The vi Editor

The *vi* editor is a nearly universal text editor included with almost every version of Unix and/or Linux on the market today. It stands for "visual" and is pronounced "vee-eye". It is important to be able to use *vi* for simple editing tasks, since it may be the only editor available in many cases, such as when X-Windows will not run correctly. Below you will find a short tutorial designed to show you around *vi* and help you gain experience in using this powerful text editor.

The *vi* editor has two very distinct and separate modes, called *text-entry* mode and *command* mode. It also has a very powerful search and replace mechanism based on *regular expressions*. At first glance, *vi* may seem a bit cryptic and hard to use, but once mastered it can actually outperform many other advanced text editors. This tutorial will introduce you to *vi*'s basic commands. To learn more about *vi*, consult the online man (or info) documentation. There are also a number of books published that will provide more comprehensive information.

Preliminary Information

Here are some commonly used keys within *vi*:

Name	***Action***
Esc	Exits *text-entry* mode and returns to *command* mode, or cancels a command.
Return	Terminates a command, or starts a newline in *text-entry* mode. (Sometimes labeled the Enter key).
Interrupt	Aborts a command (often labeled Del, Delete, or Rubout).
Bksp	*Text-entry mode:* Backspaces the cursor by one character on the current line. Removes the previously typed character from the edit buffer, but does not remove it from the display (sometimes labeled as a Left Arrow). The current line is defined as the line containing the cursor. *Command mode:* Backspaces cursor without deletion (can take a preceding *count* parameter).
Ctrl-D	*Command mode:* Scrolls down a half-screen.
Ctrl-F	*Command mode:* Scrolls forward a page.
Ctrl-B	*Command mode:* Scrolls backward a page.
Ctrl-N	*Command mode:* Moves cursor down one line (alternative to cursor arrow key).
Ctrl-P	*Command mode:* Moves cursor up one line (alternative to cursor arrow key).
Bell or Ctrl-G	*Command mode:* Displays vi status.

Name	*Action*
Ctrl-R or Ctrl-L	*Command mode:* Redraws the screen (key depends on terminal type).
Ctrl-U	*Text-entry mode:* Restores cursor to the first character inserted on the current line (further insertions can then be made from that point). *Command mode:* Scrolls up a half-screen.
Ctrl-V	*Text-entry mode:* Used to insert control characters into the text by suspending the normal action of that control character (some exceptions).
Ctrl-W	*Text-entry mode:* Moves the cursor to the first character of the last inserted word.
Ctrl-T	*Text-entry mode:* If autoindent is on, gives an indent of **shiftwidth** spaces from left-hand margin. (**Shiftwidth** can be preset or varied by *vi* commands.)
Ctrl-@	*Text-entry mode:* When entered as first character of an insertion, *vi* replaces Ctrl-@ with the last piece of text inserted (unless this exceeds 128 characters). Similar to . (dot) in command mode.

Because keyboards on terminals and computers vary so much, some of the keys shown may have alternate mappings. The keyboard mapping is controlled by a file named **/etc/termcap**. The environment variable named **TERM** is used to select the specific kind of terminal entry to use from the **/etc/termcap** file. Normally, your system login profile will select the proper terminal when you login. Should you ever need to modify this, the following (bash) commands will do so:

```
TERM=vt102
export TERM
```

or this (for csh):

```
setenv TERM vt102
```

Meet the Family

There are normally three different flavors of the *vi* editor installed on most Unix systems, called: *view*, *vedit* and *vi* itself. The *view* command invokes the *vi* editor in read-only mode, so you cannot modify the file. If available, the *vedit* command invokes the *vi* editor in novice mode, with extra prompts, more help and some simpler commands.

Invoking *vi*

```
     vi [-option...] [command...] [filename...]
     vedit [-option...] [command...] [filename...]
     view [-option...] [command...] [filename...]
```

Using the command:

```
view myfile.txt
```

is equivalent to the command:

```
vi -R myfile.txt
```

since the -R option makes *vi* operate in read-only mode. I'll cover the various *vi* options and commands a bit later.

