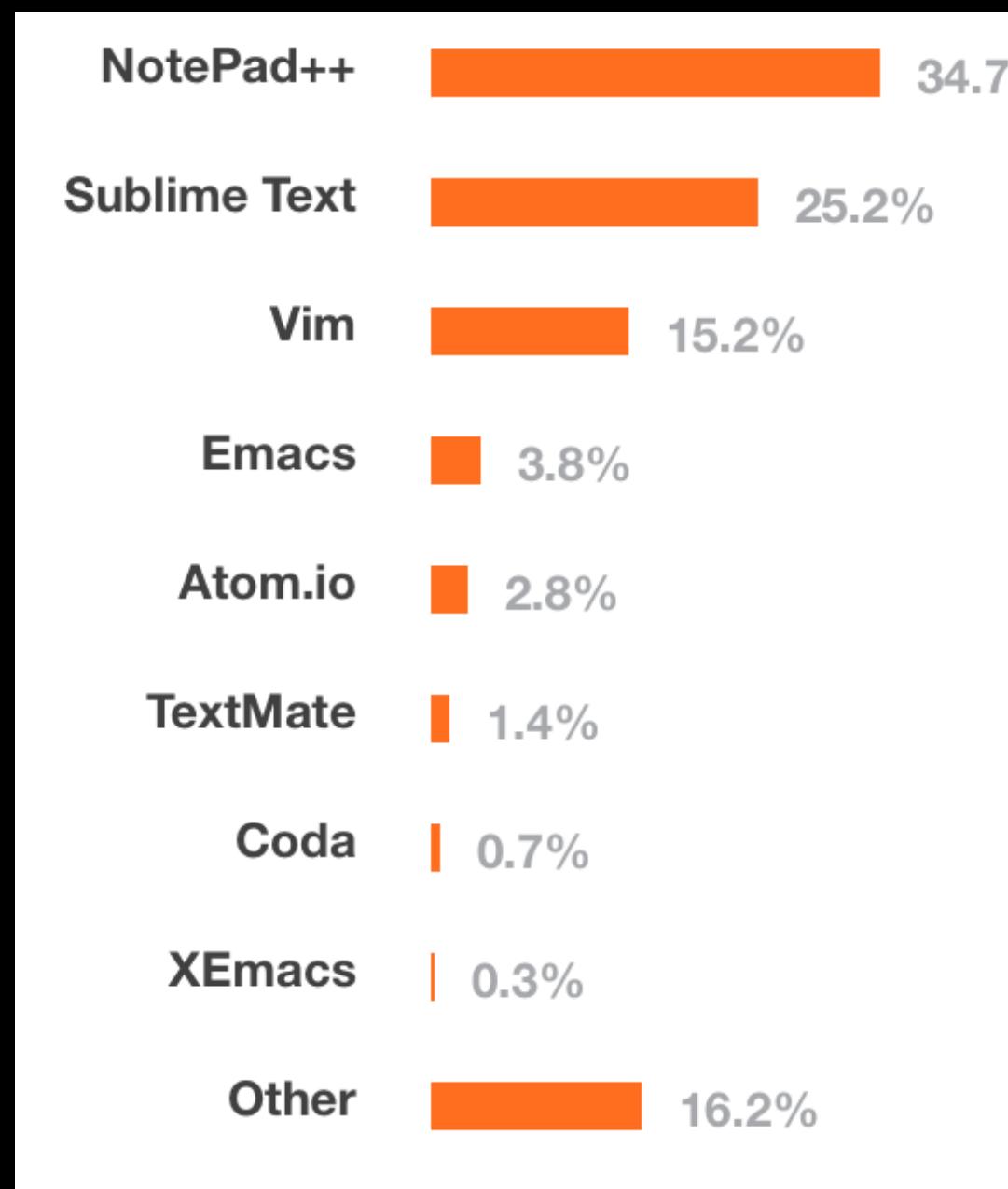
: immacomm

Mini command line for  
vim commands

# But, Why Vim?

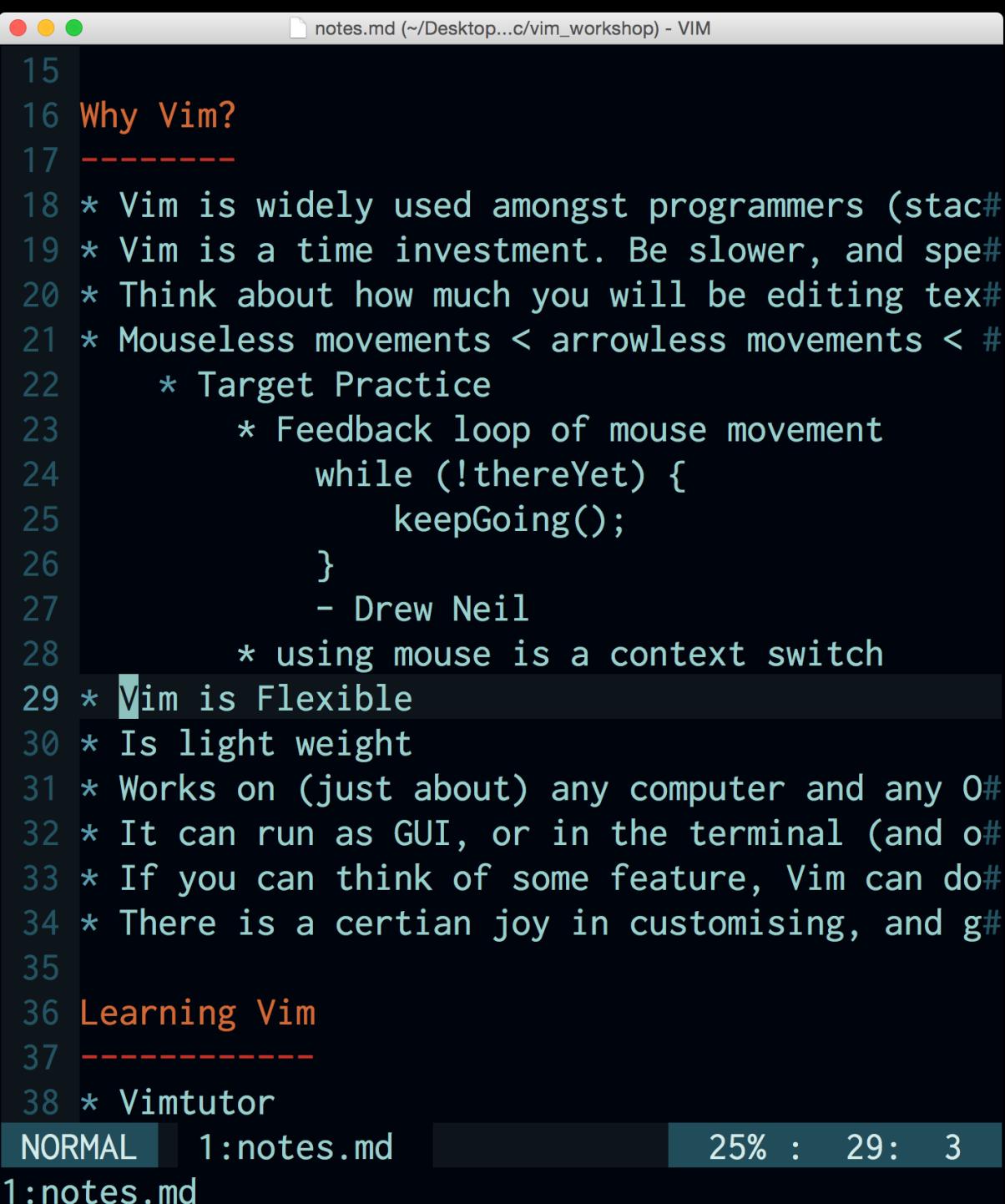
(what's so good about it anyway?)

# It's pretty popular



PREFERRED TEXT EDITOR BY OCCUPATION	
Occupation	Preferred text editor
Desktop Developers	NotePad++
Front-end web developers	Sublime Text
DevOps & Sys admins	Vim
Full-stack developers	Notepad++ and Sublime Text

# Flexible



A screenshot of a Mac OS X desktop showing a terminal window titled "notes.md (~/Desktop...c/vim\_workshop) - VIM". The terminal displays a list of reasons why Vim is flexible, starting with "Why Vim?" and ending with "Vimtutor". The text is color-coded: numbers are blue, sections like "Why Vim?" and "Vimtutor" are orange, and the quote from Drew Neil is grey. The Vim status bar at the bottom shows "NORMAL 1:notes.md 25% : 29: 3 1:notes.md".

```
15
16 Why Vim?
17 -----
18 * Vim is widely used amongst programmers (stack overflow)
19 * Vim is a time investment. Be slower, and specific
20 * Think about how much you will be editing text
21 * Mouseless movements < arrowless movements < keyboard
22     * Target Practice
23         * Feedback loop of mouse movement
24             while (!thereYet) {
25                 keepGoing();
26             }
27             - Drew Neil
28         * using mouse is a context switch
29 * Vim is Flexible
30 * Is light weight
31 * Works on (just about) any computer and any operating system
32 * It can run as GUI, or in the terminal (and on mobile devices)
33 * If you can think of some feature, Vim can do it
34 * There is a certain joy in customising, and getting things to work
35
36 Learning Vim
37 -----
38 * Vimtutor
```

