Extracted from:

Practical Vim, Second Edition

Edit Text at the Speed of Thought

The Pragmatic Bookshelf

Raleigh, North Carolina

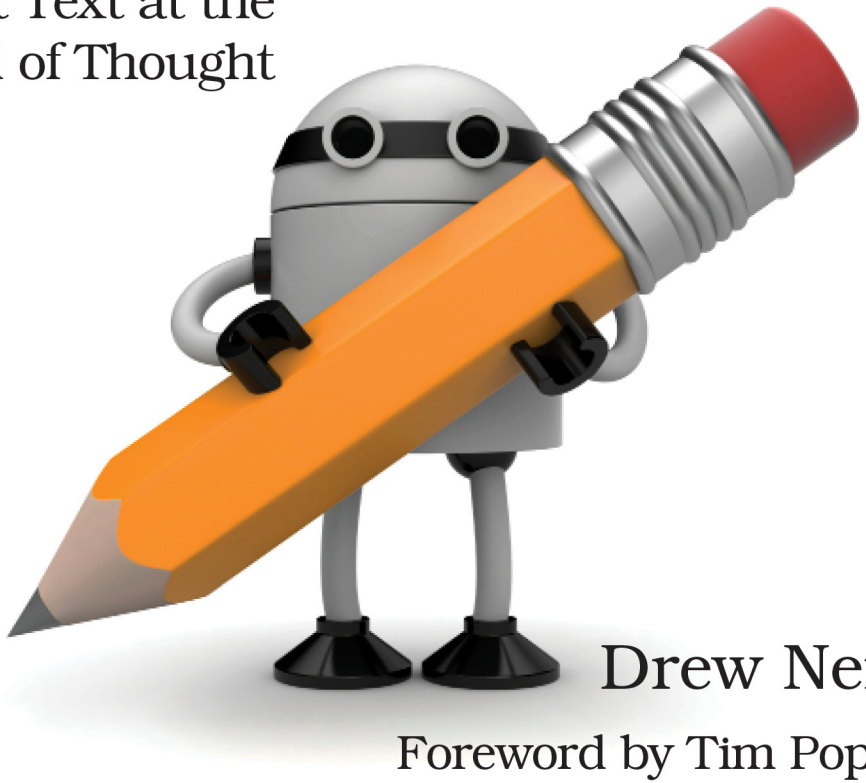
The Pragmatic Programmers

FOR Vim 8

Practical Vim

Second Edition

Edit Text at the Speed of Thought



Drew Neil

Foreword by Tim Pope

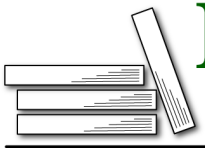
Practical Vim, Second Edition

Edit Text at the Speed of Thought

Drew Neil

The Pragmatic Bookshelf

Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Katharine Dvorak (editor)
Potomac Indexing, LLC (index)
Cathleen Small (copyedit)
Dave Thomas (layout)
Janet Furlow (producer)

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-127-8
Encoded using the finest acid-free high-entropy binary digits.
Book version: P2.0—January 2017

Meet the Substitute Command

The :substitute command is complex: in addition to providing a search pattern and replacement string, we have to specify the range over which it will execute. Optionally, we can also provide flags to tweak its behavior.

The substitute command allows us to find and replace one chunk of text with another. The command's syntax looks like this:

```
:[range]s[ubstitute]/{pattern}/{string}/[flags]
```

The substitute command has many parts to it. The rules for the [range] are just the same as for every other Ex command, which we covered in-depth in Tip 28,*Execute a Command on One or More Consecutive Lines*, on page ?. As for the {pattern}, that was covered in Chapter 12, *Matching Patterns and Literals*, on page ?.

Tweak the Substitute Command Using Flags

We can tweak the behavior of the substitute command using flags. The best way to understand what a flag does is to see it in action, so let's briefly define a handful of flags that are used in other tips. (For a complete reference, look up :h :s_flags.)

The g flag makes the substitute command act *globally*, causing it to change all matches within a line rather than just changing the first one. We'll meet it in Tip 89,*Find and Replace Every Match in a File*, on page 7.

The c flag gives us the opportunity to *confirm* or reject each change. We'll see it in action in Tip 90,*Eyeball Each Substitution*, on page 8.

The n flag suppresses the usual substitute behavior, causing the command to report the *number* of occurrences that would be affected if we ran the substitute command. Tip 86,*Count the Matches for the Current Pattern*, on page ?, gives an example of usage.

If we run the substitute command using a pattern that has no matches in the current file, Vim will report an error saying "E486: Pattern not found." We can silence these errors by including the e flag.

The & flag simply tells Vim to reuse the same flags from the previous substitute command. Tip 93,*Repeat the Previous Substitute Command*, on page ?, shows a scenario where it comes in handy.

Special Characters for the Replacement String

In Chapter 12, *Matching Patterns and Literals*, on page ?, we saw that some characters take on special meaning when used in search patterns. The replacement field also has a handful of special characters. You can find the complete list by looking up :h sub-replace-special, but some of the highlights are summarized in this table:

Symbol	Represents
\r	Insert a carriage return
\t	Insert a tab character
\\	Insert a single backslash
\1	Insert the first submatch
\2	Insert the second submatch (and so on, up to \9)
\0	Insert the entire matched pattern
&	Insert the entire matched pattern
~	Use {string} from the previous invocation of :substitute
\={Vim script}	Evaluate {Vim script} expression; use result as replacement {string}

The \r, \t and \\ tokens should be fairly self-explanatory. In Tip 93,*Repeat the Previous Substitute Command*, on page ?, we'll see how the ~ token works, but we'll also learn about a couple of shortcuts that make it even faster to repeat a substitute command. We'll see \1 and \2 in use in Tip 94,*Rearrange CSV Fields Using Submatches*, on page ?.

