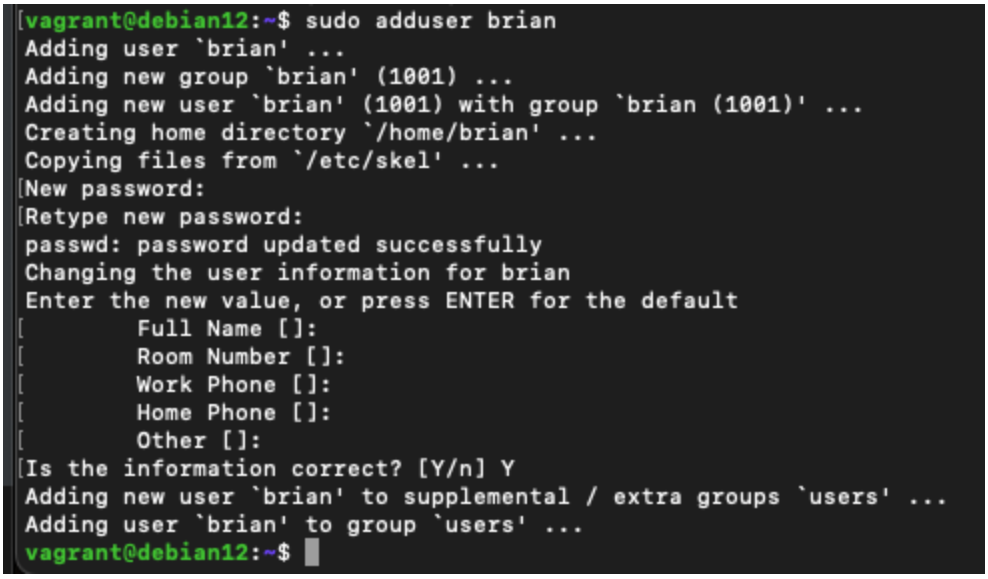
File Permissions Exercises (1)

Creating a user and group

Create a new user with `sudo adduser NAME` - I'm going to be using `brian` as an example name in these notes. When it asks for a password, you can just use `brian` or something; it will complain about the password being too short but it will create the user anyway. You can skip the GECOS information asking for a full name and phone number---it's just to help an admin contact you if needed.



Checking group membership

`group USERNAME`

```
groups nigel
```

```
nigel@debian12:/home/brian$ groups nigel
nigel : nigel users
```

Check the user and group files with `tail /etc/passwd` and `tail /etc/group` to check that the new user has been created - `tail` displays the last 10 lines of a file by default; `tail -n N FILE` would display the last N lines. Your new user `brian` (or whatever you called them) should appear in both files.

```
vagrant@debian12:~$ tail /etc/passwd
systemd-network:x:998:998:systemd Network Management:/:/usr/sbin/nologin
systemd-timesync:x:997:997:systemd Time Synchronization:/:/usr/sbin/nologin
messagebus:x:100:107::/nonexistent:/usr/sbin/nologin
sshd:x:101:65534::/run/sshd:/usr/sbin/nologin
vagrant:x:1000:1000:vagrant,,,:/home/vagrant:/bin/bash
memcache:x:102:110:Memcached,,,:/nonexistent:/bin/false
postfix:x:103:112::/var/spool/postfix:/usr/sbin/nologin
vboxadd:x:999:2::/var/run/vboxadd:/sbin/nologin
mysql:x:104:114:MySQL Server,,,:/nonexistent:/bin/false
brian:x:1001:1001:,,,:/home/brian:/bin/bash
vagrant@debian12:~$
```

Also check with `ls -l /home` that the home directory for Brian exists and is set to the correct user and group.

```
vagrant@debian12:~$ ls -l /home
total 8
drwx------ 2 brian   brian   4096 Mar  3 12:11 brian
drwx------ 3 vagrant vagrant 4096 Jan 10 04:24 vagrant
vagrant@debian12:~$
```

