

The Vim Talk

Tips from a Vim Addict

Davis Claiborne

LUG @ NC State

February 6, 2020



Linux Users Group
at NC State University

What is Vim?

- Modal, configurable, terminal-based text editor

The Vim Talk

└ Introduction

└ About Vim

└ What is Vim?

What is Vim?

- Modal, configurable, terminal-based text editor

- Brief introduction for those who don't already know Vim
- Vim is a modal, configurable, terminal-based text editor
- That's a lot of jargon, so let's define what some of those mean. I'll talk more in-depth about what these mean later, but for now I'll at least give a general overview, and I can talk more about the pros and cons of each later
- Vim is a modal text editor. This essentially means that Vim has several different modes that you can use it in
- Vim is also highly configurable. Many aspects of Vim's use can be modified through plugins or changing settings files
- Finally, Vim is terminal-based. This means that it doesn't require a fully-fledged graphical environment to use.

What is Vim?

- Modal, configurable, terminal-based text editor
- Open source charityware

The Vim Talk

└ Introduction

└ About Vim

└ What is Vim?

What is Vim?

- Modal, configurable, terminal-based text editor
- Open source charityware

- Vim is open source charityware. This means that the code is freely available online, and users are encouraged to make donations to ICCF Holland to help “poor children in Uganda”
- The revenue from ads and merchandise sales (such as t-shirts, stickers, etc.) also go to help support this organization

Why Vim?

- Keyboard-centric → Work faster

The Vim Talk

└ Introduction

└ About Vim

└ Why Vim?

Why Vim?

- Keyboard-centric → Work faster

- There are many reasons that could be given for why to use Vim
- One of the most appealing aspects of Vim is that it is keyboard-centric, which allows you to edit text more efficiently (once you learn how to)
- Think about it like this: whenever you're copying or pasting text, do you go to your mouse, right click, and then select "Copy" or "Paste" from the context menu? Or do you use Control C and Control P?
- What about when you're browsing the web? Do you use your mouse to open a new tab, or do you press Control T? In general, think about how much faster you can work when you know basic keyboard shortcuts
- Vim takes this thought process to the extreme - it essentially asks, "Why use your mouse at all?"

Why Vim?

- Keyboard-centric → Work faster
- Large user-base

The Vim Talk

└ Introduction

└ About Vim

└ Why Vim?

Why Vim?

- Keyboard-centric → Work faster
- Large user-base

- Vim also has a large user-base
- This means that, for most use-cases, whatever plugin you want has (more than likely) already been developed

Why Vim?

- Keyboard-centric → Work faster
- Large user-base
- Common

The Vim Talk

└ Introduction

└ About Vim

└ Why Vim?

Why Vim?

- Keyboard-centric → Work faster
- Large user-base
- Common

- Because Vim is popular, that also means that Vim is commonly available by default on most systems
- This means that whenever you're working on a system that isn't yours, odds are pretty good that you'll be able to copy over your configuration files and feel right at home

Why Vim?

- Keyboard-centric → Work faster
- Large user-base
- Common
- Terminal-based

The Vim Talk

└ Introduction

└ About Vim

└ Why Vim?

Why Vim?

- Keyboard-centric → Work faster
- Large user-base
- Common
- Terminal-based

- Vim is also terminal based
- This is one of the benefits that I didn't really get when I first started using Vim - what's the benefit of not having a great user-interface?
- One benefit is that it's lighter on resources - while fancy Electron-based editors look great, they also hog all of your RAM!
 - A terminal application *can* of course hog up your RAM - it's not a magic solution - but in general, terminal applications use less RAM
- Another benefit is that you can use Vim when you're remotely accessing a computer (for instance, when you SSH in), meaning you don't have to move the file back and forth to use the editor that you're used to

Why not Vim?

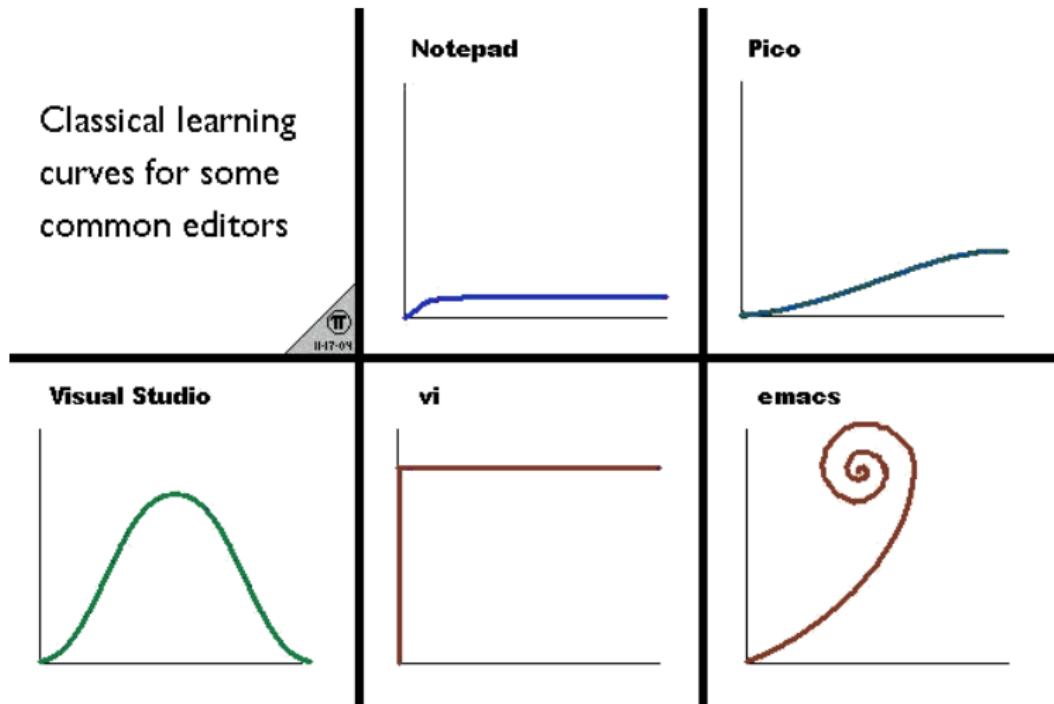


Figure 1: Vim is hard to learn

The Vim Talk

└ Introduction

└ About Vim

└ Why not Vim?

Why not Vim?

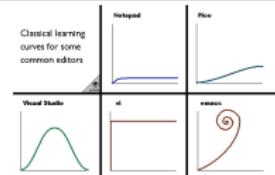


Figure 1: Vim is hard to learn

- Now, let's look at why not to learn Vim
- The main reason most people never delve too much into Vim is because of how much work it initially takes to learn it
- From this highly accurate and scientific graph shown here, we can see that vi (the older, somewhat simpler version of Vim) starts off incredibly hard and stays hard

Why not Vim?

- Hard to learn
- No built-in niceties

The Vim Talk

└ Introduction

└ About Vim

└ Why not Vim?

Why not Vim?

- Hard to learn
- No built-in niceties

- Jokes aside, it actually is pretty difficult to learn Vim - it takes a serious amount of time to entirely relearn how to use a text editor
- Another drawback to many about Vim is how bare it is out of the box - I've heard many people bemoan the lack of integrated git tools and debuggers
- My response to that is always that, while they would be nice to have built-in by default, there are plugins that exist that work well

Vim is Modal

- Insert mode
- Normal mode

The Vim Talk

└ Introduction

└ Basic Concepts

└ Vim is Modal

Vim is Modal

- Insert mode
- Normal mode

- The most important thing to understand about Vim is that it is a *modal* text editor
- What does that mean? Basically, there are different modes of operation that Vim can be in.
- The two main modes are insert mode and normal mode
- In insert mode, Vim behaves like you'd expect a normal text editor to - you type text and it shows up on screen
- Normal mode is where things start to get different - in normal mode, keys *don't* type letters
- Instead, keys do different things - some move the cursor around, some delete characters, and some change Vim to another mode
- While counter-intuitive at first, once you learn to embrace the different modes of Vim and what they allow you to do, you can work much more efficiently

Vim is Modal

- Insert mode
- Normal mode
- Replace mode
- Visual mode
- Visual block mode
- Command mode

The Vim Talk

└ Introduction

└ Basic Concepts

└ Vim is Modal

Vim is Modal

- Insert mode
- Normal mode
- Replace mode
- Visual mode
- Visual block mode
- Command mode

- Vim has many other modes as well, which, though less well-known than normal and insert mode, are also both very useful
- I'll talk about them each more in-depth later, but here's a basic run-down:
- Replace mode is for replacing text
- Visual mode is for selecting text
- Visual block mode is like visual mode, but for columns of text instead of lines of text
- Command mode is for inputting Vim commands

Key behaviors in normal mode

Types of normal mode actions:

- Motion
- Command
- Operator
- Extra

The Vim Talk

└ Introduction

└ Basic Concepts

└ Key behaviors in normal mode

Key behaviors in normal mode

Types of normal mode actions:

- Motion
- Command
- Operator
- Extra

- There are four main types of normal mode actions: motions, commands, operators, and extras
- Motion actions move the cursor around the screen. Typical motion keys are h, j, k, and l.
- Commands do something - normally they either change modes or modify the text. Common commands are i or escape.
- Operators require a motion to be given after them, and operate on the text between the cursor and the motion. Examples of operators include d and c.
- Extra keys that act as prefixes to more complex combinations. Examples of extra keys include g and z.

Practicing / learning

- Vim tutor [3]
- Online tutorials

The Vim Talk

└ Introduction

└ Learning More

└ Practicing / learning

Practicing / learning

- Vim tutor [3]

- Online tutorials

- Though Vim has a very high learning curve, luckily there are many references and tools available to help you learn
- One of the most valuable and underused tools, in my opinion, is one of the tools that comes bundled with Vim - vimtutor.
- It's an interactive tutorial that takes about a half hour to complete that teaches people the basics of Vim
- See `:help vimtutor` for more
- There are also many other ways to learn - books, interactive tutorials available online, and even some games that use Vim-like movement to help teach you the basics. No matter how you learn, there's (probably) a way that works for you

Practicing / learning

- Vim tutor [3]
- Online tutorials
- Vim User Manual [2]
- Vim Reference Manual [1]

The Vim Talk

└ Introduction

└ Learning More

└ Practicing / learning

Practicing / learning

- Vim tutor [3]
- Online tutorials
- Vim User Manual [2]
- Vim Reference Manual [1]

- Finally, there's two pieces of documentation built in to Vim itself - the Vim User Manual and the Reference manual
- The reference manual is what you'd think - good for referencing, but poor for learning. That's where the user manual comes in - this is intended to be used as a guide to more advanced information and techniques
- You can find it by typing :help usr
- Knowing how to properly use the Vim reference manual is invaluable to figuring out your problems quickly - see :help help-summary for more

Introduction

Important Concepts
Advanced Applications
Plugins

About Vim
Basic Concepts
Learning More

Simple cheat sheet

version 1.1
April 1st, 06

vi / vim graphical cheat sheet

Esc	normal mode											
~ toggle case	! external filter	@ play macro	# prev ident	\$ eol	% goto match	^ "soft" bol	& repeat is	* next ident	(begin sentence) end sentence	"soft" bol down	+ next line
* goto mark	1	2	3	4	5	6	7	8	9	0 "hard" bol	- prev line	= auto-format
Q ex mode	W next WWORD	E end WORD	R replace mode	T back till	Y yank line	U undo line	I insert at bol	O open above	P paste before	{ begin parag.	}	end parag.
q record macro	w next word	e end word	r replace char	t till	y yank	u undo	i insert mode	o open below	p paste after	l misc	m misc	b misc
A append at eol	S subs line	D delete to eol	F find ch	G eof goto in cmd	H screen top	J join lines	K help	L screen bottom	: ex cmd line	!! reg ^	bol/ gotocol	\ not used!
a append	s subst char	d delete	f find	g extra cmds	h ←	j ↓	k ↑	l →	; repeat	' goto mk, bol		
Z quit ⁴	X backspace	C change to eol	V visual lines	B prev word	N prev (find)	M screen mid!	< unindent ³	> indent ³	? find (rev.)			
Z extra ⁵	X delete char	C change	V visual mode	b prev word	n next (find)	m set mark	, reverse	, repeat	/ find			
Z extra cmds	X delete	C change	V visual mode									

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

command direct action command, if 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 char argument afterwards

bol = beginning of line, eol = end of line, mk = mark, yank = copy

words: `:nuux[foo] bar[.] baz[.]`

WORDs: `:nuux[foo] bar[.] baz[.]`

Main command line commands ('ex'): :w (save), :q (quit), :q! (quit w/o saving)
 :e (open file f), :%s/x/y/g (replace 'x' by 'y' filewide), :h (help in vim), :new (new file in vim),

Other important commands:
 CTRL-R: redo (vim),
 CTRL-F/-B: page up/down,
 CTRL-E/-Y: scroll line up/down,
 CTRL-V: block-visual mode (vim only)

Visual mode:
 Move around and type operator to act on selected region (vim only)

Notes:

- (1) use "x before a yank/paste/del command to use that register ('clipboard') (x=a, z, *) (e.g.: "ay\$ to copy rest of line to reg 'a')
- (2) type a number before any action to repeat it that number of times (e.g.: 2p, daw, 5i, d4)
- (3) duplicate operator to act on current line (dd = delete line, >> = indent line)
- (4) ZZ to save & quit, ZQ to quit w/o saving
- (5) zt: scroll cursor to top,
 zb: bottom, zz: center
- (6) gg: top of file (vim only),
 gf: open file under cursor (vim only)

For a graphical vi/vim tutorial & more tips, go to www.viemu.com - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

Figure 1: Simple Vim cheat sheet [4]

The Vim Talk

Introduction

Learning More

Simple cheat sheet

Simple cheat sheet

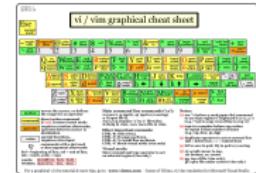


Figure 1: Simple Vim cheat sheet [4]

- Cheat sheets can be useful references, especially while you're learning but even after you consider yourself proficient
- The website that I got this cheat sheet from also features a series of useful tutorials that I recommend following - slowly incorporate the basic motions they teach until you're comfortable with them, then move to the next page

Introduction

Important Concepts Advanced Applications Plugins

About Vim
Basic Concepts
Learning More

Advanced cheat sheet

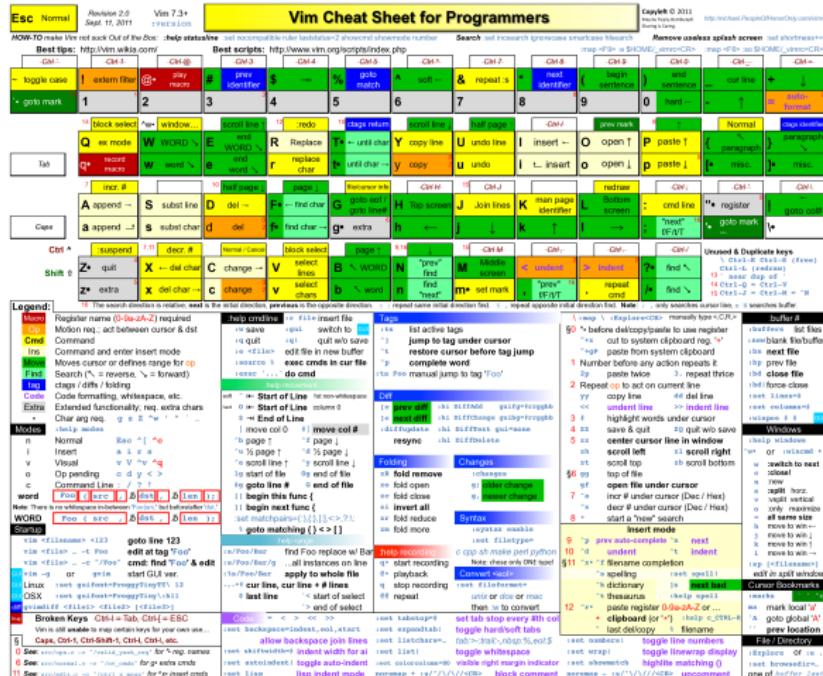


Figure 2: Advanced Vim cheat sheet [6]

The Vim Talk

Introduction

Learning More

Advanced cheat sheet

Advanced cheat sheet



- While this cheat sheet looks overwhelming, and does have lots of useful commands, it doesn't even come close to scratching the surface of all of Vim's features
- I seriously recommend at least skimming over the `usr` documentation - it's chock full of good, useful tips laid out in a very readable (for documentation) format [2]

Viewing files

- Buffer
- Window
- Tab

The Vim Talk

└ Important Concepts

└ Windows, Tabs, and Buffers

└ Viewing files

Viewing files

- Buffer
- Window
- Tab

- I'll start off talking about important concepts with one of Vim's most important concepts for advanced use - learning to distinguish between buffers, windows, and tabs
- Though they can be confusing at first, understanding how they work will open up whole new capabilities in your Vim usage

Viewing files

- Buffer
 - File in memory
- Window
 - View of buffer
- Tab
 - Collection of windows

See :help window

The Vim Talk

Important Concepts

Windows, Tabs, and Buffers

Viewing files

Viewing files

- Buffer
 - File in memory
- Window
 - View of buffer
- Tab
 - Collection of windows

See :help window

- Here's a basic explanation of each of them:
- Buffers just represent the fact that a file is being stored in memory
- Windows are views of a buffer
- Tabs are collections of windows and how they are laid out
- It's important to understand that all of these are separate from each other - multiple windows can view the same buffer. Tabs can have as many or as few windows as you want.