The *vi* modes

The *vi* editor is a modal editor, which means that depending on the current mode of operation, different keystrokes and commands will have different behaviors. It has a total of three different modes, called *text-entry, command,* and **ex** *escape* modes. In addition, while in the *text-entry* mode, there are for sub-modes, called: *insert, append, change,* and *open* sub-modes.

Mode	***Description***
Text-entry	Typed characters go to a temporary file known as the editing buffer (and eventually to a permanent file if the buffer is saved). *Visual clues:* printable characters that you type will appear on the screen. If showmode is on, the appropriate legend INSERT, APPEND, CHANGE, or OPEN will be displayed at the bottom right of the screen. *Audible clues:* pressing Esc will exit *text-entry* mode (returning you to command mode) without beeping.
Command	Keystrokes are interpreted as *vi* editing commands. Each command is usually a single or double keystroke (with possible modifiers), performing such operations as cursor movement, screen scrolling, text deletion, change and movement, string searching, and switching to other modes. *Visual clues:* the typed commands do not immediately show on screen. If showmode is on, the absence of the mode in the legend is significant! *Audible clues:* typing a character that does not correspond to a command will sound a beep. Pressing Esc will always beep (and your remain in command mode).
ex Escape	Your input is interpreted as an **ex** command. (**ex** is an older line-oriented editor upon which *vi* is based.) *Visual clues:* the **ex** command prompt : (colon) will be displayed at the beginning of the status line. The cursor appears after the colon. **ex** commands are displayed as you type them, but have no effect until you press Return (runs command) or Esc (cancels command).

Mode Navigation

From Mode	***To Mode***	***Command, Key or Action***
Command	Text-Entry	I, i (insert) A, a (append) O, o (open new line) S, s (substitute) C, c (change) R, r (replace)
Text-entry	Command	Esc
Command	ex	: (colon)
ex	Command	Return or Esc after ex command
Text-entry	ex	Must go to command mode first (Esc), then : (colon)
ex	Text-entry	Must go to command mode first (Return or Esc), followed by a text-entry command (i, a, o, s, c or r).

Creating Text with vi

Let's get started, but first make a new directory where we can create some files without possibly overwriting or corrupting any existing files. Issue these commands to make a new directory and then edit the file named *test.data* in the new directory:

```
[...]$ cd
[...]$ pwd
/home/randy
[...]$ mkdir junk
[...]$ cd junk
[...]$ vi test.data
```

Your screen should now look like this:

```
~
~
~
~
~
~
~
~
~
~
~
~
```

```
~
~
~
~
~
~
~
~
~
~
~
test.data [NEW FILE]
```

The first line will hold your cursor, with lines 2 to 24 showing a tilde (~) character. If the file was not empty, the contents of the file would have been displayed. The tilde (~) characters indicate empty lines beyond the end of the file. Line 25 (assuming you have a 25-line display) is reserved for use as a status line. It is used to display status information and for entering **ex** commands.

If the status line gets hidden, you can redisplay it by pressing Ctrl-G. We call this the *status* command. If the screen every gets garbled, perhaps by noise being received across a modem link, you can refresh the display by pressing Ctrl-L (sometimes Ctrl-R). This command is called the *screen refresh* or *screen redraw* command.

The cursor is currently display in column 1 of row 1. The cursor position indicates where the next text entry will occur, or which line a command will operate on.

Text entry

The editor started out in command mode. Before we can enter, change or rearrange any text, we must switch from command mode to text-entry mode. Adding new text is done using either append (the a command) or insert (the i command) mode. The two modes only differ in whether we will insert text just before the cursor, or append text after the cursor. In some cases, such as when working with a new file, the difference is unimportant. At other times, it is vital to use the correct mode. For example, to extend a line of text, you must append new characters after the last character on the line.

