**Author:** FOSSEE project, IIT Bombay

# Basic editing and editors

## vim

Vim is a very powerful editor. It has a lot of commands, and all of them cannot be explained here. We shall try and look at a few, so that you can find your way around in vim.

To open a file in vim, we pass the filename as a parameter to the `vim` command. If a file with that filename does not exist, a new file is created.

```
$ vim first.txt
```

To start inserting text into the new file that we have opened, we need to press the `i` key. This will take us into the *insert* mode from the *command* mode. Hitting the `esc` key, will bring us back to the *command* mode. There is also another mode of vim, called the *visual* mode which will be discussed later in the course.

In general, it is good to spend as little time as possible in the insert mode and extensively use the command mode to achieve various tasks.

To save the file, use `:w` in the command mode. From here on, it is understood that we are in the command mode, whenever we are issuing any command to vim.

To save a file and continue editing, use `:w FILENAME` The file name is optional. If you do not specify a filename, it is saved in the same file that you opened. If a file name different from the one you opened is specified, the text is saved with the new name, but you continue editing the file that you opened. The next time you save it without specifying a name, it gets saved with the name of the file that you initially opened.

To save file with a new name and continue editing the new file, use `:saveas FILENAME`

To save and quit, use `:wq`

To quit, use `:q`

To quit without saving, use `:q!`

### *Moving around*

While you are typing in a file, it is in-convenient to keep moving your fingers from the standard position for typing to the arrow keys. Vim, therefore, provides alternate keys for moving in the document. Note again that, you should be in the command mode, when issuing any commands to vim.

The basic cursor movement can be achieved using the keys, `h` (left), `l` (right), `k` (up) and `j` (down).

```
      ^
      k
< h       l >
      j
      v
```

Note: Most commands can be prefixed with a number, to repeat the command. For instance, `10j` will move the cursor down 10 lines.

#### *Moving within a line*

| Cursor Movement | Command |
|---|---|
| Beginning of line | `0` |

| | |
|---|---|
| First non-space character of line | ^ |
| End of line | $ |
| Last non-space character of line | g_ |

***Moving by words and sentences***

| Cursor Movement | Command |
|---|---|
| Forward, word beginning | w |
| Backward, word beginning | b |
| Forward, word end | e |
| Backward, word end | ge |
| Forward, sentence beginning | ) |
| Backward, sentence beginning | ( |
| Forward, paragraph beginning | } |
| Backward, paragraph beginning | { |

***More movement commands***

| Cursor Movement | Command |
|---|---|
| Forward by a screenful of text | C-f |
| Backward by a screenful of text | C-b |
| Beginning of the screen | H |
| Middle of the screen | M |
| End of the screen | L |
| End of file | G |
| Line number num | [num]G |
| Beginning of file | gg |
| Next occurrence of the text under the cursor | * |
| Previous occurrence of the text under the cursor | # |

Note: C-x is Ctrl + x

## *The visual mode*

The visual mode is a special mode that is not present in the original vi editor. It allows us to highlight text and perform actions on it. All the movement commands that have been discussed till now work in the visual mode also. The editing commands that will be discussed in the future work on the visual blocks selected, too.

## *Editing commands*

The editing commands usually take the movements as arguments. A movement is equivalent to a selection in the visual mode. The cursor is assumed to have moved over the text in between the initial and the final points of the movement. The motion or the visual block that's been highlighted can be passed as arguments to the editing commands.

| Editing effect | Command |
|---|---|
| Cutting text | d |
| Copying/Yanking text | y |
| Pasting copied/cut text | p |

The cut and copy commands take the motions or visual blocks as arguments and act on them. For instance, if you wish to delete the text from the current text position to the beginning of the next word, type `dw`. If you wish to copy the text from the current position to the end of this sentence, type `y)`.

Apart from the above commands, that take any motion or visual block as an argument, there are additional special commands.

| Editing effect | Command |
|---|---|
| Cut the character under the cursor | x |
| Replace the character under the cursor with a | ra |
| Cut an entire line | dd |
| Copy/yank an entire line | yy |

Note: You can prefix numbers to any of the commands, to repeat them.

## *Undo and Redo*

You can undo almost anything using `u`.

To undo the undo command type `C-r`

## *Searching and Replacing*

| Finding | Command |
|---|---|
| Next occurrence of `text`, forward | \text |
| Next occurrence of `text`, backward | ?text |
| Search again in the same direction | n |
| Search again in the opposite direction | N |
| Next occurrence of `x` in the line | fx |
| Previous occurrence of `x` in the line | Fx |

| Finding and Replacing | Command |
|---|---|
| Replace the first instance of `old` with `new` in the current line. | :s/old/new |
| Replace all instances of `old` with `new` in the current line. | :s/old/new/g |
| Replace all instances of `old` with `new` in the current line, but ask for confirmation each time. | :s/old/new/gc |
| Replace the first instance of `old` with `new` in the entire file. | :%s/old/new |
| Replace all instances of `old` with `new` in the entire file. | :%s/old/new/g |
| Replace all instances of `old` with `new` in the entire file but ask for confirmation each time. | :%s/old/new/gc |