The \={Vim script} token is very powerful. It allows us to execute code and use the result as our replacement {string}. In Tip 95,*Perform Arithmetic on the Replacement*, on page ?, and Tip 96,*Swap Two or More Words*, on page ?, we'll see a couple of examples of usage.

Tip 89

Find and Replace Every Match in a File

By default, the substitute command acts on the current line, changing the first occurrence only. To change every match throughout a file, we have to specify a range and use the g flag.

For demonstration purposes, we'll use this text:

substitution/get-rolling.txt

```
When the going gets tough, the tough get going.
If you are going through hell, keep going.
```

Let's try and replace every occurrence of the word *going* with *rolling*. First, we'll enable the 'hlsearch' option so that we can see what we're doing (see Tip 81, *Highlight Search Matches*, on page ?, for more details):

```
⇛ :set hlsearch
```

The simplest thing that we could do with the substitute command is to provide a target {pattern} and replacement {string}:

Keystrokes	Buffer Contents
:s/going/rolling	When the rolling gets tough, the tough get going. If you are going through hell, keep going.

See what's happened? Vim has replaced the first occurrence of "going" with "rolling," but it's left every other match untouched.

To understand why, it helps to think of a file as a two-dimensional board made up of characters along the *x*-axis and lines down the *y*-axis. By default, the substitute command only acts upon the first match on the current line. Let's see what's required to expand its scope to cover the *x*- and *y*-axes in their entirety.

To keep going on the horizontal axis we need to include the g flag. This stands for *global*, which is a rather misleading name. One might expect that this flag would cause the substitution to be carried out across the entire file, but in fact, it just means "globally within the current line." This makes more sense if you remember that Vim is a direct descendent of the line editor ed, as discussed in *On the Etymology of Vim (and Family)*, on page ?.

We'll press u to undo the last change and then try running a variation of the substitute command. This time, we'll tack the /g flag on at the end:

Keystrokes	Buffer Contents
:s/going/rolling/g	When the rolling gets tough, the tough get rolling. If you are going through hell, keep going.

This time, all occurrences of *going* on the current line have been changed to *rolling*, but that still leaves a couple of instances unchanged elsewhere in the file. So how do we tell the substitute command to act on the entire vertical axis of our file?

The answer is to provide a range. If we prefix a % at the start of the substitute command, it will be executed on every line of the file:

Keystrokes	Buffer Contents
:%s/going/rolling/g	When the rolling gets tough, the tough get rolling. If you are rolling through hell, keep rolling.

The substitute command is just one of many Ex commands, all of which can accept a range in the same manner. Tip 28,*Execute a Command on One or More Consecutive Lines*, on page ?, goes into greater depth.

To recap, if we want to find and replace all occurrences in the current file, we have to explicitly tell the substitute command to operate on the entire *x*- and *y*-axes. The `g` flag deals with the horizontal axis, while the `%` address deals with the vertical axis.

It's easy enough to forget one or the other of these details. In Tip 93,*Repeat the Previous Substitute Command*, on page ?, we'll look at a couple of techniques for repeating a substitute command.

Tip 90

Eyeball Each Substitution

Finding all occurrences of a pattern and blindly replacing them with something else won't always work. Sometimes we need to look at each match and decide if it should be substituted. The `c` flag modifies the :substitute command to make this possible.

Remember this example from Tip 5,*Find and Replace by Hand*, on page ??

the_vim_way/1_copy_content.txt

```
...We're waiting for content before the site can go live...
...If you are content with this, let's go ahead with it...
...We'll launch as soon as we have the content...
```

We couldn't use find and replace to change "content" to "copy." Instead, we used the Dot Formula to solve our problem. However, we could also have used the `c` flag on the substitute command:

⇛ `:%s/content/copy/gc`

The `c` flag causes Vim to show us each match and ask "Replace with copy?" We can then say `y` to perform the change or `n` to skip it. Vim does what we ask and then moves to the next match and asks us again.

In our example, we would respond `yny`, changing the first and last occurrences while leaving the middle one untouched.

We aren't limited to just two answers. In fact, Vim helpfully reminds us of our options with the prompt "y/n/a/q/l/^E/^Y." This table shows what each answer means:

Trigger	Effect
`y`	Substitute this match
`n`	Skip this match
`q`	Quit substituting
`l`	"last"—Substitute this match, then quit
`a`	"all"—Substitute this and any remaining matches
`<C-e>`	Scroll the screen up
`<C-y>`	Scroll the screen down

You can also find this information in Vim's help by looking up `:h :s_c`.

Discussion

Unusually, most buttons on the keyboard do nothing in Vim's Substitute-Confirmation mode. As always, the `<Esc>` key allows us to return to Normal mode, but apart from that, the landscape feels unfamiliar.

On the up side, this allows us to complete the task with a minimum of keystrokes. On the down side, all of the functionality that we're used to is unavailable to us. By contrast, if we use the Dot Formula (as in Tip 5,*Find and Replace by Hand*, on page ?), then we're in plain old Normal mode throughout. Everything works just as we expect it to.

Try both methods, and use whichever one you feel more comfortable with.