# Flexible

The screenshot shows a terminal window with several tabs open, displaying C code for a game of Naughts and Crosses. The code includes functions for transforming arrays (swap, transpose, flipV, flipH) and testing them (testCoord, testFlipV, testTranspose2, testTranspose3, testGetStrictRoot). The code is annotated with comments and assertions. The terminal also shows the output of a gcc compilation command, which fails due to a type mismatch between 'void' and 'char \*'. The status bar at the bottom indicates the file is 48 lines long, 93% complete, and has 150 errors.

```
tmux attach || tmux new -t tmux -t tmux
```

```
Thu 9 Apr 2015 00:50:32 AEST
```

```
1 // transform.h
2 // Callum Howard 2015
3
4 #define FALSE 0
5 #define TRUE 1
6
7 void swap (char *a, char *b);
8 void transpose (char a[], int size);
9 int arrayCompare (char arrayA[], char arrayB[], int size);
10 int getStrictRoot (int input);
11 int coord (int x, int y, int size);
12 void flipV (char a[], int size);
13 void flipH (char a[], int size);
```

```
1 // transform.c
2 // Callum Howard 2015
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <math.h>
7 #include <assert.h>
8 #include "transform.h"
9
10 --- 5 lines: void swap (char *a, char *b) {
11
12 --- 14 lines: void transpose (char a[], int size) {
13
14 int arrayCompare (char arrayA[], char arrayB[], int size) {
15     for (int i=0; i<size; i++) {
16         if (arrayA[i] != arrayB[i]) {
17             return FALSE;
18         }
19     }
20
21     return TRUE;
22 }
23
24 void flipH (char a[], int size) {
25     int dim = getStrictRoot(size);
26     int i = 0;
27     int j = 0;
28
29     // starting from outermost columns and moving in...
30     while (i < j) {
31         // swap columns
32         for (int k=0; k<dim; k++) {
33             swap(&a[coord(k, i, size)], &a[coord(k, j, size)]);
34         }
35         i++;
36         j--;
37     }
38
39 }
40
41 void flipV (char a[], int size) {
42     int dim = getStrictRoot(size);
43     int i = 0;
44     int j = 0;
45
46     // starting from top and bottom rows, and moving in...
47     while (i < j) {
48         // swap rows
49         for (int k=0; k<dim; k++) {
50             swap(&a[coord(i, k, size)], &a[coord(j, k, size)]);
51         }
52         i++;
53         j--;
54     }
55
56 }
57
58 --- 26 lines: void testTranspose2 () {
59
60 --- 26 lines: void testTranspose3 () {
61
62 --- 9 lines: void testGetStrictRoot () {
63
64 void testCoord () {
65     assert(coord(2, 2, 9) == 8);
66     assert(coord(1, 2, 9) == 5);
67 }
68
69 void testFlipV () {
70     const int size = 9;
71     char initial[size] = {
72         1, 2, 3,
73         4, 5, 6,
74         7, 8, 9,
75     };
76
77     char flipped[size] = {
78         4, 7, 1,
79         5, 8, 2,
80         6, 9, 3,
81     };
82
83     flipV(initial, size, flipped, size);
84
85     #ifdef DEBUG
86         // print result
87         for (int i=0; i<size; i++) {
88             printf("%d ", initial[i]);
89         }
90         printf("\n");
91     #endif
92
93     assert( arrayCompare(initial, flipped, size) );
94 }
```

```
# callumhoward at MacBook-Pro in ~/UNSW3969/COMP3411/naughtsAndCrosses on git:master *
$ gcc -Wall -Werror -O -o testTransform transform.c testTransform.c
testTransform.c:148:26: error: passing 'void' to parameter of incompatible type 'char *'
    assert( arrayCompare(flipV(initial, size), flipped, size) );
                                         ^
/usr/include/assert.h:93:25: note: expanded from macro 'assert'
    (_builtin_expect(!e, 0) ? __assert_rtn(__func__, __FILE__, __LINE__, #e) : (void)0)
                                         ^
./transform.h:9:24: note: passing argument to parameter 'arrayA' here
int arrayCompare (char arrayA[], char arrayB[], int size);
                                         ^
1 error generated.
```

```
# callumhoward at MacBook-Pro in ~/UNSW3969/COMP3411/naughtsAndCrosses on git:master *
$
```

# Light weight

21.3 MB on my computer  
(before plugins)

# Will run on anything



# GUI



MacVim



gVim (Linux, Windows)



Vim key bindings in other editors

# SSH

A screenshot of a terminal window titled "tmux attach || tmux new - tmux - tmux". The title bar also shows the date and time: "Mon 13 Apr 2015 01:19:49 AEST". The terminal has two tabs: "notes.md" and "frequency.c". The "frequency.c" tab is active and displays the following text:

```
# callumhoward at MacBook-Pro in ~
$ ssh cse

# callumh at wagner in ~

# callumh at wagner in ~
$ cd AWESOMEFOLDER/

# callumh at wagner in ~/AWESOMEFOLDER
$


