Unix 'alias' fails with 'awk' command

stackoverflow.com/questions/24245661/unix-alias-fails-with-awk-command

I'm creating an alias in Unix and have found that the following command fails..

```
alias logspace='find /apps/ /opt/ -type f -size +100M -exec ls -lh {} \; | awk
'{print $5, $9 }''
```

I get the following :

```
awk: cmd. line:1: {print
awk: cmd. line:1:       ^ unexpected newline or end of
string
```

Any ideas on why the piped awk command fails...

3 Answers

To complement's @Dropout's helpful answer:

tl;dr

The problem is the OP's attempt to use `'` *inside* a `'`-enclosed (single-quoted) string.

The **most robust** solution in this case is to **replace each *interior* `'` with** `'\''` (sic):

```
alias logspace='find /apps/ /opt/ -type f -size +100M -exec ls -lh {} \; |
                awk '\''{print $5, $9 }'\'''
```

- **Bourne-like (POSIX-compatible) shells do not support using `'` chars inside single-quoted (`'...'`-enclosed) strings AT ALL - not even with escaping**.
  - (By contrast, you CAN escape `"` inside a *double*-quoted string as `\"`, and, as in @Droput's answer, you can directly, embed `'` chars. there, but see below for pitfalls.)
- The solution above **effectively builds the string from multiple, single-quoted strings into which literal `'` chars. - escaped *outside* the single-quoted strings as `\'` - are *spliced in***.
  Another way of putting it, as @Etan Reisinger has done in a comment: `'\''` means: "close string", "escape single quote", "start new string".
- When **defining an alias**, you usually want ***single* quotes around its definition** so as to **delay evaluation** of the command until the alias is *invoked*.

Other solutions and their pitfalls:

The following discusses alternative solutions, based on the following alias:

```
alias foo='echo A '\''*'\'' is born at
$(date)'
```

Note how the `*` is effectively enclosed in single quotes - using above technique - so as to prevent pathname expansion when the alias is invoked later.

When invoked, this alias prints *literal* `A * star is born`, followed by the *then*-current date and time, e.g.: `A * is born at Mon Jun 16 11:33:19 EDT 2014`.

Use a feature called ANSI C quoting with shells that support it: `bash`, `ksh`, `zsh`

ANSI C-quoted strings, which are **enclosed in `$'...'`, DO allow escaping embedded `'` chars. as `\'`**:

```
alias foo=$'echo A \'*\' is born at
$(date)'
```

Pitfalls:

- This feature is not part of POSIX.
- By design, escape sequences such as `\n`, `\t`, ... are interpreted, too (in fact, that's the purpose of the feature).

Use of alternating quoting styles, as in @Dropout's answer:

Pitfall:

`'...'` and `"..."` have different semantics, so substituting one for the other can have unintended side-effects:

```
alias foo="echo A '*' is born at $(date)" # DOES NOT WORK AS
INTENDED
```

While syntactically correct, this will NOT work as intended, because the use of *double* quotes causes the shell to expand the command substitution `$(date)` *right away*, and thus ***hardwires* the date and time at the time of the alias *definition* into the alias**.

As stated: When **defining an alias**, you usually want ***single* quotes around its definition** so as to **delay evaluation** of the command until the alias is *invoked*.

Finally, a **caveat**:

The tricky thing in a Bourne-like shell environment is that **embedding `'` inside a single-quoted string sometimes - falsely - APPEARS to work** (instead of generating a syntax error, as in the question), when it instead does something different:

```
 alias foo='echo a '*' is born at $(date)'  # DOES NOT WORK AS
EXPECTED.
```

This definition is *accepted* (no syntax error), but won't work as expected - the right-hand side of the definition is effectively parsed as *3* strings - `'echo a '`, `*`, and `' is born at $(date)'`, which, due to how the shell parses string (merging adjacent strings, quote removal), results in the following, *single*, literal string: `a * is born at $(date)`. Since the `*` is *unquoted* in the resulting alias definition, it will expand to a list of all file/directory names in the current directory (pathname expansion).

You chould use different quotes for surrounding the whole text and for inner strings.

Try changing it to

```
alias logspace="find /apps/ /opt/ -type f -size +100M -exec ls -lh {} \; | awk
'{print $5, $9 }'"
```

In other words, your outer quotes should be different than the inner ones, so they don't mix.

Community wiki update:

- The **redeeming feature** of this answer is **recognizing that the OP's problem lies in unescaped use of the string delimiters (`'`) *inside* a string**.
- However, **this answer contains *general* string-handling truths, but does NOT apply to (Bourne-like, POSIX-compatible) shell programming *specifically***, and thus does not address the OP's problem *directly* - see the comments.

Note: Code snippets are meant to be **pseudo code**, not shell language.

Basic strings: You canNOT use the same quote within the string as the entire string is delimited with:

```
foo='hello, 'world!'';
    ^--start string
            ^-- end string
             ^^^^^^^^--- unknown "garbage" causing syntax
error.
```

You have to escape the internal strings:

```
foo='hello,
\'world!\'';
            ^--escape
```

This is true of pretty much **EVERY** programming language on the planet. If they don't provide escaping mechanisms, such as `\`, then you have to use alternate means, e.g.

```
quotechar=chr(39); // single quote is ascii #39
foo='hello ,' & quotechar & 'world!' &
quotechar;
```