Mathias Schwelghofer, photocase.com

Getting beyond vi with the vim text editor

VIM TRICKS

You don't have to use the vim text editor as a latter day vi. These simple tricks will save you time and keystrokes. **BY JON KENT**

Vim has become the de facto replacement for vi in most Linux distributions. If you have been using Unix systems for a while, you probably have not even noticed that you are now using vim instead of vi because all the usual vi commands work as you would expect in vim. However, vim is an extremely powerful editor – think emacs without the need to grow additional fingers and attend yoga classes – and therefore is much more than just a vi clone. These features are ready for you to use, and once you start using them, you might wonder how you survived without them. This article puts you on the road to vim enlightenment by highlighting some useful features you might not know about if you're accustomed to using vim as another form of vi.

Smarter Searching

Searching is easy in vim, but what happens if you misspell a word or if you are not completely sure how the word is spelled? With the standard search, you are stuck; however, vim has an incremental search search mode that starts searching as soon as you start typing the word you want to find. This feature is incredibly useful but is not enabled by default. Luckily, all you have to do is enter

```
:set incsearch
```

in the command window to enable incremental searching. Now when you enter search mode, vim will start matching straightaway and highlight each match until you find what you are looking for. Once you have found what you want, you can simply hit Return or Esc to exit the search.

One issue is outstanding, even with incremental search enabled: Matching is still case sensitive. As luck would have it, you can also set up vim to be non-case sensitive by enabling the following options in the command window:

```
:set ignorecase
:set smartcase
```

Users coming from vi will recognize the *ignorecase* option, but *smartcase* might be new. The *smartcase* option, when set in conjunction with *ignorecase*, sets vim to search for any case unless uppercase is used in the search pattern. If you use an uppercase character, vim assumes you want to perform a case-sensitive search. This gives you options when you perform a search. For example,

```
/example
```

would match Example, exAMple, and example, whereas

```
/Example
```

would only match Example.

If you find these options useful and want to enable them permanently, you need to edit or create a *.vimrc* file in your home directory and add:

```
set incsearch
set ignorecase
set smartcase
```

On most Linux systems, you should find the default *vimrc* file located in */etc* or */etc/vim*, so if you are creating your own version, it is usually best to start by copying the default version into your home directory, renaming it *.vimrc*, and then making your changes.

The following requires that you have the full featured vim package and not a cut-down version, which is usually obvious when you run *vim --version*: Usually you will see *Small version without GUI* in the version summary. If your Linux uses the *alternatives* system, another approach is to look at the symbolic link vim has within */etc/alternatives* and see if it points to the *vim.tiny* binary instead of *vim* or *vim.full*. Ubuntu is guilty of installing a stripped-down package of vim called *vim-tiny*. Ubuntu users can simply install a vim package called *vim-full* to get the fully featured version.

To make vim act as a file browser, simply enter the following:

```
:e .
```

Vim now will display the contents of the current directory, which you can transverse by arrow keys or navigation keys. To open a file within vim, navigate to the file you want to open and press Enter. In the same manner, you can traverse directories by simply navigating to the directory you want to open and pressing Enter, at which time you will be presented with the contents of the directory. If the directory is large, you can also perform the usual vim searches.

Within vim, you can open any files without using the file browser by simply entering the name of the file you want to view. But what if you are not completely sure of the name? Vim has another trick up its sleeve: file completion. This works in much the same way as shell file completion, in which you simply enter a few characters of the file in question and press the Tab key for vim to complete the file name. If multiple files match the characters you have entered, you simply press Tab until you come to the file you want to edit and then press Enter.

File completion is fully configurable – this is vim after all. Modify the *wildmode* parameter either within your *vimrc* file or within vim itself. To see all the possible settings, enter:

```
:help wildmode
```

By default, this is set to full mode and acts as outlined previously.

The previous examples all work on file names, but what if you want to find the files that contain a common string? From the shell command line, you would use *grep* to show all the files that contain a common string. To do the same with vim, you simply start vim and then run:

```
:grep <string> *
```

Vim will find files that contain *string* and open them, then you can navigate between the files with vim commands:

```
:cn - go to next file
:cp - go to previous
:cc - show current match
```

When you view each file, vim starts at the location of the match in that file. If you have moved away from the match within a file and want to go back, use *:cc* to go back to the match. To move between files, use the *:cn* and *:cp* commands instead of *:n* and *:N* because they use vim error handling; therefore, vim stores the matching files differently.