Let enter some text to get started:

1. First, let's make sure that showmode is enabled. This will add some additional status information to the status line.
2. Press the : (colon) key to enter **ex** mode. Next, type in **set showmode** and press the Return key. This activates showmode for the duration of the session. (Hint: To disable showmode, enter **:set noshowmode** from command mode).
3. Type a (lowercase) to enter append mode. The "a" will not appear on screen.
4. You are now in text-entry mode, specifically the append mode. If you have showmode turned on, the status line should change to reflect this.
5. Type in the following text (remember to press Return at the end of each line):

```
At last, I am using vi, the visual editor.  I am in append
mode, so my keystrokes are being stored (appended) into the
editing buffer.  Later on, after further editing, I will save
this text by writing from the buffer to the file test.data.
```

You should press Enter one last time after the final line of text. In addition, you should put two spaces after the period, before starting a new line of text. Some of the commands in *vi* can search for and work with sentences and the editor expects all sentences to be formatted this way.

Notice how the tildes have disappeared from lines 2 to 6, but on the other unused lines, 7 to 24, there are still tildes in column 1. If you made any typing errors, ignore them for the moment. We see how to make correction a bit later.

The above text is stored in the editing buffer. Until you write it to disk, the file **test.data** remains empty. During long editing sessions, it is good practice to save the buffer at regular intervals. To do this, follow these commands:

1. Press Esc to leave append mode and return to command mode.
2. Type a colon. You are now in **ex** mode. The status line will echo this colon. It serves as the **ex** mode prompt.
3. Type w (the **ex** write command) followed by Return. This writes the editing buffer to disk. Since you did not provide a file name, the current file name (**test.data**) is used. You will then return to command mode.

You can save the buffer in any directory to any file name for which you have write permissions. For example, if the file **test.temp** does not exist, **:w test.temp** will create the file and save the buffer to it. If **test.temp** already exists, the **:w** command will not overwrite it, but you can force *vi* to overwrite it using the command **:w! test.temp**. The current filename (**test.data**) will not be changed, so in the future using the **:w** command will save the buffer to **test.data** again. One useful variant on the **:w** command is **: w>>*filename***, which appends the editing buffer to the *filename*. In addition, there are three useful shortcuts that can be used to save, then exit *vi* immediately:

:wq{Return}	Same as **:w{Return}** followed by **:q{Return}**
:x	Same effect as above
ZZ	Same effect as above, but notice you do not enter the : (colon).

After entering the command above, the status line will show **wrote test.data, 5 lines, 242 chars**, or something very similar. You will then again be placed in command mode. Notice that even with the showmode option turned on, the status line does not tell you that you are in command mode. There is an easy way to verify command mode however. Just press the Esc key. If you are in command mode, you will hear a beep and remain in command mode. If you were in text-entry mode, pressing Esc returns you to command mode, without the beep.

Next, let's exit *vi* by entering the command **:q**. Since we have already saved the editing buffer to the file, we should exit immediately. If we had not saved the buffer yet, *vi* would complain and refuse to exit. We can force *vi* to exit without saving using the **:q!** command, or if we wanted to save the file, then exit, we can use **:wq**, **:x**, or the **ZZ** commands.