Introduction
Important Concepts
Advanced Applications
Plugins

Illustration

Windows, Tabs, and Buffers
Registers / Macros
Folds
Commands and Ranges
Text Objects

```

1 -- file opened generates class
2 -- local class = require('classest')
3 local Map = require('class.map')
4 local Draw = require('class.draw')
5 local Class = require('class.class')
6 local State = require('class.state')
7 local Frame = require('class.state.state')
8 local frame = State:create('frame')
9 -- Global values
10 maxCoordinate, maxCoordinateX = 800, 800
11
12 -- Local variables
13 tileWidth, tileHeight = 20, 24
14 local constMargin, constMarginX = coh.w, coh.b
15 local input, map
16 local numberTilesX, numberTilesY
17 local numberTiles = 99
18
19 -- Handle input
20
21 function frame:update(delta)
22   drawFrame()
23   constMargin, constMarginX = coh.w, coh.b
24   -- Create map
25   map = tileMap(tileWidth, tileHeight)
26   numberTilesX = math.ceil(numberTiles / map.tileWidth)
27   numberTilesY = math.floor(numberTiles / map.tileWidth)
28   maxCoordinateX = numberTilesX * tileWidth
29   maxCoordinateY = numberTilesY * tileHeight
30 end
31
32 -- Local functions
33 local function updateMap()
34   map:generateMap()
35   map:draw()
36   map:translate()
37   map:scale()
38 end
39
40 -- Main loop
41 while true do
42   updateMap()
43   frame:update()
44   frame:render()
45 end
46
47 function tileMap(width, height)
48   -- The actual generation classes
49   local Map = require('classest.map')
50   local Draw = require('classest.draw')
51   local Class = require('classest.class')
52   local frame = require('classest.state.state')
53   local frame = Class:create('frame')
54
55   -- Global values
56   maxCoordinate, maxCoordinateX = 800, 800
57
58   -- Local variables
59   tileWidth, tileHeight, tileMargin = 20, 24
60   constMargin, constMarginX = coh.w, coh.b
61   input, map
62
63   -- Handle input
64   function input:onInput(keys)
65     if keys == 'w' then
66       map:moveUp()
67     elseif keys == 's' then
68       map:moveDown()
69     elseif keys == 'a' then
70       map:moveLeft()
71     elseif keys == 'd' then
72       map:moveRight()
73     end
74   end
75
76   -- Local functions
77   local function updateMap()
78     map:generateMap()
79     map:draw()
80   end
81
82   -- Main loop
83   while true do
84     updateMap()
85     frame:update()
86     frame:render()
87   end
88
89   function tileMap(width, height)
90     -- The actual generation classes
91     local Map = require('classest.map')
92     local Draw = require('classest.draw')
93     local Class = require('classest.class')
94     local frame = require('classest.state.state')
95     local frame = Class:create('frame')
96
97     -- Global values
98     maxCoordinate, maxCoordinateX = 800, 800
99
100    -- Local variables
101    tileWidth, tileHeight, tileMargin = 20, 24
102    constMargin, constMarginX = coh.w, coh.b
103    input, map
104
105    -- Handle input
106    function input:onInput(keys)
107      if keys == 'w' then
108        map:moveUp()
109      elseif keys == 's' then
110        map:moveDown()
111      elseif keys == 'a' then
112        map:moveLeft()
113      elseif keys == 'd' then
114        map:moveRight()
115      end
116    end
117
118    -- Local functions
119    local function updateMap()
120      map:generateMap()
121      map:draw()
122    end
123
124    -- Main loop
125    while true do
126      updateMap()
127      frame:update()
128      frame:render()
129    end
130  end
131
132  function tileMap(width, height)
133    -- The actual generation classes
134    local Map = require('classest.map')
135    local Draw = require('classest.draw')
136    local Class = require('classest.class')
137    local frame = require('classest.state.state')
138    local frame = Class:create('frame')
139
140    -- Global values
141    maxCoordinate, maxCoordinateX = 800, 800
142
143    -- Local variables
144    tileWidth, tileHeight, tileMargin = 20, 24
145    constMargin, constMarginX = coh.w, coh.b
146    input, map
147
148    -- Handle input
149    function input:onInput(keys)
150      if keys == 'w' then
151        map:moveUp()
152      elseif keys == 's' then
153        map:moveDown()
154      elseif keys == 'a' then
155        map:moveLeft()
156      elseif keys == 'd' then
157        map:moveRight()
158      end
159    end
160
161    -- Local functions
162    local function updateMap()
163      map:generateMap()
164      map:draw()
165    end
166
167    -- Main loop
168    while true do
169      updateMap()
170      frame:update()
171      frame:render()
172    end
173  end
174
175  function tileMap(width, height)
176    -- The actual generation classes
177    local Map = require('classest.map')
178    local Draw = require('classest.draw')
179    local Class = require('classest.class')
180    local frame = require('classest.state.state')
181    local frame = Class:create('frame')
182
183    -- Global values
184    maxCoordinate, maxCoordinateX = 800, 800
185
186    -- Local variables
187    tileWidth, tileHeight, tileMargin = 20, 24
188    constMargin, constMarginX = coh.w, coh.b
189    input, map
190
191    -- Handle input
192    function input:onInput(keys)
193      if keys == 'w' then
194        map:moveUp()
195      elseif keys == 's' then
196        map:moveDown()
197      elseif keys == 'a' then
198        map:moveLeft()
199      elseif keys == 'd' then
200        map:moveRight()
201      end
202    end
203
204    -- Local functions
205    local function updateMap()
206      map:generateMap()
207      map:draw()
208    end
209
210    -- Main loop
211    while true do
212      updateMap()
213      frame:update()
214      frame:render()
215    end
216  end
217
218  function tileMap(width, height)
219    -- The actual generation classes
220    local Map = require('classest.map')
221    local Draw = require('classest.draw')
222    local Class = require('classest.class')
223    local frame = require('classest.state.state')
224    local frame = Class:create('frame')
225
226    -- Global values
227    maxCoordinate, maxCoordinateX = 800, 800
228
229    -- Local variables
230    tileWidth, tileHeight, tileMargin = 20, 24
231    constMargin, constMarginX = coh.w, coh.b
232    input, map
233
234    -- Handle input
235    function input:onInput(keys)
236      if keys == 'w' then
237        map:moveUp()
238      elseif keys == 's' then
239        map:moveDown()
240      elseif keys == 'a' then
241        map:moveLeft()
242      elseif keys == 'd' then
243        map:moveRight()
244      end
245    end
246
247    -- Local functions
248    local function updateMap()
249      map:generateMap()
250      map:draw()
251    end
252
253    -- Main loop
254    while true do
255      updateMap()
256      frame:update()
257      frame:render()
258    end
259  end
260
261  function tileMap(width, height)
262    -- The actual generation classes
263    local Map = require('classest.map')
264    local Draw = require('classest.draw')
265    local Class = require('classest.class')
266    local frame = require('classest.state.state')
267    local frame = Class:create('frame')
268
269    -- Global values
270    maxCoordinate, maxCoordinateX = 800, 800
271
272    -- Local variables
273    tileWidth, tileHeight, tileMargin = 20, 24
274    constMargin, constMarginX = coh.w, coh.b
275    input, map
276
277    -- Handle input
278    function input:onInput(keys)
279      if keys == 'w' then
280        map:moveUp()
281      elseif keys == 's' then
282        map:moveDown()
283      elseif keys == 'a' then
284        map:moveLeft()
285      elseif keys == 'd' then
286        map:moveRight()
287      end
288    end
289
290    -- Local functions
291    local function updateMap()
292      map:generateMap()
293      map:draw()
294    end
295
296    -- Main loop
297    while true do
298      updateMap()
299      frame:update()
300      frame:render()
301    end
302  end
303
304  function tileMap(width, height)
305    -- The actual generation classes
306    local Map = require('classest.map')
307    local Draw = require('classest.draw')
308    local Class = require('classest.class')
309    local frame = require('classest.state.state')
310    local frame = Class:create('frame')
311
312    -- Global values
313    maxCoordinate, maxCoordinateX = 800, 800
314
315    -- Local variables
316    tileWidth, tileHeight, tileMargin = 20, 24
317    constMargin, constMarginX = coh.w, coh.b
318    input, map
319
320    -- Handle input
321    function input:onInput(keys)
322      if keys == 'w' then
323        map:moveUp()
324      elseif keys == 's' then
325        map:moveDown()
326      elseif keys == 'a' then
327        map:moveLeft()
328      elseif keys == 'd' then
329        map:moveRight()
330      end
331    end
332
333    -- Local functions
334    local function updateMap()
335      map:generateMap()
336      map:draw()
337    end
338
339    -- Main loop
340    while true do
341      updateMap()
342      frame:update()
343      frame:render()
344    end
345  end
346
347  function tileMap(width, height)
348    -- The actual generation classes
349    local Map = require('classest.map')
350    local Draw = require('classest.draw')
351    local Class = require('classest.class')
352    local frame = require('classest.state.state')
353    local frame = Class:create('frame')
354
355    -- Global values
356    maxCoordinate, maxCoordinateX = 800, 800
357
358    -- Local variables
359    tileWidth, tileHeight, tileMargin = 20, 24
360    constMargin, constMarginX = coh.w, coh.b
361    input, map
362
363    -- Handle input
364    function input:onInput(keys)
365      if keys == 'w' then
366        map:moveUp()
367      elseif keys == 's' then
368        map:moveDown()
369      elseif keys == 'a' then
370        map:moveLeft()
371      elseif keys == 'd' then
372        map:moveRight()
373      end
374    end
375
376    -- Local functions
377    local function updateMap()
378      map:generateMap()
379      map:draw()
380    end
381
382    -- Main loop
383    while true do
384      updateMap()
385      frame:update()
386      frame:render()
387    end
388  end
389
390  function tileMap(width, height)
391    -- The actual generation classes
392    local Map = require('classest.map')
393    local Draw = require('classest.draw')
394    local Class = require('classest.class')
395    local frame = require('classest.state.state')
396    local frame = Class:create('frame')
397
398    -- Global values
399    maxCoordinate, maxCoordinateX = 800, 800
400
401    -- Local variables
402    tileWidth, tileHeight, tileMargin = 20, 24
403    constMargin, constMarginX = coh.w, coh.b
404    input, map
405
406    -- Handle input
407    function input:onInput(keys)
408      if keys == 'w' then
409        map:moveUp()
410      elseif keys == 's' then
411        map:moveDown()
412      elseif keys == 'a' then
413        map:moveLeft()
414      elseif keys == 'd' then
415        map:moveRight()
416      end
417    end
418
419    -- Local functions
420    local function updateMap()
421      map:generateMap()
422      map:draw()
423    end
424
425    -- Main loop
426    while true do
427      updateMap()
428      frame:update()
429      frame:render()
430    end
431  end
432
433  function tileMap(width, height)
434    -- The actual generation classes
435    local Map = require('classest.map')
436    local Draw = require('classest.draw')
437    local Class = require('classest.class')
438    local frame = require('classest.state.state')
439    local frame = Class:create('frame')
440
441    -- Global values
442    maxCoordinate, maxCoordinateX = 800, 800
443
444    -- Local variables
445    tileWidth, tileHeight, tileMargin = 20, 24
446    constMargin, constMarginX = coh.w, coh.b
447    input, map
448
449    -- Handle input
450    function input:onInput(keys)
451      if keys == 'w' then
452        map:moveUp()
453      elseif keys == 's' then
454        map:moveDown()
455      elseif keys == 'a' then
456        map:moveLeft()
457      elseif keys == 'd' then
458        map:moveRight()
459      end
460    end
461
462    -- Local functions
463    local function updateMap()
464      map:generateMap()
465      map:draw()
466    end
467
468    -- Main loop
469    while true do
470      updateMap()
471      frame:update()
472      frame:render()
473    end
474  end
475
476  function tileMap(width, height)
477    -- The actual generation classes
478    local Map = require('classest.map')
479    local Draw = require('classest.draw')
480    local Class = require('classest.class')
481    local frame = require('classest.state.state')
482    local frame = Class:create('frame')
483
484    -- Global values
485    maxCoordinate, maxCoordinateX = 800, 800
486
487    -- Local variables
488    tileWidth, tileHeight, tileMargin = 20, 24
489    constMargin, constMarginX = coh.w, coh.b
490    input, map
491
492    -- Handle input
493    function input:onInput(keys)
494      if keys == 'w' then
495        map:moveUp()
496      elseif keys == 's' then
497        map:moveDown()
498      elseif keys == 'a' then
499        map:moveLeft()
500      elseif keys == 'd' then
501        map:moveRight()
502      end
503    end
504
505    -- Local functions
506    local function updateMap()
507      map:generateMap()
508      map:draw()
509    end
510
511    -- Main loop
512    while true do
513      updateMap()
514      frame:update()
515      frame:render()
516    end
517  end
518
519  function tileMap(width, height)
520    -- The actual generation classes
521    local Map = require('classest.map')
522    local Draw = require('classest.draw')
523    local Class = require('classest.class')
524    local frame = require('classest.state.state')
525    local frame = Class:create('frame')
526
527    -- Global values
528    maxCoordinate, maxCoordinateX = 800, 800
529
530    -- Local variables
531    tileWidth, tileHeight, tileMargin = 20, 24
532    constMargin, constMarginX = coh.w, coh.b
533    input, map
534
535    -- Handle input
536    function input:onInput(keys)
537      if keys == 'w' then
538        map:moveUp()
539      elseif keys == 's' then
540        map:moveDown()
541      elseif keys == 'a' then
542        map:moveLeft()
543      elseif keys == 'd' then
544        map:moveRight()
545      end
546    end
547
548    -- Local functions
549    local function updateMap()
550      map:generateMap()
551      map:draw()
552    end
553
554    -- Main loop
555    while true do
556      updateMap()
557      frame:update()
558      frame:render()
559    end
560  end
561
562  function tileMap(width, height)
563    -- The actual generation classes
564    local Map = require('classest.map')
565    local Draw = require('classest.draw')
566    local Class = require('classest.class')
567    local frame = require('classest.state.state')
568    local frame = Class:create('frame')
569
570    -- Global values
571    maxCoordinate, maxCoordinateX = 800, 800
572
573    -- Local variables
574    tileWidth, tileHeight, tileMargin = 20, 24
575    constMargin, constMarginX = coh.w, coh.b
576    input, map
577
578    -- Handle input
579    function input:onInput(keys)
580      if keys == 'w' then
581        map:moveUp()
582      elseif keys == 's' then
583        map:moveDown()
584      elseif keys == 'a' then
585        map:moveLeft()
586      elseif keys == 'd' then
587        map:moveRight()
588      end
589    end
590
591    -- Local functions
592    local function updateMap()
593      map:generateMap()
594      map:draw()
595    end
596
597    -- Main loop
598    while true do
599      updateMap()
600      frame:update()
601      frame:render()
602    end
603  end
604
605  function tileMap(width, height)
606    -- The actual generation classes
607    local Map = require('classest.map')
608    local Draw = require('classest.draw')
609    local Class = require('classest.class')
610    local frame = require('classest.state.state')
611    local frame = Class:create('frame')
612
613    -- Global values
614    maxCoordinate, maxCoordinateX = 800, 800
615
616    -- Local variables
617    tileWidth, tileHeight, tileMargin = 20, 24
618    constMargin, constMarginX = coh.w, coh.b
619    input, map
620
621    -- Handle input
622    function input:onInput(keys)
623      if keys == 'w' then
624        map:moveUp()
625      elseif keys == 's' then
626        map:moveDown()
627      elseif keys == 'a' then
628        map:moveLeft()
629      elseif keys == 'd' then
630        map:moveRight()
631      end
632    end
633
634    -- Local functions
635    local function updateMap()
636      map:generateMap()
637      map:draw()
638    end
639
640    -- Main loop
641    while true do
642      updateMap()
643      frame:update()
644      frame:render()
645    end
646  end
647
648  function tileMap(width, height)
649    -- The actual generation classes
650    local Map = require('classest.map')
651    local Draw = require('classest.draw')
652    local Class = require('classest.class')
653    local frame = require('classest.state.state')
654    local frame = Class:create('frame')
655
656    -- Global values
657    maxCoordinate, maxCoordinateX = 800, 800
658
659    -- Local variables
660    tileWidth, tileHeight, tileMargin = 20, 24
661    constMargin, constMarginX = coh.w, coh.b
662    input, map
663
664    -- Handle input
665    function input:onInput(keys)
666      if keys == 'w' then
667        map:moveUp()
668      elseif keys == 's' then
669        map:moveDown()
670      elseif keys == 'a' then
671        map:moveLeft()
672      elseif keys == 'd' then
673        map:moveRight()
674      end
675    end
676
677    -- Local functions
678    local function updateMap()
679      map:generateMap()
680      map:draw()
681    end
682
683    -- Main loop
684    while true do
685      updateMap()
686      frame:update()
687      frame:render()
688    end
689  end
690
691  function tileMap(width, height)
692    -- The actual generation classes
693    local Map = require('classest.map')
694    local Draw = require('classest.draw')
695    local Class = require('classest.class')
696    local frame = require('classest.state.state')
697    local frame = Class:create('frame')
698
699    -- Global values
700    maxCoordinate, maxCoordinateX = 800, 800
701
702    -- Local variables
703    tileWidth, tileHeight, tileMargin = 20, 24
704    constMargin, constMarginX = coh.w, coh.b
705    input, map
706
707    -- Handle input
708    function input:onInput(keys)
709      if keys == 'w' then
710        map:moveUp()
711      elseif keys == 's' then
712        map:moveDown()
713      elseif keys == 'a' then
714        map:moveLeft()
715      elseif keys == 'd' then
716        map:moveRight()
717      end
718    end
719
720    -- Local functions
721    local function updateMap()
722      map:generateMap()
723      map:draw()
724    end
725
726    -- Main loop
727    while true do
728      updateMap()
729      frame:update()
730      frame:render()
731    end
732  end
733
734  function tileMap(width, height)
735    -- The actual generation classes
736    local Map = require('classest.map')
737    local Draw = require('classest.draw')
738    local Class = require('classest.class')
739    local frame = require('classest.state.state')
740    local frame = Class:create('frame')
741
742    -- Global values
743    maxCoordinate, maxCoordinateX = 800, 800
744
745    -- Local variables
746    tileWidth, tileHeight, tileMargin = 20, 24
747    constMargin, constMarginX = coh.w, coh.b
748    input, map
749
750    -- Handle input
751    function input:onInput(keys)
752      if keys == 'w' then
753        map:moveUp()
754      elseif keys == 's' then
755        map:moveDown()
756      elseif keys == 'a' then
757        map:moveLeft()
758      elseif keys == 'd' then
759        map:moveRight()
760      end
761    end
762
763    -- Local functions
764    local function updateMap()
765      map:generateMap()
766      map:draw()
767    end
768
769    -- Main loop
770    while true do
771      updateMap()
772      frame:update()
773      frame:render()
774    end
775  end
776
777  function tileMap(width, height)
778    -- The actual generation classes
779    local Map = require('classest.map')
780    local Draw = require('classest.draw')
781    local Class = require('classest.class')
782    local frame = require('classest.state.state')
783    local frame = Class:create('frame')
784
785    -- Global values
786    maxCoordinate, maxCoordinateX = 800, 800
787
788    -- Local variables
789    tileWidth, tileHeight, tileMargin = 20, 24
790    constMargin, constMarginX = coh.w, coh.b
791    input, map
792
793    -- Handle input
794    function input:onInput(keys)
795      if keys == 'w' then
796        map:moveUp()
797      elseif keys == 's' then
798        map:moveDown()
799      elseif keys == 'a' then
800        map:moveLeft()
801      elseif keys == 'd' then
802        map:moveRight()
803      end
804    end
805
806    -- Local functions
807    local function updateMap()
808      map:generateMap()
809      map:draw()
810    end
811
812    -- Main loop
813    while true do
814      updateMap()
815      frame:update()
816      frame:render()
817    end
818  end
819
820  function tileMap(width, height)
821    -- The actual generation classes
822    local Map = require('classest.map')
823    local Draw = require('classest.draw')
824    local Class = require('classest.class')
825    local frame = require('classest.state.state')
826    local frame = Class:create('frame')
827
828    -- Global values
829    maxCoordinate, maxCoordinateX = 800, 800
830
831    -- Local variables
832    tileWidth, tileHeight, tileMargin = 20, 24
833    constMargin, constMarginX = coh.w, coh.b
834    input, map
835
836    -- Handle input
837    function input:onInput(keys)
838      if keys == 'w' then
839        map:moveUp()
840      elseif keys == 's' then
841        map:moveDown()
842      elseif keys == 'a' then
843        map:moveLeft()
844      elseif keys == 'd' then
845        map:moveRight()
846      end
847    end
848
849    -- Local functions
850    local function updateMap()
851      map:generateMap()
852      map:draw()
853    end
854
855    -- Main loop
856    while true do
857      updateMap()
858      frame:update()
859      frame:render()
860    end
861  end
862
863  function tileMap(width, height)
864    -- The actual generation classes
865    local Map = require('classest.map')
866    local Draw = require('classest.draw')
867    local Class = require('classest.class')
868    local frame = require('classest.state.state')
869    local frame = Class:create('frame')
870
871    -- Global values
872    maxCoordinate, maxCoordinateX = 800, 800
873
874    -- Local variables
875    tileWidth, tileHeight, tileMargin = 20, 24
876    constMargin, constMarginX = coh.w, coh.b
877    input, map
878
879    -- Handle input
880    function input:onInput(keys)
881      if keys == 'w' then
882        map:moveUp()
883      elseif keys == 's' then
884        map:moveDown()
885      elseif keys == 'a' then
886        map:moveLeft()
887      elseif keys == 'd' then
888        map:moveRight()
889      end
890    end
891
892    -- Local functions
893    local function updateMap()
894      map:generateMap()
895      map:draw()
896    end
897
898    -- Main loop
899    while true do
900      updateMap()
901      frame:update()
902      frame:render()
903    end
904  end
905
906  function tileMap(width, height)
907    -- The actual generation classes
908    local Map = require('classest.map')
909    local Draw = require('classest.draw')
910    local Class = require('classest.class')
911    local frame = require('classest.state.state')
912    local frame = Class:create('frame')
913
914    -- Global values
915    maxCoordinate, maxCoordinateX = 800, 800
916
917    -- Local variables
918    tileWidth, tileHeight, tileMargin = 20, 24
919    constMargin, constMarginX = coh.w, coh.b
920    input, map
921
922    -- Handle input
923    function input:onInput(keys)
924      if keys == 'w' then
925        map:moveUp()
926      elseif keys == 's' then
927        map:moveDown()
928      elseif keys == 'a' then
929        map:moveLeft()
930      elseif keys == 'd' then
931        map:moveRight()
932      end
933    end
934
935    -- Local functions
936    local function updateMap()
937      map:generateMap()
938      map:draw()
939    end
940
941    -- Main loop
942    while true do
943      updateMap()
944      frame:update()
945      frame:render()
946    end
947  end
948
949  function tileMap(width, height)
950    -- The actual generation classes
951    local Map = require('classest.map')
952    local Draw = require('classest.draw')
953    local Class = require('classest.class')
954    local frame = require('classest.state.state')
955    local frame = Class:create('frame')
956
957    -- Global values
958    maxCoordinate, maxCoordinateX = 800, 800
959
960    -- Local variables
961    tileWidth, tileHeight, tileMargin = 20, 24
962    constMargin, constMarginX = coh.w, coh.b
963    input, map
964
965    -- Handle input
966    function input:onInput(keys)
967      if keys == 'w' then
968        map:moveUp()
969      elseif keys == 's' then
970        map:moveDown()
971      elseif keys == 'a' then
972        map:moveLeft()
973      elseif keys == 'd' then
974        map:moveRight()
975      end
976    end
977
978    -- Local functions
979    local function updateMap()
980      map:generateMap()
981      map:draw()
982    end
983
984    -- Main loop
985    while true do
986      updateMap()
987      frame:update()
988      frame:render()
989    end
990  end
991
992  function tileMap(width, height)
993    -- The actual generation classes
994    local Map = require('classest.map')
995    local Draw = require('classest.draw')
996    local Class = require('classest.class')
997    local frame = require('classest.state.state')
998    local frame = Class:create('frame')
999
1000   -- Global values
1001   maxCoordinate, maxCoordinateX = 800, 800
1002
1003   -- Local variables
1004   tileWidth, tileHeight, tileMargin = 20, 24
1005   constMargin, constMarginX = coh.w, coh.b
1006   input, map
1007
1008   -- Handle input
1009   function input:onInput(keys)
1010     if keys == 'w' then
1011       map:moveUp()
1012     elseif keys == 's' then
1013       map:moveDown()
1014     elseif keys == 'a' then
1015       map:moveLeft()
1016     elseif keys == 'd' then
1017       map:moveRight()
1018     end
1019   end
1020
1021   -- Local functions
1022   local function updateMap()
1023     map:generateMap()
1024     map:draw()
1025   end
1026
1027   -- Main loop
1028   while true do
1029     updateMap()
1030     frame:update()
1031     frame:render()
1032   end
1033 end
1034
1035  function tileMap(width, height)
1036    -- The actual generation classes
1037    local Map = require('classest.map')
1038    local Draw = require('classest.draw')
1039    local Class = require('classest.class')
1040    local frame = require('classest.state.state')
1041    local frame = Class:create('frame')
1042
1043    -- Global values
1044    maxCoordinate, maxCoordinateX = 800, 800
1045
1046    -- Local variables
1047    tileWidth, tileHeight, tileMargin = 20, 24
1048    constMargin, constMarginX = coh.w, coh.b
1049    input, map
1050
1051    -- Handle input
1052    function input:onInput(keys)
1053      if keys == 'w' then
1054        map:moveUp()
1055      elseif keys == 's' then
1056        map:moveDown()
1057      elseif keys == 'a' then
1058        map:moveLeft()
1059      elseif keys == 'd' then
1060        map:moveRight()
1061      end
1062    end
1063
1064    -- Local functions
1065    local function updateMap()
1066      map:generateMap()
1067      map:draw()
1068    end
1069
1070    -- Main loop
1071    while true do
1072      updateMap()
1073      frame:update()
1074      frame:render()
1075    end
1076  end
1077
1078  function tileMap(width, height)
1079    -- The actual generation classes
1080    local Map = require('classest.map')
1081    local Draw = require('classest.draw')
1082    local Class = require('classest.class')
1083    local frame = require('classest.state.state')
1084    local frame = Class:create('frame')
1085
1086    -- Global values
1087    maxCoordinate, maxCoordinateX = 800, 800
1088
1089    -- Local variables
1090    tileWidth, tileHeight, tileMargin = 20, 24
1091    constMargin, constMarginX = coh.w, coh.b
1092    input, map
1093
1094    -- Handle input
1095    function input:onInput(keys)
1096      if keys == 'w' then
1097        map:moveUp()
1098      elseif keys == 's' then
1099        map:moveDown()
1100      elseif keys == 'a' then
1101        map:moveLeft()
1102      elseif keys == 'd' then
1103        map:moveRight()
1104      end
1105    end
1106
1107    -- Local functions
1108    local function updateMap()
1109      map:generateMap()
1110      map:draw()
1111    end
1112
1113    -- Main loop
1114    while true do
1115      updateMap()
1116      frame:update()
1117      frame:render()
1118    end
1119  end
1120
1121  function tileMap(width, height)
1122    -- The actual generation classes
1123    local Map = require('classest.map')
1124    local Draw = require('classest.draw')
1125    local Class = require('classest.class')
1126    local frame = require('classest.state.state')
1127    local frame = Class:create('frame')
1128
1129    -- Global values
1130    maxCoordinate, maxCoordinateX = 800, 800
1131
1132    -- Local variables
1133    tileWidth, tileHeight, tileMargin = 20, 24
1134    constMargin, constMarginX = coh.w, coh.b
1135    input, map
1136
1137    -- Handle input
1138    function input:onInput(keys)
1139      if keys == 'w' then
1140        map:moveUp()
1141      elseif keys == 's' then
1142        map:moveDown()
1143      elseif keys == 'a' then
1144        map:moveLeft()
1145      elseif keys == 'd' then
1146        map:moveRight()
1147      end
1148    end
1149
1150    -- Local functions
1151    local function updateMap()
1152      map:generateMap()
1153      map:draw()
1154    end
1155
1156    -- Main loop
1157    while true do
1158      updateMap()
1159      frame:update()
1160      frame:render()
1161    end
1162  end
1163
1164  function tileMap(width, height)
1165    -- The actual generation classes
1166    local Map = require('classest.map')
1167    local Draw = require('classest.draw')
1168    local Class = require('classest.class')
1169    local frame = require('classest.state.state')
1170    local frame = Class:create('frame')
1171
1172    -- Global values
1173    maxCoordinate, maxCoordinateX = 800, 800
1174
1175    -- Local variables
1176    tileWidth, tileHeight, tileMargin = 20, 24
1177    constMargin, constMarginX = coh.w, coh.b
1178    input, map
1179
1180    -- Handle input
1181    function input:onInput(keys)
1182      if keys == 'w' then
1183        map:moveUp()
1184      elseif keys == 's' then
1185        map:moveDown()
1186      elseif keys == 'a' then
1187        map:moveLeft()
1188      elseif keys == 'd' then
1189        map:moveRight()
1190      end
1191    end
1192
1193    -- Local functions
1194    local function updateMap()
1195      map:generateMap()
1196      map:draw()
1197    end
1198
1199    -- Main loop
1200    while true do
1201      updateMap()
1202      frame:update()
1203      frame:render()
1204    end
1205  end
1206
1207  function tileMap(width, height)
1208    -- The actual generation classes
1209    local Map = require('classest.map')
1210    local Draw = require('classest.draw')
1211    local Class = require('classest.class')
1212    local frame = require('classest.state.state')
1213    local frame = Class:create('frame')
1214
1215    -- Global values
1216    maxCoordinate, maxCoordinateX = 800, 800
1217
1218    -- Local variables
1219    tileWidth, tileHeight, tileMargin = 20, 24
1220    constMargin, constMarginX = coh.w, coh.b
1221    input, map
1222
1223    -- Handle input
1224    function input:onInput(keys)
1225      if keys == 'w' then
1226        map:moveUp()
1227      elseif keys == 's' then
1228        map:moveDown()
1229      elseif keys == 'a' then
1230        map:moveLeft()
1231      elseif keys == 'd' then
1232        map:moveRight()
1233      end
1234    end
1235
1236    -- Local functions
1237    local function updateMap()
1238      map:generateMap()
1239      map:draw()
1240    end
1241
1242    -- Main loop
1243    while true do
1244      updateMap()
1245      frame:update()
1246      frame:render()
1247    end
1248  end
1249
1250  function tileMap(width, height)
1251    -- The actual generation classes
1252    local Map = require('classest.map')
1253    local Draw = require('classest.draw')
1254    local Class = require('classest.class')
1255    local frame = require('classest.state.state')
1256    local frame = Class:create('frame')
1257
1258    -- Global values
1259    maxCoordinate, maxCoordinateX = 800, 800
1260
1261    -- Local variables
1262    tileWidth, tileHeight, tileMargin = 20, 24
1263    constMargin, constMarginX = coh.w, coh.b
1264    input, map
1265
1266    -- Handle input
1267    function input:onInput(keys)
1268      if keys == 'w' then
1269        map:moveUp()
1270      elseif keys == 's' then
1271        map:moveDown()
1272      elseif keys ==
```

