Basic vi Commands

What is vi?

The default editor that comes with the UNIX operating system is called vi (**vi**sual editor). [Alternate editors for UNIX environments include pico and emacs, a product of GNU.]
The UNIX vi editor is a full screen editor and has two modes of operation:

1. *Command mode* commands which cause action to be taken on the file, and
2. *Insert mode* in which entered text is inserted into the file.

In the command mode, every character typed is a command that does something to the text file being edited; a character typed in the command mode may even cause the vi editor to enter the insert mode. In the insert mode, every character typed is added to the text in the file; pressing the <Esc> (*Escape*) key turns off the Insert mode.
While there are a number of vi commands, just a handful of these is usually sufficient for beginning vi users. To assist such users, this Web page contains a sampling of basic vi commands. The most basic and useful commands are marked with an asterisk (* or star) in the tables below. With practice, these commands should become automatic.
NOTE: Both UNIX and vi are **case-sensitive**. Be sure not to use a capital letter in place of a lowercase letter; the results will not be what you expect.

To Get Into and Out Of vi

To Start vi

To use vi on a file, type in vi filename. If the file named filename exists, then the first page (or screen) of the file will be displayed; if the file does not exist, then an empty file and screen are created into which you may enter text.

*	**vi filename**	*edit filename starting at line 1*
	vi -r filename	*recover filename that was being edited when system crashed*

To Exit vi

Usually the new or modified file is saved when you leave `vi`. However, it is also possible to quit `vi` without saving the file.
Note: The cursor moves to bottom of screen whenever a colon (`:`) is typed. This type of command is completed by hitting the `<Return>` (or `<Enter>`) key.

*	**`:x<Return>`**	*quit `vi`, writing out modified file to file named in original invocation*
	`:wq<Return>`	*quit `vi`, writing out modified file to file named in original invocation*
	`:q<Return>`	*quit (or exit) `vi`*
*	**`:q!<Return>`**	*quit `vi` even though latest changes have not been saved for this `vi` call*

Moving the Cursor

Unlike many of the PC and MacIntosh editors, **the mouse does not move the cursor** within the `vi` editor screen (or window). You must use the the key commands listed below. On some UNIX platforms, the arrow keys may be used as well; however, since `vi` was designed with the Qwerty keyboard (containing no arrow keys) in mind, the arrow keys sometimes produce strange effects in `vi` and should be avoided.
If you go back and forth between a PC environment and a UNIX environment, you may find that this dissimilarity in methods for cursor movement is the most frustrating difference between the two.
In the table below, the symbol `^` before a letter means that the `<Ctrl>` key should be held down while the letter key is pressed.

*	**`j` *or* `<Return>` [*or* down-arrow]**	*move cursor down one line*
*	**`k` [*or* up-arrow]**	*move cursor up one line*
*	**`h` *or* `<Backspace>` [*or* left-arrow]**	*move cursor left one character*
*	**`l` *or* `<Space>` [*or* right-arrow]**	*move cursor right one character*
*	**`0` (zero)**	*move cursor to start of current line (the one with the cursor)*
*	**`$`**	*move cursor to end of current line*
	`w`	*move cursor to beginning of next word*
	`b`	*move cursor back to beginning of preceding word*
	`:0<Return>` *or* `1G`	*move cursor to first line in file*
	`:n<Return>` *or* `nG`	*move cursor to line `n`*
	`:$<Return>` *or* `G`	*move cursor to last line in file*

Screen Manipulation

The following commands allow the `vi` editor screen (or window) to move up or down several lines and to be refreshed.

	^f	*move forward one screen*
	^b	*move backward one screen*
	^d	*move down (forward) one half screen*
	^u	*move up (back) one half screen*
	^l	*redraws the screen*
	^r	*redraws the screen, removing deleted lines*

Adding, Changing, and Deleting Text

Unlike PC editors, you cannot replace or delete text by highlighting it with the mouse. Instead use the commands in the following tables.
Perhaps the most important command is the one that allows you to back up and *undo* your last action. Unfortunately, this command acts like a toggle, undoing and redoing your most recent action. You cannot go back more than one step.

*	**u**	*UNDO WHATEVER YOU JUST DID; a simple toggle*

The main purpose of an editor is to create, add, or modify text for a file.

Inserting or Adding Text

The following commands allow you to insert and add text. Each of these commands puts the `vi` editor into insert mode; thus, the `<Esc>` key must be pressed to terminate the entry of text and to put the `vi` editor back into command mode.

*	**i**	*insert text before cursor, until `<Esc>` hit*
	I	*insert text at beginning of current line, until `<Esc>` hit*
*	**a**	*append text after cursor, until `<Esc>` hit*
	A	*append text to end of current line, until `<Esc>` hit*
*	**o**	*open and put text in a new line below current line, until `<Esc>` hit*
*	**O**	*open and put text in a new line above current line, until `<Esc>` hit*

Changing Text

The following commands allow you to modify text.

*	**r**	*replace single character under cursor (no* `<Esc>` *needed)*
	R	*replace characters, starting with current cursor position, until* `<Esc>` *hit*
	cw	*change the current word with new text, starting with the character under cursor, until* `<Esc>` *hit*
	cNw	*change* `N` *words beginning with character under cursor, until* `<Esc>` *hit; e.g.,* `c5w` *changes 5 words*
	C	*change (replace) the characters in the current line, until* `<Esc>` *hit*
	cc	*change (replace) the entire current line, stopping when* `<Esc>` *is hit*
	Ncc ***or*** **cNc**	*change (replace) the next N lines, starting with the current line, stopping when* `<Esc>` *is hit*

Deleting Text

The following commands allow you to delete text.

*	**x**	*delete single character under cursor*
	Nx	*delete N characters, starting with character under cursor*
	dw	*delete the single word beginning with character under cursor*
	dNw	*delete* `N` *words beginning with character under cursor; e.g.,* `d5w` *deletes 5 words*
	D	*delete the remainder of the line, starting with current cursor position*
*	**dd**	*delete entire current line*
	Ndd ***or*** **dNd**	*delete* `N` *lines, beginning with the current line; e.g.,* `5dd` *deletes 5 lines*

Cutting and Pasting Text

The following commands allow you to copy and paste text.

	yy	*copy (yank, cut) the current line into the buffer*
	Nyy ***or*** **yNy**	*copy (yank, cut) the next N lines, including the current line, into the buffer*
	p	*put (paste) the line(s) in the buffer into the text after the current line*

Other Commands

Searching Text

A common occurrence in text editing is to replace one word or phase by another. To locate instances of particular sets of characters (or strings), use the following commands.

/string	*search forward for occurrence of* `string` *in text*
?string	*search backward for occurrence of* `string` *in text*
n	*move to next occurrence of search string*
N	*move to next occurrence of search string in opposite direction*

Determining Line Numbers

Being able to determine the line number of the current line or the total number of lines in the file being edited is sometimes useful.

:.=	*returns line number of current line at bottom of screen*
:=	*returns the total number of lines at bottom of screen*
^g	*provides the current line number, along with the total number of lines, in the file at the bottom of the screen*

Saving and Reading Files

These commands permit you to input and output files other than the named file with which you are currently working.

:r filename<Return>	*read file named* `filename` *and insert after current line (the line with cursor)*
:w<Return>	*write current contents to file named in original* `vi` *call*
:w newfile<Return>	*write current contents to a new file named* `newfile`
:12,35w smallfile<Return>	*write the contents of the lines numbered 12 through 35 to a new file named* `smallfile`
:w! prevfile<Return>	*write current contents over a pre-existing file named* `prevfile`
