

# 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

- 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.



# 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

- 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?

- 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

- 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?

Classical learning curves for some common editors

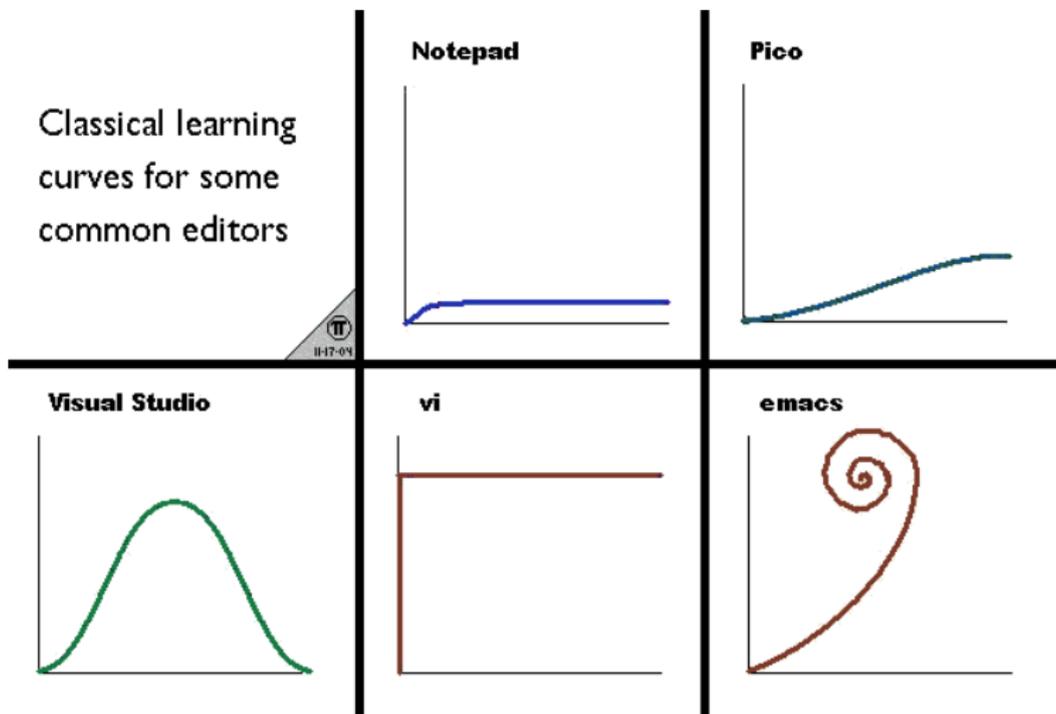


Figure 1: Vim is hard to learn

## The Vim Talk

## Introduction

## About Vim

## Why not Vim?

Why not Vim?

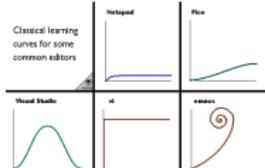


Figure 1: Vim is hard to learn

- 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



# The Vim Talk

## └ Introduction

## └ About 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

- 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

- 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

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

- 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]

- 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

# Simple cheat sheet

version 1.1  
April 1st, 06

## vi / vim graphical cheat sheet

<b>Esc</b> normal mode												
~ toggle case	external filter	@, play macro	# prev ident	\$ eol	% goto match	^ "soft" bol	& repeat :s	% 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 word	E end word	R replace word	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	d paste after	f misc	F	misc
A append at eol	S subst line	D delete to eol	F, "back find ch	G eof/goto ln	H screen top	J join lines	K help	L screen bottom	. ex cmd line	! reg. spec	^	bol/goto col
a append	s subst char	d delete	f, find char	g, extra cmds	h ←	j ↓	k ↑	l →	* repeat	g goto mk. bol	/	not used!
Z quit	X back-space	C change to eol	V visual lines	B prev word	N prev (find)	M screen mid!	< un-indent	> indent	? find (rev.)			
z extra cmds	x delete char	c change	v visual mode	b prev word	n next (find)	m set mark	reverse t/T/f/F	repeat cmd	/ find			

