LINUX VOICE

GET TO KNOW VIM

MASTERCLASS

Essential Linux tools explained – this month, say hello to the Vim text editor and some advanced features in Firefox

AN INTRODUCTION TO VIM

Whatever Unix system you're using, it will probably have Vim – let's learn this essential editor.

JOHN LANE

Vim, which is a hugely flexible text editor for Linux, is "Vi Improved". It carries a legacy that can be traced back to the 1970s, when Vi was born as the 'visual mode' for the Unix line editor, Ex (Ex itself was the eXtended version of the editor, Ed). The most jarring part of that legacy for new users is that Vim has two main operating modes.

LV PRO TIP

":help vim-modes" helps to explain Vim's operating modes

Command Mode is where you interact with Vim to move around, alter, save and exit; pretty much everything you can think of except the one thing you might want to do: type text into it. That's what the other mode (Insert Mode) is for. Sadly, many people take the time to learn one command at this point: **:q**, the quit command, so they can go off and find another editor. But *Linux Voice* readers are a hardy bunch, so we're staying with it...

Getting around

LV PRO TIP

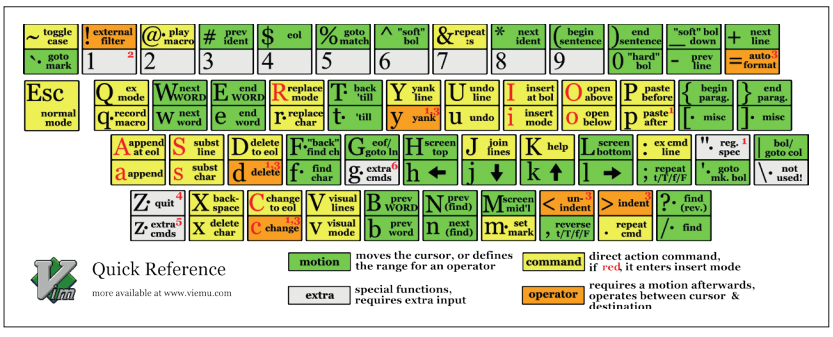
Most linux distros alias "vi" to "vim"

Another surviving part of its legacy is its quirky use of the **h**, **j**, **k** and **l** keys to move around the file. This can be traced back to Vi's creator using a terminal that didn't have separate cursor arrow keys but used these keys instead. Vim supports the normal arrow keys as well, but it's worth learning to use those letter keys because they're there directly underneath your fingertips and it'll make you much quicker once you've trained your muscle memory. For die-hards, the arrow keys can be turned off.

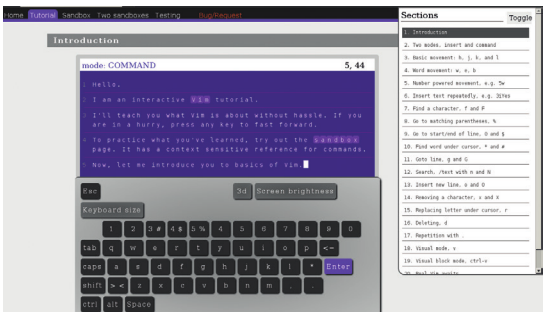
Quirks aside, let's get into Vim:

```
vim myfile
```

Like we mentioned, you'll find yourself in Command Mode. For basic navigation, you use the **hjkl** keys to move left, down, up and right within the file or there are commands that navigate words, sentences, paragraphs and pages.



Vim's Command Mode offers heaps of powerful commands to manipulate files.



Vim benefits from many online learning resources, like this interactive tutorial (www.openvim.com).

Commands are case-sensitive so **j** and **J** do different things. Most commands are one or two keystrokes and the first you'll want to learn is **i**, which enters Insert Mode. Once in Insert Mode, anything typed will be entered into the file until you press the Escape key, which returns you to command mode. Insert Mode is meant for short bursts of text entry – you're meant to place the cursor, enter Insert Mode, type some text and Escape back to Command Mode. New users might try to enter Insert Mode and then work as if they are using a simpler text editor such as Notepad, but to do so misses the point and power of Command Mode.

The next characteristic of Vim to understand is its ability to repeat commands. Any keystroke command given may be preceded with a number and that command will be repeated that many times. For example. enter **5h** to move left five characters. Using repeats like this with the navigation commands can help you move around your file very quickly.

Moving around is fine, but command mode also offers many ways to make modifications to your file. Begin with the basic (**i**)nsert, (**r**)eplace and e(**x**)tract. Some commands, like (**c**)hange and (**d**)elete, are

followed by a mandatory operator specifying what to operate on – these operators are the same as the navigation commands, so **3dw** deletes three words, **5x** deletes five characters.

