Learning the vi text editor

William Totten

University of Delaware

January 06, 2017

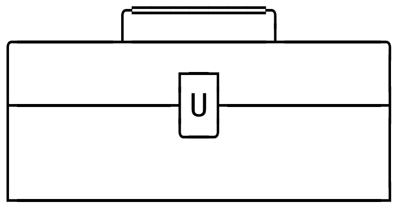
Learning how the vi text editor works for use on research systems. The vi editor is so named because it is a visual interface to the ex editor. By today's standards, vi might not seem very visual, but in 1977 it was revolutionary.

Because vi is both powerfull and small, you will find it on any UNIX or Linux system.

Overview

Building a vi toolbox

- vi is fairly orthogonal
- Start with a small set of features
- Learn the vi modes
- Get comfortable moving around a file
- Learn how to use a few commands
- Build and expand those features over time



vi is a Modal Editor

Command Mode

Press the "Escape" key to enter command mode.

ex engine

Press colon `:` from within command mode to enter the ex engine.

Insert or Overtype mode

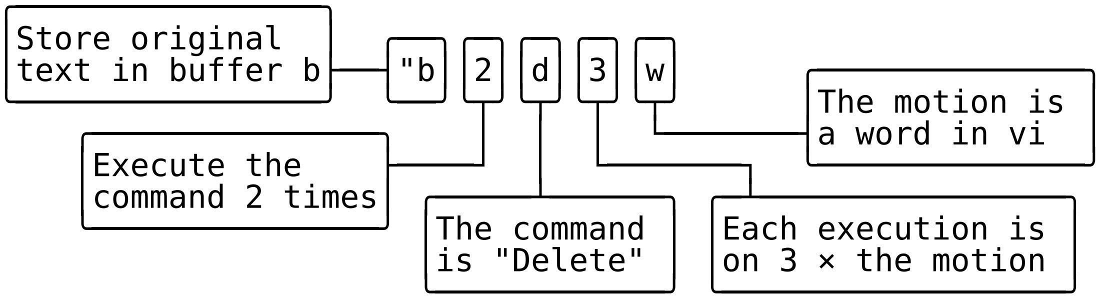
From command mode, the following keys enter different types of input modes:

Command	Description
i	Enter insert mode at the current cursor position.
o	Create a new line below the current one, and enter insert mode.
O	Create a new line before the current one, and enter insert mode.
R	Enter Overtype/Replace mode. Press Escape to exit.

Anatomy of an Action in vi

- How complex can an action get?

"b2d3w

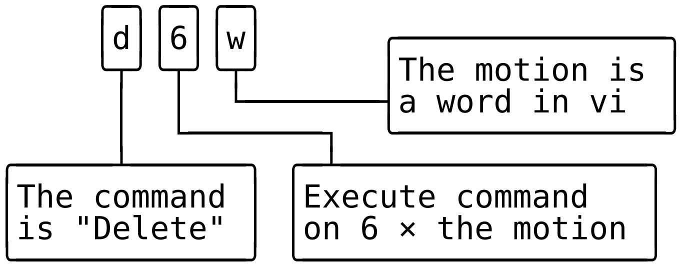

Store original text in buffer b
"b
2
d
3
w
The motion is a word in vi
Execute the command 2 times
The command is "Delete"
Each execution is on 3 × the motion

- This action will delete 6 words and store that text in a buffer named ‘b’.
- Luckily, you only really need to remember half of that.
- Buffers are rarely used, and while powerful, you should wait to learn them.
- Those two numbers always get multiplied together, let's focus on one of them.

Anatomy of a (Simple) Action in vi

- Let's simlify down to something which can fit into our toolbox

d6w



- This action will delete 6 words and store that text the general buffer.
- You can add the complete action to your toolkit when you are comfortable.
- This is a pattern we can remember and expand.
- Master this pattern first, then add more advanced commands to your toolbox.

Commands with Motions

The following table lists commands which follow the above methodology of accepting a motion which describes the text to act upon. If any of these command characters are entered twice (e.g. d d, c c, ...), then the action will be executed against the entire current line.

