Vim Editor Essentials

Vim (Vi Improved) is a text editor that is one of the direct derivatives, and one of the prime ingredients of the UNIX hacker culture. With its extensive support and simplicity, Vim stands out as a unique text editor with numerous features. It's both simple and powerful!

We come across numerous GUI text editors like KWrite and Gedit, with which you need to use the mouse extensively to access some of the features. Unlike these, Vim doesn't support the mouse. Instead, it stays close to the command-line with extensive key-bindings support. Many believe this is a feature, because it facilitates better control over text processing with simple commands.

Vim is a mode editor. Its three modes are: INSERT, REPLACE and VISUAL. This article will focus on how Vim can help get going with your text editing tasks. If you are new to Vim, open a terminal session and type the following command:

```
$ vim test.txt
```

This will launch the editor. Press 'i' and start writing. You might feel a bit awkward at this stage since you are more comfortable with GUI text editors and probably might think: what's the point? Well, Vim has its many advantages. Let's create a large file using the *dd* command, filled with 0s. Now open that file in GUI editors like Gedit or KWrite. What happens? It takes a lot of time to render the text, and eventually crashes, right? Now, try to open it with Vim, and see the difference.

Let us have a roundup of the treasures of Vim:

- Mode-controlled text editing
- Regular expressions support
- Text replacement, searching, deletion
- Auto completion
- Auto indentation
- Split windows
- Multiple file tabs
- Syntax colouring
- Dictionary
- Better help
- Sleek and simple

Installing Vim

On most of the distros, Vim is installed by default. However, if you're on an Ubuntu system like me, you'll need to download a few extra packages to get the full functionality.

Open a terminal and run the following command:

```
$ apt-get install vim-full.
```

Time to get started now. Open a terminal and type the command given below:

```
$ vim
```

You'll see something similar to the screenshot in Figure 1.

To open or create a new file, type...

```
$ vim filepath
```

Vim opens in the command mode where it accepts Vim commands only. To edit the text, press 'i' to switch to *Insert* mode. Now you can type text. The cursor can be moved using standard keyboard arrows or using the keys given in Figure 2.

To exit from the Vim editing mode, press ESC. Now, to save the changes to the file, type *:w* ; to save and quit, type *:wq* ; and to quit without saving changes, type *:q!*

Let's now look at some of the important operating modes of Vim.

Command mode: We can use different letters or letter sequences to command Vim operations. Vim commands are case sensitive. The ESC keystroke can terminate the command.

Insert mode: When the editor is in *Insert* mode, on the footer part you will see the '-- INSERT --' status. To switch back to command mode, press ESC. Text can be inserted in this mode. Press 'i' to initiate *Insert* mode. It is possible to initiate *Insert* at different portions of the text. To insert at the next new line, press 'o'; to insert at the new line before the current one, press 'O'; and to insert at the end of the line, press 'A'.

Command line mode: In command line mode, we use commands starting with ':' which will show commands that we're tying at the moment, at the footer part of the editor. For example, *:help*, *:wq*, etc.

Basic operations

To open and save files

To open files with Vim, use the following command: *vim file.* To open multiple files as split windows, type: *$ vim -O file1 file2*. To open a file from the Vim command-line, use the following: *:e file* // Auto complete is supported by pressing TAB.

To save the changes to the currently opened file, press ESC to switch to the command mode from the editing mode, then type *:w* and press ENTER. To exit Vim after saving a file, type *:wq*. To just quit without saving, use *:q!* To save the current file as a new file, use the command line *:w new_filepath*.

Figure 1: The Vim editor

Figure 2: Cursor navigation

Manipulating files

Cut, copy and *paste* are essential features of any text editor. To cut the current word, place the cursor at the starting character of the word and use 'cw'. To cut the current line, use 'cc' in the command mode. To copy the current word, place the cursor at the starting letter and use 'yw', while to copy an entire line, use 'yy'. To paste content, place the cursor at the position where the contents of the buffer need to be placed, and press 'p'.

To remove a word, place the cursor at the starting letter of the word, and press 'dw', while to remove a line, press 'dd', and to remove a character, press 'x'. To undo the previous action, press 'u', while to redo an action, 'Ctrl + r'. To repeat a previously performed task, press '.' To list the available *undo tasks*, ':undolist'.

