Understanding when to use root

The root user account exists on all Linux distributions and is the most powerful user account on the planet. The root user account can be used to do anything within your server, and I do mean anything. Want to create files and directories virtually anywhere on the filesystem? Want to install software? These processes are easily performed with root. The root account can even be used to destroy your entire installation with one typo or ill-conceived command: if you instruct root to delete all the files on your entire hard disk, it won't hesitate to do so. It's always assumed on a Linux system that if you are using root, you are doing so because you know what you are doing. So, there's often not so much as a confirmation prompt while executing any command as root. It will simply do as instructed, for better or worse.

Using sudo to run privileged commands

just keep in mind that the purpose of sudo is to enable you to use your user account to do things that normally only root would be able to do. For example, as a normal user, you cannot issue a command such as the following to install a software package:

```
apt install nmap
```

But if you prefix the command with sudo (assuming your user account has access to it), the command will work just fine:

```
sudo apt install nmap
```

When you use sudo, you'll be asked for your user's password for confirmation, and then the command will execute. Subsequent commands prefixed with sudo may not prompt for your password, as it will cache your password for a short period of time until it times out or the terminal is closed. Creating and removing users Creating user using adduser When you run this command, you will be asked a series of questions regarding how you want the user to be created. Run the following commands to create two users:

```
sudo adduser user1
```

Then answer all questions asked and then run:

```
sudo adduser user2
```

To see all users exist on your system:

```
cat /etc/passwd
```

If you want to see the exact user you are searching:

```
cat /etc/passwd | grep user1
```

This will display user1 information only, thanks to `pipe | and grep.` Removing users Removing access is very important when a user no longer needs to access a system, as unmanaged accounts often become a security risk. To remove a user account, use the userdel command. To delete user2, run:

```
sudo userdel user2
```

By default, the userdel command does not remove the contents of the user's home directory. Here, We can see that the files for the 'user1' user still exist by entering the following command:

```
ls -l /home
```

If you do actually want to remove a user's home directory at the same time that you remove an account, just add the -r option. This will eliminate the user and their data in one shot: First, check if the home directory of user1 exist in /home

```
ls -l /home
```

Then attempt to remove user1 and its home directory: sudo userdel -r user1 Can you verify if the home directory of user1 is removed as well?

```
ls -l /home
```

Switching users Create two more users on the system:

```
adduser ali
adduser asha
```

Check which user you logged in on the system(who you really are):

```
whoami
```

Do you see your user? There is another you can recognize the user you're currently logged in by looking at your shell prompt: Now that we have several users on our system, we need to know how to switch between them. Of course, you can always just log in to the server as one of the users, but you can actually switch to any user account at any time, provided you either know that user's password or have sudo access. The command you will use to switch from one user to another is the su command. If you enter su with no options, it will assume that you want to switch to root and will ask you for your root password. As we mentioned earlier, Ubuntu locks the root account by default,

so at this point you may not have a root password. To switch from somnog user to asha, run:

```
su - asha
```

Then provide user asha's password. To logout, just run:

```
exit
```

Managing groups

Now that we understand how to create, manage, and switch between user accounts, we'll need to understand how to manage groups as well. With Linux, it's just one-to-one ownership: just one user and just one group assigned to each file or directory. If you list the contents of a directory on a Linux system, you can see this for yourself:

```
ls -l
```

The following is a sample line of output from a directory on one of my servers:

```
-rw-r—r-- 1 root bind 490 2020-04-15 22:05 named.conf
```

If you were curious as to which groups exist on your server, all you would need to do is cat the contents of the `/etc/group` file. Similar to the /etc/passwd file we covered earlier, the /etc/group file contains information regarding the groups that have been created on your system. Go ahead and take a look at this file on your system:

```
cat /etc/group
```

Creating new group

```
sudo groupadd sales
sudo groupadd admins
```

If you take a look at the `/etc/group` file again after adding a new group, you'll see that a new line was created in the file. Removing group Removing a group is just as easy. Just issue the groupdel command followed by the name of the group you wish to remove: sudo groupdel admins If we wanted to add a user to our admins group, we would issue the following command: sudo usermod -aG admins myuser Setting permissions on files and directories Viewing permissions on a file:

```
ls -l
```

```
-rw-rw-r-- 1 somnog somnog 26 Oct 23 07:27 test
```

As you can see, permissions are read differently depending on their context: whether they apply to

a file or a directory. Changing permissions Create a new empty file:

```
touch file.txt
```

View the permissions of this file:

```
ls -l file.txt
```

`-rw-rw-r-- 1 somnog somnog 0 Oct 23 16:00 file.txt`

The user has read and write, and read and write permission for the group, and only read for others. Remove read bit from others:

```
chmod o-r file.txt
```

With this example, we're removing the r bit from other (o-r). If we wanted to add this bit instead, we would simply use + instead of -.

```
chmod u+x file.txt
```

In addition, you can also use octal point values to manage and modify permissions. This is actually the most common method of altering permissions. Basically, each of the permission bits (r, w, and x) has its own octal equivalent, as follows: `Read = 4 Write = 2 Execute = 1` For example to give full permission for on file.txt:

```
chmod 667 file.txt
```

Which means: read and write for users and groups and read, write and execute for others. To give all groups full permission:

```
chmod 777 file.txt
```

You really don't need to do that though! Changing the ownership of files Switch to user asha:

```
touch newfile.txt
```

Check the ownership of the file:

```
ls -l newfile.txt
```

`-rw-rw-r-- 1 somnog somnog 0 Oct 23 16:10 newfile.txt` As you can from above command, the

user owner of this somnog and the group owner is somnog, the primary group of somnog user. Let's the user owner of this to asha (remember we created this user earlier, if not create it now with adduser)

```
sudo chown user:group filename
sudo chown asha newfile.txt
```

Then check now:

```
ls -l newfile.txt
```

Or you can change the user and the group in one command:

```
sudo chown ali:sales newfile.txt
```

To only change the group owner you use, chgrp command:

```
sudo chgrp somnog newfile.txt
```

Managing Software Packages To update the repository:

```
apt update
```

To upgrade:

```
apt upgrade
```

To install a new package:

```
apt install htop
```

To remove this software:

```
apt remove htop
```

To remove the package and all its files:

```
apt purge packagename
```

To list installed packages on your server:

```
apt list --installed
```