Command	Description
d	Delete the text associated with the motion, an place the deleted text into the buffer.
c	Change the text. This action will delete the text associated with the motion, then enter **insert mode**
y	Yank text (vi's term for copy). This will place the text associated with the motion into the buffer.
< >	Shift the lines left and right respectively, based on the shiftwidth variable, which defaults to a tab.
!	Filter the lines through an external program. After you specify the motion which includes complete lines, and so has a limited motion set) you will be prompted for the command to filter through.

Commands without Motions

This table contains commands which do not accept motions as arguments. The motion for these commands is implicitly defined as a single character. You may specify a number before these commands, which will specify the number of characters to act upon.

Command	**Description**
s	Substitute characters, this works like the change command forced to a motion of 1 character.
x	Remove characters, this works like the delete command forced to a motion of 1 character.
r	Replace characters. The same character will replace all characters if preceded by a number.
~	Change the case of a character between upper and lower cases. Non-alpha characters are left unaffected.

Miscellaneous Commands

Commands which don't fit into the previous constraints. These are the non-orthogonal commands.

Command	**Description**
p	Paste the buffer contents after the cursor.
P	Paste the buffer contents before the cursor.
.	Redo last action sequence at the current location.
u	Undo last action.
U	Undo last actions on current line.
J	Join the next line to this one with a space between
D	Delete the rest of the current line from the cursor on.
C	Change the rest of the current line from the cursor on.
Y	Yank the current line into the buffer.
m*c*	Mark current position with label *c* .

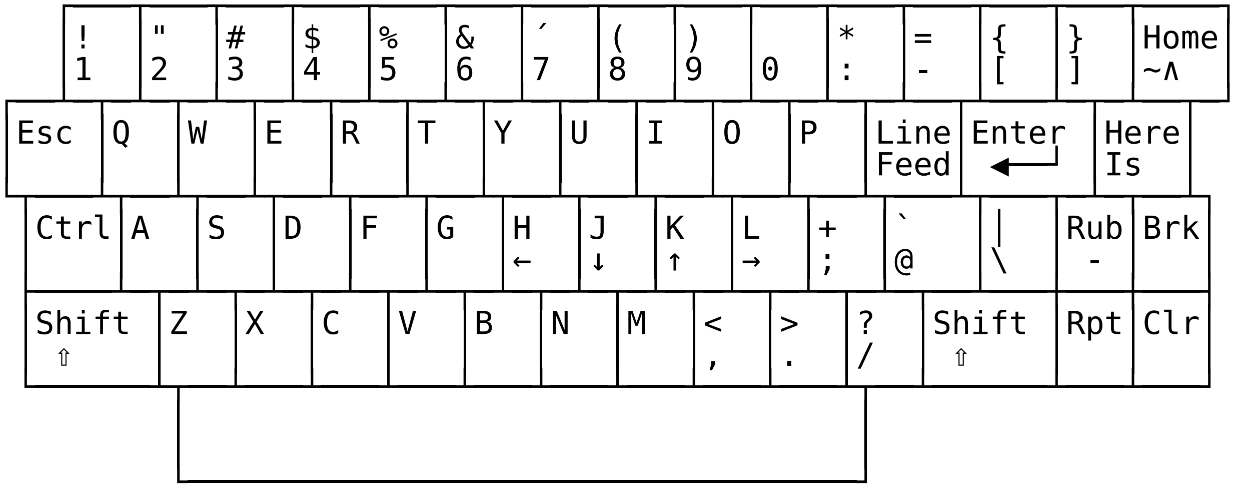
Common Motions

Motion	Description
h	Left one character
k	Up one line
b	Back to beginning of word
%	Match fence pair: *()[]{}*
^	Back to start of line
?re	Back to *re*
n|	Column *n* of current line
{	Back to paragraph start
ctrl-U	Back by one screen
ctrl-E	Shift screen down one line

Motion	Description
l	Right one character
j	Down one line
w	Forward to before word
e	Forward to end of word
$	Forward to end of line
/re	Forward to *re*
nG	To line number *n*
}	Forward to paragraph end
ctrl-F	Forward by one screen
ctrl-D	Forward by one-half screen

Why h, j, k, l to move around?