<p><b>motion</b> moves the cursor, or defines the range for an operator</p> <p><b>command</b> direct action command, if <b>red</b>, it enters insert mode</p> <p><b>operator</b> requires a motion afterwards, operates between cursor &amp; destination</p> <p><b>extra</b> special functions, requires extra input</p> <p><b>Q.</b> commands with a dot need a char argument afterwards</p> <p>bol = beginning of line, eol = end of line, mk = mark, yank = copy</p> <p>words: <code>quux(foO) baz, baz;</code>          WORDS: <code>quux(foO) baz, baz;</code></p>	<p><b>Main command line commands ('ex'):</b></p> <p>:w (save), :q (quit), :q! (quit w/o saving)          :e f (open file f),          :%s/s/y/g (replace 's' by 'y' filewide),          :h (help in vim), :new (new file in vim),</p> <p><b>Other important comands:</b></p> <p>CTRL-R: redo (vim),          CTRL-F/-B: page up/down,          CTRL-E/-Y: scroll line up/down,          CTRL-V: block-visual mode (vim only)</p> <p><b>Visual mode:</b>          Move around and type operator to act on selected region (vim only)</p> <p><b>Notes:</b></p> <ol style="list-style-type: none"> <li>(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')</li> <li>(2) type in a number before any action to repeat it that number of times (e.g.: 2p, dzw, \$i, daj)</li> <li>(3) duplicate operator to act on current line (dd = delete line, &gt;&gt; = indent line)</li> <li>(4) ZZ to save &amp; quit, ZQ to quit w/o saving</li> <li>(5) zt: scroll cursor to top, zb: bottom, zz: center</li> <li>(6) gg: top of file (vim only), gf: open file under cursor (vim only)</li> </ol>
---	---

For a graphical vi/vim tutorial & more tips, go to [www.viemu.com](http://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

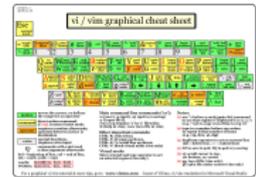


Figure 1. Simple Vim cheat sheet [4]

- 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



# The Vim Talk

## Introduction

## Learning More

## Advanced cheat sheet



Figure 2. Advanced Vim cheat sheet [6]

- 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

- 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`

- 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.

# Illustration

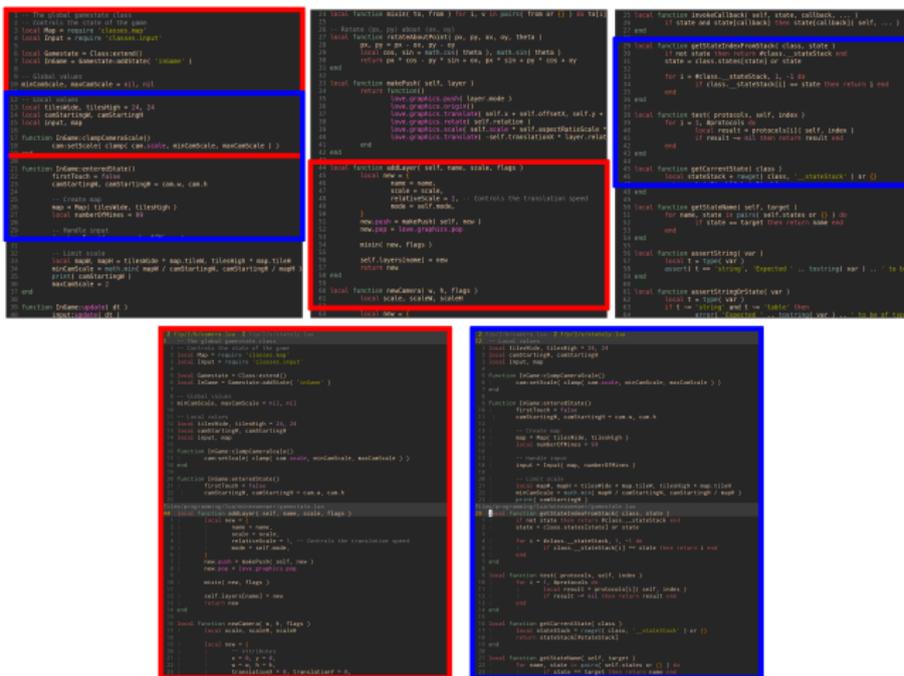


Figure 3: Visualization of buffers, tabs, and windows

# The Vim Talk

## Important Concepts

### Windows, Tabs, and Buffers

#### Illustration

Illustration



Figure 3. Visualization of buffers, tabs, and windows

- 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

- 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 states 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