To edit the file again, enter the command: **`vi test.data`**

Cursor Movement

Command	***Action***
l or spacebar or {right arrow}	Moves cursor to the right, but not beyond the end of a line (note the warning beep). The spacebar does *not* blank out any characters being traversed.
h or Bksp or {left arrow}	Moves the cursor to the left, but not beyond the start of the current line (a beep sounds).
+ or Return	Moves the cursor to the start of the next line, or beeps if already at the last line in the file.
j or Ctrl-N or Ctrl-J or LF or {down arrow}	Moves cursor down one line, in the same column. Beeps if no next line. If column in the lower line is beyond the end of the line, the cursor moves to the last character in the line. (Some terminals do not have a separate LF key).
k or Ctrl-P or {up arrow}	Moves cursor up one line in the same column. Beeps if you are on the first line in the file. If column in the upper line is beyond the end of the line, the cursor will move to the last character of that line.
- {dash}	Moves cursor up one line and over to the first character on the line. Beeps if no previous line.
^ {caret}	Moves cursor to the first nonblank character in the current line.
0 {zero}	Moves cursor to column 1 of the current line (whether blank or not).
$	Moves cursor to the last character of the current line.
w	Moves cursor forward to the start of the next word. Words are taken to be strings separated by whitespace (newlines, spaces or tabs), or punctuation symbols, so "heavy,metal,rock" will be treated as three words by *vi*. If at the last word in a line, advances to the next word on the next line, if available.
W	Same as 'w', but punctuation does not count, so "heavy,metal,rock" would be skipped as a single word.
b	Works like 'w', but moves cursor backwards to the previous word.
B	Works like 'b', except ignores punctuation symbols.

Command	***Action***
e	Works like 'w', but cursor stops under the last character of the next word. If the cursor is already in the middle of a word, stops at the end of the word.
E	Works like 'e', but ignores punctuation symbols.
(	Moves cursor to the start of the current sentence, or the start of the previous sentence if the cursor is already at the start of a sentence.
)	Moves cursor to the end of the next sentence. A sentence is any string terminated by a period, question mark, or exclamation point, followed by two spaces or a newline.
H	Moves cursor to the home position, which is column 1 of the top line on the screen.
L	Moves cursor to the last line on the screen.
G	Moves cursor to the end of the file.
:*line_number*	Immediately jumps to the line number supplied. (Hint: **:1** always returns you to the top of the file.)

Practice moving the cursor around using these keys on your sample file.

We need to add more text to the sample file in order to gain experience using the screen control commands for scrolling forward and backward. We are going to use one of *vi*'s tricks to quickly add some text to our file.

Edit your sample file (**test.data**) and position the cursor at under the last period of the last sentence (Hint: use commands **'G' '$'**). Now, enter append mode with the command **'a'**. Enter the following text (press {Return} first):

```
This line is being added to test.data.
```

Next, press Esc to leave *text-entry* mode and enter *command* mode, followed by a press of the . (period) key. This invokes the *repeat* command, which will repeat the last action we performed. Next, try out the *undo* command, which is invoked by pressing the 'u' key. Notice our new line of text disappears. Press 'u' again to invoke *undo* one more time. You should see the line of text reappear. So the *undo* command reverses the last command we issued, including possibly undoing an *undo* command.

Using the *repeat* command, we can now add as many copies of the line of text as desired. Press . {period} repeatedly, or hold it down, until the screen begins to scroll. Now we can try out some of the screen control commands.

Screen Control Commands

Command	***Action***
[*n*]Ctrl-U	Scrolls the screen up *n* lines. The default value for *n* gives a half-screen scroll.
[*n*]Ctrl-D	Scrolls the screen down *n* lines. The default value for *n* gives a half-screen scroll.
[*count*]Ctrl-F	Pages the screen forward, leaving two lines between pages for continuity, if possible. Note that *count* gives the number of *pages*, with a default of 1.
[*count*]Ctrl-B	Pages the screen backward, leaving two lines between pages for continuity, if possible. *Count* gives the number of pages to scroll, with a default of 1.
Ctrl-G or {Bell}	Displays the status line.
z{Return}	"Zeroes" the screen by redrawing the display with the current line placed at the top of the screen. This command is an apparent exception to the "No Returns Needed" rule. Actually the 'z' can be followed by {Return}, . {period}, or - {dash} with different results (see below).
z.	Similar to z{Return}, except places the current line in the middle of the screen.
z-	Similar to z{Return}, except places the current line at the bottom of the screen.
Ctrl-R or Ctrl-L	Refreshes the screen, clearing any garbage characters that may have showed up in the screen, perhaps from a network broadcast, modem signal noise, etc... Usually also clears the status line.

