

What does this do in regards to an incoming argument in a shell script

Asked 10 years, 3 months ago Modified 10 years, 3 months ago Viewed 98 times



0



I have been using this in bash and sh scripts because I read somewhere, which I can't seem to locate any longer, that it is the best way to deal with inbound arguments that have spaces, slashes, or other characters that may need special treatment like escaping.

```
INCOMING_ARG=${1%/}
```

Can someone break that down to me, or tell me if there is an even better way?

For the longest time I simply used `$1`, `$2`, `$3`, `$4` etc., as they came in, never was bit in the butt, but I did do some sanitizing on the argument which I no longer do with `${1%/}`.

bash

shell

args

Share

Improve this question

Follow

edited Sep 22, 2014 at 7:47



Jonathan Leffler

752k ● 145 ● 946 ● 1.3k

asked Sep 20, 2014 at 5:35



Scott

611 ● 1 ● 5 ● 12


Please fix the auto-correct behaviour on your gadget — or at least fix up the mess it leaves behind. — Jonathan Leffler Sep 20, 2014 at 7:15

If you always double-quote your variable references, you won't go far wrong. As the answers suggest, the notation you're using now doesn't really buy you anything useful — indeed, it could even be harmful in some edge cases. — Jonathan Leffler Sep 20, 2014 at 7:18


I don't know what is meant by my "gadget", I just typed in a question. — Scott Sep 22, 2014 at 7:38

In this context, your 'gadget' is the device you typed the question on. I'm guessing it was a smartphone or tablet with spelling 'correction' enabled, so that various words were mangled from what you typed into what might be sensible in a non-computer discussion but which were severely out of place here ('inkling' for 'incoming'; 'in bounce' for 'inbound'). Of course, if I got the scenario wrong and you in fact intended the typos to be entered, then you need to explain what an 'inkling argument' and an 'in bounce argument' actually is — you would be using non-standard terminology. — Jonathan Leffler Sep 22, 2014 at 7:43

"Inkling Argument", it's what we're having right now :) "In Bounce", that's when we've made up and are in sync with each other's steps, or "bounce" as the kids say these days. Seriously,

sorry for a terrible post, there were answers that helped regardless, so I appreciate you all for the help! – [Scott](#) Sep 24, 2014 at 1:18 

2 Answers

Sorted by: Highest score (default) 



4



`${1%/}` just removes the trailing slash from `$1`. That is all.

This is an example of the bash parameter expansion `${parameter%word}` where `parameter` is the name of the shell variable and `word` is whatever you want to remove from the end of the parameter. For example:

```
$ a=/home/me/dir/ ; echo ${a%/}
/home/me/dir
$ a=/home/me/dir/ ; echo ${a%/dir/}
/home/m
```

Nothing about this helps with args that have spaces, slashes, or other characters that may need special treatment like escaping.

Trailing slashes and directory names

Unix/Linux is quite tolerant of surplus slashes. When given duplicate slashes, Unix ignores them. Observe, for example, that the following all point to the same directory:

```
$ ls -altd /usr/bin/ /usr/bin// /usr/bin///
drwxr-xr-x 2 root root 81920 Sep 19 11:25 /usr/bin/
drwxr-xr-x 2 root root 81920 Sep 19 11:25 /usr/bin//
drwxr-xr-x 2 root root 81920 Sep 19 11:25 /usr/bin///
```

And the following all point to the same file:

```
$ ls -alt /usr/bin/xpdf /usr/bin//xpdf /usr/bin///xpdf
-rwxr-xr-x 1 root root 2358 Apr 17 2012 /usr/bin/xpdf
-rwxr-xr-x 1 root root 2358 Apr 17 2012 /usr/bin//xpdf
-rwxr-xr-x 1 root root 2358 Apr 17 2012 /usr/bin///xpdf
```

Consequently, removing a trailing slash from a directory name usually is not necessary.

Share

edited Sep 20, 2014 at 6:31

answered Sep 20, 2014 at 5:43

Improve this answer

Follow



[John1024](#)

114k ● 14 ● 150 ● 181

Thanks that explains a lot to me. I wish I could find the source, as it was one of those "what not to do in bash" type of tutorials and what I posted was the solution. At any rate, the answer is easy enough, so I shall quote my vars in all cases and should be fine more often than not.

– [Scott](#) Sep 22, 2014 at 7:43



All that does is strip a trailing slash on `$1`.

1

```
$ X=foo/  
$ echo ${X%/}  
foo
```



But it's not safe against whitespace or unusual characters, and can actually make things worse, because its expansion is subject to further pathname expansion.

Suppose you had a script where `$1` was equal to a literal `*/`, that was passed quoted into the script, and you want to preserve its value without the script instead of having it expand in a weird way. Then `${X%/}` is definitely not what you want to do:

```
$ touch foo.txt  
$ X='*/'  
$ echo $X  
*/  
$ echo ${X%/}  
foo.txt
```

This happens because it's not quoted.

Don't use this.

In shell scripts, in general, always quote everything.

```
$ echo "${X%/}"  
*
```

Share

edited Sep 20, 2014 at 16:37

answered Sep 20, 2014 at 5:44

Improve this answer

Follow



[andrewdotn](#)

34.7k ● 10 ● 109 ● 133

- 1 You should try `echo $X` in a directory with sub-directories. You will find that it does indeed expand the `*`; it didn't in your example because `*/` is constrained to directories and not regular files. – [rici](#) Sep 20, 2014 at 6:10

@rici The point is that `${X%/}` is not "the best way to deal with in bound arguments that have spaces, slashes, or other characters that may need special treatment like escaping," because it does something completely different, and without quoting, it will expand special characters instead of escaping them. `X` is presumably an already-expanded filename or

argument containing special characters. The fact that `${X%/}` mangles the value of `X` in more cases than I mentioned doesn't disprove my point, but reinforces it. – [andrewdotn](#) Sep 20, 2014 at 16:36

- 1 My point is that the unquoted `$X` also undergoes pathname expansion, so your example is misleading. If you added `mkdir foo` after `touch foo.txt`, you would see a more useful comparison. **Both** forms need to be quoted. Of course, you are correct that `${X%/}` is not a magical incantation to safeguard a script, but you shouldn't imply that it affects the application of pathname expansion, because it doesn't. – [rici](#) Sep 20, 2014 at 17:44
-

@rici Why are you harassing me? I'm not claiming `$X` is safe at all. What could be clearer than saying, "Don't use this ... always quote everything"? – [andrewdotn](#) Sep 20, 2014 at 21:09