- 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

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

- 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
- CTRL-W n creates a new window
- CTRL-W s splits the current window horizontally
- CTRL-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

- 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

- 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`

- 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  init10
6  begin
5      $monitor ( $time,
4          "  a + b + c + d + e + f + g + h + i + j + k + l + m + n + o + p + q + r + s + t + u + v + w + x + y + z + AA",
3          in[32],
2          in[31:28], in[27:24], in[23:20], in[19:16],
1          in[15:12], in[11: 8], in[ 7: 4], in[ 3: 0],
15         out[16], out[15:12], out[11:8], out[7:4], out[3:0],
1      );
2
3      // 0 + 3456 + 7890 = 11346
4      in = 33'b0...0011_0100_0101_0110...0111_1000_1001_0000;
5      #10;
6
7      // 1 + 3456 + 7890 = 11347
8      in = 33'b1...0011_0100_0101_0110...0111_1000_1001_0000;
9      #10;
10
11     // 0 + 1234 + 1235 = 02469
12     in = 33'b0...0001_0010_0011_0100...0001_0010_0011_0101;
13     #10;
14
15     // 1 + 1234 + 1235 = 02470
16     in = 33'b1...0001_0010_0011_0100...0001_0010_0011_0101;
17     #10;
18
19     // 0 + 9999 + 9999 = 19998
20     in = 33'b0...1001_1001_1001_1001...1001_1001_1001_1001;
21     #10;
22
23     // 1 + 9999 + 9999 = 19999
24     in = 33'b1...1001_1001_1001_1001...1001_1001_1001_1001;
25     #10;
26
27     // 0 + 0000 + 0000 = 00000
28     in = 33'b0...0000_0000_0000_0000...0000_0000_0000_0000;
29     #10;
30
31     // 1 + 0000 + 0000 = 00001
32     in = 33'b1...0000_0000_0000_0000...0000_0000_0000_0000;
33     #10;
34
35     // 0 + 0500 + 0500 = 01000
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
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
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490

```

# The Vim Talk

## Important Concepts

### Windows, Tabs, and Buffers

#### 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`

- Copy / paste / cut:
  - Copy: "yymotion"
  - Cut: "dymotion"
  - Paste: "ap"

- 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

- Copy / paste / cut:
  - Copy: "y<motion>
  - Cut: "d<motion>
  - Paste: "p
- 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

- What is a macro?
  - Creating a macro:
    - Start: `qregisters`
    - Stop: `q`
  - Using a macro:
    - Play: `@registers`
    - 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 - {{{(ons and )}}}`end`
- Usage:
  - Create a fold: `zFmotion`
  - Delete a fold: `zD`
  - Open a fold: `zO`
  - Close a fold: `zC`
  - Close all folds: `zA`
  - Open all folds: `zR`

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

- Types of commands:
  - Ex commands: ;
  - Search patterns: / and ?
  - Filter commands: |
- Example commands:
  - `write`: :help, :quit
  - `find`
  - `!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 use 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

- 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

- 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

- 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: ' "
    - HTML/XML tags: t
  - Selections:
    - "A(n) objects" : aobjects
    - "inner objects" : iobjects
- 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

## Advanced Applications

### Buffers / Windows / Tabs

#### Opening Files

#### 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`

- 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`