```

---

```
19 int main (int argc, char **argv) {
20     // declare an array to store the frequency
21     int frequency[MAX_ASCII];
22
23     // make sure the array is empty by initialising to 0
24     int i = 0;
25     while (i < MAX_ASCII) {
26         frequency[i] = 0;
27         i++;
28     }
29
30     // read characters in from STDIN (standard input) until Ctrl-d
```

The cursor is positioned at line 22, character 1. The status bar at the bottom shows "NORMAL" and "1:frequency.c" on the left, and "main() < c utf-8[unix]" and "36% : 22: 0" on the right.



```
while (!thereYet) {  
    keepGoing();  
}
```

- Drew Neil, “Precision Editing at the Speed of Thought”

# But can Vim do XYZ?

...yes, yes it can.

# Vim is a time investment

Editing in Vim is FAST, but it takes time to learn

How to get good at  
Vim

# How to get good at Vim

- Vimtutor
  - installed with vim
  - just type `vimtutor` at the terminal
- [VimCasts.org](http://VimCasts.org)
- Vim :help - built in help/documentation
- Google / StackOverflow

# How to get good at Vim

- Practice!
- Don't stop learning new things,
- Seek a faster way

Let's Learn some Vim!

# http://callumh.web.cse.unsw.edu.au/vimcheatsheet.pdf



# Cheat Sheet



esc															
normal mode															
~	toggle case	! external filter	@ play macro	# prev ident	\$ EOL	% goto match	^ BOL (soft)	& repeat :s	* next ident	( begin sentence	) end sentence	- BOL down (soft)	+ next line		
`.	goto mark	1 <sup>(2)</sup>	2	3	4	5	6	7	8	9	0	- BOL (hard)	- prev line	= auto indent	
		Q ex mode	W next WORD	E end WORD	R replace mode	T back until	Y yank line	U undo line	I insert at BOL	O open above	P paste before	{ begin paragraph	}	end paragraph	BOL/ goto
		q record macro	W next word	e end word	r replace char	t seek until	y yank	u undo	i insert mode	o open below	p paste after	{ <sup>(1)</sup> misc	} <sup>(7)</sup> misc	<sup>(7)</sup> <leader>	
		A append at EOL	S subst. line	D delete to EOL	F "back" findchar	G EOF/ goto line	H screen top	J join lines	K ident manual	L screen bottom	:	ex cmd line	" register	select	
		a append	s subst. char	d <sup>(1)(3)</sup> delete	f find char	g <sup>(1)(3)</sup> extra commands	h <sup>(8)</sup> commands	j	k <sup>(1)</sup> ▲	l <sup>(1)</sup> ▶	:	repeat seek	' goto	mark BOL	
		Z <sup>(5)</sup> quit	X back-space	C change to EOL	V visual lines	B prev WORD	N prev match	M screen middle	< un-indent	> indent	? search backward	/ search forward			
		Z <sup>(6)</sup> extra commands	X delete char	C <sup>(1)(3)</sup> change	V <sup>(4)</sup> visual mode	B <sup>(1)</sup> prev word	N <sup>(1)</sup> next match	M <sup>(1)</sup> set mark	,	,					

**motion**

moves the cursor, or defines the range for an operator

**command**

direct action command, if labelled red, it enters insert mode

**operator**

requires a motion afterwards, operates between cursor & destination

**extra**

special functions, requires extra input

:w :write save <filename>

:q :quit

:wq :quit!

:q!: quit and discard changes

:e :edit open <filename> (can be a new filename)

:h :help open manual and documentation

:bn :bnext show next buffer (file)

:bp :bprevious show previous buffer

:ls :list all buffers

:%s/<pattern>/<string>/g replace <pattern> with <string> (can use regex)

<C-r> redo

<C-v> block-visual mode

<C-g> show current filename

<C-e> / <C-y> scroll line up / line down

<C-u> / <C-d> half page up / half page down

<C-f> / <C-b> page up / page down

<C-o> / <C-I> jump to last / next cursor position (in jumplist)

<C-w>s split window horizontally

<C-w>v vertically split window

<C-w><C-w> cycle next window (split)

<C-w>r rotate window positions

<C-w>c close window

(1) use "<char> before a yank/paste/delete command to specify which register ('clipboard') to use  
eg. "ay\$ to copy rest of line to register 'a'

(2) enter a number before a motion, command or operator to repeat it that many times  
eg. 2p, c2w, 5j, d4j