While the arrow keys on modern keyboards work to move the cursor in vi, why was the original choice to use the h, j, k, and l characters made?



ADM3A Keyboard used by Bill Joy when he originally wrote vi

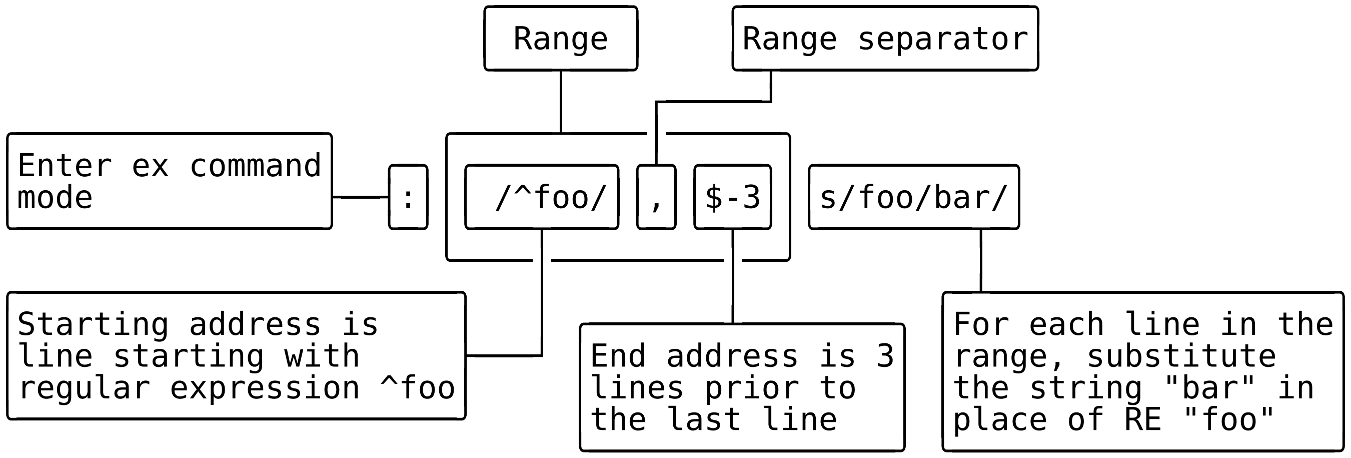
ex General Commands

Historically, vi is just the visual mode of the ex editor. As such, many operations are accomplished via entering the ex editor. This is done by pressing the colon : key.

Command	Description
q q!	Quit vi, with exclamation (!) forcibly
w w!	Write file, with exclamation (!) forcibly
w file	Write to file named *file*
e file	Edit new file named *file*
r file	Read in the contents of file named *file* to buffer
x	Write file, but only if it modified since last write
n	Start editing next file in command-line file list
w	Rewind to the first file mentioned on the command-line

Anatomy of Buffer Modification Syntax in ex

```
:/^foo/,$-3s/foo/bar/
```


Range
Range separator
Enter ex command mode
:
/^foo/
,
$-3
s/foo/bar/
Starting address is line starting with regular expression ^foo
End address is 3 lines prior to the last line
For each line in the range, substitute the string "bar" in place of RE "foo"

ex Buffer Modification commands

Command	**Description**
s/RE/text/	Substitute text for Regular Expression
d	Delete lines
p	Print lines (not very useful in vi, but is in ex)
m addr	Move lines to *addr*
ya c	Yank into buffer *c* or general buffer w/o *c*
g/RE/cmd	Run *cmd* on all lines matching *RE*
>	Shift lines right by shiftwidth (defaults to tab)
<	Shift lines left by shiftwidth (defaults to tab)

Different Implementations of vi

When you type `v` `i`, you don't always get the same editor.