The Vim Talk

Important Concepts

Windows, Tabs, and Buffers

Illustration

Illustration



- He's an image that may help illustrate the differences between the three
- The three code blocks in the top row are the three buffers that are currently active
- The red and blue boxes in the top row show where the windows are viewing the buffers. Note that a single buffer can have multiple windows viewing it.
- The bottom row shows how the windows are combined into tabs - the color around the code corresponds to the windows in the top row

Buffers

- Buffer states:
 - Active
 - Hidden
 - Inactive

The Vim Talk

Important Concepts

Windows, Tabs, and Buffers

Buffers

Buffers

- Buffer states:
 - Active
 - Hidden
 - Inactive

- In my opinion, buffers are the hardest of the three to grasp, since you can't necessarily see them all the time
- The first thing to understand about buffers is that they are just a file in memory - nothing more, nothing less. They do not need to be visible.
- In fact, buffers are given different statuses based on their visibility:
- Buffers that are currently in a window are marked as active
- Buffers that are not displayed but are loaded are hidden
- Buffers that are not displayed and not loaded are inactive

Buffers

- Buffer states:
 - Active
 - Hidden
 - Inactive
- Navigating buffers:
 - `:buffers`
 - `:bnext`
 - `:buff <name>`

See `:help :buffers` and `:help buffer-hidden` for more

The Vim Talk

Important Concepts

Windows, Tabs, and Buffers

Buffers

Buffers

- Buffer states:
 - Active
 - Hidden
 - Inactive

- Navigating buffers:
 - :buffers
 - :bnext
 - :buff <name>

See :help :buffers and :help buffer-hidden for more

- The ability to navigate buffers quickly and easily is what makes them so powerful
- You can use :buffers to view the current buffers list
- Using that information, you can use :bnext and :bprev to move around between buffers
- Each buffer is also numbered - you can use :buff N, where N is that buffer's number, to move to it
- The most useful way to navigate buffers, however, is using :buff <name>, where <name> is at least a partial match for the file you want that is uniquely identifying

Windows

- Creating windows:

- CTRL-W n
- CTRL-W s
- CTRL-W v
- :new
- :split
- :vsplit

The Vim Talk

Important Concepts

Windows, Tabs, and Buffers

Windows

Windows

- Creating windows:
 - `CTRL-W n`
 - `CTRL-W s`
 - `CTRL-W v`
- `new`
- `split`
- `zsplit`

- In order to use windows, you have to be able to create them
- Most all Window commands are prefixed by pressing control w
- These all also have corresponding commands as well
- `CTLR-W n` creates a new window
- `CTLR-W s` splits the current window horizontally
- `CTLR-W v` splits the current window vertically

Windows

- Creating windows:
 - `CTRL-W n`
 - `CTRL-W s`
 - `CTRL-W v`
 - `:new`
 - `:split`
 - `:vsplit`
- Navigating / moving windows:
 - `CTRL-W h, j, k, l`
 - `CTRL-W H, J, K, L`

See `:help windows` for more

The Vim Talk

Important Concepts

Windows, Tabs, and Buffers

Windows

Windows

- Creating windows:
 - `CTRL-W n`
 - `CTRL-W s`
 - `CTRL-W v`
 - `cnew`
 - `csplit`
 - `cwsplit`
- Navigating / moving windows:
 - `CTRL-W h, j, k, l`
 - `CTRL-W H, J, K, L`

See `:help windows` for more

- `CTRL-W` followed by `h, j, k, or l` changes which window you're currently selecting in that direction
- `CTRL-W` followed by capital `H, J, K, or L` the window you're currently selecting as much as possible in that direction