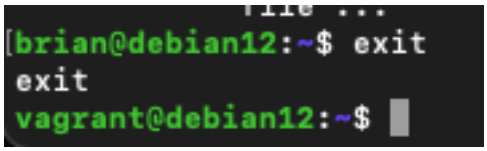
Time to change user: `su brian` and enter the password. Notice that the prompt has changed to `brian@debian12:/home/vagrant$` (at least if you started off in that folder). So the user has changed, and because `/home/vagrant` is no longer the current user's home directory, it gets written out in full.

```
vagrant@debian12:~$ su brian
Password:
brian@debian12:/home/vagrant$
```

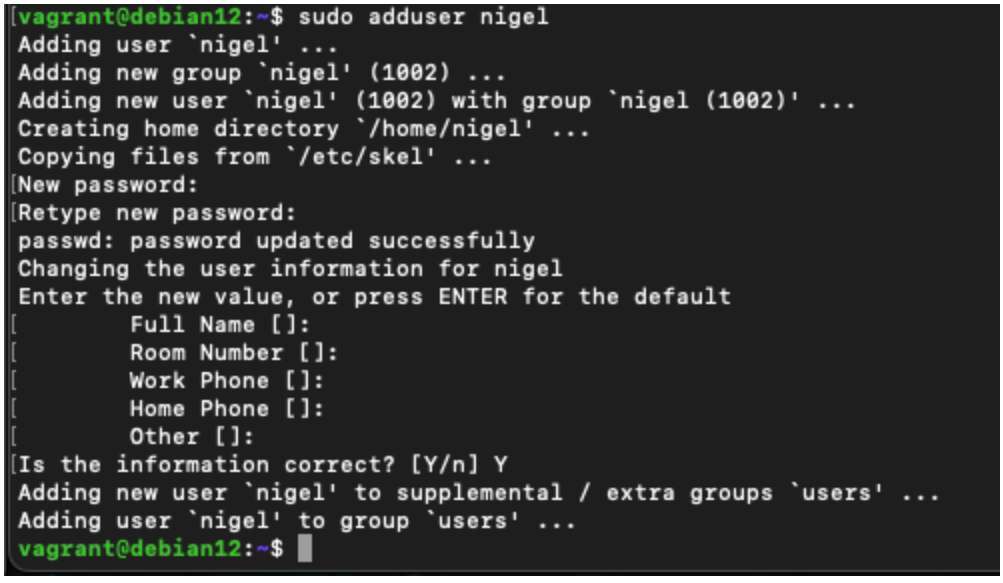
Run `cd` to go home followed by `pwd` and check that you are now in `/home/brian` or whatever you called your new user.

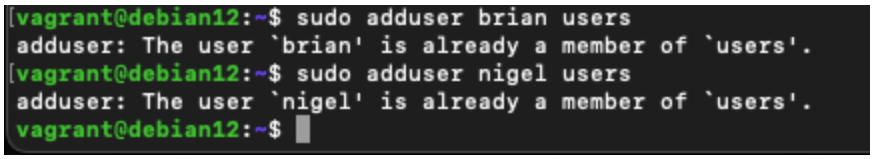
Note: `brian` cannot use sudo, so you have to exit his terminal to get back to one running as vagrant for this.

- could use either `exit` or *Ctrl+D*



Next, create a user `nigel` (or some other name) add both your two new users, but not `vagrant`, to the group `users` (which already exists) using the command `sudo adduser USERNAME GROUPNAME`, where group and username are changed accordingly.

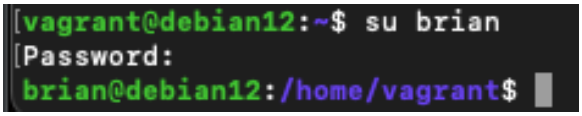




Explore file permissions

As user `brian` (or whatever you called your first new user)

- to switch users use `su USERNAME` :



Then change to brians home directory (`cd /home/brian` or just `cd` or `cd~` if youre logged in as brian):

Then set up your home directory using what you learnt in the videos so that

Use `ls -l /home` to check out brians home directory permissions

- You can do everything (rwx).
- Members of the `users` group can list files and change to your home directory, but not add/remove files. You will need to change the group of your home directory to `users` for this, using the command `chgrp -R GROUPNAME DIRECTORY`.

```
chmod g+rx /home/brian     // when you are logged in as brian
```

Now nigel can use `ls` in brian's home directory, he can view the readme file but cannot write to it.

