# IPSL BootCamp: vi[*]

Institute Pierre Simone Laplace, IPSL BootCamp

The content of the BootCamp can be found in:
https://forge.ipsl.jussieu.fr/igcmg_doc/wiki/Train

March 23, 2016

## Contents

[*]Author of this chapter: Marco van Hulten, Laboratoire des Sciences du Climat et de l'Environnement (LSCE). This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License.

## 1 Introduction

You're at your desk and decided, after reloading Slashdot for the fiftieth time and realising that there won't be a new XKCD for another two days, to become productive. You are thus confronted with the decades old dilemma: the need to choose an editor: *vi* or *Emacs*. This introduction is about *vi*.

*vi* is a text editor originally created by Bill Joy in 1976 for the Unix operating system. While originally proprietary software, nowadays commonly used variants of vi are free software[1]. For this course you can use any vi variant as I tended to ensure compatibility with the POSIX standard and the OpenBSD implementation, which are more minimal than 'Vi IMproved' (vim) that has many more features. E.g., vim has syntax highlighting, vi has not. Several properties of vi:

- It is always installed on GNU/Linux and BSD systems,
- it is fast,
- but you have to learn to use it.

vi is a *modal* editor, which means that

- in *command mode* you can input commands (move, delete, copy, save etc.)
- in *insert mode* you type text (like you do in gedit or Notepad)

## 2 Basics

You can start up with any of these commands in a terminal:

| | |
|---|---|
| `$ vi` | on GNU systems this is, in fact, usually vim |
| `$ vim` | *Vi IMproved*: much extended version of vi |
| `$ gvim` | GUI version of vim |

When you started vim, you'll get this welcome screen:

```
~
~                          VIM - Vi IMproved
~
~                           version 7.4.827
~                      by Bram Moolenaar et al.
~                  Modified by <bugzilla@redhat.com>
~             Vim is open source and freely distributable
~
~                      Sponsor Vim development!
~           type  :help sponsor<Enter>    for information
~
~           type  :q<Enter>               to exit
~           type  :help<Enter>  or  <F1>  for on-line help
~           type  :help version7<Enter>   for version info
~
                                                   0,0-1         All
```

[1] In the context of software 'free' refers to liberty, not price: `http://www.gnu.org/philosophy/free-sw.html`, but you'll find many free software vi implementations to be gratis as well.

Here you are already confronted with some basic information. Many commands, like `:q<Enter>`, start with a colon (`:`). The help text presented to you is not actually in the buffer: it'll disappear as soon as you enter a command. The *buffer* is the in-memory text of a file. At the left of the screen are tildes (∼), which are the lines beyond the text buffer. Since the first line has a tilde, the buffer is empty. The left of the bottom of the screen is empty, which means vi has no messages to display, you're not entering a command, and you are in *command mode* (or *normal mode*). At the bottom near the right of the screen you see the current line number (0) and the column number (0–1). At the far right you'll see how far you are in the buffer: 'All' at the moment, since all is shown.[2]

Press `i` to change the mode to *insert mode*. Now you can insert text at the position of the cursor (column 1). As soon as you start typing, the first line is created and the column number is increased:

```
My first line in vi.
My second line.
And so on...
~
~
~
~
~
-- INSERT --                                         3,13          All
```

You can see that you are in *insert mode* as that is presented in the bottom-left corner of your screen. We see that we finished typing when the cursor was at line 3, column 13. If you press Escape now, you'll get back to *command mode*. If you are not sure in what mode you are and want to get back to command mode (the normal mode), press the Escape key until your terminal starts beeping.