Text-Entry Modes

Command	***Action***
a[*text*]	Appends *text* after the cursor.
A[*text*]	Appends *text* after the end of the current line, no matter where the cursor is.
i[*text*]	Inserts *text* before the cursor.
I[*text*]	Inserts *text* in front of the current line no matter where the cursor is.
o[*text*]	Opens a new line below the current line and inserts *text*.
O[*text*]	Opens a new line above the current line and inserts *text*.

There is one other command to remember that is the opposite of the *open* command. It is the *join* command, invoked by pressing the letter J (uppercase). This brings the line below the current line up and appends it to the current line. Unlike the other commands, this command does not enter *text-entry* mode, but leaves you in *command* mode.

Deleting Text

Command	***Action***
x	Deletes the character under the cursor.
[*count*]x	Deletes *count* characters forward starting at the cursor.
X	Deletes the character to the left of the cursor.
[*count*]X	Deletes *count* characters backward starting at the one to the left of the cursor.
dd	Deletes the current line.
D	Deletes from the cursor to the end of the line.
d<*cursor_movement*>	Deletes from the cursor or from the current line to a point determined by the *cursor_movement* argument.

Understanding the delete with *cursor_movement* command is best done with some simple examples. As you are trying these commands out, remember the *undo* command 'u'. If you mess things up really bad, you can also use the *master_undo* command, which undoes all the changes made to the edit buffer since the file was loaded. The command for the *master_undo* is 'U'.

Cursor Movement Deletes

Command	***Deletion***
dw	From the cursor to the end of the word (w is word advance).
db	From the cursor to the beginning of the word (b is word back).
d{Return}	Deletes the current line and the next line.
d0	Deletes from the cursor the beginning of the line.
d^	Deletes from the cursor to the first printable character in the line.
d$	Deletes from the cursor to the end of the line.
d)	Deletes from the cursor to the end of the sentence.
d(	Deletes from the cursor to the beginning of the sentence.
dL	Deletes from the cursor to the end of the screen.
dH	Deletes from the cursor to the beginning of the screen.

Command	***Deletion***
dG	Deletes from the cursor to the end of the file.

In addition, most of these commands also accept a *count* parameter between the 'd' and the *cursor_movement* key, so **d3w** will delete the next 3 words and **d4j**, or **d4{down arrow}**, will delete the next four lines of text.

Changing Text

The *vi* editor supports a *change text* command, invoked with the **'c'** key, that works similar to the *delete text* command just discussed. When changing text, *vi* will replace the last character that will be changed with a '$' as a visual aid. For example, if you place the cursor on the word 'visual' in the sample text, then press **'cw'**, the 'l' in visual will be replaced with a '$' to show you that any text you enter will replace the entire word. Like the *delete* command, the *change* command accepts both an optional *count* and a *cursor_movement* key, so a general description of the command looks like this:

```
c[count]<cursor_movement>[text]{Esc}
```

Other Change Text Commands

Command	***Action***
[*count*]r<*char*>	Overstrikes the character at the cursor with *count* copies of another character, namely *char*, while remaining in command mode. The default for *count* is 1.
[*count*]R<*text*>{Esc}	Overstrikes the current line with *count* copies of *text* (default is 1).
cc<*text*>{Esc} or C<*text*>{Esc}	Changes the current line and replaces it with *text*.
s<*text*>{Esc}	Substitutes current character with *text*.
[*count*]s<*text*>{Esc}	Substitutes *count* characters with *text*.
S<*text*>{Esc}	Substitutes the current line with *text*.
[*count*]S<*text*>{Esc}	Substitutes *count* lines with *text*.
><*cursor_movement*>	Shifts all lines determined by the *cursor_movement* key to the right by **shiftwidth** spaces (defaults to 8).
<<*cursor_movement*>	Shifts all lines determined by the *cursor_movement* key to the left by **shiftwidth** spaces (defaults to 8).
>>	Shifts the current line to the right.
<<	Shifts the current line to the left.