To search, substitute and replace

To search for a word/sentence while in the command mode, type */word* and press ENTER, where 'word' is the word you're searching for. To repeat the search in the forward direction, press *n*, while to reverse the direction of the search, press *?*.

To replace a part of the text, use:

- :%s/word_pattern/replacement_text/ – replaces

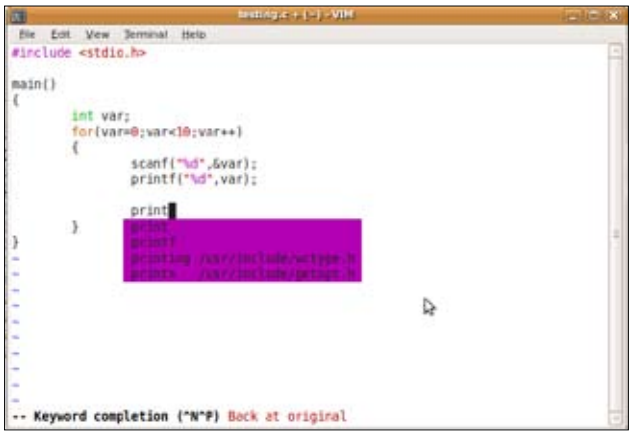

Figure 3: Auto completion in Vim

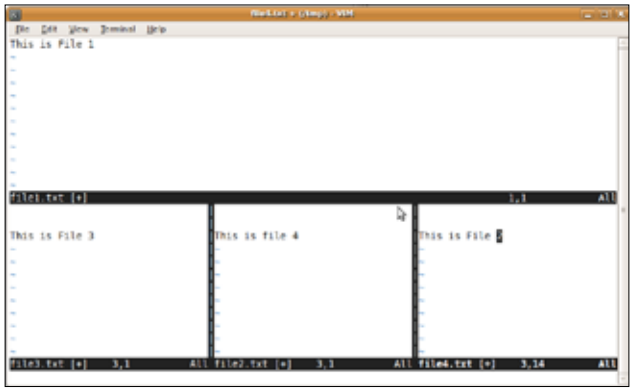

Figure 4: Using multiple files side-by-side

the first word of every line

- :%s/word_pattern/replacement_text/ – global replacement of the word
- :%s/word_pattern/replacement_text/gc – replacement with interactive *Yes/No* confirmation

%s represents the entire text in the file. It is also possible to use a range of lines for replacement. In case we want to use a replacement only between the lines 1 and 25, the above command can be used as:

```
:1,25 s/word_pattern/replacement_text/
```

You can also use regular expressions for replacement and searching. This is similar to the *sed* (stream editor) syntax.

Repeating commands *N* times

If you want to repeat a specific command *N* number of times, prefix the number of times before the command. For example, to copy the next five lines, use 5yy, where yy is the command for copying 1 line. To move the cursor to the eighth word of a line, use '8w'.

Selecting text

Vim has an awesome option to select a portion of text, just as we do using a mouse in GUI-based editors. The special purpose option is called Visual Mode.

Press 'v' in the command mode and navigate the cursor—the selected text will be highlighted. After making the necessary selection, press a command for an operation.

For example, press 'v', and make selections by moving the cursor down and right. For deletion, press 'd', or any other required operation.

Advanced Vim features

Vim is an advanced editor that has capabilities like spell check, auto indentation for different programming languages, auto-completion, syntax colouring and more. Syntax colouring and auto indentation enables programmers to write code in an artistic way, making the code more pleasing to the eye. Vim also supports colouring schemes with themes.

Syntax colours and auto indent

To enable syntax colouring, use the command line *: syntax on*.

Vim supports many programming languages for indentation. To enable indentation, use the command line, *:set autoindent*. To explicitly enable C syntax indentation, use *:set cindent*.

Spell check and auto complete

Vim also supports spell checking. It highlights wrongly spelled words with a different colour. To enable spell checking, at the command line, use *:setlocal spell spelllang=en_us*, for US English while writing a document.

For auto completion of a word or a keyword, type a part of the word and press *Ctrl+N*, after which a pop-up will show the list of available words (Figure 3). You can scroll through the word list by pressing *Ctrl+N* repeatedly. Auto completion works for words that have already been typed previously in the current file.