Tabs

- Creating / closing tabs:
 - `:tabnew`
 - `:tabclose`

The Vim Talk

Important Concepts

Windows, Tabs, and Buffers

Tabs

Tabs

- Creating / closing tabs:
 - :tabnew
 - :tabclose

- Using tabs is pretty straightforward
- `:tabnew` is used to create a new tab
- `:tabclose` is used to close a new tab
- `:tabs` lists the current tabs and the windows they're using
- `gt` and `gT` are used to navigate back and forth between tabs
- As you can see, navigation with tabs is not as fluid as with buffers or windows, which is why they are generally recommended over tabs

Tabs

- Creating / closing tabs:
 - `:tabnew`
 - `:tabclose`
- Navigating tabs:
 - `:tabs`
 - `gt` and `gT`

See `:help tabpage` for more

Demonstration

```

7 initial
6 begin
5 $monitor ( $time,
4     + $s10d10d + $d10d10d + $d10d10d,
3     in[32], in[31:28], in[27:24], in[23:20], in[19:16],
2     in[15:12], in[11:8], in[7:4], in[3:0],
1     out[16], out[15:12], out[11:8], out[7:4], out[3:0],
0 );
1
2 // 0 + 2456 + 7890 = 111348
3 in = 33'b0____0011_0100_0101_0110____0111_1000_1001_0000;
4 $10;
5
6 // 1 + 3456 + 7890 = 111347
7 in = 33'b1____0011_0100_0101_0110____0111_1000_1001_0000;
8 $10;
9
10 // 0 + 1234 + 1235 = 02469
11 in = 33'b0____0001_0010_0011_0100____0001_0010_0011_0101;
12 $10;
13
14 // 1 + 1234 + 1235 = 02470
15 in = 33'b1____0001_0010_0011_0100____0001_0010_0011_0101;
16 $10;
17
18 // 0 + 9999 + 9999 = 19998
19 in = 33'b0____1001_1001_1001_1001____1001_1001_1001_1001;
20 $10;
21
22 // 1 + 9999 + 9999 = 19999
23 in = 33'b1____1001_1001_1001_1001____1001_1001_1001_1001;
24 $10;
25
26 // 0 + 0000 + 0000 = 00000
27 in = 33'b0____0000_0000_0000____0000_0000_0000_0000;
28 $10;
29
30 // 1 + 0000 + 0000 = 00001
31 in = 33'b1____0000_0000_0000____0000_0000_0000_0000;
32 $10;
33
34 // 0 + 0500 + 0500 = 01000
35
36 w4d_qlc.v
37 buffers
38 "adder_4.v"           line 1
39 "full_adder_bcd.v"    line 0
40 "full_adder.v"         line 0
41 "get_carry.v"          line 0
42 "w4d_qlb.v"            line 0
43 "%w4d_qlc.v"           line 15

```

15,12-26 Top|full_adder_bcd.v 12,1 All

Figure 4: Buffers, windows, and tabs

The Vim Talk

└ Important Concepts

└ Windows, Tabs, and Buffers

└ Demonstration

Demonstration



Figure 4: Buffers, windows, and tabs

- Now I'll walk through a demonstration to hopefully explain some of the differences between tabs and buffers
- In the image, you can see there are 6 buffers, 3 of which are currently active
- There are 3 windows open currently
- Show navigating buffers

Registers

- Copy / paste / cut:
 - Copy: "xy<motion>
 - Cut: "xd<motion>
 - Paste: "xp

The Vim Talk



Registers

- Copy / paste / cut:
 - Copy: "ayymotion>
 - Cut: "ad'motion>
 - Paste: "xp

- Registers can be used for copying and pasting text
- Think of it as having over 40 different clipboards to copy and paste with
- You can copy text into a register by specifying the register (here, x), then y for yank, then the motion for capturing text
- Cutting text is similar to copying, but with commands that remove text, like d or c
- Pasting text from a register is done with p

Registers

- Copy / paste / cut:
 - Copy: "xy<motion>
 - Cut: "xd<motion>
 - Paste: "xp
- Names:
 - Unnamed register: ""
 - Numbered registers: "0 - "9
 - Named registers: "a - "z or "A - "Z

See :help registers for more

The Vim Talk

Important Concepts

Registers / Macros

Registers

Registers

- Copy / paste / cut:
 - Copy: "`ayymotion`"
 - Cut: "`ad'motion`"
 - Paste: "`xp`"
- Names:
 - Unnamed register: "`"`"
 - Numbered registers: "`0` - "`9`"
 - Named registers: "`a` - "`z`" or "`A` - "`Z`"

See `:help registers` for more

- There are tons of registers - some of them are more useful than others. All registers are prefix with a double quote.
- The unnamed register is where text deleted or yanked in normal mode go into this register
- The most recently yanked or deleted text goes to register 0, and the text that used to be in "0 goes to "1 and so on
- Named registers only get text when you specify that the text should go in there. Use a lowercase letter to replace the text that used to be in the register, or a capital letter to append the text to the register.

Macros

- What is a macro?
- Creating a macro:
 - Start: `q<register>`
 - Stop: `q`
- Using a macro:
 - Play: `@<register>`
 - Replay: `@@`
- Macros use registers

See :help recording for more

The Vim Talk

Important Concepts

Registers / Macros

Macros

Macros

- What is a macro?
- Creating a macro:
 - Start: `q@register>`
 - Stop: `q`
- Using a macro:
 - Play: `@register`
 - Replay: `@@`
- Macros use registers

See `:help recording` for more

- Macros are used to record actions to recreate complex movements
- To create a macro, press `q` and then a letter
- Do whatever actions you'd like to be recorded, then press `q` to quit
- To play back a macro, use `@` and the register used
- You can repeat the last macro by pressing `@@`
- Macros are actually tightly linked with registers - they're used to store the macro's content
- You can view and modify a macro by putting / yanking into a reg
- Because macros use registers, and because using a capital letter for a register appends, you can append to a macro by using a capital letter
- This can be useful - for instance if you realize you forgot to include part of the macro but don't want to rerecord it

Folds

- Hide code
- Methods:
 - Indent
 - Syntax
 - Marker - {{{<n>} and }}}<n>
- Usage:
 - Create a fold: zf<motion>
 - Delete a fold: zd
 - Open a fold: zo
 - Close a fold: zc
 - Close all folds:zM
 - Open all folds: zR

See :help folds for more

The Vim Talk

Important Concepts

- Folds
 - Folds

Folds

- Hide code
- Methods:
 - Indent
 - Syntax
 - Marker - {{{<n> and }}}<n>
- Usage:
 - Create a fold: zf:motion
 - Delete a fold: zd
 - Open a fold: zo
 - Close a fold: zc
 - Close all folds: zR
 - Open all folds: zH

See :help folds for more

- Folds are useful tools used for hiding sections of code that you don't always want to see
- There are many ways of defining folds - the most useful are by indentation, methods defined by the syntax file, or manually, by inserting curly braces
- The number, n, is not required - you can also define nested folds with the marker method by having a number after the braces
- My preferred method is with markers, since I have more control over where and how the folds are defined, and it allows for others to have the same folds as you, plus you can manually set the fold level
- Using folds is easy - to create a fold, press zf, then a motion for all the text you want to be folded
- Pressing zd when inside of a fold allows you to delete that fold, and so on

Commands

- Types of commands:
 - Ex commands: :
 - Search patterns: / and ?
 - Filter commands: !
- Example commands:
 - :write, :help, :quit
 - /asdf
 - !cat file.txt

See :help cmdline for more

The Vim Talk

Important Concepts

Commands and Ranges

Commands

Commands

- Types of commands:
 - Ex commands: : !
 - Search patterns: / ?
 - Filter commands: !

- Example commands:
 - :write, :help, :quit
 - ./sdf
 - :!cat file.txt

See :help cmdline for more

- There are three main types of commands
- “Ex” commands, which are prefixed with colon, search commands, which are prefixed with slash or question mark, and filter commands, which are prefixed with an exclamation point
- Most of us are familiar with the most common commands, like write, quit, help, etc.
- Most of us should also be familiar with searching - slash searches forwards and question mark searches backwards
- An interesting little tidbit is that slash has come to mean search in many areas - on Twitter and GitHub, for instance, pressing slash brings you to the search bar
- Finally, there are filters. Filters allow you to run external commands from the command line to alter your file.

Command navigation

- History
- Separated for types
- Partial matches

The Vim Talk

Important Concepts

Commands and Ranges

Command navigation

Command navigation

- History
- Separated for types
- Partial matches

- You can use the up/down arrow keys to go through the command history
- Vim keeps a separate history for all of the different kinds of commands - this means searching for something doesn't interfere with your ex commands, for instance
- If you start typing something in the command window and then press up or down, only partial matches from your history will be shown

Ranges

- Ranges specify section of buffer for commands
 - `n,m:command`
- Specifying ranges:
 - Line numbers: numbers
 - Current line: .
 - Last line: \$
 - Entire file: %
- E.g. `:%sort` to alphabetize entire file

See `:help [range]` for more

The Vim Talk

Important Concepts

Commands and Ranges

Ranges

Ranges

- Ranges specify section of buffer for commands
 - `n,m`: command
 - Specifying ranges:
 - Line numbers: `numbers`
 - Current line: `.`
 - Last line: `$`
 - Entire file: `%`
 - E.g. :`Xsort` to alphabetize entire file
- See :help [range] for more

- Ranges prefix a command, and are used to specify a region of the current buffer for the command to operate on
- There are many ways to specify ranges
- The simplest is by using the line numbers you want
- You can also make the range relative to where the cursor currently is with a period
- The dollar sign and percent sign can also be used
- For instance, you can take advantage of Vim's sort command to sort an entire file like so

Text objects

- Used for selecting text
- Objects:
 - Word: w
 - Parenthesis: (or)
 - Quotes: "
 - HTML/XML tags: t
- Selections:
 - “A(n) <object>” : a<object>
 - “Inner <object>” : i<object>

See :help text-objects for more

The Vim Talk

- └ Important Concepts
 - └ Text Objects
 - └ Text objects

Text objects

- Used for selecting text
- Objects:
 - Word: w
 - Parenthesis: { or }
 - Quotes: " or ''
 - HTML/XML tags: t
- Selections:
 - "A[n] object<>" : a<object>
 - "In[n] object<>" : i<object>

See :help text-objects for more

- Most of us are likely familiar with text motions - w, W, e, etc. - they are used for navigating through text
- Text objects are similar to motions, but for **selecting** text instead
- There are several different objects you can use - w for words, double quotes for quotes, etc.
- One of the coolest is t for HTML tags
- You can select these objects with “a” for a/an, or “i” for inner
- The difference between these two being “a” includes surrounding white space while “i” does not

Moving your cursor / view

- Move current line:
 - To top of window: zt
 - To middle of window: zz
 - To bottom of window: zb
 - Up one line: CTRL-E
 - Down one line: CTRL-Y
- Move cursor to:
 - Top of window: H
 - Middle of window: M
 - Bottom of window: L

See :help scroll-cursor for more

The Vim Talk

Advanced Applications

Buffers / Windows / Tabs

Moving your cursor / view

Moving your cursor / view

- Move current line:
 - To top of window: zt
 - To middle of window: zz
 - To bottom of window: zb
 - Up one line: CTRL-E
 - Down one line: CTRL-Y

- Move cursor to:
 - Top of window: H
 - Middle of window: M
 - Bottom of window: L

See :help scroll-cursor for more

- Being able to quickly navigate with your cursor is another valuable skill to know
- zt, zz, and zb can be used to reposition the current line you're viewing within the window, as can CTRL-E and CTRL-Y
- Note that you should be careful with zz! If you're holding down shift or have cap lock on, ZZ saves and closes the file!
- Additionally, you can move the cursor relative to the Window using H, M, and L

Opening Files

- Opening files:
 - In current window: `gf`
 - In new window: `CTRL-W f`
 - In new tab: `CTRL-W gf`
 - At specific line: `<motion>F`

The Vim Talk



Opening Files

- Opening files
 - In current window: `gf`
 - In other windows: `CTRL-W f`
 - In new tab: `CTRL-W gf`
 - At specific line: `motion>F`

- If you're lazy like me, you prefer to avoid typing as much as possible - that includes file names
- To open the file in the current window, you can use `gf`. You can think of `gf` as being short for "goto file"
- For each of these motions, you can also use a capital `f`, which generally works the same as lowercase `f`, but if a number follows the file, the file will be opened at that line

Opening Files

- Opening files:
 - In current window: `gf`
 - In new window: `CTRL-W f`
 - In new tab: `CTRL-W gf`
 - At specific line: `<motion>F`
- Opening tabs: `:tab <command>`
 - `:tab help gf`
 - `:0tabnew file.txt, :$tabnew file.txt`
 - `:+tabnew file.txt, :$-tabnew file.txt`

The Vim Talk

Advanced Applications

Buffers / Windows / Tabs

Opening Files

Opening Files

- Opening file:
 - In current window: gf
 - In new window: CTRl-W F
 - In new tab: CTRl-W gf
 - At specific line: motion>F
- Opening tabs: :tab <command>
 - :tab help gf
 - :tabnew file.txt,:\$tabnew file.txt
 - :tabnew file.txt,:\$-tabnew file.txt

