JOE 'ZONKER' BROCKMEIER (/USERS/ZONKER) | NOVEMBER 20, 2009

Vim 101: A Beginner's Guide to Vim

Ever wanted to learn Vim, but weren't sure how to start? No problem. We have you covered! This will be the first of a four-part (possibly longer) series covering how to use Vim and where to start using the world's best text editor.

Note that this is an article explicitly for beginners new to Vim. If you've been using Vim for a while, all of this should look entirely familiar--and you might

wonder why some topics aren't covered. Patience. We'll get to everything in due time, but there's a lot to cover!

What's Vim, and Why Do I Want It?

Vim is the editor of choice for many developers and power users. It's a "modal" text editor based on the vi (http://en.wikipedia.org/wiki/Vi) editor written by Bill Joy in the 1970s for a version of UNIX. It inherits the key bindings of vi, but also adds a great deal of functionality and extensibility that are missing from the original vi.

What the heck do we mean by *modal*? When you're using most word processors and text editors, the alphanumeric keys (i.e., a through z, 1 through 9) are only used to input those characters unless they're modified by a control key. In Vim, the *mode* that the editor is in determines whether the alphanumeric keys will input those characters or move the cursor through the document.

For example, many text editors and word processors require you to use the mouse to click a menu item or icon, or use the Ctrl-s hotkey combination, to save a file.

In Vim, you can save a file without your hands leaving the keyboard, and sometimes without even leaving the home keys. From Vim's *insert* mode, hit Escape and then :w. That's all. More on that later.

If that all sounds complicated, it's not--but it can take a little getting used to, like driving a manual transmission instead of an automatic. So, why would you want to learn Vim? Even though Vim is my favorite editor, I'll be very blunt: you might *not* want to. If you're never going to do any system administration or heavy editing of text, and if you don't want to invest any time in learning the capabilities that Vim has, then learning Vim *might* not be the best use of your

time.

On the other hand, if do any of the following, you probably want to look into Vim:

- System administration
- Programming
- Working with HTML, LaTeX, or other markup languages
- Heavy editing of plain text files

Even though Vim isn't as easy to use *initially* as standard GUI text editors like Gedit or word processors like OpenOffice.org's Writer, over a longer term you can become more productive using Vim. If you're a touch-typist, you'll find that your speed will improve even more with Vim because your hands rarely need to leave the "home" keys--and you'll only need to use the mouse if you choose to do so.

System administrators need to know at least a little Vim/vi to get by simply because it's the editor most likely to be on any given *nix system that you need to work on. Vi is the *lingua franca* of system administration.

If you're a programmer or working with structured markup languages like HTML, LaTeX, DocBook, etc., Vim is the bee's knees. It offers a number of features that I'll cover in later installments that make working with programming and markup languages much easier.

You might be skeptical. I was too, 10 years ago when my boss insisted I learn Vim to work on the company's Web site. The first week was painful. The month after that was okay. Within two months, I'd have sooner typed with my feet than to switch away from Vim.

The Modes

Some people disagree on how many modes Vim actually has. I'm going to define three: insert mode, command mode, and last-line mode. Let's start with the

default mode you'll see when you start up Vim--command mode.

When you run vim *filename* to edit a file, Vim starts out in command mode. This means that all the alphanumeric keys are bound to commands, rather than inserting those characters. Typing j won't insert the character "j"--it will move the cursor down one line. Typing dd will delete an entire line, rather than inserting "dd."

To enter the *insert* mode, type i (for "insert") and now the keys will behave as you'd expect. You can type normally until you want to make a correction, save the file, or perform another operation that's reserved for command mode or last-line mode. To get out of insert mode, hit the Escape key.

Once you press Escape, you're in command mode again. What if you'd like to save your file or search through your document? No problem, press : and Vim will switch to *last-line* mode. Vim is now waiting for you to enter a command like :w to write the file or :q to exit the editor.

If that all sounds complicated, it's really not. It does take a few days to start training your brain to move between the modes and memorizing the most important keys for movement, commands, and so on. But once you start getting the hang of it, you'll be surprised by how fluid it is editing a file in Vim. Let's walk through some of the most common commands that you need to know.

The Basics of Moving in Vim

The first thing you'll want to learn is how to move around a file. When you're in command mode, you'll want to remember the following keys and what they do:

- h moves the cursor one character to the left.
- j moves the cursor down one line.
- k moves the cursor up one line.
- l moves the cursor one character to the right.

- 0 moves the cursor to the beginning of the line.
- $ moves the cursor to the end of the line.
- w move forward one word.
- b move backward one word.
- G move to the end of the file.
- gg move to the beginning of the file.
- `. move to the last edit.

Here's a handy tip: prefacing a movement command with a number will execute that movement multiple times. So, if you want to move up six lines, enter 6k and Vim will move the cursor up six lines. If you want to move over five words, enter 5w. To move 10 words back, use 10b.

Keep that tip in mind--you'll find that prefacing other commands with a number can come in handy as well.

The best way to learn is practice. Take a few minutes to try Vim out. If you're on a Linux system right now, open up a terminal and type vim filename. Enter insert mode and type a bit (or copy some of the text from this article into Vim) and then hit Escape to start practicing movement around the file. Once you feel you're getting the hang of it, it's time to try some editing.

