# Crib Sheet for Visual Editor *vi*

## Entering the Editor

- To edit a file `filename` with `vi`, issue the command `vi filename`.
- To view a file `filename` with `vi` but without the permission to modify the file, issue the command `view filename`. (This is in case you want to prevent yourself from making an error by changing the contents.)

## Scrolling

*Control*-`d` scrolls down half a screen
*Control*-`u` scrolls up half a screen
*Control*-`f` scrolls down a full screen (forward)
*Control*-`b` scrolls up a full screen (backward)
*Control*-`e` scrolls down a line
*Control*-`y` scrolls up a line

## Cursor Control

`h` moves left one space
`j` moves down one line
`k` moves right one space
`l` moves up one line
space bar moves right one space
carriage return moves to beginning of the next line
`G` ("go") moves to the end of the file
`5G`, for example, moves to line 5 of the file
`$` moves to the end of the current line
`^` moves to the beginning of the current line

## Deletion

`x` deletes the current character
`5x`, for example, deletes the current and following four characters
`dd` deletes the current line
`7dd`, for example, deletes the current and following six lines
`dw` deletes the current word
`ra` replaces the current character with the character `a`

`d` followed by a `$` deletes from the cursor to the end of the line

## Changing/Insertion

The following commands are all terminated when *ESCape* is hit. Subsequent *ESCape* commands only fleep the screen and are harmless. The `vi` editor is by default a “command” mode editor. By default, keystrokes are not inserted into the text but rather direct the movement of the cursor around in the text. This is in contrast with many other editors (emacs, Notepad, Wordpad, etc.), which are by default in “insert” mode, so that typed characters are inserted at the cursor location and the cursor movement is done by characters (the arrow keys) other than the usual typed characters.

Note that the command `xp` causes the character at the cursor and the character after the cursor to be flipped. For example, if the cursor is pointing to the `h` character of a string `htis`, and the command `xp` is issued, then the string will be changed to `this`. This is the best way to fix the effect of dyslexic fingers doing the typing.

`a` to begin appending text after the cursor
`i` to begin inserting text before the cursor
`cw` to change the text from the cursor location to the end of current word
`cc` to change the entire current line
`C` to change the text from the cursor location to the end of current line
`o` to put text in a new line beginning after the current line
`O` to put text in a new line beginning before the current line
`R` to begin overwriting (replacing) text with new text

## General Stuff

*Control*-`g` causes display of the current line number in the file, the number of lines in the file, and the file name.
*Control*-`L` causes redisplay of the screen
`ZZ` saves the current edit and exits `vi`
`u` undoes the most recent change
a period (`.`) causes a repeat of the most recent change

Note that the `u` (“undo”) command acts differently with different systems. In some systems and versions and setups for `vi`, a sequence of `u` commands causes all the most recent changes to be undone, one at a time. This has the advantage, which is sometimes desired, of being able to back out of several changes you decide you don't want after all. In other systems, the `u` command itself becomes the most recent edit change. This means that a second `u` command will undo that undo, and put back the change you just decided you didn't want. This also can be the right thing to have available. If you can't remember your most recent change, or if your most recent change was made by your cat walking on the keyboard, sometimes you want to see what that change was and be able to toggle back and forth to decide which version is really correct. Neither option is a clear winner, and you should be aware of which is used in the system you are using, so you won't be surprised by what happens with repeated undo commands.

## Block Moves and Copies

This is perhaps the least convenient feature of `vi`. One way to move a block of 7 lines, say, is to move the cursof to the first of the seven lines, then issue a `7dd` command to delete then. Actually, this puts the "deleted" lines into a buffer. If you then move the cursor to where you wish to put them, the command `p` will put the buffer back *after* the current line, and the command `P` will put the buffer back *before* the current line. To copy text, do the same but also put back the buffer where you originally did the deletion before moving to where the copy is to be made. See also the next section.

## Shell and *ed* Commands

Typing a colon while in command mode causes the colon to be displayed at the bottom of the screen and an escape to a different command level. At this point, `!command` will cause the shell command `command` to be executed without leaving the editor.

`:w` writes the edit buffer to the file name with which you entered `vi`
`:w junk` writes the edit buffer to a file named `junk`
`:r junk` reads the file named `junk` into the edit buffer
`:q` quits `vi` without saving the current buffer
`:!chmod +w filename` changes the `filename` permission to allow writing

The quit command is useful if you've done massive damage to a file by mistake. The `!chmod` command is useful if you've edited a read-only file and now want to save it. It changes the permissions on the file so you can save it with the original name. Another fix is to issue a `:w junk` command. This write the current buffer as a file named `junk`. Then change the permissions on the original file (or delete it entirely) and move the file `junk` to have the original name. (Trust me, you are likely to need this maneuver sooner or later.)

**Global Searching and Substituting Examples**

- `:1,$s/old/new/` For all lines (`$` refers to the last line), substitute the string `old` for the first occurrence in any line of the string `new`.
- `:1,$s/old/new/g` For all lines, substitute the string `old` for all occurrences in any line of the string `new`. (The `g` at the end means "global.")
- `:5,13s/old/new/` For lines 5 through 13, substitute the string `old` for the first occurrence in any of these lines of the string `new`.
- `:.,13s/old/new/` For all lines from the current line (that's the meaning of the period) through line 13, substitute the string `old` for the first occurrence in any of these lines of the string `new`.
- `/gribble` Move the cursor to the next occurrence of the string `gribble`.
- `?gribble` Move the cursor to the previous occurrence of `gribble`.
- `n` Having done one of the above searches, repeat the search.