UNIX & LINUX

What characters do I need to escape when using sed in a sh script?

Asked 12 years, 1 month ago Modified 4 years, 1 month ago Viewed 759k times

411

Take the following script:

```
#!/bin/sh
sed 's/(127\.0\.1\.1)\s/\1/' [some file]
```

If I try to run this in `sh` (`dash` here), it'll fail because of the parentheses, which need to be escaped. But I *don't* need to escape the backslashes themselves (between the octets, or in the `\s` or `\1`). What's the rule here? What about when I need to use `{...}` or `[...]`? Is there a list of what I do and don't need to escape?

shell-script sed quoting

Share Edit Follow Flag

edited Feb 28, 2012 at 23:55
Gilles 'SO- stop being evil'
829k 195 1.7k 2.2k

asked Feb 28, 2012 at 4:42
detly
5,160 6 25 29

2 Here is a bash function for converting paths for use with SED: `function sedPath { path=$((echo $1|sed -r 's/([\$\.\*\/\[\\^])/\\\1/g'|sed 's/[]]/\[]]/g')>&1) } #Escape path for use with sed` – user2428118 May 10, 2016 at 12:57

See also: Which characters need to be escaped in Bash? How do we know it? – codeforester Feb 28, 2018 at 7:02

2 Dura lex, sed sed – Nemo May 11, 2018 at 19:56

4 Answers

Sorted by: Highest score (default)

466

There are two levels of interpretation here: the shell, and sed.

In the shell, everything between single quotes is interpreted literally, except for single quotes themselves. You can effectively have a single quote between single quotes by writing `'\''` (close single quote, one literal single quote, open single quote).

Sed uses basic regular expressions. In a BRE, in order to have them treated literally, the characters `$.*[\^` need to be quoted by preceding them by a backslash, except inside character sets (`[…]`). Letters, digits and `(){}+?|` must not be quoted (you can get away with quoting some of these in some implementations). The sequences `\(`, `\)`, `\n`, and in some implementations `\{`, `\}`, `\+`, `\?`, `\|` and other backslash+alphanumerics have special meanings. You can get away with not quoting `$^` in some positions in some implementations.

Furthermore, you need a backslash before `/` if it is to appear in the regex outside of bracket expressions. You can choose an alternative character as the delimiter by writing, e.g., `s~/dir~/replacement~` or `\~/dir~p`; you'll need a backslash before the delimiter if you want to include it in the BRE. If you choose a character that has a special meaning in a BRE and you want to include it literally, you'll need three backslashes; I do not recommend this, as it may behave differently in some implementations.

In a nutshell, for `sed 's/…/…/'`:

- Write the regex between single quotes.
- Use `'\''` to end up with a single quote in the regex.
- Put a backslash before `$.*/[\]^` and only those characters (but not inside bracket expressions). (Technically you shouldn't put a backslash before `]` but I don't know of an implementation that treats `]` and `\]` differently outside of bracket expressions.)
- Inside a bracket expression, for `-` to be treated literally, make sure it is first or last (`[abc-]` or `[-abc]`, not ~~`[a-bc]`~~).
- Inside a bracket expression, for `^` to be treated literally, make sure it is *not* first (use `[abc^]`, not ~~`[^abc]`~~).
- To include `]` in the list of characters matched by a bracket expression, make it the first character (or first after `^` for a negated set): `[]abc]` or `[^]abc]` (not ~~`[abc]]` nor `[abc\]]`~~).

In the replacement text:

- `&` and `\` need to be quoted by preceding them by a backslash, as do the delimiter (usually `/`) and newlines.
- `\` followed by a digit has a special meaning. `\` followed by a letter has a special meaning (special characters) in some implementations, and `\` followed by some other character means `\c` or `c` depending on the implementation.
- With single quotes around the argument (`sed 's/…/…/'`), use `'\''` to put a single quote in the replacement text.

If the regex or replacement text comes from a shell variable, remember that

- The regex is a BRE, not a literal string.
- In the regex, a newline needs to be expressed as `\n` (which will never match unless you have other `sed` code adding newline characters to the pattern space). But note that it

won't work inside bracket expressions with some `sed` implementations.

- In the replacement text, `&`, `\` and newlines need to be quoted.
- The delimiter needs to be quoted (but not inside bracket expressions).
- Use double quotes for interpolation: `sed -e "s/$BRE/$REPL/"`.

Share Edit Follow Flag

edited Dec 26, 2018 at 10:41

answered Feb 29, 2012 at 1:06
Gilles 'SO- stop being evil'
829k 195 1.7k 2.2k

1 Escaping the actual wildcard character (*) you can use double backslash (`\\*`). Example: `echo "***NEW***" | sed /\\*\\*\\*NEW\\*\\*\\*/s/^/#/` – Melroy van den Berg Mar 20, 2019 at 16:44

1 "Use '\'' to end up with a single quote in the regex." didn’t work for me on macOS Catalina. I had to switch to using double quotes and putting the single-quotes inside. Tried everything between 0-2 backslashes. – Florian Wendelborn Dec 13, 2019 at 19:43

3 I had to escape `+` as well, for it to have the regexy meaning. – hoijui Apr 3, 2021 at 8:43

@danger89: You discovered the difference between single quotes and double quotes (or no quotes), for example: `grep -rl access\\.log .` = `grep -rl "access\\.log" .` = `grep -rl 'access\.log' .`. @hoijui: Same here with Ubuntu 18.04.5. – uav Apr 8, 2021 at 17:04

1 wow could it be any more easier than that! – ononononon May 3, 2022 at 12:15

|

63

The problem you're experiencing isn't due to shell interpolating and escapes - it's because you're attempting to use extended regular expression syntax without passing sed the `-r` or `--regexp-extended` option.

