Sudo and Globbing

Finnbarr P. Murphy

(fpm@fpmurphy.com)

The question is how we can use the *sudo* utility to display a list of files in a directory to which we have absolutely no Unix filesystem privileges

Consider the following directory and files contained therein:

```
$ ls -l
total 4
drwxrwx---. 2 root root 4096 May 22 21:14 demo
$ su
Password: XXXXXXXX
# ls -l demo
total 0
-rw-r--r--. 1 root root 0 May 22 21:14 file1
-rw-r--r--. 1 root root 0 May 22 21:14 file2
-rw-r--r--. 1 root root 0 May 22 21:14 file3
# exit
exit
```

Note the directory permissions are 770 and the user and group owner is *root*.

Here is the output when I attempt to list the subdirectory contents:

```
$ whoami
fpm
$ ls -l demo
ls: cannot open directory demo: Permission denied
$
```

This is what I expect because I am not *root* nor a member of the *root* group. To enable me to view the directory contents, the directory permissions need to be at least 771 (drwxrwx–x) for a non-root user such as myself to list the directory.

So how can I work around this limitation? One workaround is by using *sudo*. Assuming I have root privledges via *sudo* either explicitly or as a member of the *wheel* group, here is what happens when I naively use *sudo* to attempt to list the subdirectory contents:

```
$ sudo ls -l demo/file1
-rw-r--r--. 1 root root 0 May 22 21:14 demo/file1
$ sudo ls -l demo/*
ls: cannot access demo/*: No such file or directory
$
```

I still cannot list the files in the *demo* subdirectory.

The above does not work because the shell attempts to glob *demo/** before passing the result of globing to *sudo* which then passes it unchanged to *ls*. Since the shell is not running as *root*, it is not able to traverse the directory and globbing fails. Thus, *ls* receives the unexpanded globbing pattern *demo/** in its argument list.

Note that *ls* and other Linux utilities do not perform globbing. The utilities expect a list of filenames as command line arguments.

The most straight forward way to get correct results is to invoke *ls -l demo/** in a sub-shell. Quote the command string using either double quotes (“) or single quotes (‘).

```
$ sudo sh -c "ls -l demo/*"
-rw-r--r--. 1 root root 0 May 22 21:14 demo/file1
-rw-r--r--. 1 root root 0 May 22 21:14 demo/file2
-rw-r--r--. 1 root root 0 May 22 21:14 demo/file3
$ sudo sh -c 'ls -l demo/*'
-rw-r--r--. 1 root root 0 May 22 21:14 demo/file1
-rw-r--r--. 1 root root 0 May 22 21:14 demo/file2
-rw-r--r--. 1 root root 0 May 22 21:14 demo/file3
$ sudo -s ksh -c "ls -l demo/*"
-rw-r--r--. 1 root root 0 May 22 21:14 demo/file1
-rw-r--r--. 1 root root 0 May 22 21:14 demo/file2
-rw-r--r--. 1 root root 0 May 22 21:14 demo/file3
$
```

As you can see, there is no difference in output.

Note that there may be a difference if you use shell variables. Here is an example where there is a difference.

```
$ sudo sh -c 'echo $USER'
root
$ sudo sh -c "echo $USER"
fpm
```

There are other workarounds to this issue besides *sudo*. For example, I could be added to the *root* group but that has security implications. Another group of which I am a member could be given group ownership of the subdirectory. POSIX ACLs could be used to grant browsing and other permissions to the subdirectory.