Install the *ctags* package if you want inbuilt auto-completion support for various programming languages as well.

Split windows

There are certain occasions when you need to work with multiple files simultaneously. Normally when you want to refer to the contents of a file while working on another file, you open two instances of editors open side-by-side. But Vim is intelligent enough to open multiple files simultaneously.

To start a horizontal window by splitting the current Vim instance, use the command *Ctrl+w+s*, or at the command line, use *:split*. For a vertical split, type *Ctrl+w+h,* or at the command line, *:vsplit*. Refer to Figure 4. You can now switch between different split instances using the Ctrl+w+w shortcut. These multiple splits can be further split using the same commands. The split windows act as separate instances of Vim so that we can

open and work on different files, side by side.

To exit from a split window, use the Ctrl+w+q shortcut, or at the command line, use *:q!*

While we're on the subject of split windows, there is a command called *vimdiff* for viewing the difference between two files. Suppose you have two versions of a text file, and you want to look at the changes made and compare it with the original, use the command:

```
$ vimdiff file1 file2
```

When you scroll through the file, both file instances will be scrolled. Sounds interesting, right? Take a look at Figure 5.

Vim also supports window tabs. To open a new tab, at the command line, use *:tabnew*. Now to switch between tabs, use the Ctrl+PAGEDOWN or Ctrl+PAGEUP shortcuts.

Line numbers

In certain cases, especially while writing code or editing a script, it is convenient to see line numbers for the text file. To enable line numbers, use the command *:set number*. To disable the function, use *:set nonumber*.

Execute external commands

While working from Vim itself, you can execute external commands. To execute and view the output of a command, at the command line, use *:!command*, where 'command' is the command you want to execute. For example, to execute ls, try *:!ls*

It is also possible to paste the output of the executed command to the currently working file by using *:r !command*. For example, *:r ls /*

Vim has awesome features to process the contents of the file using external commands and to paste the output back to the file. It is also able to process text between a range of lines.

There are situations when a part of the text needs to be processed. Vim makes it convenient to process a given range of lines using external commands.

tr is a utility to perform translations based on the sequence inputs. We can easily perform translations such as lower case to upper case conversion using *tr* as the following bash command line.

```
slynux@slynux-laptop:~$ echo "This is a line of text 1234" | tr 'a-z' 'A-Z'
THIS IS A LINE OF TEXT 123
```

Suppose we have a large text file and the text between the lines 3-5 is to be converted to upper case. We can perform it easily using the external commands methodology like *:3,5 !tr 'a-z' 'A-Z'*,

which instructs Vim to perform *tr 'a-z' 'A-Z'* for the text between the lines 3-5.

Figure 5: Files opened using the vimdiff command

In this way, any kind of external command can be coupled with Vim to process any part of the text in a file, in a handy way.

Moving the cursor in a file

Vim enables moving the cursor to different positions by accepting different references. For programming tasks, often compile/runtime errors occur and compiler return errors are noticed with line numbers. In such cases, we need to move the cursor immediately to the error line referenced by the line number. To move to a line referenced by a line number, use *: line_number*. For example, to move to line 50 use *:50*

Essentially, we can also perform some actions using the corresponding commands:

- Move cursor to beginning of line – *0*
- Move cursor to end of line – *$*
- Move cursor to beginning of next line – +
- Move cursor to end of file – *G*

Scrolling page by page is very useful while dealing with large files. To do so, use *Ctrl+f* for forward scroll and *Ctrl+b* for backward scroll.

Getting help

Vim has an embedded command line called *:help* to get help with different operations. For more information on making substitutions, use *:help substitutions*; for search, *use :help search* and so on. To quit from help mode, use *:q*

If you would like more insights into the Vim editor, I would like to recommend *A byte of Vim* written by Swaroop C H [*www.swaroopch.com/notes/Vim*].

I guess that's all for now. Hope you will join our Vim club soon. Happy hacking, till we meet again! END

By: Sarath Lakshman

The author is a Hacktivist of Free and Open Source Software from Kerala. He loves working on the GNU/Linux environment and contributes to the PiTiVi video editor project. He is also the developer of SLYNUX, a distro for newbies. He blogs at *www.sarathlakshman.info*