- Another useful tool is the tab command, which can be combined with any command that would normally open a new window to offer more flexibility
- For instance, running tab help gf opens the help file for gf in a new tab
- You can also use numbers and symbols to specify where your tab should go
- A number will make it that number tab, while a dollar sign will make it the last tab
- Pluses and minuses can be used to make the tab relative - if no number or symbol is given, the tab will be relative to the current tab

Marks

- Help with file navigation
- Set a mark: `m<letter>`
 - Local: lower
 - Global: capital
- Going to a mark:
 - Line of mark: '`<symbol>`
 - Location of mark: ``<symbol>`

See :help mark-motions for more

The Vim Talk



Marks

- Help with file navigation
- Set a mark: `m<letter>`
 - Local: lower
 - Global: capital
- Going to a mark:
 - Line of mark: `'<symbol>`
 - Location of mark: ``<symbol>`

See :help mark-motions for more

- Marks are useful tools that help with file navigation
- They can be made by pressing `m`, and then the letter you'd like to name the mark
- If you use a lowercase letter, the mark is local to the current buffer and can only be accessed while working in that buffer
- If you use an uppercase letter, the mark is global and can be accessed from any file, even if the buffer is not open
- You can go back to a mark that's been set by using either single quote or backtick
- Single quote moves you to the line of the mark, while backtick moves you to the exact location where the mark was made

Ranges

- Expanding: `.+1,$-5<command>`
- Searches: `?back?,/forth/<command>`
- Marks: `'a,'b<command>`
- , vs ;:
 - ; moves the cursor
 - , does not

See :help range and :help 10.3 for more

The Vim Talk

Advanced Applications

Ranges / Commands / Searching

Ranges

Ranges

- Expanding: `.+1,$-5command`
- Searches: `?back?,/Forth/<command>`
- Marks: `'a,,`b<command>`
- `,` vs `;`:
 - `;` moves the cursor
 - `,` does not

See :help range and :help 18.3 for more

- While you saw earlier that ranges can be based on lines and files, ranges can also do so much more
- For instance, you use numbers to expand the search range
- You can also use searches and even marks to define ranges
- There is one other aspect that I haven't mentioned yet. While all of my examples have used commas as the range separators, you can also use semicolons.
- The difference between these two is easier to understand with a visual, but basically comma moves the cursor and semicolon doesn't.

Range example

```
function counter( i )
    local a = 0

    do
        a = a + i
    end

    return function()
        a = a + 1
        return a
    end
end
```

```
:?^function?,/end/p
```

The Vim Talk

Advanced Applications

Ranges / Commands / Searching

Range example

Range example

```
function counter( l )
    local a = 0
    do
        a = a + 1
    end
    return function()
        a = a + 1
        return a
    end
end
```

:?^function?,/end/p

- So let's do an example to explain the difference between the two with an extremely contrived example
- Imagine your cursor is currently on the line with the arrow and you're performing the following command, which prints the lines that fall within the range between a line starting with the word "function" and the word "end"

Range example

```
function counter( i )           ←
  local a = 0

  do
    a = a + i
  end

  return function()
    a = a + 1
    return a
  end
end
```

```
:?^function?,/end/p
```

The Vim Talk

Advanced Applications

Ranges / Commands / Searching

Range example

Range example

```
function counter( i )
    local a = 0
    do
        a = a + 1
    end
    return function()
        a = a + 1
        return a
    end
end
```

```
:?^function?,/end/p
```

- If you perform that function, the indicated lines will be printed
- That's because the pattern looks for function at the start of a line before the cursor, then the “end” after the cursor

Range example

```
function counter( i )
    local a = 0

    do
        a = a + i
    end

    return function()
        a = a + 1
        return a
    end
end
```

```
:?^function?;/end/p
```

The Vim Talk

Advanced Applications

Ranges / Commands / Searching

Range example

Range example

```
function counter( l )
    local a = 0
    do
        a = a + 1
    end
    return function()
        a = a + 1
        return a
    end
end
```

:?^function?;/end/p

- Now let's return back to the original scenario, but this time using a semicolon instead of a comma

Range example

```
function counter( i )          ←
    local a = 0

    do
        a = a + i
    end           ←

    return function()
        a = a + 1
        return a
    end
end
```

```
:?^function?;/end/p
```

The Vim Talk

Advanced Applications

Ranges / Commands / Searching

Range example

Range example

```
function counter( i )
    local a = 0
    do
        a = a + i
    end
    return function()
        a = a + 1
        return a
    end
end
```

```
:?^function?;/end/p
```

- This time, these lines will be printed
- In contrast to the comma, this time the first occurrence of end after the pattern match is found

Inputting commands

- Matching:
 - Show matches: CTRL-D
 - Cycle through options: <tab> and <S-tab>
- Command window: q:, q/
 - Normal mode navigation / editing
 - Run commands in insert mode

See :help cmdline-completion and :help cmdline-window for more

The Vim Talk

Advanced Applications

Ranges / Commands / Searching

Inputting commands

Inputting commands

- Matching:
 - Show matches
 - Cycle through optionsCTRL-D
ctab/ and <S-tab>
- Command window: q!, q/
 - Normal mode navigation / editing
 - Run commands in insert mode

See :help cmdline-completion and :help cmdline-window for more

- As you're typing in the command window, some formats, like ex commands, have matching options available
- You can show all the available matches by pressing CTRL-D
- You can press tab or shift tab to autocomplete forwards or backwards
- Repeated tabs continue to cycle through the available options
- You can open a command window to see previous ex commands or searches by using q: or q/
- The command window shows the entire history of commands entered, and can be searched though / operated on like normal text
- To run the command or start the search, press enter on the line or go into insert mode on the desired line and hit enter

Searching

- Replace using last search: `:<range>s//<replace>`
- Replace using last search: `:<range>s/<p>/<r>/<flag>`
 - Confirm: c
 - Global: g
 - Ignore case: i

See `:help :substitute` and `:help pattern-atoms` for more

The Vim Talk

Advanced Applications

Ranges / Commands / Searching

Searching

Searching

- Replace using last search: `:change>s//<replace>`
- Replace using last search: `:change>s/<p>/<r>/<flag>`
 - Confirm: `c`
 - Global: `g`
 - Ignore case: `i`

See `:help :substitute` and `:help pattern-atoms` for more

- Searching has tons of parts to it. This slide has a few tips to searching that I found very helpful
- The first one is using the last search to perform a replacement
- When no search pattern is given, the last search is used instead
- This lets you test out a search first to see what it matches before performing the replacement
- The substitute command also takes flags - some of the more useful ones are “c”, which prompts for confirmation for each change, global, which does all substitutions in the range, and i, which ignores the case

Searching

- Replace using last search: `:<range>s//<replace>`
- Replace using last search: `:<range>s/<p>/<r>/<flag>`
 - Confirm: c
 - Global: g
 - Ignore case: i
- Useful patterns:
 - Word boundary: \< and \>
 - Matching: \(\) and \1, \2, ...

See :help :substitute and :help pattern-atoms for more

The Vim Talk

Advanced Applications

Ranges / Commands / Searching

Searching

Searching

- Replace using last search: `:change>s//<replace>`
- Replace using last search: `:change>s/<p>/<r>/<flag>`
 - Confirm: `c`
 - Global: `g`
 - Ignore case: `i`
- Useful patterns:
 - Word boundary: `\< and \>`
 - Matching: `\(\<\)\> and \(\<1\)\>, \(\<2\)\> ...`

See `:help :substitute` and `:help pattern-atoms` for more

- Patterns can get very complex - I certainly wouldn't call myself an expert on them by any means
- These patterns are pretty simple, though, can be very helpful
- The first two are first starting and stopping word boundaries
- Word boundaries are pretty much what they sound like - they come at the start and end of words and are useful for keeping partial matches from messing up your search
- Matching is also very useful - a back slash followed by (or) indicates what's called a "grouping," which can then be referred to in the replacement pattern by its number

Useful / interesting commands

- Man pages in Vim:
 - Open man page under cursor: K
 - Open man in Vim:
 - :runtime! ftplugin/man.vim
 - <leader>K or :Man xxx
- File to HTML: :TOhtml

The Vim Talk

Advanced Applications

Ranges / Commands / Searching

Useful / interesting commands

Useful / interesting commands

- Man pages in Vim:
 - Open man page under cursor: `K`
 - Open man page in Vim:
 - `:r runtime! ftplugin/man.vim`
 - `:leader>K` or `:Man xxx`
- File to HTML: `:TOhtml`

- This slide just has various things that I didn't think fit anywhere else really well, but were still cool / useful
- Vim has the ability to view man pages - capital K will open the man page of the word under the cursor in the terminal
- But you can also view the man pages within Vim
- First, run the following command (or add it to your `.vimrc` if you use it enough)
- Now you can use “leader K”, or use the `:Man` command, to open the man page within Vim

Useful / interesting commands

- Man pages in Vim:
 - Open man page under cursor: K
 - Open man in Vim:
 - :runtime! ftplugin/man.vim
 - <leader>K or :Man xxx
- File to HTML: :TOhtml
- View / change options: :options
- Special characters:
 - Current file: %
 - E.g. :cd %:h

See :help Man, :help cmdline-special, and :help filename-modifiers for more

The Vim Talk

Advanced Applications

Ranges / Commands / Searching

Useful / interesting commands

Useful / interesting commands

- Man pages in Vim:
 - Open man page under cursor: K
 - Open man in Vim:
 - :runtime! f#plugin/man.vim
 - :cleaderX or :Man xxx
- File to HTML: :TOhtml
- View / change options: :options
- Special characters:
 - Current file %
 - E.g. :cd %h

See :help Man, :help cmdline-special, and :help filename-modifiers for more

- Using the :options command, you can view and even change specific variables
- There are certain special characters that are replaced in the command line - the most important of which is percent, which represents the current buffer
- In combination with filename modifiers, you can easily do things like change Vim's current directory to be relative to the current buffer's directory

Normal mode

- Increment / decrement: **CTRL-A** and **CTRL-X**
 - Binary, octal, decimal, hex, alpha
 - See `:help nrformats` for more

¹ As of Vim 8, `:terminal` can be used to bring up a terminal in a new window

The Vim Talk



Normal mode

- Increment / decrement: CTRL-A and CTRL-X
 - Binary, octal, decimal, hex, alpha
 - See :help nrformats for more

¹ As of Vim 8, :terminal can be used to bring up a terminal in a new window

- Like the last slide, these are more useful things that don't quite fit anywhere else
- The first one is to increment with CTRL-A and decrement w/ CTRL-X
- This may seem like it's not very useful at first, but, combined with macros, you can use this command easily to make a list

Normal mode

- Increment / decrement: **CTRL-A** and **CTRL-X**
 - Binary, octal, decimal, hex, alpha
 - See `:help nrformats` for more
- Suspend Vim:
 - Command line: **CTRL-Z**¹
 - Resume Vim: **fg**

¹ As of Vim 8, `:terminal` can be used to bring up a terminal in a new window

The Vim Talk

- └ Advanced Applications
 - └ Useful bindings
 - └ Normal mode

Normal mode

- Increment / decrement: CTRL-A and CTRL-X
 - Binary, octal, decimal, hex, alpha
 - See :help nrformats for more

- Suspend Vim:
 - Command line: `CTRL-Z2`
 - Resume Vim: `fg`

¹ As of Vim 8, `:terminal` can be used to bring up a terminal in a new window

- You can also suspend Vim with `CTRL-Z`
- This pauses your Vim session and brings you to the terminal that you used to open Vim
- Again, at first this might not seem so useful, but there are certain scenarios where this is valuable
- For instance, when using a remote connection or dumb terminal you can quickly run something on the command line without ending your session
- Even in graphical environments, this command can be useful - since it uses the same terminal that Vim was launched in, variables set in that session still exist
- When you're done using the command line, run the command `fg` to resume Vim
- As of Vim 8, the terminal command has somewhat made this obsolete

Insert mode

- Autocomplete: CTRL-X, CTRL-N and CTRL-P to cycle
 - File names: CTRL-F
 - Tags: CTRL-O
 - Keywords: CTRL-N
 - Spelling: s²
- Registers:
 - Paste: CTRL-R <reg>

See :help i_CTRL-X and :help compl-omni-filetypes for more

² CTRL-S will suspend the terminal; use CTRL-Q to resume

The Vim Talk

- └ Advanced Applications
 - └ Useful bindings
 - └ Insert mode

Insert mode

- Autocomplete: CTRL-X, CTRL-N and CTRL-P to cycle
 - File names: CTRL-F
 - Tags: CTRL-O
 - Keywords: CTRL-N
 - Spelling: s²

- Registers:
 - Paste: CTRL-R *creg*

See :help i_CTRL-X and :help compl-omni-Filetypes for more

² CTRL-S will suspend the terminal; use CTRL-Q to resume

- In insert mode, CTRL-X is used to bring up different autocomplete menus - CTRL-F for files, CTRL-O omni completing from tags files like exuberant ctags, etc.
- Note that for spelling, you press just s and not control s, since control s suspends Vim. If you happen to do this, press control q to resume it
- You can also insert text from registers while in insert mode by pressing CTRL-R then pressing the register

Introduction
Important Concepts
Advanced Applications
Plugins

Undo tree
Marks
Window Movement

Undotree

- I'll say this first before I start getting in to plugins - as a general rule, I try to avoid using lots of plugins unless I think they're very valuable
- I try to keep the number of plugins I use low so that I can easily get Vim set up and started without too much hassle
- Obviously plugins are awesome and part of what makes Vim so useful - and certain plugins probably would speed up my development process
- That being said, these are the only plugins I use for now, mostly because I haven't spent a ton of time researching plugins - so if you have any suggestions, feel free to let me know!