- Everyone else cannot do anything with your home directory.

use the `chmod u=rwx,g=rw,o= /home/brian`

```
brian@debian12:~$ chmod u=rwx,g=rw,o= /home/brian
```

This could also be achieved with `chmod 750 /home/brian`

- 7 in binary = 111. This grants rwx permissions
- 5 in binary = 101. This grants read and executablee permissions to the group
- 0 in binary = 000. This grants no rwx permiossions to others

clarify the below:

110 = read write

111 = read write execute

001 = execute only

100 = read only???

101 = read and execute

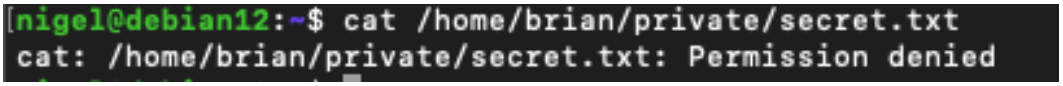
Also as `brian`, make a `private` subdirectory in your home folder that no-one but you can access (read, write or execute).

```
chmod go-rwx private
```

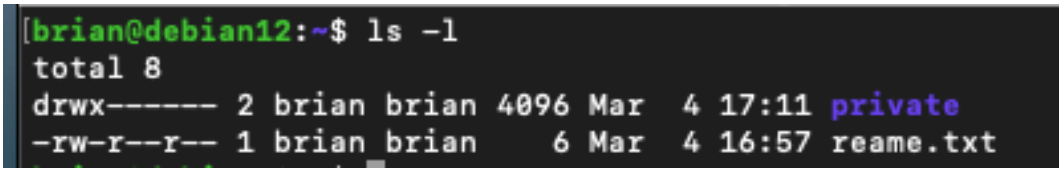
```
[brian@debian12:~$ chmod go-rwx private
[brian@debian12:~$ ls -l
total 8
drwx------ 2 brian brian 4096 Mar  4 17:05 private
-rw-r--r-- 1 brian brian    6 Mar  4 16:57 reame.txt
```

Create a file `secret.txt` in there with `nano private/secret.txt` as user `brian` from Brian's home directory, and put something in it. Do not change any permissions on `secret.txt` itself.

Check as Nigel that you can see the folder itself, but not cd into it nor list the file. Check that even knowing the file name (`cat /home/brian/private/secret.txt`) as Nigel doesn't work.



Using `ls -l` as Brian in both `~` and `~/private`, compare the entries for the files `~/readme.txt`, `~/private/secret.txt` and the folder `~/private`. Why do the groups of the two files differ?



Note that, even though the secret file has read permissions for everyone by default, Nigel cannot read it. The rule is that you need permissions on the whole path from `/` to a file to be able to access it.

This is another reminder that if you want to store private files on a lab machine, then put it in a folder that is only accessible to you. Other students can read your home directory by default, and they would be able to look at your work. This has led to plagiarism problems in the past, but good news: we keep logs and can usually figure out what happened! `:-)`.

Altenatively you could remove permissions from everyone else on your home directory there, but this prevents you from being able to share files in specific folders that you do want to share with other students.

setuid

We are going to create a file to let Nigel (and others in the users group) send Brian messages which go in a file in his home directory.

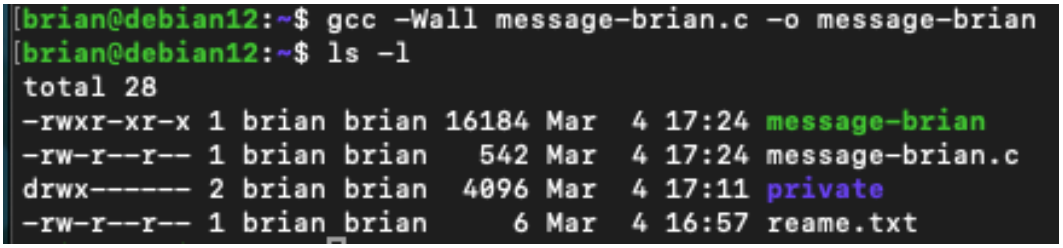
As Brian, create a file `message-brian.c` in your home directory and add the following lines:

```
#include <stdio.h>
#include <stdlib.h>

const char *filename ="/home/brian/messages.txt";

int main(int argc, char **argv) {
  if (argc != 2) {
    puts("Usage: message-brian MESSAGE");
    return 1;
  }
  FILE *file = fopen(filename, "a");
  if (file == NULL) {
    puts("Error opening file");
    return 2;
  }
  int r = fputs(argv[1], file);
  if (r == EOF) {
    puts("Error writing message");
    return 2;
  }
  r = fputc('\n', file);
  if (r == EOF) {
    puts("Error writing newline");
    return 2;
  }
  fclose(file);
```