(3) enter an operator twice to act on the current line  
eg. dd, >, ==

(4) use 'v' when in normal mode to enter visual mode and select text by entering motions

(5) use 'ZZ' to save and quit (same as :wq)  
use 'ZQ' to quit and discard changes

(6) zt scroll cursor is at top of screen  
zb scroll cursor is at bottom of screen  
zz scroll cursor to middle of screen

(7) [[ / ]] jump cursor to previous / next whitespace line

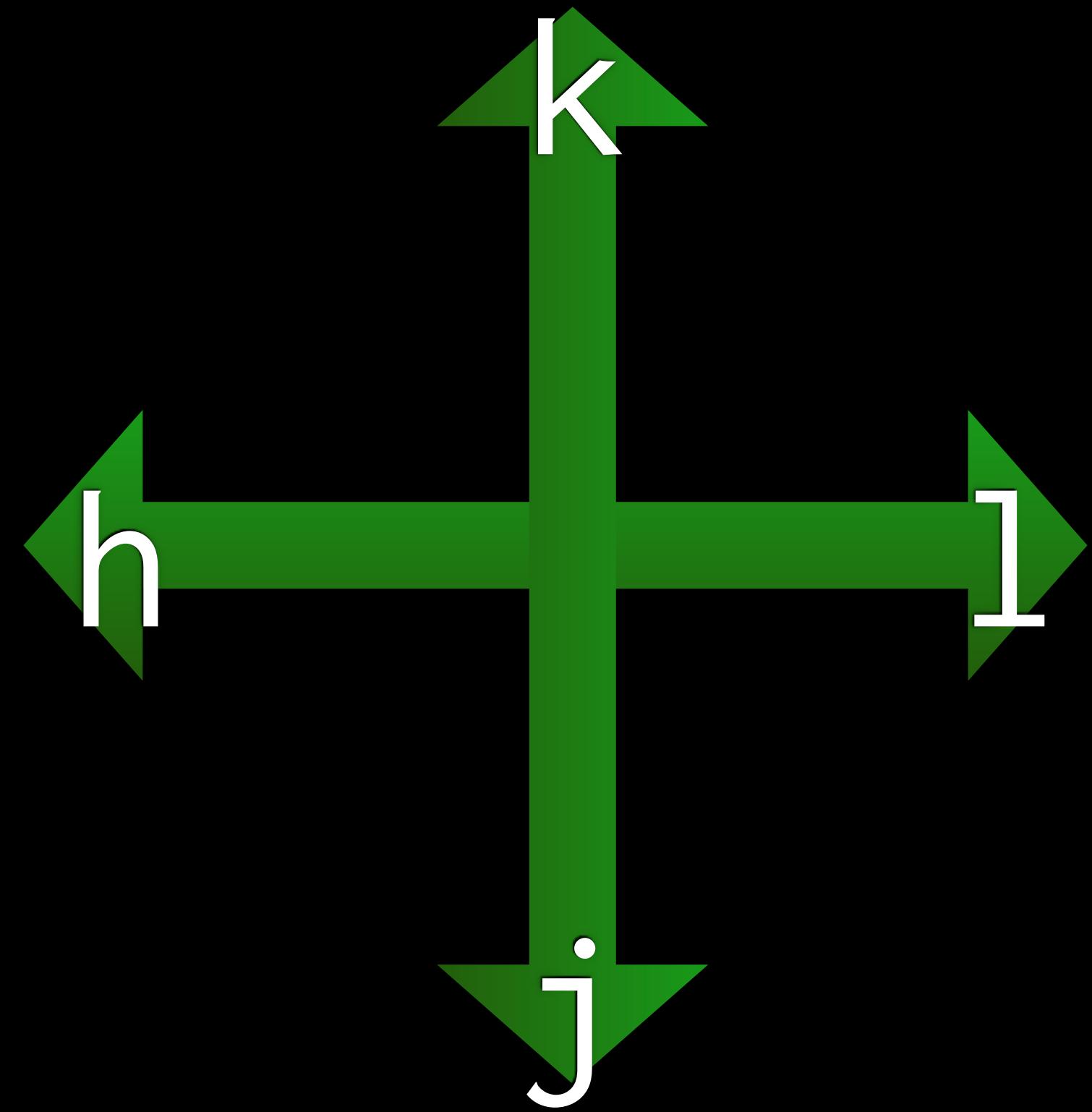
(8) gg jump cursor to top of file  
<number>gg jump cursor to line <number>  
gf open file under cursor

CSEsoc Vim Workshop, Callum Howard 2015  
based on vi/vim graphical cheat sheet: www.viemu.com

Vim can be frustrating at first,  
but persevere! Soon enough  
it will be muscle memory