- 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`
  - `:Stabnew file.txt :!Stabnew file.txt`
  - `:+stabnew file.txt :!S-stabnew 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

- Advanced Applications
  - Marks
    - Marks

## 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

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

See `:help range` and `:help 10.3` 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( i )  
  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( i )  
  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 + 1  
  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

- 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

- 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

- Replace using last search: :<range>sf/<replace>
- Replace using last search: :<range>sf/cp/<rs>/<flag>
  - Confirm: c
  - Global: g
  - Ignore case: I

See :help | substitute and :help pattern-atom 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

- Replace using last search: :<range>sf/<replace>
- Replace using last search: :<range>sf/cp/<rs/<flag>
  - Confirm: c
  - Global: g
  - Ignore case: I
- Useful patterns:
  - Word boundary: \  - Matching: \( \) and \1, \2, ...

See :help : substitute and :help pattern-atom 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! ftpplugin/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 in Vim:
    - `:runtime! $PLUGINSDIR/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! $PLUGINSDIR/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

- 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

---

<sup>1</sup> 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

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

<sup>1</sup>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<sup>1</sup>
  - Resume Vim: fg

---

<sup>1</sup> 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-Z<sup>1</sup>
  - Resume Vim: fg

<sup>1</sup> 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: s<sup>2</sup>
- Registers:
  - Paste: CTRL-R <reg>

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

---

<sup>2</sup> 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: s<sup>1</sup>

- Registers:
  - Paste: CTRL-R <reg>

See :help i\_CTRL-X and :help compl-omni-filetypes for more

<sup>1</sup> 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

# Undotree

---

└─ Plugins

└─ Undo tree

└─ 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!



# The Vim Talk

## └ Plugins

### └ Undo tree

#### └ Undotree

- Built-in undo tree
- Plugin to visualize

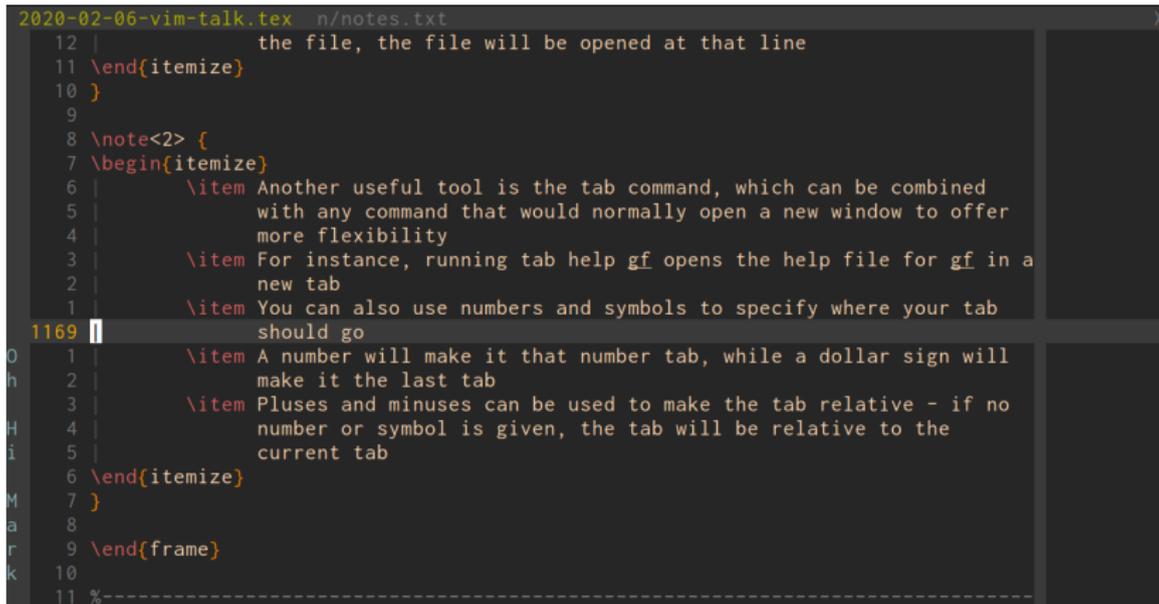


Figure 5. Undotree in action [5]

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



```
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
1169 | should go
O |     \item A number will make it that number tab, while a dollar sign will
h |     make it the last tab
H |     \item Pluses and minuses can be used to make the tab relative - if no
i |     number or symbol is given, the tab will be relative to the
M |     current tab
a |     \end{itemize}
r |     }
k |     \end{frame}
10
11 %-----
```

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

# The Vim Talk

- └ Plugins
- └ Marks
- └ Signature

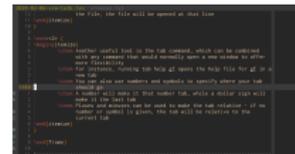


Figure 6. Example of local and global mark visualization [1]

- 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