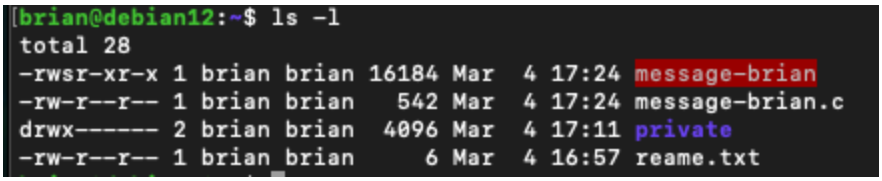
```
  return 0;
}
```

Compile it with `gcc -Wall message-brian.c -o message-brian` (you should not get any warnings) and check with `ls -l`, you will see a line like:

```
-rwxr-xr-x    1 brian     brian         19984 Oct 28 13:26 me
ssage-brian
```



These are the default permissions for a newly created executable file; note that gcc has set the three `+x` bits for you. Still as Brian, run `chmod u+s message-brian` and check the file again: you should now see `-rwsr-xr-x` for the file permissions. The `s` is the setuid bit.



As Nigel (`su nigel`), go into Brian's home directory and run `./message-brian "Hi from Nigel!"`. The quotes are needed here because the program accepts only a single argument.

Now run `ls -l` and notice that a `messages.txt` has appeared with owner and group `brian`. Check the contents with `cat messages.txt`. Although Nigel cannot create and edit files in Brian's home directory himself (he can't

edit `messages.txt` for example, although he can read it), the program `message-brian` ran as Brian, which let it create the file. Nigel can send another message like this (`./message-brian "Hi again!"`), which gets appended to the file: try this out.

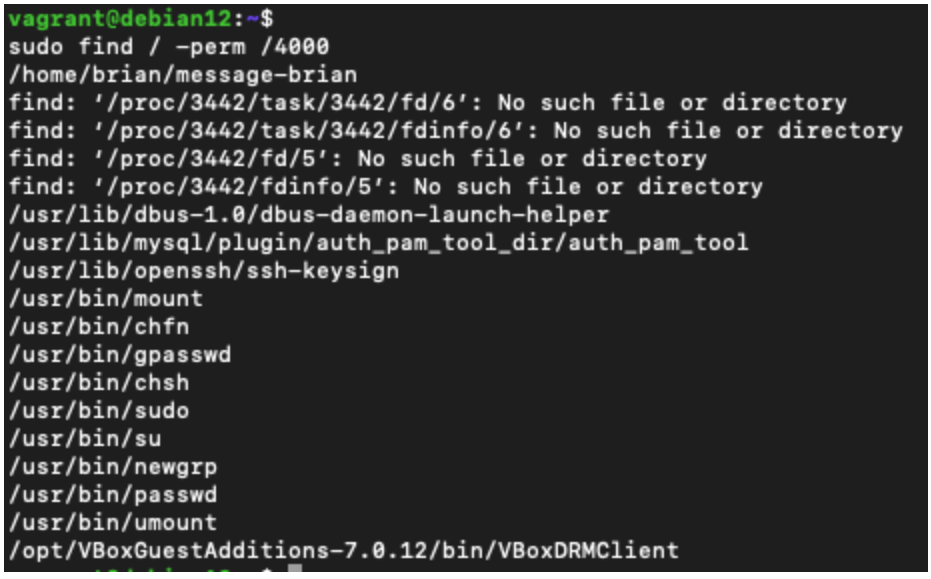
This shows how setuid programs can be used to allow other users to selectively perform specific tasks under a different user account.

Warning: writing your own setuid programs is extremely dangerous if you don't know the basics of secure coding and hacking C programs, because a bug in such a program could let someone take over your user account. The absolute minimum you should know is the contents of our security units up to and including 4th year.