A more sophisticated way to modify files is to use Ex commands. These hark back to the days before Vi when people used teletypes and didn't even have a monitor in front of them. Ex commands are typed at a prompt and perform modifications to the file being edited. You enter an Ex command from Vim's command mode by first entering a colon (**:**) and then the command. Ex commands take the form

```
:[range] command [args]
```

where the range and arguments are optional. The range defines the line, or lines, to perform the command on, and can be specified using line numbers, marks or special symbols: a full stop character represents the current line and **$** represents the last line. **%** is equivalent to **1,$**. You can create marks to represent lines of your choice: to define a mark, **a**, at your current position in a file, enter **ma** and you can then refer to that line as **a**. You can also use simple expressions like **+3** to refer to the line three ahead of the current line, or **$-10** to refer to the tenth-before-last line. Probably the most commonly used ex-command is to perform a global search and replace: **:s/%/foo/bar/g** replaces all occurrences of **foo** with **bar**. **foo** is a regular expression.

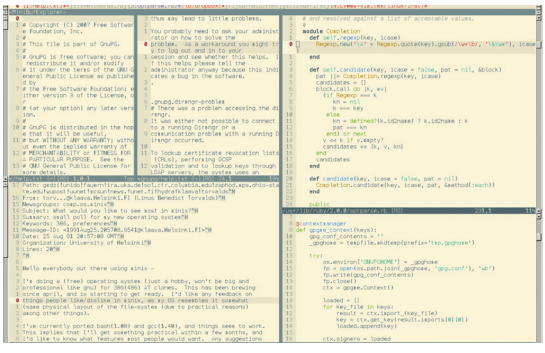
Vi has commands to cut or copy and paste text but it calls the copy action Yank instead. The commands are (**d**)elete, (**y**)ank and (**p**)aste. Deleting and yanking place the affected text into a general purpose buffer (which is not the desktop's clipboard). You can also use a named buffer by prefixing a command with **"** and a character (which names the buffer).

```
"a3dw
"byy
```

will delete three words into buffer **a** and yank the whole line into buffer **b**.

Buffers, windows and tabs

When Vim loads a file, it reads it into a buffer, which is the in-memory copy of the file that is altered as you edit it. The original file is only modified when you write the buffer back to disk. You use the **:w** command to do this. Vim can have many files open at the same time, each being held in its own buffer. You can list all buffers using **:ls** and navigate between them with **:b**. Split windows is a feature of Vim that enables you to view the file being edited in multiple places, or even view multiple different files. To split the window, enter **:sp** for a horizontal split or **:vs** for a vertical one. This will display the current buffer twice. Append a filename to instead load another file into the split window. If two windows display the same buffer, changes made in one will be reflected in the other. Window control commands are initiated with **Ctrl+W**. Moving between windows involves using **Ctrl+W** followed by a movement key, for example **Ctrl+W k** moves to the window above.



There are lots of ways to organise multiple files in Vim.

The other way to organise multiple files is to use tabs. You can have multiple tabs, each displaying multiple windows and, remember, each of those windows can display content from any buffer. Tab commands begin with **:tab**.

Pimp your Vim

One of the big features of Vim is that it is heavily customisable. You can view and change settings with the **:set** command. To see all settings:

```
:set all
```

And to enable a setting, for example:

```
:set number
```

will display line numbers. Prefix the setting name with **no** to disable it:

```
:set nonumber
```

To apply your setting preferences when starting Vim, you can store them in a file called **~/.vimrc**. This file can do more than just settings, though: by writing Vimscripts, you can create new functionality or modify existing features.

A good place to start customising is by key mapping. Any Command-Mode key sequence can be mapped to a key:

```
:map - dd
```

will cause Vim to delete the current line (that's the **dd** command) whenever you press **-**. Beware, however, mapping single keys will remap the main commands (there are no free single characters). To help avoid this you can use modifiers:

```
:map <c-d> dd
```

will map **Ctrl+d** to perform **dd**, which deletes the current line. You can also map multiple characters but you'll need to type them quickly to get their effect:

```
:map ,d dd
```

A coder's paradise

You can control whether tabs are hard or soft (implemented with spaces, often preferable these days). You can set up automatic indentation and code folding (hiding function bodies). The **=** command will indent the current selection. **:fold** will fold any selected block; **zo** opens a fold and **zc** closes it.

Finally, this tour would be incomplete without mentioning Gvim, which wraps Vim in a GUI window and exposes some of its commands through a traditional menu interface.

Do you want to know more?

- The Vim Tips Wiki **http://vim.wikia.com/wiki/Vim_Tips_Wiki**
- Open Vim Tutorial **http://openvim.com**
- ViEmu cheat sheet and tutorial **www.viemu.com**
- Vim website at **www.vim.org**