Undotree

- Built-in undo tree
- Plugin to visualize

```
Press ? for help.
+ [934] (1 second ago) 24 \end{frame}
+ (933) (3 seconds ago) 22 %
\\
| 932 (55 seconds ago) 21
| 931 (1 minute ago) 20 {
| 930 (1 minute ago) 19 | % Removes space of header
| 929 (1 minute ago) 18 | % See https://tex.stackexchange.com/questions/44983
| 928 (1 minute ago) 17 | \maketitle
| 927 (1 minute ago) 16 | \setbeamertemplate{headline}{\default}
| 926 (1 minute ago) 15 | \defbeamertemplate{headline}{\vspace{(-)\headheight}}
| 925 (1 minute ago) 14 | \makeatother
| 924 (1 minute ago) 13
| 923 (2 minutes ago) 12 | \begin{frame}[noframenumbering] % allowframebreaks
| 922 (2 minutes ago) 11 | ] % Remove first comment for multiple pages of refs
| 921 (2 minutes ago) 10 | \frametitle{References}
| 920 (2 minutes ago) 9 | \footnotetext{%
| 919 (2 minutes ago) 8 | \begin{thebibliography}{99} % Beamer does not support BibTeX so references must be inserted manually as below
| 918 (2 minutes ago) 7 | \tiny
| 917 (4 minutes ago) 6 | \bibitem{vim-ref} See \texttt{:help help-section}
| 916 (4 minutes ago) 5 | \bibitem{vim-usr} See \texttt{:help usr\_to\_user}
| 915 (4 minutes ago) 4 | \bibitem{vim-c} See \texttt{:help c}
| 914 (4 minutes ago) 3 | \bibitem{vim-tutor} See \texttt{:help vimtutor}
| 913 (4 minutes ago) 2 | \bibitem{vim-cheat-sheet} Graphical vi-vim Cheat Sheet and Tutorial
| 912 (4 minutes ago) 1 | \url{http://www.viwm.com/a_vim_vim_graphical_cheat_sheet_tutorial.html}
| 911 (4 minutes ago) 0 | 1705 \bibitem{advanced-vim-cheat-sheet} Vim Cheat Sheet For Programmers
| 910 (4 minutes ago) 1 | \url{http://michael.peopleofhonorconley.com/vim/_cheat_sheet_for_programmers_print.pdf}
| 909 (4 minutes ago) 2 | \bibitem{vim-sign} vim signature
| 908 (4 minutes ago) 3 | \url{https://github.com/kahen/vim-signature}
| 907 (5 minutes ago) 4 |
| 906 (5 minutes ago) 5 |
| 905 (5 minutes ago) 6 |
| 904 (5 minutes ago) 7 |
| 903 (5 minutes ago) 8 |
| 902 (7 minutes ago) 9 |
current: 909 redo: 933
- seq: 909 -
1705c1705
+ asdf| \bibitem{advanced-vim-cheat-sheet} See \texttt{:help vim-graphical\_cheat\_sheet\_tutorial.html}
+ asdf| \end{thebibliography}
+ asdf| \end{frame}
```

Figure 5: Undotree in action [5]

See :help undo-tree and :help undotree.txt

The Vim Talk

- └ Plugins
 - └ Undo tree
 - └ Undotree

Undotree

- Built-in undo tree
- Plugin to visualize



Figures 5: Undotree in action [S]

See :help undo-tree and :help undotree.txt

- Have you ever been editing, decided you needed to undo to some point to look at a change you made, then accidentally made a new change, deleting all your progress?
- Thanks to Vim's built-in undo tree, you never have to worry about doing that again - Vim stores your undoes in a tree structure, so new changes just make new branches, keeping you from accidentally wiping your redo history
- This plugin helps you visualize the undo tree graphically

Signature

The screenshot shows a Vim session with the file `2020-02-06-vim-talk.tex` open. The file contains a LaTeX document with several `\itemize` environments. A cursor is positioned at line 1169, which starts with a local mark (`m`). The text on the screen includes:

```
2020-02-06-vim-talk.tex  n/notes.txt
12 |           the file, the file will be opened at that line
11 \end{itemize}
10 }
9
8 \note<2> {
7 \begin{itemize}
6 |       \item Another useful tool is the tab command, which can be combined
5 |           with any command that would normally open a new window to offer
4 |           more flexibility
3 |       \item For instance, running tab help gf opens the help file for gf in a
2 |           new tab
1 |       \item You can also use numbers and symbols to specify where your tab
should go
1169 |       \item A number will make it that number tab, while a dollar sign will
0 |           make it the last tab
h |       \item Pluses and minuses can be used to make the tab relative - if no
H |           number or symbol is given, the tab will be relative to the
i |           current tab
5 |
6 \end{itemize}
M |
7 }
8
9 \end{frame}
K |
10 %
11 %-----
```

Figure 6: Example of local and global mark visualization [7]

The Vim Talk

Plugins

Marks

Signature

Signature



Figure b: Example of local and global mark visualization [?]

- Here is an illustration of the different marks using
- As you can see in the picture, there are three global marks set, as well as several local marks
- I have run into some issues with this plugin, however. If you delete a global mark using the delmarks command, the gutter won't update correctly. To fix this, reload the buffer.
- See here for more:
<https://github.com/kshenoy/vim-signature/issues/162>