A general task for a security analyst might be finding all files with the setuid bit set on a system. You can try this yourself, but return to a vagrant shell first so that you're allowed to use sudo:

```
sudo find / -perm /4000
```

You might get some errors relating to `/proc` files, which you can ignore: these are subprocesses that find uses to look at individual files.



Apart from `message-brian`, you'll find a few files by default: `sudo`, `mount`, `umount` and `su`. The first one you already know; look up what the next two do and think about why they are setuid. Specifically, what kinds of (un)mounting are non-root users allowed to do according to the manual pages?

`mount` & `unmount`

From `man mount` :

"All files accessible in a Unix system are arranged in one big tree, the file hierarchy, rooted at /. These files can be spread out over several devices. The **mount** command serves to attach the filesystem found on some device to the big file tree. Conversely, the **umount**(8) command will detach it again. The filesystem is used to control how data is stored on the device or provided in a virtual way by network or other services."

Look up the `passwd` program in the manual pages. Why might that program need to be setuid?

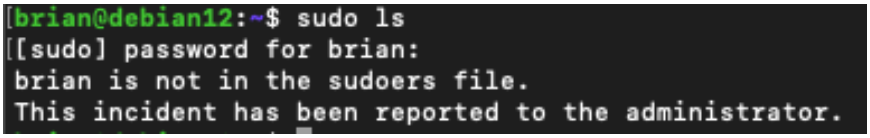
The reason `passwd` needs to be setuid (set user ID upon execution) is because the password files it needs to modify are usually owned by the root user and are not writable by regular users. The setuid bit on the `passwd` executable ensures that when any user runs the `passwd` program, the program runs with the permissions of the file's owner, which is usually the root user, instead of running with the permissions of the user who launched the program. This allows the `passwd` program to modify the protected password files and update the user's password while maintaining the security of those files against unauthorized access.

Setting the setuid bit is a way to grant specific privileges to users for tasks that require higher permissions than those users typically possess, without giving

them full root access. However, it's crucial to manage and monitor setuid files carefully, as they can introduce security risks if misused or if there are vulnerabilities in the program being granted elevated privileges.

sudo

Make sure your terminal is running as `brian` and try a `sudo ls`. You will see a general message, you will be asked for your password, and then you will get the error `brian is not in the sudoers file. This incident will be reported.` (This means that an entry has been logged in `/var/log/messages`.)



So, `brian` can currently not use sudo. Switch back to `vagrant` and run the command `sudo cat /etc/sudoers`. Everything is commented out except `root ALL=(ALL) ALL` and the last line `#includedir /etc/sudoers.d` (this is not a comment!) which contains a single file `vagrant` with the line `vagrant ALL=(ALL) NOPASSWD: ALL` which is why vagrant can use sudo in the first place.

```
vagrant@debian12:/home/brian$ sudo cat /etc/sudoers
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset
Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:
/sbin:/bin"

# This fixes CVE-2005-4890 and possibly breaks some versions of kdesu
# (#1011624, https://bugs.kde.org/show_bug.cgi?id=452532)
Defaults        use_pty

# This preserves proxy settings from user environments of root
# equivalent users (group sudo)
#Defaults:%sudo env_keep += "http_proxy https_proxy ftp_proxy all_proxy no_prox
y"

# This allows running arbitrary commands, but so does ALL, and it means
# different sudoers have their choice of editor respected.
#Defaults:%sudo env_keep += "EDITOR"

# Completely harmless preservation of a user preference.
#Defaults:%sudo env_keep += "GREP_COLOR"

# While you shouldn't normally run git as root, you need to with etckeeper
#Defaults:%sudo env_keep += "GIT_AUTHOR_* GIT_COMMITTER_*"

# Per-user preferences; root won't have sensible values for them.
#Defaults:%sudo env_keep += "EMAIL DEBEMAIL DEBFULLNAME"

# "sudo scp" or "sudo rsync" should be able to use your SSH agent.
#Defaults:%sudo env_keep += "SSH_AGENT_PID SSH_AUTH_SOCK"

# Ditto for GPG agent
#Defaults:%sudo env_keep += "GPG_AGENT_INFO"

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "@include" directives:

@includedir /etc/sudoers.d
vagrant@debian12:/home/brian$
```

However, note the commented lines such as

```
# %wheel ALL=(ALL) NOPASSWD: ALL
# %sudo ALL=(ALL) ALL
```

If uncommented, the first one would let everyone in group wheel run commands using sudo (this is the default on some other linux distributions), whereas the

second one would allow everyone in the group `sudo` to do this, but would prompt for their own password beforehand.

