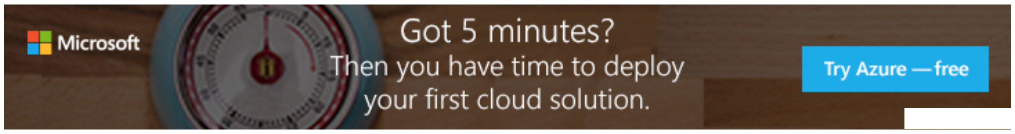
vim regex replace multiple consecutive spaces with only one space



I often work with text files which have a variable amount of whitespaces as word separators (text processors like Word do this, to distribute fairly the whitespace amount due to different sizes of letters in certain fonts and they put this annoying variable amount of spaces even when saving as plain text).

I would like to automate the process of replacing these sequences of whitespaces that have variable length with single spaces. I suspect a regex could do it, but there are also whitespaces at the beginning of paragraphs (usually four of them, but not always), which I would want to let unchanged, so basically my regex should also not touch the leading whitespaces and this adds to the complexity.

I'm using vim, so a regex in the vim regex dialect would be very useful to me, if this is doable.

My current progress looks like this:

```
:%s/ \+/ /g
```

but it doesn't work correctly.

I'm also considering to write a vim script that could parse text lines one by one, process each line char by char and skip the whitespaces after the first one, but I have a feeling this would be overkill.

regex vim

asked Oct 5 '10 at 2:48
jedi_coder
740 1 7 13

Good for reformatting vertically-aligned code :) – JackHasaKeyboard Aug 7 '16 at 22:06

7 Answers

In the interests of pragmatism, I tend to just do it as a three-stage process:

```
:g/^    /s//XYZZYPARA/g
:g/ \+/s// /g
:g/^XYZZYPARA/s//    /g
```

I don't doubt that there may be a better way (perhaps using macros or even a pure regex way) but I usually find this works when I'm in a hurry. Of course, if you have lines starting with `XYZZYPARA` , you may want to adjust the string :-)

It's good enough to turn:

```
    This is a new paragraph
spanning       two lines.
    And    so    is   this but on one line.
```

into:

```
    This is a new paragraph
spanning two lines.
    And so is this but on one line.
```

> *Aside:* If you're wondering why I use `:g` instead of `:s`, that's just habit mostly. `:g` can do everything `:s` can and so much more. It's actually a way to execute an *arbitrary* command on selected lines. The command to execute happens to be `s` in this case so there's no real difference but, if you want to become a `vi` power user, you should look into `:g` at some point.

edited Oct 5 '10 at 3:02 answered Oct 5 '10 at 2:56
paxdiablo
529k 129 1061 1508

2 Yeah, the purist/idealist in me started taking a back seat a long time ago. Now I just like to get the job done, especially if the alternative is a 600-character regex with back-tracking and look-ahead, that I won't understand when I have to come back and debug it in three months :-) – paxdiablo Oct 5 '10 at 3:12

+1 xyzzy plover – SingleNegationElimination Oct 5 '10 at 3:56

I used a variant on the above: :g/ \+/s// /g I understand the space and \+ to match one or more, no idea what the /s/ does, anyone know? – anteatersa Nov 28 '12 at 11:47

1 @anteatersa, the `s` is the substitute command itself. If you read the last part of my answer it explains the `g` simply *selects* the lines, then executes an arbitrary command on each of them, of which `s` is one possibility. For example `:g/^$/d` will run the `d` command (delete line) on all empty lines. You can have all sorts of fun such as with `:g/^/m0` :-) – paxdiablo Nov 28 '12 at 11:59

Did you find this question interesting? Try our newsletter

Sign up for our newsletter and get our top new questions delivered to your inbox (see an example).

this will replace 2 or more spaces

```
s/ \{2,}/ /g
```

or you could add an extra space before the `\+` to your version

```
s/  \+/ /g
```

answered Oct 5 '10 at 2:51
mikerobi
14k 3 29 36

4 I think this is probably the best and simplest answer. It also has the added benefit of working in other RegEx dialects too! – TrinitronX Jul 16 '12 at 16:27

This will do the trick:

```
%s![^ ]\zs  \+! !g
```

Many substitutions can be done in Vim easier than with other regex dialects by using the `\zs` and `\ze` meta-sequences. What they do is to exclude part of the match from the final result, either the part before the sequence (`\zs`, "s" for "start here") or the part after (`\ze`, "e" for "end here"). In this case, the pattern must match one non-space character first (`[^ ]`) but the following `\zs` says that the final match result (which is what will be replaced) starts *after* that character.