Tradewinds

```
9 module adder_4 (
  10    input [3:0] A, [3:0] B,
  11    input C_in,
  12    output [3:0] F,
  13    output C
  14);
  15    // Adder outputs
  16    wire [2:0] C_out;
  17
  18    full_adder adder_1 (.A(A[0]), .B(B[0]), .C_in(C_in), .S(F[0]), .C_out(C_out[0]));
  19    full_adder adder_2 (.A(A[1]), .B(B[1]), .C_in(C_out[0]), .S(F[1]), .C_out(C_out[1]));
  20    full_adder adder_3 (.A(A[2]), .B(B[2]), .C_in(C_out[1]), .S(F[2]), .C_out(C_out[2]));
  21    full_adder adder_4 (.A(A[3]), .B(B[3]), .C_in(C_out[2]), .S(F[3]), .C_out(C_out[3]));
  22
  23 endmodule

 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229
 230
 231
 232
 233
 234
 235
 236
 237
 238
 239
 240
 241
 242
 243
 244
 245
 246
 247
 248
 249
 250
 251
 252
 253
 254
 255
 256
 257
 258
 259
 260
 261
 262
 263
 264
 265
 266
 267
 268
 269
 270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323
 324
 325
 326
 327
 328
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 340
 341
 342
 343
 344
 345
 346
 347
 348
 349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 400
 401
 402
 403
 404
 405
 406
 407
 408
 409
 410
 411
 412
 413
 414
 415
 416
 417
 418
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 430
 431
 432
 433
 434
 435
 436
 437
 438
 439
 440
 441
 442
 443
 444
 445
 446
 447
 448
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459
 460
 461
 462
 463
 464
 465
 466
 467
 468
 469
 470
 471
 472
 473
 474
 475
 476
 477
 478
 479
 480
 481
 482
 483
 484
 485
 486
 487
 488
 489
 490
 491
 492
 493
 494
 495
 496
 497
 498
 499
 500
 501
 502
 503
 504
 505
 506
 507
 508
 509
 510
 511
 512
 513
 514
 515
 516
 517
 518
 519
 520
 521
 522
 523
 524
 525
 526
 527
 528
 529
 530
 531
 532
 533
 534
 535
 536
 537
 538
 539
 540
 541
 542
 543
 544
 545
 546
 547
 548
 549
 550
 551
 552
 553
 554
 555
 556
 557
 558
 559
 559
 560
 561
 562
 563
 564
 565
 566
 567
 568
 569
 570
 571
 572
 573
 574
 575
 576
 577
 578
 579
 580
 581
 582
 583
 584
 585
 586
 587
 588
 589
 589
 590
 591
 592
 593
 594
 595
 596
 597
 598
 599
 599
 600
 601
 602
 603
 604
 605
 606
 607
 608
 609
 609
 610
 611
 612
 613
 614
 615
 616
 617
 618
 619
 619
 620
 621
 622
 623
 624
 625
 626
 627
 628
 629
 629
 630
 631
 632
 633
 634
 635
 636
 637
 638
 639
 639
 640
 641
 642
 643
 644
 645
 646
 647
 648
 649
 649
 650
 651
 652
 653
 654
 655
 656
 657
 658
 659
 659
 660
 661
 662
 663
 664
 665
 666
 667
 668
 669
 669
 670
 671
 672
 673
 674
 675
 676
 677
 678
 679
 679
 680
 681
 682
 683
 684
 685
 686
 687
 688
 689
 689
 690
 691
 692
 693
 694
 695
 696
 697
 698
 699
 699
 700
 701
 702
 703
 704
 705
 706
 707
 708
 709
 709
 710
 711
 712
 713
 714
 715
 716
 717
 718
 719
 719
 720
 721
 722
 723
 724
 725
 726
 727
 728
 729
 729
 730
 731
 732
 733
 734
 735
 736
 737
 738
 739
 739
 740
 741
 742
 743
 744
 745
 746
 747
 748
 748
 749
 750
 751
 752
 753
 754
 755
 756
 757
 758
 759
 759
 760
 761
 762
 763
 764
 765
 766
 767
 768
 769
 769
 770
 771
 772
 773
 774
 775
 776
 777
 778
 779
 779
 780
 781
 782
 783
 784
 785
 786
 787
 788
 788
 789
 789
 790
 791
 792
 793
 794
 795
 796
 797
 798
 799
 799
 800
 801
 802
 803
 804
 805
 806
 807
 808
 809
 809
 810
 811
 812
 813
 814
 815
 816
 817
 818
 819
 819
 820
 821
 822
 823
 824
 825
 826
 827
 828
 829
 829
 830
 831
 832
 833
 834
 835
 836
 837
 838
 838
 839
 840
 841
 842
 843
 844
 845
 846
 847
 848
 848
 849
 850
 851
 852
 853
 854
 855
 856
 857
 858
 859
 859
 860
 861
 862
 863
 864
 865
 866
 867
 868
 869
 869
 870
 871
 872
 873
 874
 875
 876
 877
 878
 878
 879
 879
 880
 881
 882
 883
 884
 885
 886
 887
 888
 888
 889
 889
 890
 891
 892
 893
 894
 895
 896
 897
 898
 898
 899
 899
 900
 901
 902
 903
 904
 905
 906
 907
 908
 909
 909
 910
 911
 912
 913
 914
 915
 916
 917
 918
 919
 919
 920
 921
 922
 923
 924
 925
 926
 927
 928
 929
 929
 930
 931
 932
 933
 934
 935
 936
 937
 938
 938
 939
 940
 941
 942
 943
 944
 945
 946
 947
 948
 948
 949
 950
 951
 952
 953
 954
 955
 956
 957
 958
 959
 959
 960
 961
 962
 963
 964
 965
 966
 967
 968
 969
 969
 970
 971
 972
 973
 974
 975
 976
 977
 978
 978
 979
 979
 980
 981
 982
 983
 984
 985
 986
 987
 988
 988
 989
 989
 990
 991
 992
 993
 994
 995
 996
 997
 998
 998
 999
 999
 1000
 1001
 1002
 1003
 1004
 1005
 1006
 1007
 1008
 1009
 1009
 1010
 1011
 1012
 1013
 1014
 1015
 1016
 1017
 1018
 1019
 1019
 1020
 1021
 1022
 1023
 1024
 1025
 1026
 1027
 1028
 1029
 1029
 1030
 1031
 1032
 1033
 1034
 1035
 1036
 1037
 1038
 1038
 1039
 1040
 1041
 1042
 1043
 1044
 1045
 1046
 1047
 1048
 1048
 1049
 1050
 1051
 1052
 1053
 1054
 1055
 1056
 1057
 1058
 1059
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1078
 1079
 1079
 1080
 1081
 1082
 1083
 1084
 1085
 1086
 1087
 1088
 1088
 1089
 1089
 1090
 1091
 1092
 1093
 1094
 1095
 1096
 1097
 1097
 1098
 1098
 1099
 1099
 1100
 1101
 1102
 1103
 1104
 1105
 1106
 1107
 1108
 1109
 1109
 1110
 1111
 1112
 1113
 1114
 1115
 1116
 1117
 1118
 1119
 1119
 1120
 1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1129
 1130
 1131
 1132
 1133
 1134
 1135
 1136
 1137
 1138
 1138
 1139
 1140
 1141
 1142
 1143
 1144
 1145
 1146
 1147
 1148
 1148
 1149
 1149
 1150
 1151
 1152
 1153
 1154
 1155
 1156
 1157
 1158
 1159
 1159
 1160
 1161
 1162
 1163
 1164
 1165
 1166
 1167
 1168
 1169
 1169
 1170
 1171
 1172
 1173
 1174
 1175
 1176
 1177
 1178
 1178
 1179
 1179
 1180
 1181
 1182
 1183
 1184
 1185
 1186
 1187
 1188
 1188
 1189
 1189
 1190
 1191
 1192
 1193
 1194
 1195
 1196
 1197
 1197
 1198
 1198
 1199
 1199
 1200
 1201
 1202
 1203
 1204
 1205
 1206
 1207
 1208
 1209
 1209
 1210
 1211
 1212
 1213
 1214
 1215
 1216
 1217
 1218
 1218
 1219
 1219
 1220
 1221
 1222
 1223
 1224
 1225
 1226
 1227
 1228
 1228
 1229
 1229
 1230
 1231
 1232
 1233
 1234
 1235
 1236
 1237
 1238
 1238
 1239
 1239
 1240
 1241
 1242
 1243
 1244
 1245
 1246
 1247
 1248
 1248
 1249
 1249
 1250
 1251
 1252
 1253
 1254
 1255
 1256
 1257
 1258
 1259
 1259
 1260
 1261
 1262
 1263
 1264
 1265
 1266
 1267
 1268
 1269
 1269
 1270
 1271
 1272
 1273
 1274
 1275
 1276
 1277
 1278
 1278
 1279
 1279
 1280
 1281
 1282
 1283
 1284
 1285
 1286
 1287
 1288
 1288
 1289
 1289
 1290
 1291
 1292
 1293
 1294
 1295
 1296
 1297
 1297
 1298
 1298
 1299
 1299
 1300
 1301
 1302
 1303
 1304
 1305
 1306
 1307
 1308
 1309
 1309
 1310
 1311
 1312
 1313
 1314
 1315
 1316
 1317
 1318
 1318
 1319
 1319
 1320
 1321
 1322
 1323
 1324
 1325
 1326
 1327
 1328
 1328
 1329
 1329
 1330
 1331
 1332
 1333
 1334
 1335
 1336
 1337
 1338
 1338
 1339
 1339
 1340
 1341
 1342
 1343
 1344
 1345
 1346
 1347
 1348
 1348
 1349
 1349
 1350
 1351
 1352
 1353
 1354
 1355
 1356
 1357
 1358
 1359
 1359
 1360
 1361
 1362
 1363
 1364
 1365
 1366
 1367
 1368
 1369
 1369
 1370
 1371
 1372
 1373
 1374
 1375
 1376
 1377
 1378
 1378
 1379
 1379
 1380
 1381
 1382
 1383
 1384
 1385
 1386
 1387
 1388
 1388
 1389
 1389
 1390
 1391
 1392
 1393
 1394
 1395
 1396
 1397
 1397
 1398
 1398
 1399
 1399
 1400
 1401
 1402
 1403
 1404
 1405
 1406
 1407
 1408
 1409
 1409
 1410
 1411
 1412
 1413
 1414
 1415
 1416
 1417
 1418
 1418
 1419
 1419
 1420
 1421
 1422
 1423
 1424
 1425
 1426
 1427
 1428
 1428
 1429
 1429
 1430
 1431
 1432
 1433
 1434
 1435
 1436
 1437
 1438
 1438
 1439
 1439
 1440
 1441
 1442
 1443
 1444
 1445
 1446
 1447
 1448
 1448
 1449
 1449
 1450
 1451
 1452
 1453
 1454
 1455
 1456
 1457
 1458
 1459
 1459
 1460
 1461
 1462
 1463
 1464
 1465
 1466
 1467
 1468
 1468
 1469
 1469
 1470
 1471
 1472
 1473
 1474
 1475
 1476
 1477
 1478
 1478
 1479
 1479
 1480
 1481
 1482
 1483
 1484
 1485
 1486
 1487
 1488
 1488
 1489
 1489
 1490
 1491
 1492
 1493
 1494
 1495
 1496
 1497
 1497
 1498
 1498
 1499
 1499
 1500
 1501
 1502
 1503
 1504
 1505
 1506
 1507
 1508
 1509
 1509
 1510
 1511
 1512
 1513
 1514
 1515
 1516
 1517
 1518
 1518
 1519
 1519
 1520
 1521
 1522
 1523
 1524
 1525
 1526
 1527
 1528
 1528
 1529
 1529
 1530
 1531
 1532
 1533
 1534
 1535
 1536
 1537
 1538
 1538
 1539
 1539
 1540
 1541
 1542
 1543
 1544
 1545
 1546
 1547
 1548
 1548
 1549
 1549
 1550
 1551
 1552
 1553
 1554
 1555
 1556
 1557
 1558
 1559
 1559
 1560
 1561
 1562
 1563
 1564
 1565
 1566
 1567
 1568
 1568
 1569
 1569
 1570
 1571
 1572
 1573
 1574
 1575
 1576
 1577
 1578
 1578
 1579
 1579
 1580
 1581
 1582
 1583
 1584
 1585
 1586
 1587
 1588
 1588
 1589
 1589
 1590
 1591
 1592
 1593
 1594
 1595
 1596
 1597
 1597
 1598
 1598
 1599
 1599
 1600
 1601
 1602
 1603
 1604
 1605
 1606
 1607
 1608
 1609
 1609
 1610
 1611
 1612
 1613
 1614
 1615
 1616
 1617
 1618
 1618
 1619
 1619
 1620
 1621
 1622
 1623
 1624
 1625
 1626
 1627
 1628
 1628
 1629
 1629
 1630
 1631
 1632
 1633
 1634
 1635
 1636
 1637
 1638
 1638
 1639
 1639
 1640
 1641
 1642
 1643
 1644
 1645
 1646
 1647
 1648
 1648
 1649
 1649
 1650
 1651
 1652
 1653
 1654
 1655
 1656
 1657
 1658
 1659
 1659
 1660
 1661
 1662
 1663
 1664
 1665
 1666
 1667
 1668
 1668
 1669
 1669
 1670
 1671
 1672
 1673
 1674
 1675
 1676
 1677
 1678
 1678
 1679
 1679
 1680
 1681
 1682
 1683
 1684
 1685
 1686
 1687
 1688
 1688
 1689
 1689
 1690
 1691
 1692
 1693
 1694
 1695
 1696
 1697
 1697
 1698
 1698
 1699
 1699
 1700
 1701
 1702
 1703
 1704
 1705
 1706
 1707
 1708
 1709
 1709
 1710
 1711
 1712
 1713
 1714
 1715
 1716
 1717
 1718
 1718
 1719
 1719
 1720
 1721
 1722
 1723
 1724
 1725
 1726
 1727
 1728
 1728
 1729
 1729
 1730
 1731
 1732
 1733
 1734
 1735
 1736
 1737
 1738
 1738
 1739
 1739
 1740
 1741
 1742
 1743
 1744
 1745
 1746
 1747
 1748
 1748
 1749
 1749
 1750
 1751
 1752
 1753
 1754
 1755
 1756
 1757
 1758
 1759
 1759
 1760
 1761
 1762
 1763
 1764
 1765
 1766
 1767
 1768
 1768
 1769
 1769
 1770
 1771
 1772
 1773
 1774
 1775
 1776
 1777
 1778
 1778
 1779
 1779
 1780
 1781
 1782
 1783
 1784
 1785
 1786
 1787
 1788
 1788
 1789
 1789
 1790
 1791
 1792
 1793
 1794
 1795
 1796
 1797
 1797
 1798
 1798
 1799
 1799
 1800
 1801
 1802
 1803
 1804
 1805
 1806
 1807
 1808
 1809
 1809
 1810
 1811
 1812
 1813
 1814
 1815
 1816
 1817
 1818
 1818
 1819
 1819
 1820
 1821
 1822
 1823
 1824
 1825
 1826
 1827
 1828
 1828
 1829
 1829
 1830
 1831
 1832
 1833
 1834
 1835
 1836
 1837
 1838
 1838
 1839
 1839
 1840
 1841
 1842
 1843
 1844
 1845
 1846
 1847
 1848
 1848
 1849
 1849
 1850
 1851
 1852
 1853
 1854
 1855
 1856
 1857
 1858
 1859
 1859
 1860
 1861
 1862
 1863
 1864
 1865
 1866
 1867
 1868
 1868
 1869
 1869
 1870
 1871
 1872
 1873
 1874
 1875
 1876
 1877
 1877
 1878
 1878
 1879
 1879
 1880
 1881
 1882
 1883
 1884
 1885
 1886
 1887
 1888
 1888
 1889
 1889
 1890
 1891
 1892
 1893
 1894
 1895
 1896
 1897
 1897
 1898
 1898
 1899
 1899
 1900
 1901
 1902
 1903
 1904
 1905
 1906
 1907
 1908
 1909
 1909
 1910
 1911
 1912
 1913
 1914
 1915
 1916
 1917
 1918
 1918
 1919
 1919
 1920
 1921
 1922
 1923
 1924
 1925
 1926
 1927
 1928
 1928
 1929
 1929
 1930
 1931
 1932
 1933
 1934
 1935
 1936
 1937
 1938
 1938
 1939
 1939
 1940
 1941
 1942
 1943
 1944
 1945
 1946
 1947
 1948
 1948
 1949
 1949
 1950
 1951
 1952
 1953
 1954
 1955
 1956
 1957
 1958
 1959
 1959
 1960
 1961
 1962
 1963
 1964
 1965
 1966
 1967
 1968
 1968
 1969
 1969
 1970
 1971
 1972
 1973
 1974
 1975
 1976
 1977
 1977
 1978
 1978
 1979
 1979
 1980
 1981
 1982
 1983
 1984
 1985
 1986
 1987
 1988
 1988
 1989
 1989
 1990
 
```

Vim tradewinds [8]

The Vim Talk

Plugins

Window Movement

Tradewinds

Tradewinds



- Vim's window movement leaves a lot to be desired by default - this plugin, tradewinds, helps to alleviate some of these issues
- You have your windows laid out like so

Tradewinds

```

11 module full_adder_bcd (
12   input [3:0] A, [3:0] B,
13   input C_in,
14   output [3:0] Z,
15   output C
16 );
17   wire [3:0] S;
18   wire [3:0] add;
19   wire C_out;
20
21   adder_4 adder_1 (.A(A[0]), .B(B[0]), .C_in(C_in), .F(S[0]), .C_out(C_out));
22   get_carry carry (.S(S[3]), .S2(S[2]), .S1(S[1]), .C_out(C_out), .C(C));
23
24   assign add[0] = S;
25   assign add[1] = C;
26   assign add[2] = S;
27   assign add[3] = 0;
28
29   adder_4 adder_2 (.A(S), .B(add), .C_in(1'b0), .F(Z), .C(X));
30
31 endmodule

```



```

10 module adder_4 (
9   input [3:0] A, [3:0] B,
8   input C_in,
7   output [3:0] F,
6   output C
5 );
4
3 // Adder outputs
2 wire [2:0] C_out;
1
11 full_adder adder_1 (.A(A[0]), .B(B[0]), .C_in(C_in), .S(F[0]), .C_out(C_out));
12 full_adder adder_2 (.A(A[1]), .B(B[1]), .C_in(C_out[0]), .S(F[1]), .C_out(C_out[1]));
13 full_adder adder_3 (.A(A[2]), .B(B[2]), .C_in(C_out[1]), .S(F[2]), .C_out(C_out[2]));
14 full_adder adder_4 (.A(A[3]), .B(B[3]), .C_in(C_out[2]), .S(F[3]), .C_out(C));
15
16 endmodule

```



```

1 get_carry.v
2 module full_adder (
3   input A, B, C_in,
4   output S, C_out
5 );
6
7   wire [2:0] w;
8
9   xor U1 (w[0], A, B);
10  and U2 (w[1], A, B);
11  and U3 (w[2], C_in);
12  xor U4 (S, C_in, w[0]);
13  or U5 (C_out, w[2], w[1]);
14
15 endmodule

```

Vim tradewinds [8]

The Vim Talk

Plugins

Window Movement

Tradewinds

Tradewinds



Vim tradewinds [8]

- Now imagine you want to move one of those windows over for a bit to the side, like so
- Once your done, how do you move it back?
- Vim's window movements by default are all extreme - you can only move windows all the way to the left, right, etc.
- (As a side note, you *can* rotate horizontally split tabs as well, but that still doesn't really help solve this issue)
- That's where this plugin comes in - tradewinds allows for what it calls "soft moves"
- Tradewinds' key mappings are intuitive - they are prefixed by `CTRL-W g`, and it implements the `HJKL` keys to soft move windows in specific directions
- The fix, here, for instance, is `CTRL-W gl`, which moves the left window back to where it was

References

- [1] See :help help-summary
- [2] See :help usr_toc
- [3] See :help vimtutor
- [4] Graphical vi-vim Cheat Sheet and Tutorial
http://www.viemu.com/a_vi_vim_graphical_cheat_sheet_tutorial.html
- [5] undotree <https://github.com/mrbill/undotree>
- [6] Vim Cheat Sheet for Programmers
http://michael.peopleofhonoronly.com/vim/vim_cheat_sheet_for_programmers_print.pdf
- [7] vim signature <https://github.com/kshenoy/vim-signature>
- [8] vim trade winds <https://github.com/andymass/vim-tradewinds>