Let's allow people in the users group to reboot the machine. Open a root shell with `sudo su` as vagrant; this is so we don't get locked out if we break sudo.

Edit the sudoers file with `visudo` as root, and add the following line:

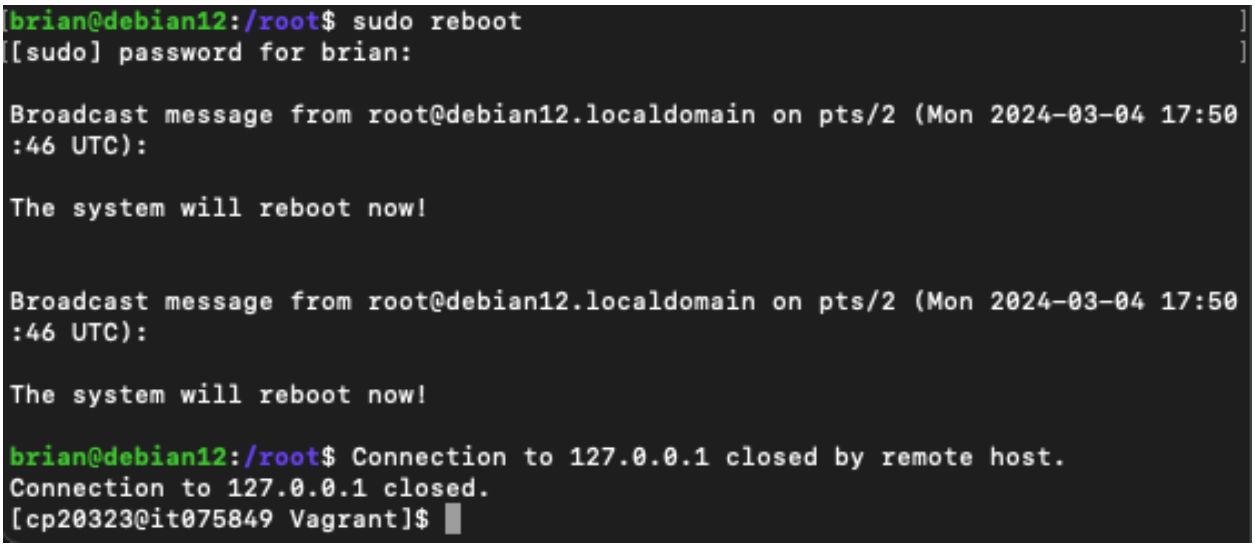
```
%users ALL=(ALL) /sbin/reboot
```

and save the sudoers file.

Warning:: Never edit `/etc/sudoers` directly and *always* use `visudo` instead. If you make a mistake and add a syntax error to the file then `sudo` will refuse to work. If your root account doesn't have a password (some people don't like that as a security precaution) then you'll have to spend the next half-hour figuring out how to break into your own computer and wrestle back control. There is almost always a command to check a config file before replacing the current one: the same advice also applies to the ssh config files. If you break them you might have to travel to wherever the server is with a keyboard and a monitor.

%users ALL=(ALL) /sbin/reboot

You can now switch back to `brian` (check the prompt to make sure you are Brian) and do `sudo reboot`. After asking for Brian's password, the virtual machine will now reboot, which you notice because you get kicked out of your ssh connection. Another `vagrant ssh` after a few seconds will get you back in again.


brian@debian12:/root$ sudo reboot
[sudo] password for brian:
Broadcast message from root@debian12.localdomain on pts/2 (Mon 2024-03-04 17:50
:46 UTC):
The system will reboot now!
Broadcast message from root@debian12.localdomain on pts/2 (Mon 2024-03-04 17:50
:46 UTC):
The system will reboot now!
brian@debian12:/root$ Connection to 127.0.0.1 closed by remote host.
Connection to 127.0.0.1 closed.
[cp20323@it075849 Vagrant]$