Table 1: Navigating Vim Tabs

Command	Description
:gt	Next tag
:tabn	Next tag
:tabp	Previous tag
:tabfirst	First tag
:tablast	Last tag

Figure 1: Vim's tabbing feature helps you organize open files.

Editing Multiple Files

Vim, like vi, can edit multiple files at the same time. To transverse these files, you can use the *:n* (next) and *:N* (previous) commands, but vim also lets you use split screen mode to edit files:

```
:split
```

Now the screen is split in half with a horizontal rule, and you can use further split commands to split the screen into additional segments. To move among the windows, use Ctrl + W and arrow keys.

This feature makes editing multiple files easy, especially if you want to view both files at the same time. However, if splitting the screen is not to your liking, or if you have multiple files you want to edit, version 7 or later of vim has a new feature that could help: tabbing. Tabbing works by adding a tab to, by default, the top of the screen with the file you are editing. To use tabbing within vim, enter:

```
:tabnew <file name>
```

Don't forget that you can also use the auto-completion feature of vim with the *tabnew* command. Vim will open this file in a new tabbed window ready for you to edit. Another, possibly quicker, method is to supply all the files you want to open at the shell command line with the *-p* flag:

```
vim -p file1 file2 ➥
file3 file4
```

To navigate between tabs, use the commands shown in Table 1.

Figure 2: Vim's visual mode is useful for selecting and moving text.

If you don't like the tabs at the top of the screen, turn them off by running

```
:set showtabline=0
```

and then switch them back on with:

```
:set showtabline=2
```

If you prefer not to see the tabs, simply add the following to your *.vimrc* file:

```
set showtabline=0
```

With the tabbing display switched off, at times, you'll want to remind yourself of the tabs you have running. The following command provides such a summary:

```
:tabs
```

Vim will display a list of tabs and the file associated with each tab.

Finally, one of the most useful features of tabbing is the *tabdo* command, which lets you perform an action on all of the open tabs (Figure 1). As a simple example, the following command replaces the text *fred* with *joe* in all open tabs:

```
:tabdo %s/fred/joe/g
```

Purists look away, but yes, vim now comes with a built-in spell checker. The spell checker is often disabled by default. To enable spell checking, enter the following command:

```
:setlocal spell ➥
spelllang=en_gb
```

Replace *en_gb* with the language region code you require. Vim will highlight words that are incorrectly spelled in red and rare or uncapitalized words in blue. To navigate the words vim thinks are incorrect, use the key combinations shown in Table 2. The vim spell checker comes with other key combinations. Enter

```
: help spell
```

to see the extended documentation.

Vi users know that if you want to edit a section of code or text within a file, you can place the cursor on the last line you want to edit and run the following command to place a marker:

```
:ma
```

To copy this section, place the cursor at the start of the section and enter

```
:y'a
```

which yanks all the code or text from the cursor to the marker *a* you just placed. Then you can move the cursor to where you want to place the section and enter:

```
:p
```

With vim, this also works, but it has a slightly quicker way of achieving this result. In visual mode, you highlight the text you want to edit and then type an editing command. Three modes are available: line-by-line, character, and block mode. Line-by-line and character modes are probably the most useful, but block mode could be handy if you need to edit text tables. These modes are available via the key combinations shown in Table 3.

With vim's visual mode, to copy a section of text, simply enter visual mode, highlight the text you want, and type

```
:y
```

to yank the highlighted text into an unnamed buffer (Figure 2). Move the cursor to where you want to locate the text and type

```
:p
```

to place a copy of the text there.

It is also possible to use your mouse to highlight text by typing

```
:set mouse=a
```

or placing this command within your *.vimrc* file. Now when you highlight a section of text with your mouse, vim will go into visual mode automatically.

Conclusion

If you migrate from the vi world, some of the recent improvements to vim could be hard to find. In this article, I summarized some of the tricks you can play with the vim text editor. ■

Table 2: Spell Checker Key Combinations

Sequence	Description
]s	Go to the next misspelled word
[s	Go the previous misspelled word
z=	Display suggestions for correct spelling
zg	Add word as a correctly spelled word

Table 3: Visual Mode Options

Sequence	Description
Shift+v	Line-by-line mode
v	Character mode
Ctrl+v	Block mode