root (super) user, su and sudo

coursera.org/learn/linux-for-developers/supplement/XSxag/root-super-user-su-and-sudo

It is possible to enter the system as the root user either for a series of operations or only for one. As a general rule, you should assume so-called root privileges only when absolutely necessary and for as short a time as necessary.

In order to temporarily sign on as another user, you can use the **su** command, as in the following examples:

```
1

2

3

$ su anotheruser

$ su root

$ su
```

You will be prompted for the password of the user whose name was specified. If you do not give a user name (as in the third example), root will be assumed.

The superuser session ends when you type **exit** in the shell.

If you use a naked dash as an option, as in:

```
1

$ su -
```

there is a subtle difference; you are signed into a login shell, which means your working directory will shift to the home directory of the account you are logging into, paths will change, etc.

If you use the **-c** option as in:

```
1

2

$ su root -c ls

$ su - root -c ls
```

you execute only one command, in this case **ls**. In the first case, this will be in the current working directory, in the second, in the root's home directory.

Suppose a normal user needs temporary root privilege to execute a command, say to put a file in a directory that requires root privilege. You can do that with the **su** command, but there is one obvious drawback; the user needs to have the root password in order to do this.

Once you have made the root password known to a normal user, you have abandoned all notions of security. While this may be an acceptable day-to-day method on a system on which you are the only normal user and you are trying to respect good system hygiene by avoiding privilege escalation except when absolutely necessary, there is a better method involving the **sudo** command.

To use **sudo** you merely have to do:

```
1

2

$ sudo -u anotheruser command

$ sudo command
```

where in the second form, the implicit user is root. While this resembles doing **su -c**, it is quite different in that the user's own password is required; **su** requires the other user's password (often that of root).

However, this will not work unless the superuser has already updated **/etc/sudoers** to grant you permission to use the **sudo** command. Furthermore, it is possible to limit exactly which subset of commands a particular user or group has access to, and to permit usage with a password prompt.

It is not recommended to edit this file in a normal manner. You should use either the **visudo** or **sudoedit** programs (as root) instead, as in:

```
1
2
# /usr/sbin/visudo
# /usr/sbin/sudoedit /etc/sudoers
```

because they check the resulting edited file to make sure it has no errors in it before exiting. Otherwise, you can wind up in a difficult-to-fix situation, especially on Linux distributions such as Ubuntu, which hide the root account and rely heavily on the use of **sudo**.

Rather than discussing the details of how to do such fine tuning, we recommend you read this file as it is self-documenting, or do **man sudoers**.

The simplest line you could add to this file would be (for user **student**):

```
1
student ALL=(ALL) ALL
```

which would let the user have all normal root privileges.

On all recent Linux distributions, you should not modify the file **/etc/sudoers**. Instead, there is a sub-directory **/etc/sudoers.d** in which you can create individual files for individual users.

Thus, you can simply make a short file in **/etc/sudoers.d** containing the above line, and then give it proper permissions as in:

```
1
$ chmod 440 /etc/sudoers.d/student
```

Note that some Linux distributions may require **chmod 400 /etc/sudoers.d/student**.

If a file named **/tmp/rootfile** is owned by root, the command:

```
1

$ sudo echo hello > /tmp/rootfile
```

will fail due to permission problems.

The proper way to do this would be:

```
1

$ sudo bash -c "echo hello > /tmp/rootfile"
```

Do you see why?

Some Linux distributions, notably Ubuntu, do not work with root user accounts in the traditional UNIX fashion.

Instead, there appears to be only a normal user account, and the same password is used to log into the system as a normal user, and to use **sudo**. In fact, there is no direct **su** command. However, the equivalent is easily accomplished through the command: **sudo su**.

To some UNIX traditionalists, this method of dealing with the root account is a bad idea, and some attribute its use to an attempt to make Windows users more comfortable when they move to Linux.