	vi	**vim**	**vile**	**nvi**
Notes	Uses original code by Bill Joy	The most enhanced vi implementation	Enhanced, but less than vim, popular with SAs	Relatively true to original
Syntax Highlights	No	Yes, extensive & on by default	Yes, but fairly clumsy	No
Graphical Version?	No	Yes, gvim	Yes, xvile	No
Where?	Solaris (Composers) Some Linux (Arch)	Most Linux & Mac OSX (Mills/Farber)	Optional, commonly in /usr/local/bin at UD	FreeBSD, OpenBSD, & NetBSD

vi: More than just an editor

Many applications allow editing with vi key bindings. Once you are more comfortable with vi, you can enable these features if you like.

readline The readline library is used to obtain input for a number of programs, most notably "bash". Bash can be put into vi input mode using the command `set -o vi`. Readline is also used by python, lua, mathomatic, and gnuplot.

zsh The zsh shell has its own built-in line editing, and is widely heralded as the best command-line among UNIX shells. It can also be enabled with `set -o vi`.

abiword The abiword word processor contains an incomplete, but fairly featureful vi key binding. Although, enabling it is a little complicated.

kate The kate text editor is a feature rich text editor with menus tabs, color coding, etc. This is an excellent choice for people wanting vi key bindings, but with a modern gui interface

MonoDevelop Cross platform IDE for C#, F#, and more

Firefox Firefox has multiple extensions which can be added which help configure Firefox for vi key bindings for text area boxes, or for moving around Firefox itself.

Switches Cisco, and other swiches have vi input modes for their command-line interfaces.

ksh, tcsh, and other shells also have vi input modes.

Vimtutor

The most commonly installed vi clone, vim, comes with a file which walks people through the basics needed to be comfortable using the vi editor. It can be accessed at the shell prompt by running:

```
$ vimtutor↲
```

This will start vim on a copy of the vim tutor file. This file contains instructions on how to edit it to learn how to use the most important functionality.

Appendix

What follows are additional commands

Add these to your toolbox later

More Motions

Motion	Description
Tc	Back almost character *c*
Fc	Back to character *c*
Fc	Inverse of last: f, F, t, T
(	Back to sentence start
[	Back to section start
H	First line on screen
_	Entire current line
-	Back to first non-whitespace on previous line
' '	Start of line of last jump
'c	Start of line marked *c*

Motion	Description
tc	To before character *c*
fc	To character *c*
,	Repeat last: f, F, t, T
)	Forward to sentence end
]	Forward to section end
L	Last line on screen
M	Middle line on screen
+	Forward to first non-whitespace on next line
` `	Goto spot of last jump
`c	Goto mark *c*

Ranges

Range	**Description**
14	Line 14 of the current buffer
12,42	All lines from 12 to 42 (inclusive)
%	Every line in the current buffer
/RE/	The next line (not all) which matches Regular Expression RE
/RE1/,/RE2/	Starting at Regular Expressin RE1, and ending at Regular Expression RE2 inclusive)
g/RE/	Every line matching regular expression RE

Variables (settings)

Different vi implementations have different variables for their own extensions from the original. However, a basic set is fairly common across all implementations. You should review the manual for your implementation if you want to customize it. vim, for example, has a very large number of settings variables and custom functionality. These very common variables are consistent across implementations (aliases are in parenthesis after the full name):

Boolean Variables

Boolean variables can be unset by prefixing them with the string "no".

autoindent (ai) New lines should be indented at the same level
autowrite (aw) Write buffers before leaving
number (nu) Display line numbers in the left margin
showmatch (sm) Automatically show fence pairs on *()[]{}*
showmode (smd) Show if the editor is in *Insert* or *Replace* modes
wrapscan (ws) When searches reach the bottom of the file, start at top

Scalar Variables

shiftwidth (sw) Number of spaces to insert on a shift operation
tabstop The width to use when displaying tabs
wrapmargin (wm) Split long lines at a specified column

Buffers

- A buffer is an internal concept which is similar to a clipboard.
- There is a general buffer, one which is used as the default for actions which require a buffer.
- Buffers can have names, and can exist to store text for the entirety of a session.
- Buffer names can be any letter
- Buffers can be referenced by preceding the buffer name with a double quote (") character.