<http://callumh.web.cse.unsw.edu.au/vimcheatsheet.pdf>

# Movement



<http://callumh.web.cse.unsw.edu.au/vimcheatsheet.pdf>

# Exiting Vim

:w	:write	save <filename>
:q	:quit	quit
:wq		save and quit
:q!	:quit!	quit and discard changes
:e	:edit	open <filename> (can be a new filename)
:h	:help	open manual and documentation

use ‘ZZ’ to save and quit (same as :wq)  
use ‘ZQ’ to quit and discard changes

x to Delete

it's like crossing it out

# Inserting (with i)

NORMAL

INSERT

VISUAL

:immacommand

Keys are used to move around and modify text

Type like you are used to, just like nano

For selecting text

Mini command line for vim commands

This is a example sentence.

i to insert before cursor

a to insert after cursor

|This is a **example** sentence.|

I to insert **before** line

A to append at end of line



Emacs users hate him, you  
won't believe these 6 AMAZING  
productivity tips... [more](#)

# Memorise using the mnemonics



# Cheat Sheet



esc															
normal mode															
~	toggle case	! external filter	@ play macro	# prev ident	\$ EOL	% goto match	^ BOL (soft)	& repeat :s	* next ident	( begin sentence	) end sentence	- BOL down (soft)	+ next line		
`.	goto mark	1 <sup>(2)</sup>	2	3	4	5	6	7	8	9	0	- BOL (hard)	- prev line	= auto indent	
		Q ex mode	W next WORD	E end WORD	R replace mode	T back until	Y yank line	U undo line	I insert at BOL	O open above	P paste before	{ begin paragraph	}	end paragraph	BOL/ goto
		q record macro	W next word	e end word	r replace char	t seek until	y yank	u undo	i insert mode	o open below	p paste after	{ misc	}	misc	\ <leader>
		A append at EOL	S subst. line	D delete to EOL	F "back" findchar	G EOF/ goto line	H screen top	J join lines	K ident manual	L screen bottom	:	ex cmd line	" register	select	
		a append	s subst. char	d <sup>(1)(3)</sup> delete	f find char	g <sup>(1)(3)</sup> extra commands	h <sup>(8)</sup> commands	j	k <sup>(1)</sup> ▲	l <sup>(1)</sup> ▶	:	repeat seek	' goto	mark BOL	
		Z <sup>(5)</sup> quit	X back-space	C change to EOL	V visual lines	B prev WORD	N prev match	M screen middle	< un-indent	> indent	? search backward		/ search forward		
		Z <sup>(6)</sup> extra commands	X delete char	C <sup>(1)(3)</sup> change	V <sup>(4)</sup> visual mode	B <sup>(1)</sup> prev word	N <sup>(1)</sup> next match	M <sup>(1)</sup> set mark	,	,					

**motion** moves the cursor, or defines the range for an operator

**command** direct action command, if labelled red, it enters insert mode

**operator** requires a motion afterwards, operates between cursor & destination

**extra** special functions, requires extra input

**q.** commands with a dot need a character argument afterwards

BOL Beginning Of Line

EOL End Of Line

ident word under cursor

yank copy

words WORDs

```
int main(int argc, char *argv[]) {  
int main(int argc, char *argv[]) {
```

:w :write save <filename>

:q :quit

:wq :quit!

:q!: quit and discard changes

:e :edit open <filename> (can be a new filename)

:h :help open manual and documentation

:bn :bnext show next buffer (file)

:bp :bprevious show previous buffer

:ls :list all buffers

:%s/<pattern>/<string>/g replace <pattern> with <string> (can use regex)

<C-r> redo

<C-v> block-visual mode

<C-g> show current filename

<C-e> / <C-y> scroll line up / line down

<C-u> / <C-d> half page up / half page down

<C-f> / <C-b> page up / page down

<C-o> / <C-I> jump to last / next cursor position (in jumplist)

<C-w>s split window horizontally

<C-w>v vertically split window

<C-w><C-w> cycle next window (split)

<C-w>r rotate window positions

<C-w>c close window

(1) use "<char> before a yank/paste/delete command to specify which register ('clipboard') to use  
eg. "ay\$ to copy rest of line to register 'a'

(2) enter a number before a motion, command or operator to repeat it that many times  
eg. 2p, c2w, 5j, d4j

(3) enter an operator twice to act on the current line  
eg. dd, >, ==

(4) use 'v' when in normal mode to enter visual mode and select text by entering motions

(5) use 'ZZ' to save and quit (same as :wq)  
use 'ZQ' to quit and discard changes

(6) zt scroll cursor is at top of screen  
zb scroll cursor is at bottom of screen  
zz scroll cursor to middle of screen

(7) [[ / ]] jump cursor to previous / next whitespace line

(8) gg jump cursor to top of file  
<number>gg jump cursor to line <number>  
gf open file under cursor

