Advice: Safely Removing a File or Directory in Unix

Removing files and directories is a very common task, so in some environments support engineers or automation scripts delete hundreds of files per day. That’s why I think it’s important to be familiar with different ways and safety mechanisms you should use when it comes to removing Unix directories. This article has a number of principles that should help you make your day-to-day files and directories operations safer.

DISCLAIMER: one can never be too careful when using a command line, and removing files or directories in Unix is not an exception. That’s why please take extra care and spend time...
planning and understanding commands and command line options before executing them on production data. I’m sharing my own advice and my approach, but DO YOUR OWN RESEARCH as I accept no responsibility for any possible loss caused by direct or indirect use of the suggested commands.

Safely Removing Files and Directories

Advice in this article is equally applicable to commands you type and to the automation solutions you create. Be it a single command line or a complex Ansible playbook – safety mindset should be applied whenever you’re creating an actionable plan for working with important data.

If you can think of any more advice related to this topic, please let me know!

1. Double-check Directories Before Removing

I wouldn’t call this out if it hasn’t saved me so many times. No matter who made the request, no matter how urgent the task is, no matter how basic and obvious the directory name seems, always double-check directories before removing!

Basic approach is: replace `rm/rmdir` with `ls -l command`. 
So instead of

```bash
$ rm -rf /etc /bin
```

you type `ls` command and review the output:

```bash
$ ls -l /etc /bin
```

Things you’re checking for are:

1. Is this a user/task specific directory or a global directory?
2. Does it seem to be part of the core OS?
3. Will removing these files break any functionality you can think of?
4. Does the directory contain any files?
5. Does the number of files seem different from what you expected?

**For instance:** you’re asked to delete an empty directory. Do a quick `ls`, and if it has files – double-check if they should be deleted as well. Also, check if it’s one of the common core OS directories like `/etc` or `/bin` or `/var` – it could be that you got the name by mistake but removing directory without checking would become an even bigger mistake.

2. Consider Moving Instead of
Removing

In troubleshooting, many requests are made so that you free up a directory or tidy up filesystem structure. But the issues are mostly around file and directory names, rather than the space they take up.

So if filesystem space is not an issue right now, consider moving the directory instead of removing (deleting) it completely:

$ mv /home/greys/dir1 /home/greys/dir1.old

The end result will be that /home/greys/dir1 directory is gone, but you still have time to review and recover files from /home/greys/dir1.old if necessary.

3. Use root Privilege Wisely

**Hint: don’t use root unless you absolutely have to.** If the request is to remove a subdirectory in some application path – find out what user the application is running as and become that user before removing the directory.

For instance:

# su - javauser
# rm -rf /opt/java/logs/debug
Run as root user, this will let you become a javauser and attempt to remove the /opt/java/logs/debug directory (debug subdirectory in /opt/java/logs directory).

If there’s an issue (like getting permissions denied error) – review and find out what the problem is instead of just becoming root and removing the directory or files anyway. **Specifically:** permission denied suggests that files belong to another user or group, meaning they are potentially used and needed for something else and not just the application you’re working on.

4. Double-check Any Masks or Variables

If you’re dealing with expanding filename masks, double-check them to have correct and non-zero values.

Consider this:

```bash
$ rm -rf /${SOMEDIR}
```

if you’re not careful validating it, then it’s quite possible that $SOMEDIR is not initialised (or initialised under some other user session), thus resulting in the vastly different command with catastrophic results (yes, I know: this exact example below is NOT that bad, because as regular user it simply won’t work. But run as run will result in OS self-destruct):

Similarly, if there are filenames to be expanded, verify that expansion works as intended. Very important thing to realise is that your filename masks will be expanded as your current user.

$ <SUDO> rm /$(ls /root)
ls: cannot access '/root/': Permission denied
   rm: cannot remove '/': Is a directory

This example above is using shell expansion: it runs ls /root command that will return valid values if you have enough permissions. But run as regular use this will give an error and also alter the path used for the rm command. It will be as if you tried to run the following with sudo privileges:

$ rm /

Again, I’m not giving you the full commans as it’s all too easy to break your Unix like OS beyond repair when you run full commands as root without double-checking.

5. echo Each Command Before Running

The last principle I find very useful is to prepend any potentially dangerous command with echo. This means your shell will attempt expanding any command line parameters and substitutions, but then show the resulting command line instead of actually executing it:
```bash
greys@becky:~ $ echo rm -rf /opt/java/logs/${HOSTNAME}
rm -rf /opt/java/logs/becky

See how it expanded `${HOSTNAME}` variable and replaced it with the actual hostname, becky?

Use echo just to be super sure about what you think the Unix shell will execute.

That’s it for today, hope you like this collection of safety principles. Let me know if you want more articles of this kind!

See Also

- mkdir permission denied
- rm command
- rmdir command
- ls command

return nothing