Vi(m)

Alles zum Texteditor

- Vi(m) - search and replace

Vi(m) - search and replace

Quelle: https://vim.fandom.com/wiki/Search_and_replace

Intro

Vim provides the `:s` (substitute) command for search and replace; this tip shows examples of how to substitute. On some systems, gvim has *Find and Replace* on the Edit menu (:help :promptrepl), however it is easier to use the `:s` command due to its command line history and ability to insert text (for example, the word under the cursor) into the search or replace fields.

Basic search and replace

The `:substitute` command searches for a text pattern, and replaces it with a text string. There are many options, but these are what you probably want:

`:%s/foo/bar/g`
: Find each occurrence of 'foo' (in all lines), and replace it with 'bar'.

`:s/foo/bar/g`
: Find each occurrence of 'foo' (in the current line only), and replace it with 'bar'.

`:%s/foo/bar/gc`
: Change each 'foo' to 'bar', but ask for confirmation first.

`:%s/\<foo\>/bar/gc`
: Change only whole words exactly matching 'foo' to 'bar'; ask for confirmation.

`:%s/foo/bar/gci`
: Change each 'foo' (case insensitive due to the `i` flag) to 'bar'; ask for confirmation.
`:%s/foo\c/bar/gc` is the same because `\c` makes the search case insensitive.
This may be wanted after using `:set noignorecase` to make searches case sensitive (the default).

`:%s/foo/bar/gcI`
: Change each 'foo' (case sensitive due to the `I` flag) to 'bar'; ask for confirmation.
`:%s/foo\C/bar/gc` is the same because `\C` makes the search case sensitive.
This may be wanted after using `:set ignorecase` to make searches case insensitive.

The `g` flag means *global* – each occurrence in the line is changed, rather than just the first. This tip assumes the default setting for the `'gdefault'` and `'edcompatible'` option (off), which requires that the `g` flag be included in `%s///g` to perform a global substitute. Using `:set gdefault` creates confusion because then `%s///` is global, whereas `%s///g` is not (that is, `g` reverses its meaning).

When using the `c` flag, you need to confirm for each match what to do. Vim will output something like: `replace with foobar (y/n/a/q/l/^E/^Y)?` (where foobar is the replacement part of the `:s/.../.../` command. You can type `y` which means to substitute this match, `n` to skip this match, `a` to substitute this and all remaining matches ("all" remaining matches), `q` to quit the command, `l` to substitute this match and quit (think of "last"), `^E` to scroll the screen up by holding the Ctrl key and pressing E and `^Y` to scroll the screen down by holding the Ctrl key and pressing Y. However, the last two choices are only available, if your Vim is a normal, big or huge built or the insert_expand feature was enabled at compile time (look for `+insert_expand` in the output of `:version`).

Also when using the `c` flag, Vim will jump to the first match it finds starting from the top of the buffer and prompt you for confirmation to perform replacement on that match. Vim applies the `IncSearch` highlight group to the matched text to give you a visual cue as to which match it is operating on (set to `reverse` by default for all three term types as of Vim 7.3). Additionally, if more than one match is found and you have search highlighting enabled with `:set hlsearch`, Vim highlights the remaining matches with the `Search` highlight group. If you do use search highlighting, you should make sure that these two highlight groups are visually distinct or you won't be able to easily tell which match Vim is prompting you to substitute.

Details

Search range:

`:s/foo/bar/g`	Change each 'foo' to 'bar' in the current line.
`:%s/foo/bar/g`	Change each 'foo' to 'bar' in all the lines.
`:5,12s/foo/bar/g`	Change each 'foo' to 'bar' for all lines from line 5 to line 12 (inclusive).
`:'a,'bs/foo/bar/g`	Change each 'foo' to 'bar' for all lines from mark a to mark b inclusive (see **Note** below).
`:'<,'>s/foo/bar/g`	When compiled with `+visual`, change each 'foo' to 'bar' for all lines within a visual selection. Vim automatically appends the visual selection range ('<,'>) for any ex command when you select an area and enter `:`. Also, see **Note** below.
`:.,$s/foo/bar/g`	Change each 'foo' to 'bar' for all lines from the current line (.) to the last line ($) inclusive.
`:.,+2s/foo/bar/g`	Change each 'foo' to 'bar' for the current line (.) and the two next lines (+2).