NOTE: As a general rule of thumb, doubled commands affect only the current line. Examples include dd (delete current line), cc (change current line), << (shift current line left) and >> (shift current line right).

Yanking and Putting Text

The *vi* editor has several other buffers, in addition to the editing buffer, that help you perform "cut and paste" operations. There are 26 named buffers, called *a, b, c ... z* and nine delete buffers called *1-9*. In addition, there is one unnamed buffer that serves two purposes: it acts as the default buffer for many operations, and it serves as the receptacle for the most recently deleted piece of text. It is often referred to as delete buffer 0.

You can "yank" text from the editing buffer into one of the 26 named buffers, or into the unnamed buffer, as follows:

["<*letter*>]y<*cursor_movement*>

If you specific a letter, text will be yanked from the editing buffer and stored in the buffer under that name. Using a lowercase letter gives you a destructive yank (sometimes called a *General Sherman*), which overwrites any text that was previously held in the named buffer. Using an uppercase letter for the named buffer makes *vi* do an appending yank (sometimes called a *Lincoln*), which appends the yanked text to the named buffer instead of overwriting the buffer.

By default, the *unnamed* buffer is used. The amount of text saved is determined by the *cursor_movement* key used. For example, **"ayw** will yank the current word into buffer **a**, overwriting anything that was in buffer **a**. The command **"Ay(** will yank the text from the cursor to the beginning of the line, appending the text to buffer **a**.

The following variants should not surprise you:

["<*letter*>]yy

or

["<*letter*>Y

which both will yank the current line and place it into one of the buffers.

Every time you delete any text, *vi* automatically places the deleted text into both the unnamed buffer and onto a stack of delete buffers named 1-9. The nine most recent deletions are therefore always available for you to paste back into your file, using the numbers 1-9, instead of a-z.

To transplant the yanked or deleted text back into your document, use the *put* command.

Syntax:

["<*letter*|*number*>]p

or

```
["<letter|number>]P
```

The lowercase version puts the text below the current line, or after the cursor, while the uppercase version inserts the text above the current line, or before the cursor.

It is very common to use one of the delete commands, move the cursor, then use p or P to paste the deleted text back into another area of the same file. Since *vi* will allow you to work on more than one file at a time, it is also possible to cut and paste between files.

Searching Text

There are four basic ways in *vi* to search for text. The syntax looks like this:

```
/[pattern]/[offset]{Return}
/[pattern]{Return}
```

or

```
?[pattern]?[offset]{Return}
?[pattern]{Return}
```

Using the / character performs a forward search, while the ? character performs a backward search. The *pattern* accepts simple characters, or more complicated search patterns called *regular expressions*. If no pattern is given, *vi* will use the last pattern and repeat the search (or beeps if no previous pattern). The *offset* value is a positive or negative number that modifies your search as follows:

```
/sun/+2{Return}
```

will stop two lines after the line having the first occurrence of the word "sun", starting the search with the current line.

```
?sun?-4{Return}
```

will stop four lines prior to the line having the first match for "sun" during a backward search from the current cursor position.

You can use the **:set ignorecase** (or **:set ic**) **ex** command to make *vi* perform a case-insensitive search. By default, *vi* performs case sensitive searches. To reverse this, use the **ex** command **:set noic**.

You can search for a single character in the current line using the command:

```
f<character>
```

while

```
F<character>
```

searches backwards in the current line for the character specified.

One last command that is quite useful, is the *mark* command. This command allows you to set a bookmark that remembers your current position in the file, which you can later return to very easily. It is used often to remember where we are, search for some text, then jump back to where we started.

To set a bookmark, use this command:

```
m<lowercase_letter>
```

to jump back to a saved bookmark, use this command:

```
'<lowercase_letter>
```

You can have up to 26 bookmarks, named *a-z*.