Now we may want to move around in the text. For that we'll use the navigation keys h, j, k and l. The letter `h` and `l` are for left and right, respectively (since they are the first and last in the sequence on the keyboard). The `j` is for down (looks like an downward arrow), and `k` is for up. For instance, when typing `hhhhkk`, you'll end up at the end of the word 'first' in the buffer:

```
My first line in vi.
My second line.
And so on...
~
~
~
~
~
                                                     1,8           All
```

There are other ways to navigate to that place. Especially with longer texts it would be handy to know how. So, let's first go back to the end of the file by pressing the inverse navigation series: `jjllll`. Now press `1Gee`. You are again at the end of the word 'first'. The `1G` sets the cursor at the start of the buffer.[3] The letter `e` jumps to the end of a word. For completeness, the inverse navigation is `G$`, where `G` sends you to the last line and `$` to the end of the line.

You can save the file with `:w` (the colon puts you in a state that some call the *command-line mode*), where `w` stands for *write*. At first try, vim may display the error "`E32: No file name`".[4] That means it does not know where to write the file to, so you need an argument to the write command:

[2]Plain vi (on OpenBSD) does not show this information, but you can show some information by pressing Ctrl-g.
[3]That is plain vi; in vim `gg` works as well, so here you may press `ggee`.
[4]Plain *vi* does not do that and saves your buffer in a temporary file.

```
My first line in vi.
My second line.
And so on...
~
~
~
~
~
:w myfile.txt
```

and you'll get back in command mode, now with some information at the bottom:

```
My first line in vi.
My second line.
And so on...
~
~
~
~
~
"myfile.txt" [New] 3L, 50C written                          1,8           All
```

Usually I don't give the file argument within vi. Instead I inform vi what files to open by supplying them at start-up:

```
$ vi mydoc.tex
```

If `mydoc.tex` exists in the current directory, the buffer gets filled with the content of mydoc.tex. When I then make changes and write, this file gets updated on the disk. If the file did not exist, a new file is created with the name I supplied at the command shell, as soon as I give the write command (`:w`).

A final command that you must know is how to quit a buffer or vi. That is `:q` to simply quit the current buffer gracefully. You can use `:wq` to write the buffer to disk and quit. Or if you made changes to the buffer but *don't* want to save, `:q!`, the exclamation mark meaning *do it no matter what!*.

## 3 Exercises

The exercises are in *VIM Tutor*:

| | |
|---|---|
| `$ vimtutor` | interactive vim tutorial |
| `$ gvimtutor` | GUI version of vimtutor |

Use the right locale for different languages:

| | |
|---|---|
| `$ locale -a` | list the available locales |
| `$ LC_ALL=fr_FR.utf8 vimtutor` | in French French |
| `$ LC_ALL=en_GB.utf8 vimtutor` | in British English |

The interactive tutorial gets copied to the temporary directory of the operating system (`/tmp/`):