Editing Vim Style

Now that you know how to move around a bit, let's try editing. Move the cursor to the beginning of a word. Now type x. What happened? You should have deleted the character that the cursor was on. Want to undo it? No problem. Type u (for undo) and it will be restored.

Want to delete an entire word? Move your cursor to the beginning of a word again. Use dw. Note that this will only delete the word from the cursor on--so if you have the cursor in the middle of a word, it will only delete from that point on. Again, u will undo it. Note that Vim has multiple levels of undo, so you can undo

the change before that and the change before that, etc.

Want to undo your undo? Hit Ctrl-r. That will redo your last undo.

Again, here's a longer list of the commands you'll definitely want to know starting out:

- d starts the delete operation.
- dw will delete a word.
- d0 will delete to the beginning of a line.
- d$ will delete to the end of a line.
- dgg will delete to the beginning of the file.
- dG will delete to the end of the file.
- u will undo the last operation.
- Ctrl-r will redo the last undo.

You may have noticed that several commands combine a text operation and movement key. gg takes you to the end of a file, and d is used to delete. Combining them gives you something more powerful. Vim's like that. If you're working in Vim and think "hey, I wonder if I can combine two things I know to make something easier," the answer is often (but not always) yes.

Let's move on a bit and talk briefly about searching and replacing.

Searching and Replacing

Now that you know how to enter text, make some changes and so forth, it's time to learn how to use search and replace in Vim. It's really pretty easy. If you want to search through the document from command mode, use / followed by the text you want to search for. So, if I want to search for "bunny" I can enter / and then bunny and hit enter.

If I want to find it again, I hit n. If I want to look for a previous instance of the text, I'll use N instead, which will search the opposite direction through the

document.

Want to reverse the direction of your search? Use ? instead of / and Vim will move backwards through the document. Using n and N as above will reverse the direction of the search.

- /*text* search for *text* in the document, going forward.
- n move the cursor to the next instance of the text from the last search. This will wrap to the beginning of the document.
- N move the cursor to the previous instance of the text from the last search.
- ?*text* search for *text* in the document, going backwards.
- :%s/*text*/*replacement text*/g search through the entire document for *text* and replace it with *replacement text*.
- :%s/*text*/*replacement text*/gc search through the entire document and *confirm* before replacing text.

That's all pretty easy, isn't it? Now to move on to an important operation: Cutting and pasting text.

Copying and Pasting

You've already learned to delete text. The last text that you've deleted is stored in the buffer ready to be pasted back into the document. So if you've run dd and deleted an entire line, you can now hit p or P to paste it back into the document. This goes for single lines, multiple lines, and even entire documents.

Want to select text? In command mode, hit V and you'll be able to move the cursor using the arrow keys or the standard movement keys (h, k, j, l) to highlight text. This is pretty easy, but can be slow. Want to copy entire lines at a time? Use V instead of v and you'll highlight entire lines at a time. Again, you can use the movement keys to highlight additional lines.

Vim has a really cool trick as well. You can highlight in columns. Use Ctrl-v and you'll be able to highlight a column instead of an entire line. This can be useful

when working with some text files that have data in columns and you want to select an entire column, but not an entire line.

When you've highlighted what you want, hit y and it will "yank" the text into the buffer to be pasted later. So a usual paste operation might look like this:

Hit v to highlight some text. Then hit y to yank it into the buffer. Then move the cursor where you want it, and use p in command mode. There you go--you've just pasted some text!

The commands you most need to start out:

- v highlight one character at a time.
- V highlight one line at a time.
- Ctrl-v highlight by columns.
- p paste text after the current line.
- P paste text on the current line.
- y yank text into the copy buffer.

You've done enough editing for one day and you're ready to pack it in. No problem. Here's how you can save the file and quit Vim.

Saving and Quitting

If you're in insert mode, hit Escape. Then enter : and you'll see a line at the bottom of the screen with a cursor ready to take input.

To write the file you're editing, enter w. (So, you'll have :w.) That will write the file to the existing filename. If you don't have a filename or want to write out to a different filename, use :w *filename*.

To quit Vim after you've finished, hit :q. Since Vim is your friend, it won't just pop out on you if you haven't saved your file. It will say "no write since last change," and suggest that you add ! to override.

If you really want to quit, go ahead and use :q! to leave without being nagged.You can also exit Vim using ZZ, which will save and quit the file. Again, all this might sound a bit complex, but it really isn't. It's a bunch of smaller things to learn that when you add them all up, make for a very powerful package.

For now that should be enough to get you started. Don't fret, though. We'll go through more Vim usage next week, and keep working on Vim until you're an expert.

Joe 'Zonker' Brockmeier (http://www.dissociatedpress.net/about/) is a longtime FOSS advocate, and currently works for Novell (http://www.novell.com/) as the community manager (http://zonker.opensuse.org/) for openSUSE (http://www.opensuse.org/). Prior to joining Novell, Brockmeier worked as a technology journalist covering the open source beat for a number of publications, including Linux Magazine, Linux Weekly News, Linux.com, UnixReview.com, IBM developerWorks, and many others.