Since there is no way to have a non-space character in front of line-leading whitespace, it will be not be matched by the pattern, so the substitution will not replace it. Simple.

edited Oct 5 '10 at 16:39 answered Oct 5 '10 at 3:48
Aristotle Pagaltzis
72.9k 12 83 85

1 I would like to propose this alternative: `%s!\S\@<= \+! !g`. The `\@<=` is such a beautiful duck that I like using it. See also `:help /\@<=` – Benoit Oct 5 '10 at 16:36

1 I just prefer the reduced finger acrobatics of `zs` over typing `@<=` … in much the same way (if to a lesser

extent) that I enjoy Vim better than E(scape)M(eta)A(lt)C(ontrol)S(hift). :) OTOH, one's sense of flair is always worth some sacrifice, so feel free. – Aristotle Pagaltzis Oct 5 '10 at 16:48

depends on what keyboard layout you're using of course… – Benoit Oct 5 '10 at 18:22

There are lots of good answers here (especially Aristotle's: `\zs` and `\ze` are well worth learning). Just for completeness, you can also do this with a negative look-behind assertion:

```
:%s/\(^ *\)\@<! \{2,}/ /g
```

This says "find 2 or more spaces (`' \{2,}'`) that are NOT preceded by 'the start of the line followed by zero or more spaces'". If you prefer to reduce the number of backslashes, you can also do this:

```
:%s/\v(^ *)@<! {2,}/ /g
```

but it only saves you two characters! You could also use `' +'` instead of `' {2,}'` if you don't mind it doing a load of redundant changes (i.e. changing a single space to a single space).

You could also use the negative look-behind to just check for a single non-space character:

```
:%s/\S\@<!\s\+/ /g
```

which is much the same as (a slightly modified version of Aristotle's to treat spaces and tabs as the same in order to save a bit of typing):

```
:%s/\S\zs \+/ /g
```

See:

```
:help \zs
:help \ze
:help \@<!
:help zero-width
:help \v
```

and (read it all!):

```
:help pattern.txt
```

edited Mar 30 '14 at 16:12
Community ♦
1 1

answered Oct 5 '10 at 10:42
DrAl
46.6k 8 74 92

Does this work?

```
%s/\([^ ]\)  */\1 /g
```

answered Oct 5 '10 at 3:34
frogstarr78
358 1 9

better use `%s/[^ ]\zs \+/ /g` in this case (`:help /\zs`) – Benoit Oct 5 '10 at 16:32

Ah! Nice. I agree much better. Thank you. – frogstarr78 Oct 6 '10 at 23:23

I like this version - it is similar to the look ahead version of Aristotle Pagaltzis, but I find it easier to understand. (Probably just my unfamiliarity with \zs)

```
s/\([^ ]\) \+/\1 /g
```

or for all whitespace

```
s/\(\S\)\s\+/\1 /g
```

I read it as "replace all occurences of something other than a space followed by multiple spaces with the something and a single space".

answered Oct 5 '10 at 4:13
Michael Anderson
34.5k 4 75 128

Of course this version is an order of magnitude more finicky to type, and to formulate on the fly – and that's for almost as trivial a pattern as it gets. You'll be well served to familiarise yourself with `\zs` and `\ze` , they can do wonders for the writability and readability of more complex patterns (particularly when you have reason to use both at once!). – Aristotle Pagaltzis Oct 5 '10 at 4:43

I surely will look at `\zs` and `\ze` , but I also often use my regexs in python and sed. So it can be nice to have a solution that will work across multiple applications. – Michael Anderson Oct 5 '10 at 5:11

Answered; but though i'd toss my work flow in anyway.

```
%s/  / /g
@:@:@:@:@:@:@:@:@:@:@:@:(repeat till clean)
```

Fast and simple to remember. There are a far more elegant solutions above; but just my .02.

answered Oct 5 '10 at 12:50
wom
320 2 14

this is not a good solution: first it will remove leading whitespace, which the author of the question wishes to avoid. Second, you can do 100@: to run 100 times contents of register : (which is the last ex command) – Benoit Oct 5 '10 at 16:34

1 hence I said it's not the best answer in my reply :) – wom Oct 6 '10 at 18:18

1 I still find this answer useful, even though it does not answer OP's questions well. – xeon Feb 9 '15 at 23:44