```

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
1   _out[0]) );
1   full_adder adder_2 ( .A(A[1]), .B(B[1]), .C_in(C_out[0]), .S(F[1]), .C_out(C
2   _out[1]) );
2   full_adder adder_3 ( .A(A[2]), .B(B[2]), .C_in(C_out[1]), .S(F[2]), .C_out(C
3   _out[2]) );
3   full_adder adder_4 ( .A(A[3]), .B(B[3]), .C_in(C_out[2]), .S(F[3]), .C_out(C
4   ) );
4
5 endmodule

adder_4.v 11:1 All get_carry.v 6:1 All
12 module full_adder_bcd (
11  input [3:0] A, [3:0] B,
10  input C_in,
9   output [3:0] Z,
8   output C
7 );
6
5   wire [3:0] S;
4   wire [3:0] add;
3   wire C_out;
2
1   adder_4 adder_1 ( .A(A), .B(B), .C_in(C_in), .F(S), .C(C_out) );
13  get_carry carry ( .S1(S[3]), .S2(S[2]), .S1(S[1]), .C_out(C_out), .C(C) );
1
2   assign add[0] = 0;
3   assign add[1] = C;
4   assign add[2] = C;
5   assign add[3] = 0;
6
7   adder_4 adder_2 ( .A(S), .B(add), .C_in(1'b0), .F(Z), .C(X) );
8
9 endmodule

full_adder_bcd.v 13:1 All full_adder.v 8:1 All
    
```

Vim tradewinds [8]

2020-02-06

# The Vim Talk

## └ Plugins

## └ Window Movement

## └ Tradewinds

Tradewinds



Vim tradewinds [8]

- 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

```

12 module full_adder_bcd (
11   input [3:0] A, [3:0] B,
10   input C_in,
9   output [3:0] Z,
8   output C
7);
6
5   wire [3:0] S;
4   wire [3:0] add;
3   wire C_out;
2
1   adder_4 adder_1 ( .A(A), .B(B), .C_in(C_in), F
(S), .C(C_out));
13   get_carry carry ( .S(S[3]), .S2(S[2]), .S1(S[1]
), .C_out(C_out), .C(C));
1
2   assign add[0] = 0;
3   assign add[1] = C;
4   assign add[2] = C;
5   assign add[3] = 0;
6
7   adder_4 adder_2 ( .A(S), .B(add), .C_in('b0),
F(Z), .C(X));
8
9 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
13   full_adder adder_1 ( .A(A[0]), .B(B[0]), .C_in(
C_in), .S(F[0]), .C_out(C_out[0]));
1   full_adder adder_2 ( .A(A[1]), .B(B[1]), .C_in(
C_out[0]), .S(F[1]), .C_out(C_out[1]));
2   full_adder adder_3 ( .A(A[2]), .B(B[2]), .C_in(
C_out[1]), .S(F[2]), .C_out(C_out[2]));
3   full_adder adder_4 ( .A(A[3]), .B(B[3]), .C_in(
C_out[2]), .S(F[3]), .C_out(C));
4
5 endmodule

5 module get_carry (
4   input S3, S2, S1, C_out,
3   output C
2);
1
6   wire [1:0] w;
1
2   and U1A ( w[0], S3, S2 );
3   and U1B ( w[1], S3, S1 );
4   or U2A ( C, C_out, w[0], w[1] );
5
6 endmodule

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

full_adder_bcd.v 13,1 All
adder_4.v 11,1 All
full_adder.v 8,1 All
get_carry.v* 12L, 175C
    
```

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](http://www.viemu.com/a_vi_vim_graphical_cheat_sheet_tutorial.html)
- [5] undotree <https://github.com/mbbill/undotree>
- [6] Vim Cheat Sheet for Programmers  
[http://michael.peopleofhonoronly.com/vim/vim\\_cheat\\_sheet\\_for\\_programmers\\_print.pdf](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>