Change your sed line from

```
sed 's/(127\.0\.1\.1)\s/\1/' [some file]
```

to

```
sed -r 's/(127\.0\.1\.1)\s/\1/' [some file]
```

and it will work as I believe you intend.

By default sed uses uses basic regular expressions (think grep style), which would require the following syntax:

```
sed 's/\(127\.0\.1\.1\)[ \t]/\1/' [some file]
```

Share Edit Follow Flag

answered Feb 29, 2012 at 2:56
R Perrin
3,049 20 11

I had this problem again, and forgot to scroll down to find the solution I upvoted last time. Thanks again. – isaaclw Apr 4, 2014 at 20:17

1 Thanks a lot. Adding `-r` as an option was what was necessary in my case. – HelloGoodbye May 21, 2015 at 8:23

1 Note that `-r` is not available on MacOS – Hubert Grzeskowiak Jan 31, 2020 at 0:10

2 @HubertGrzeskowiak Using `-E` on MacOS did the same trick as `-r` see – Felix Mar 30, 2020 at 8:14

1 `-r` is available on MacOs at least on 10.14.6 (Mojave) – Juan Antonio Jan 29, 2021 at 9:22

24

Unless you want to interpolate a shell variable into the sed expression, use single quotes for the whole expression because they cause everything between them to be interpreted as-is, including backslashes.

So if you want sed to see `s/\(127\.0\.1\.1\)\s/\1/` put single quotes around it and the shell won't touch the parentheses or backslashes in it. If you need to interpolate a shell variable, put only that part in double quotes. E.g.

```
sed 's/\(127\.0\.1\.1\)/'"$ip"'/'
```

This will save you the trouble of remembering which shell metacharacters are not escaped by double quotes.

Share Edit Follow Flag

answered Feb 28, 2012 at 5:58
Kyle Jones
15k 3 42 52

I want `sed` to see `s/(127\.0\.1\.1)/...`, but putting that in a shell script as-is doesn't work. What you're saying about the shell not touching the parentheses seems wrong. I've edited my question to elaborate. – detly Feb 28, 2012 at 6:14

4 The shell isn't touching the parentheses. You need the backslases because **sed** needs to see them. `sed 's/(127\.0\.1\.1)/IP \1/'` fails because sed needs to see `\(` and `\)` for group syntax, not `(` and `)`. – Kyle Jones Feb 28, 2012 at 6:31

facepalm It's not in the man page, but it IS in some online manual I found. Is this normal for regex, because I've never had to use it in regex libraries (in, eg. Python)? – detly Feb 28, 2012 at 6:33

3 For traditional Unix commands, there are basic regular expressions and extended regular expressions. Details. sed uses basic regular expressions, so the backslashes are needed for group syntax. Perl and Python went beyond even extended regular expressions. While I was poking

around I found an extremely informative chart that illustrates what a confusing bramble we conjure up when we glibly say "regular expression." – Kyle Jones Feb 28, 2012 at 7:07

1 I would also add that the only character that cannot be used inside single quotes is a single quote. – enzotib Feb 28, 2012 at 9:08

|

4

I think it's worth mentioning that, while sed is based on the the POSIX standard, which specifies support only for basic regular expression (BRE), two different versions of the sed command actually exist - BSD(Mac OS) and GNU(Linux distros). Each version implements similar, as well as unique extensions to the POSIX standard, and can affect the functionality of sed across different platforms. As a result, proper syntax of the sed command, functioning as expected on one system, might actually translate to completely different results on another. This can lead to unexpected behavior with regards to the usage of escaped and special characters.

These extensions to the POSIX standard tend to be more prevalent on the GNU version of sed, often times providing the convenience of less strict formatting, especially in comparison to the BSD version. However, while GNU sed does allow for the functionality of some special characters, they are still not actually POSIX-compliant. Additionally, the only real difference between basic and extended regular expression(ERE), within GNU sed, is the behavior of the following special characters:

'?', '+', parentheses, braces ('{}'), and '|'

While this may be the case, some special characters have limited or no support at all on BSD sed, such as '|', '?', and '+', as it more closely adheres to the POSIX syntax standards. The inclusion of those characters, in a fashion similar to that of GNU sed, will often result in issues with portability and functionality of scripts utilizing sed. It's also worth noting, POSIX BRE syntax does not define a meaning for some escape sequences, most notably: \|, +, \?, `, \', \<, >, \b, \B, \w, and \W,.

For those running the BSD/Mac OS version of sed, emulating behavior of some special characters can be a bit tricky, but it can be done in most cases. For example, + could be emulated in a POSIX-compliant fashion like this: {1,} and \? would look like this: {0,1} Control character sequences, however, are typically not supported. If at all possible, it's certainly easiest to utilize GNU sed, but if you need functionality on both platforms, remember to use POSIX features only, to ensure portability. If you're a Mac user and would like to take advantage of GNU sed as opposed to BSD sed, you might try installing Homebrew, and downloading GNU sed via command line with: $brew install gnu-sed.

To wrap things up, differences in version can really dictate what the proper syntax might look like, or what characters are necessary to escape. I hope this provides some additional context for the initial question as well as the accepted answer, and helps others consider how they should proceed, based on the end goal of their script and command usage.

Share Edit Follow Flag answered Mar 1, 2020 at 3:41

forthelulz
41 2