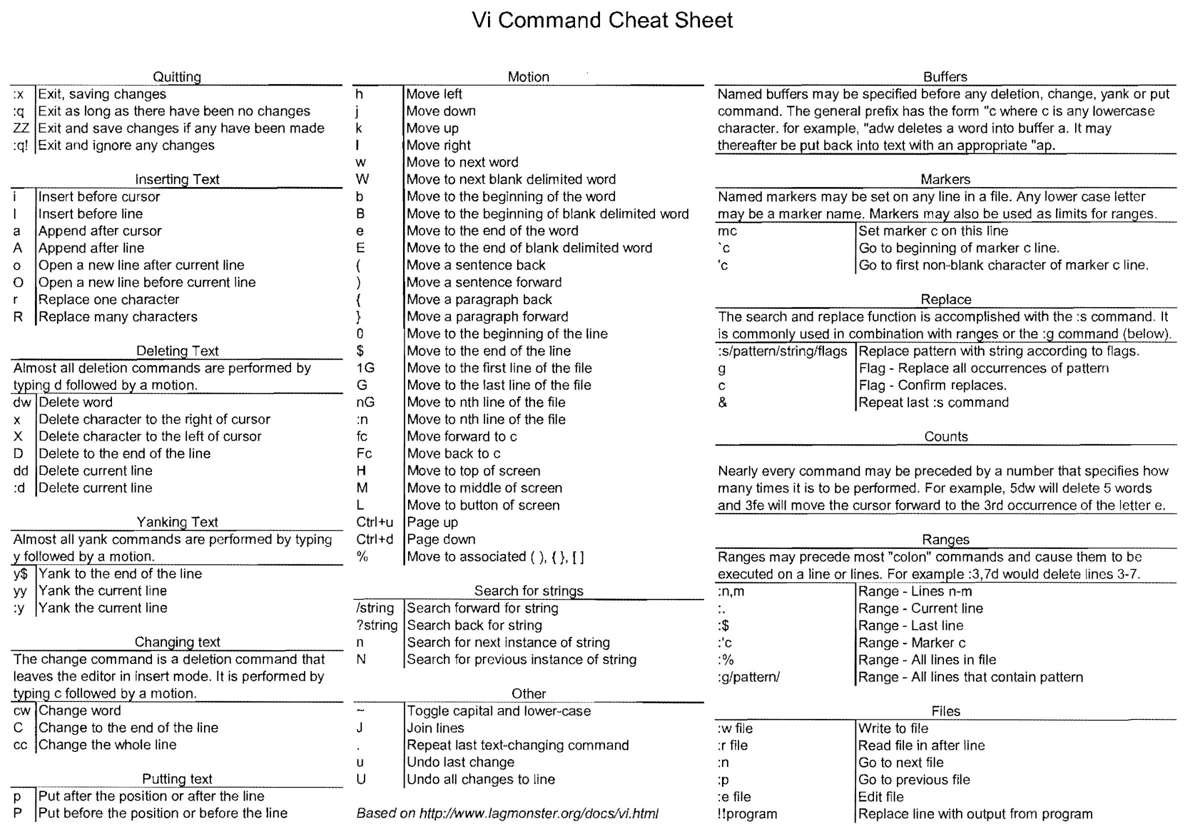
```
===============================================================================
=    W e l c o m e   t o   t h e   V I M   T u t o r    -    Version 1.7      =
===============================================================================

     Vim is a very powerful editor that has many commands, too many to
     explain in a tutor such as this.  This tutor is designed to describe
     enough of the commands that you will be able to easily use Vim as
     an all-purpose editor.

     The approximate time required to complete the tutor is 25-30 minutes,
     depending upon how much time is spent with experimentation.

"/tmp/tutorHhuEaK" 970 lines, 33257 characters
```

Edit the file to do the exercises.[5] Since you will be asked to remove parts of the manual or move around many lines, it may be useful to open another terminal and open there a vim session (by creating a new file, editing an existing or starting another vimtutor session).

## 4 Useful links

- Arguments for using vi, with examples: `http://www.viemu.com/a-why-vi-vim.html`
- Vi Lovers Home Page: `http://thomer.com/vi/vi.html`
- Wiki book: `https://en.wikibooks.org/wiki/Learning_the_vi_Editor/Getting_acquainted`
- Vim documentation: `http://vimdoc.sourceforge.net/htmldoc/help.html`
- vi(1): `http://man.openbsd.org/OpenBSD-current/man1/ex.1`
- Regular expressions: `http://man.openbsd.org/OpenBSD-current/man7/re_format.7`
- POSIX standard: `http://pubs.opengroup.org/onlinepubs/9699919799/utilities/vi.html`

[5]Most exercises are compatible with plain (OpenBSD's) vi, except for some of the sections 2.7, 4.1, 4.2, 5.3, 6.4, 6.5 and 7.



