Vim tips: The basics of search and replace

Wednesday, 28 June 2006 08:00 Joe 'Zonker' Brockmeier

- 
- 

Let's start by looking at searches and doing search and replace operations within Vim. You can do a search in normal mode by using /*searchstring* . This will search forward through the file for *searchstring* . Likewise, running ?*searchstring* will search backwards through the file.

After running a search once, you can repeat it by using n in command mode, or N to reverse direction.

When you want to search for a string of text and replace it with another string of text, you can use the syntax :[range]s/*search*/*replace*/. The range is optional; if you just run :s/*search*/*replace*/, it will search only the current line and match only the first occurrence of a term.

Most of the time, that's not sufficient, so you can add a range like so:

:8,10 s/*search*/*replace*/g

In that example the range is from line 8 to line 10. I've also added the "global" option, which tells Vim to replace every occurrence on a line, and not just the first occurrence. Without adding g, your search will match only the first instance of a string in any given line.

Another way to specify the range is to enter visual mode and select the lines that you want to search, and then press : to enter command mode. To enter visual mode from normal mode, press v to select regular visual mode, or V for line selection, or Ctrl-v for block selection. Then select the range in visual mode and press :, followed by the search command you wish to use.

If you want to search an entire file, you can use % to indicate that as the range:

:%s/*search*/*replace*/g

You may also wish to be asked for confirmation before Vim makes a substitution. To do this, add the confirm (c) option to the end of the search and replace command: :%s/*search*/*replace*/gc. When you run this search, Vim will give you a prompt that looks something like this:

replace with foo (y/n/a/q/l/^E/^Y)?

The "y" and "n" are self-explanatory, but what about the rest? To tell Vim to go ahead and replace all instances of the matched string, answer with a. If you realize that you don't really want to make the changes, you can tell Vim to quit the operation using q. To tell Vim to make the current change and then stop, use l, for last.

^E and ^Y allow you to scroll the text using Ctrl-e and Ctrl-y.

Where you land

Searches in Vim put the cursor on the first character of the matched string by default; if you search for Debian, it would put the cursor on the D. That's usually fine, but Vim has a few options for offsetting the cursor placement from the beginning of the matched string.

To land on the last character in the matched string, rather than the beginning, add an /e to your search:

/Debian/e

That will place the cursor on the n rather than the D. Vim also allows you to specify a cursor offset by line, or from the beginning or end of the string. To have the cursor land two lines above a matched string, for example, use /*string*/-2. To place the cursor two lines below the string, use /*string*/+2.

To offset from the beginning of the string, add a /b or /s with the offset that you want. For example, to move three characters from the beginning of the search, you'd use /*string*/s+3 or /*string*/b+3 -- "s" for "start" or "b" for "begin." To count from the end of the string, use /e instead, so /*string*/e-3 will place the cursor on the third character from the last character of the matched string.

Next: Vim's special characters

Now that we've got basic search syntax covered, it's time to look at some of Vim's special characters to make searches a little more efficient. Searching for a literal string is out of the question if you're trying to match every URL in a file, or every comment in a bash or Perl script.

The first character you want to get to know is the humble dot. In a search, a dot or period (.), will match any single character. Do a quick search using /. and you'll see that it matches, literally, everything -- letters, numbers, whitespace, the whole kit and kaboodle.

What if you want to match a term at the beginning or end of a line? Vim uses the caret to match the beginning of a line (^) and the dollar sign to match the end of a line ($). For instance, to find any line in a bash script that begins with a comment, you could use ^#. To find empty lines, just use ^$ which will match any line without any characters.

To match whitespace, use the \\s operator. If you wanted to find empty lines that contain nothing but whitespace, you could use ^\\s.*$. This will match lines with whitespace, but no other characters -- it won't match empty lines without white space. By contrast, the \\S operator will match non-whitespace characters.

The \\d operator will match any digit in a search, while the \\D operator will match any non-digit in a search. To match any uppercase character, use \\u, while \\l will match any lowercase character. Using \\U will match any non-uppercase character, and \\L will match any non-lowercase character.

There's a subtle difference between matching any lowercase character and matching any non-uppercase character, and matching any uppercase character and matching any non-lowercase character. If you use \\u it will only match an uppercase letter -- but if you use \\L it will match any uppercase letter *and* any other character that's not a lowercase character; so it would match whitespace, digits, punctuation, and so on.

What happens when you actually want to search for a special character, such as a dollar sign or caret? Special characters can be escaped with the backslash character, so use \\$ to search for a dollar sign in your file, or \\^ to search for the caret, and so forth. Note that the backslash character is itself a special character, so you'd need to use two backslashes together.