CSE Soc Vim Workshop, Callum Howard 2015  
based on vi/vim graphical cheat sheet: www.viemu.com

# ProTip: Touch-Type

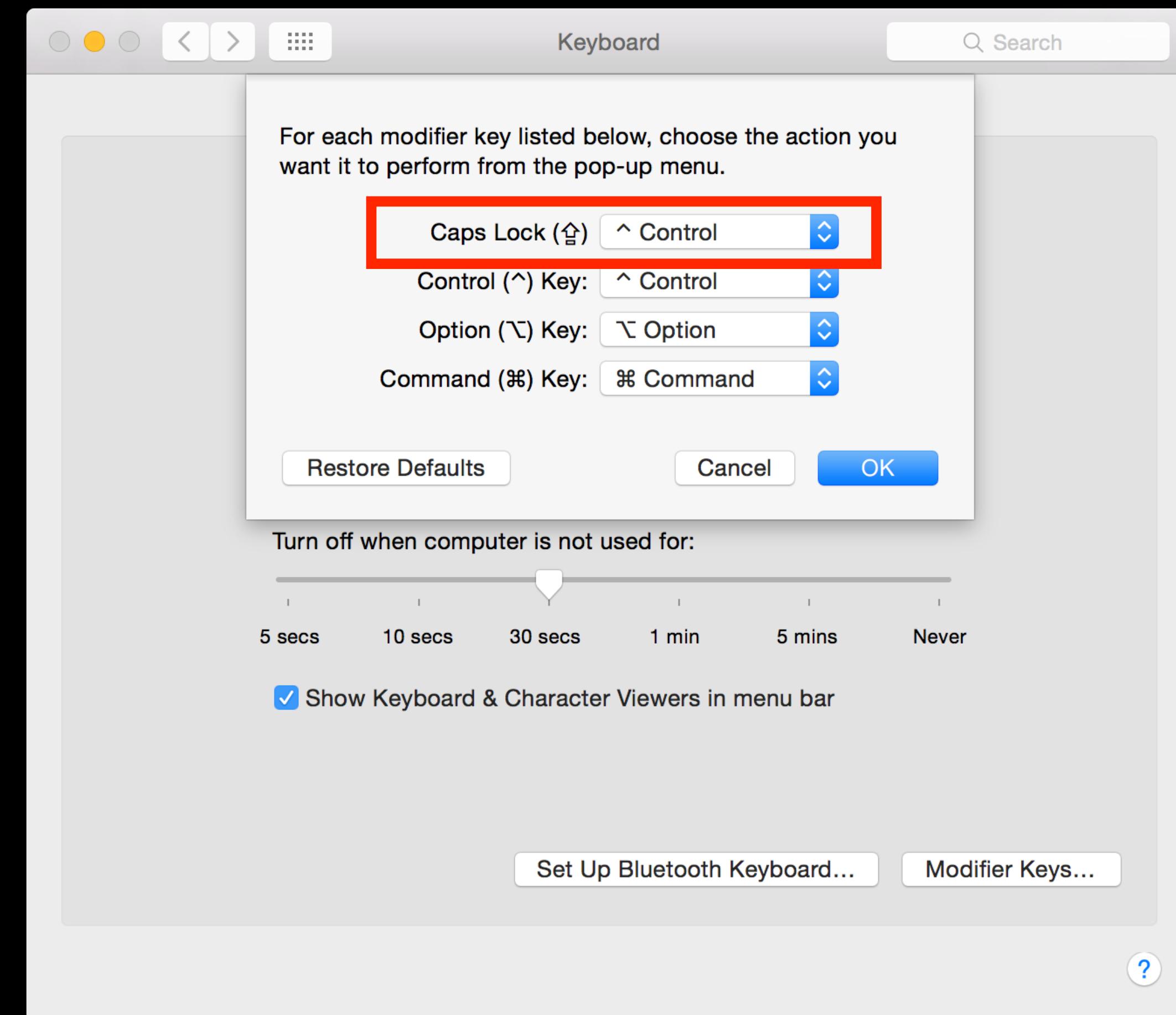


ProTip:

