VI COMMANDS

MODES: There are two modes, command and text entry.

Command: Any keys you hit will be interpreted as a command rather than as text. vi is initially in command mode. Hit ESC to return to command mode from text entry mode. If in doubt where you are, hit ESC until terminal beeps; you are then in command mode.

Text entry: There are several commands which put you into text entry mode:

i	insert before cursor
a	append after cursor
o	open new line below cursor line
O	Open new line above cursor line

(There are also text entry commands for change and replace given below. You can correct errors using text entry by backspacing without leaving text entry mode. For more complicated corrections you should return to command mode.)

QUITTING AND SAVING:

ZZ	save changes and exit editor
:x	same as ZZ
:w	write file (save without exiting editor)
:wq	write and quit
:q!	quit without saving changes

(The ":" puts you on a command line at bottom of screen on which any line editor [ed or ex] command can be given.)

CURSOR MOVEMENT: The cursor moves only within the bounds of existing text.

h	left one position	(On most terminals the cursor or
j	down one line	arrow keys give same results.
k	up one line	^H [backspace], ^N [next], ^P
l	right one position	[previous] and space also give same results.)
H	Home cursor (first line of screen)	
M	Middle line of screen	
L	Last line of screen	
nG	Go to line n of file, where n is a numeric argument. (defaults to last line of file if no argument is given.)	
n|	move to column n of current line	

```
        0       beginning of line
        $       end of line
        ^       first nonwhite on current line
        +       first nonwhite on next line (<cr> does same thing)
        -       first nonwhite on previous line

        e or E  end of word             (Difference between e,w,b and
        w or W  forward word            E,W,B is whether punctuation is
        b or B  backward word           ignored.  Sentences are not
        )       forward sentence        recognized unless punctuation is
        (       backward sentence       followed by the standard two
        }       forward paragraph       spaces (or an end of line).
        {       backward paragraph      Paragraphs are delimited by
        ]]      forward section         paragraph macros (e.g., .P) or
        [[      backward section        blank lines.  Sections are
                                        delimited by heading macros
                                        or end of file.  These
                                        objects (words, sentences,
                                        etc.) can be used to indicate
                                        scope of various commands
                                        such as delete, change, etc.)
SCREEN MOVEMENT:

        ^D      scroll Down
        ^U      scroll Up
        ^F      page Forward
        ^B      page Backward
        ^E      one line forward
        ^Y      one line backward

TEXT DELETION:

        x       delete character                (These commands can be
        X       delete previous character       preceded by an integer
        dd      delete line                     argument to indicate the
        d$      delete to end of line           number of characters,
        d)      delete to end of sentence       words, etc., to be
        d}      delete to end of paragraph      deleted.)
        dw      delete word

TEXT ALTERATION:

        r       replace character under cursor with next character typed
                (Leaves you in command mode)
        R       Replace text with new text      (Replacement, substitu-
        s       substitute text for character   tion, or change begins
```

```
        cw      change word to new text         at cursor position and
        c)      change sentence to new text     continues until you hit
                                                ESC.)

TEXT MOVING:

        yy      yank copy of line into buffer (can be preceded by
                numeric argument; can also do yw, y), y}, etc., to
                yank other objects.)
        p       put after cursor last item yanked or deleted
        P       Put before cursor last item yanked or deleted

CUTTING AND PASTING:

        mx      mark cursor location as position "x", where x is a
                letter a-z
        `x      move cursor to position previously marked by "x"
        "x      use named buffer "x", where x is a letter a-z, for
                following command

(Notes: ` can be used to indicate scope of a yank or delete.  Thus to
move a chunk of text, first move the cursor one position past the last
character you want to move and mark it, say ma.  Then move the cursor to
the first character of the chunk and either yank or delete (depending on
whether you want to copy or move) into a named buffer, say either "by`a
or "bd`a .  (This either yanks or deletes text down to position marked
"a" into buffer "b".)  Now you simply move the cursor to the new
location at which you want to place the chunk of text and say "bp .
This operation not only works to move or copy within a file but also to
move or copy from one file to another.  After you have yanked or deleted
text into a named buffer, just type :e filename and the editor will
switch to the new file (which may or may not already exist), then put
the text whereever you want it in the new file.  :e# toggles between the
two files.  The editor will not let you abandon a changed file in this
manner without writing changes.  Cut and paste operations can be mapped
to your terminal's function keys so that you do not have to name buffers
or markers explicitly.)

SEARCHING:

    /pattern    search for next occurrence of "pattern"
    ?pattern    search for preceding occurrence of "pattern"
        n       repeat the last search command ("/" or "?") for next
                occurence
        fx      find "x" on current line, where "x" is any single
                character
```

```
        Fx      same as "f" except searches backward
```

(Note: After typing the "/" or "?" the cursor will be positioned on a command line at bottom of screen. Type search pattern there, terminating with <cr> or ESC. / search wraps to beginning of file, ? to end of file.)

SHELL AND LINE EDITOR COMMANDS:

```
        :       execute line editor command
        !       execute shell command
```

Examples:

```
  :se wm=8      set wrap margin to 8 (i.e., 72 columns). Creates new
                lines (breaking at whole words) as needed so you don't
                have to type carriage returns.
  :se noai      turn off autoindent
 :g/p1/s//p2/g  global replacement of pattern p1 by pattern p2
    !}fmt       format paragraph (i.e., fill). default is 72 column
                lines.
    !}sort      sort paragraph (usually a list of items terminated by a
                blank line)
  !}tr a-z A-Z  translate paragraph to upper case
  !}tr A-Z a-z  translate paragraph to lower case
    !sh         create shell (e.g., to do ls). ^D to return to editor.
```

(I have used paragraphs here. You can use other objects, of course. These, and other more complicated commands (such as spell checking) can be conveniently mapped to your terminal's function keys so that you can perform them with a single keystroke.)

MISCELLANEOUS:

```
        J       Join current line and next line into one line
        u       undo effect of previous command
        .       repeat previous command
        %       find parenthesis or brace matching one under cursor
        ^L      redraw screen (useful if screen becomes garbled by
                messages)
        ^V      in text entry mode, next character to be typed is a
                control (non-printing) character
        ^D      in text entry mode, backs up over autoindent whitespace
```