You also may wish to delineate search terms by word boundaries. Say you do a search for *sig* using Vim. This will match sig, signature, signing, and a number of other strings that you do not wish to match. It might seem like a good idea to include spaces on each side of the search term, like so:

s/ *term* / *replace* /gc

That will do a better job of matching the search string only, but it will miss the string if it's at the end of a sentence, or separated by a comma. To make sure you're getting the right match, use \\ to match the end of a word. So, the search phrase would be better written as:

s/\\<*term*\\>/*replace*/gc

Quantity counts

If you're lucky, your parents taught you at an early age not to be greedy. That's a good rule in life, and usually a good rule in searching and replacing using Vim as well.

Vim allows you to specify how many of something you wish to match by adding quantifiers to the term. Some of the quantifiers are considered "greedy" because they match as many of the specified characters as possible. Others are non-greedy because they match a specific number or as few as possible.

As an example, the * quantifier tells Vim to match 0 or more of a character. So, a search like /abc* will match abc, abby, and absolutely.

To match one or more, use the \\+ quantifier. A search for /abc\\+ will match abc, but not abby or absolutely. For zero or one, use \\=, which would match abc, abby, and absolutely.

Vim can be even more precise, and will allow you to specify an exact number or range. The syntax for this is \\{0,10}, where the search would match 0 to 10 instances of the character. For example, to match a string with at least five uppercase characters, but no more than seven, you could use /\\; this breaks down as "match the beginning of a word, then five to seven uppercase characters, then the end of a word."

If you want to match a number of characters *exactly*, use \\{n} where n is the number. Want to find all of the three-letter words in a file? Use /\\ and you'll find every three-letter string in the file. Of course, that might match non-word strings, so you could use /\\ instead. The \\w tells Vim to match "word characters," so it won't match digits, punctuation, and suchlike.

You can also narrow it down to "at most" or "at least" a certain number of characters. As you've already learned, the syntax for matching a minimum to maximum number of occurrences is \\{x,y}, with x being the minimum and y being the maximum. So, just drop the minimum or maximum number, and keep the comma. To see that in action, run /\\<\\w\\{5,} . That will match words at least five characters long. To match words no longer than five letters, use /\\<\\w\\{,5} .

Finally, you can use \\{-} to tell Vim to match as little as possible. Let's say you're trying to match HTML tags in a file. Searching for matches too much. If you have a paragraph enclosed in

tags, Vim would match the whole paragraph rather than individual tags. To match individual tags, use . This would tell Vim to match character and then stop.

Let's be insensitive

As you're no doubt aware, searches in Vim are case-sensitive. GNOME, Gnome, and gnome are completely different, as far as Vim is concerned. But what if you want to search for all three at the same time, without resorting to serious regex-foo? Simple -- tell Vim to be case-insensitive using set ignorecase. Once ignorecase is set, a search for gnome will match GNOME, Gnome, and gnome.

If, after a while, you decide you want the case-sensitivity, toggle it back on using :set noignorecase.

If you don't want to toggle case-sensitivity on and off all the time, you can just use the \\c and \\C modifiers. The \\c modifier tells Vim to be case-insensitive, and \\C tells Vim to be case-sensitive.

For example, to search backwards through the text, ensuring that the search is case-sensitive, use ?\\C*pattern* , so ?\\CGNOME will only match GNOME, not gnome or Gnome. Of course, this works for forward searches as well, so you could use /\\CGNOME instead.

Another trick is to use the smartcase option. Use :set ignorecase smartcase, and if your search term has at least one capital letter, Vim will switch to case-sensitive; otherwise it will use case-insensitive search.

Finally, if you want to preserve case-sensitivity but search for a word that may have a capital letter or lower-case letter in one position, you could use a range instead. To match SuSE or SUSE, you'd use /S[Uu]SE.

Ranges are pretty useful in their own right. To match a through j, for example, you can use [a-j], or [^a-j] to match any character that's not a through j. This works with capital letters and digits too; [A-Q] would match any capital letter from A to Q, and [1-5] would match any digit from 1 to 5.

Matching one or more terms

Let's say you want to replace *Kang* or *Kodos* with the more generic term *alien*. Instead of running two searches, you can use branches, which are separated by a backslash and pipe character.

:%s/Kang\\|Kodos/alien/gc

You're not limited to two terms, either. If you want to replace Larry, Moe, and Curly with Stooge, you could use :%s/Larry\\|Moe\\|Curly/Stooge/gc.

Joe 'Zonker' Brockmeier

GURU

Joe 'Zonker' Brockmeier is a freelance writer and editor with more than 10 years covering IT. Formerly the openSUSE Community Manager for Novell, Brockmeier has written for Linux Magazine, Sys Admin, Linux Pro Magazine, IBM developerWorks, Linux.com, CIO.com, Linux Weekly News, ZDNet, and many other publications. Brockmeier is also a FLOSS advocate and participates in several projects, including GNOME as the PR team lead. You can reach Zonker at jzb@zonker.netThis e-mail address is being protected from spambots. You need JavaScript enabled to view it and follow him on Twitter.