`:g/^baz/s/foo/bar/g`	Change each 'foo' to 'bar' in each line starting with'baz'.

> **Note**: As of Vim 7.3, substitutions applied to a range defined by marks or a visual selection (which uses a special type of marks '< and '>) are not bounded by the column position of the marks by default. Instead, Vim applies the substitution to the entire line on which each mark appears unless the `\%V` atom is used in the pattern like: `:'<,'>s/\%Vfoo/bar/g`.

When searching:

> `.`, `*`, `\`, `[`, `^`, and `$` are metacharacters.
> `+`, `?`, `|`, `&`, `{`, `(`, and `)` must be escaped to use their special function.
> `\/` is / (use backslash + forward slash to search for forward slash)
> `\t` is tab, `\s` is whitespace (space or tab)
> `\n` is newline, `\r` is CR (carriage return = Ctrl-M = ^M)
>
> After an opening `[`, everything until the next closing `]` specifies a /collection. Character ranges can be represented with a `-`; for example a letter a, b, c, or the number 1 can be matched with `[1a-c]`. Negate the collection with `[^` instead of `[`; for example `[^1a-c]` matches any character except a, b, c, or 1.
> `\{#\}` is used for repetition. `/foo.\{2\}` will match foo and the two following characters. The `\` is not required on the closing `}` so `/foo.\{2}` will do the same thing.
> `\(foo\)` makes a backreference to foo. Parenthesis without escapes are literally matched. Here the `\` is required for the closing `\)`.

When replacing:

> `\r` is newline, `\n` is a null byte (0x00).
> `\&` is ampersand (& is the text that matches the search pattern).
> `\0` inserts the text matched by the entire pattern
> `\1` inserts the text of the first backreference. `\2` inserts the second backreference, and so on.

You can use **other delimiters** with substitute:

```
:s#http://www.example.com/index.html#http://example.com/#
```

Save typing by using `\zs` and `\ze` to set the **start and end of a pattern**. For example, instead of:

```
:s/Copyright 2007 All Rights Reserved/Copyright 2008 All Rights Reserved/
```

Use:

```
:s/Copyright \zs2007\ze All Rights Reserved/2008/
```

Using the current word or registers

`:%s//bar/g`

Replace each match of the last search pattern with 'bar'.
For example, you might first place the cursor on the word `foo` then press `*` to search for that word.
The above substitute would then change all words exactly matching 'foo' to 'bar'.

`:%s/foo/<c-r><c-w>/g`

Replace each occurrence of 'foo' with the word under the cursor.
`<c-r><c-w>` means that you press Ctrl-R then Ctrl-W.
The word under the cursor will be inserted as though you typed it.

`:%s/foo/<c-r><c-a>/g`

Replace each occurrence of 'foo' with the WORD under the cursor (delimited by whitespace).
`<c-r><c-a>` means that you press Ctrl-R then Ctrl-A.
The WORD under the cursor will be inserted as though you typed it.

`:%s/foo/<c-r>a/g`

Replace each occurrence of 'foo' with the contents of register 'a'.
`<c-r>a` means that you press Ctrl-R then `a`.
The contents of register 'a' will be inserted as though you typed it.

`:%s/foo/<c-r>0/g`

Same as above, using register 0 which contains the text from the most recent yank command. Examples of yank (copy) commands are `yi(` which copies the text inside parentheses around the cursor, and `y$` which copies the text from the cursor to the end of the line. After a yank command which did not specify a destination register, the copied text can be entered by pressing Ctrl-R then `0`.

`:%s/foo/\=@a/g`

Replace each occurrence of 'foo' with the contents of register 'a'.
`\=@a` is a reference to register 'a'.
The contents of register 'a' is not shown in the command. This is useful if the register contains many lines of text.

`:%s//<c-r>//g`

Replace each match of the last search pattern with the `/` register (the last search pattern).
After pressing Ctrl-R then `/` to insert the last search pattern (and before pressing Enter to perform the command), you could edit the text to make any required change.