Vi Command Cheat Sheet

Quitting

| | |
|---|---|
| :x | Exit, saving changes |
| :q | Exit as long as there have been no changes |
| ZZ | Exit and save changes if any have been made |
| :q! | Exit and ignore any changes |

Inserting Text

| | |
|---|---|
| i | Insert before cursor |
| I | Insert before line |
| a | Append after cursor |
| A | Append after line |
| o | Open a new line after current line |
| O | Open a new line before current line |
| r | Replace one character |
| R | Replace many characters |

Deleting Text

Almost all deletion commands are performed by typing d followed by a motion.

| | |
|---|---|
| dw | Delete word |
| x | Delete character to the right of cursor |
| X | Delete character to the left of cursor |
| D | Delete to the end of the line |
| dd | Delete current line |
| :d | Delete current line |

Yanking Text

Almost all yank commands are performed by typing y followed by a motion.

| | |
|---|---|
| y$ | Yank to the end of the line |
| yy | Yank the current line |
| :y | Yank the current line |

Changing text

The change command is a deletion command that leaves the editor in insert mode. It is performed by typing c followed by a motion.

| | |
|---|---|
| cw | Change word |
| C | Change to the end of the line |
| cc | Change the whole line |

Putting text

| | |
|---|---|
| p | Put after the position or after the line |
| P | Put before the position or before the line |

Motion

| | |
|---|---|
| h | Move left |
| j | Move down |
| k | Move up |
| l | Move right |
| w | Move to next word |
| W | Move to next blank delimited word |
| b | Move to the beginning of the word |
| B | Move to the beginning of blank delimited word |
| e | Move to the end of the word |
| E | Move to the end of blank delimited word |
| ( | Move a sentence back |
| ) | Move a sentence forward |
| { | Move a paragraph back |
| } | Move a paragraph forward |
| 0 | Move to the beginning of the line |
| $ | Move to the end of the line |
| 1G | Move to the first line of the file |
| G | Move to the last line of the file |
| nG | Move to nth line of the file |
| :n | Move to nth line of the file |
| fc | Move forward to c |
| Fc | Move back to c |
| H | Move to top of screen |
| M | Move to middle of screen |
| L | Move to button of screen |
| Ctrl+u | Page up |
| Ctrl+d | Page down |
| % | Move to associated ( ), { }, [ ] |

Search for strings

| | |
|---|---|
| /string | Search forward for string |
| ?string | Search back for string |
| n | Search for next instance of string |
| N | Search for previous instance of string |

Other

| | |
|---|---|
| ~ | Toggle capital and lower-case |
| J | Join lines |
| . | Repeat last text-changing command |
| u | Undo last change |
| U | Undo all changes to line |

*Based on http://www.lagmonster.org/docs/vi.html*

Buffers

Named buffers may be specified before any deletion, change, yank or put command. The general prefix has the form "c where c is any lowercase character. for example, "adw deletes a word into buffer a. It may thereafter be put back into text with an appropriate "ap.

Markers

Named markers may be set on any line in a file. Any lower case letter may be a marker name. Markers may also be used as limits for ranges.

| | |
|---|---|
| mc | Set marker c on this line |
| `c | Go to beginning of marker c line. |
| 'c | Go to first non-blank character of marker c line. |

Replace

The search and replace function is accomplished with the :s command. It is commonly used in combination with ranges or the :g command (below).

| | |
|---|---|
| :s/pattern/string/flags | Replace pattern with string according to flags. |
| g | Flag - Replace all occurrences of pattern |
| c | Flag - Confirm replaces. |
| & | Repeat last :s command |

Counts

Nearly every command may be preceded by a number that specifies how many times it is to be performed. For example, 5dw will delete 5 words and 3fe will move the cursor forward to the 3rd occurrence of the letter e.

Ranges

Ranges may precede most "colon" commands and cause them to be executed on a line or lines. For example :3,7d would delete lines 3-7.

| | |
|---|---|
| :n,m | Range - Lines n-m |
| :. | Range - Current line |
| :$ | Range - Last line |
| :'c | Range - Marker c |
| :% | Range - All lines in file |
| :g/pattern/ | Range - All lines that contain pattern |

Files

| | |
|---|---|
| :w file | Write to file |
| :r file | Read file in after line |
| :n | Go to next file |
| :p | Go to previous file |
| :e file | Edit file |
| !!program | Replace line with output from program |

Figure 1: Cheat sheet with basic vi(1) commands, downloaded from `http://d.umn.edu/~becke405/`. Many others can be found through a simple web search.