60 5 help

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

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

edited Feb 28 '12 at 23:55
Gilles
387k 75 714 1166

asked Feb 28 '12 at 4:42
detly
863 4 11 22

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 '16 at 12:57

3 Answers

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 quotes between single quotes by writing `'\''` (close single quote, one literal single quote, open single quote).

Sed uses basic regular expressions. In a BRE, 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. You can choose an alternate 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.

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.

In the replacement text:

- `&` and `\` need to be quoted, 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` ;
- in the replacement text, `&` , `\` and newlines need to be quoted;
- the delimiter needs to be quoted.
- Use double quotes for interpolation: `sed -e "s/$BRE/$REPL/"`

edited Jun 16 '15 at 16:41

answered Feb 29 '12 at 1:06
Gilles

387k 75 714 1166

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]
```

answered Feb 29 '12 at 2:56
R Perrin
1,551 6 7

I had this problem again, and forgot to scroll down to find the solution I upvoted last time. Thanks again. – isaaclw Apr 4 '14 at 20:17

Thanks a lot. Adding `-r` as an option was what was necessary in my case. – HelloGoodbye May 21 '15 at 8:23

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.

answered Feb 28 '12 at 5:58
Kyle Jones
9,468 1 18 41

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 '12 at 6:14

3 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 '12 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 '12 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 '12 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 '12 at 9:08

|