`:%s/<c-r>*/bar/g`

Replace all occurrences of the text in the system clipboard (in the `*` register) with 'bar' (see next example if multiline).
On some systems, selecting text (in Vim or another application) is all that is required to place that text in the `*` register.

`:%s/<c-r>a/bar/g`

Replace all occurrences of the text in register 'a' with 'bar'.
`<c-r>a` means that you press Ctrl-R then `a`. The contents of register 'a' will be inserted as though you typed it.
Any newlines in register 'a' are inserted as `^M` and are not found.
The search works if each `^M` is manually replaced with '\n' (two characters: backslash, 'n').
This replacement can be performed while you type the command:

`:%s/<c-r>=substitute(@a,"\n",'\\n','g')<CR>/bar/g`

The `"\n"` (double quotes) represents the single character newline; the `'\\n'` (single quotes) represents two backslashes followed by '`n`'.
The `substitute()` function is evaluated by the `<c-r>=` (Ctrl-R `=`) expression register; it replaces each newline with a single backslash followed by '`n`'.
The `<CR>` indicates that you press Enter to finish the `=` expression.

`:%s/<c-r>0/bar/g`

Same as above, using register 0 which contains the text from the most recent yank command.

See Paste registers in search or colon commands instead of using the clipboard.

Additional examples

`:%s/foo/bar/`

On each line, replace the first occurrence of "foo" with "bar".

`:%s/.*\zsfoo/bar/`

On each line, replace the last occurrence of "foo" with "bar".

`:%s/\<foo\>//g`

On each line, delete all occurrences of the whole word "foo".

`:%s/\<foo\>.*//`

On each line, delete the whole word "foo" and all following text (to end of line).

`:%s/\<foo\>.\{5}//`

On each line, delete the first occurrence of the whole word "foo" and the following five characters.

`:%s/\<foo\>\zs.*//`

On each line, delete all text following the whole word "foo" (to end of line).

`:%s/.*\<foo\>//`

On each line, delete the whole word "foo" and all preceding text (from beginning of line).

`:%s/.*\ze\<foo\>//`

On each line, delete all the text preceding the whole word "foo" (from beginning of line).

`:%s/.*\(\<foo\>\).*/\1/`
: On each line, delete all the text preceding and following the whole word "foo".

`:%s/\<foo\(bar\)\@!/toto/g`
: On each line, replace each occurrence of "foo" (which starts a word and is not followed by "bar") by "toto".

`:s/^\(\w\)/\u\1/`
: If the first character at the beginning of the *current line only* is lowercase, switch it to uppercase using `\u` (see switching case of characters).

`:%s/\(.*\n\)\{5\}/&\r/`
: Insert a blank line every 5 lines.
The pattern searches for `\(.*\n\)` (any line including its line ending) repeated five times (`\{5\}`).
The replacement is `&` (the text that was found), followed by `\r` (newline).

`:%s/\<foo\(\a*\)\>/\=len(add(list, submatch(1)))?submatch(0):submatch(0)/g`
: Get a list of search results. (the list must exist)
Sets the `modified` flag, because of the replacement, but the content is unchanged.
Note: With a recent enough Vim (version 7.3.627 or higher), you can simplify this to:

`:%s/\<foo\(\a*\)\>/\=add(list, submatch(1))/gn`
: This has the advantage, that the buffer won't be marked modified and no extra undo state is created. The expression in the replacement part is executed in the sandbox and not allowed to modify the buffer.

Special cases

For substituting patterns with corresponding case-sensitive text, Michael Geddes's keepcase plugin can be used, e.g.:

`:%SubstituteCase/\cHello/goodBye/g`
: Substitute 'Hello hello helLo HELLO' by 'Goodbye goodbye goodBye GOODBYE'

For changing the offsets in a patch file (line number of a block), this little snippet can be used:

```
s/^@@ -\(\d\+\),\(\d\+\) +\(\d\+\),\(\d\+\) @@$/\="@@ -".eval(submatch(1)+offsetdiff
).",".submatch(2)." +".eval(submatch(3)+offsetdiff).",".submatch(4)." @@"/g
```

Useful when we want to strip some blocks from a patch, without patch having to complain about offset differences.

> **Note** Should try to make the expression more compact, but don't know how without having the possibility of modifying unwanted lines.