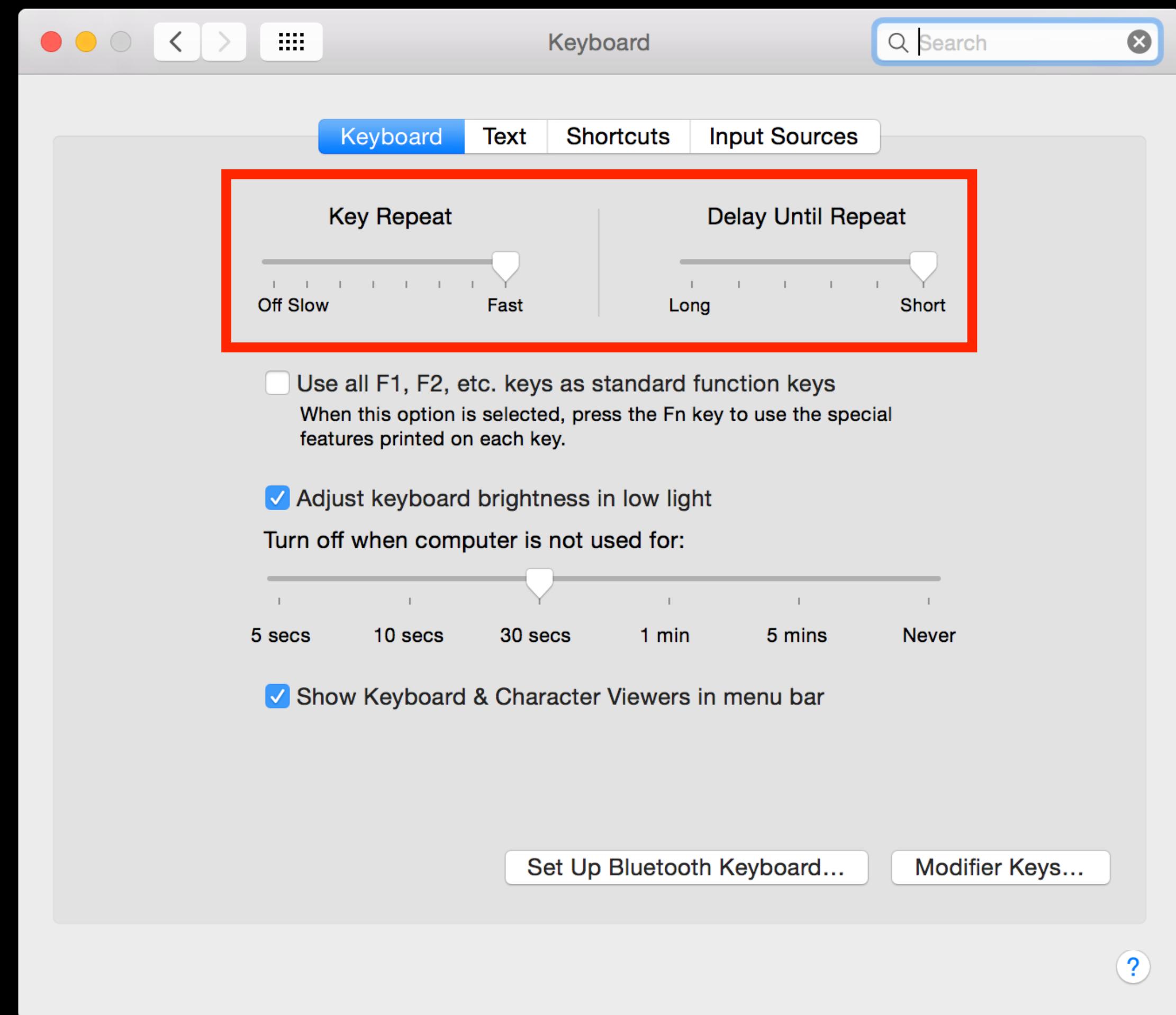
:imap kj <ESC>

Got to first :set nocp

# Protip: Remap caps-lock to CTRL



# Protip: speed up key-repeat



# ProTip: Disable arrow keys

```
map <Up> <NOP>
map <Down> <NOP>
map <Left> <NOP>
map <Right> <NOP>
```

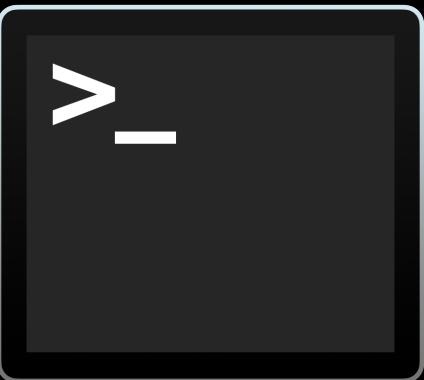
put this in your .vimrc

# gVim vs Terminal Vim



gVim

- + Smoother Scrolling
- + Dynamic cursor (| in insert mode, ━ in normal mode)
- + Can display Millions of colours
- Has toolbar that you should turn off!



Terminal Vim

- + Access to the Command Line <C-z>
- + 256 colours
- + Tmux
- + SSH

# Plugins or Vanilla Vim

## Plugins

- + Super Powerful
- + Super Customisable
- + Super
- + Can slow down Vim

## Vanilla Vim

- + Can use in an exam
- + Can use another person's account/computer without your customisations
- + Can use over SSH