Week 05

Users, Groups and Permissions. and more git.

Melvyn Ian Drag

February 19, 2020

Abstract

Users, groups, permissions, and a teensy bit more about git and github to make your life easier.

1 Intro

On the one hand we are going to learn many things today. On the other, just a couple of things. The main idea:

nix is a multiuser operating system. *nix* can bunch users into groups. Based on who the user is and what what groups he belongs to , *nix* can change the ability of the user to do things on the machine, like install software, access certain files, enter certain folders, etc..

Now we will look at the details.

2 Users

Linux/Unix is a multiuser operating system. This may not seem revolutionary, but when Unix came out the concept of a multiuser machine was a big deal. Windows is also multiuser and macs run a verion of BSD (like linux) so this may not impress you. I'm going to show you how to add/remove and manage users now.

There is even discussion in modern operating systems of being single user! `https://discuss.haiku-os.org/t/suggestion-we-remain-single-user-read-on/2031`. Look at the cook haiku os!

You have already seen that when logged into a machine you login as a user - *whoami* will tell you your username. You have a home directory for storing your personal files at */home/$(whoami)*. Users can be added, removed, and modified in your system.

2.1 Adding Users

You add users with the *adduser* command. Try it to add a user to your machine!

```
user@machine$ sudo adduser newusername
# Then answer the following questions.
# The only two essential questions are the password questions, you can just hit <enter> to go through the
    rest.
```

To see that the user created, look under '/home'. You will now have a home directory for this user.

```
user@machine$ ls /home
you thenewuseryouadded
```

You don't have permission to touch edit this user's files. The permissions you have depends on the way your system is configured! For example, on this system you can look in the user's home directory.

```
user@machine$ touch /home/thenewuseryouadded/file.txt
#error
```

but

```
user@machine$ ls /home/thenewuseryouadded
#No error message
```

And look what vim says if you try to :wq a file there:

```
user@machine$ vim /home/thenewuseryouadded/file.txt
#no problem. Now add some text and try to :wq. You will see
# something to the effect of "readonly is set"
# exit with :q!
```

To switch to using the user you created, you can user the 'substitute user' command 'su'.

```
user@machine$ sudo su - thenewuseryouadded
thenewuseryouadded@machine$ whoami
thenewuseryouadded
thenewuseryouadded@machine$ exit
user@machine$
```

Now notice that if you switch to that user using 'su' you can edit files there.

```
$ sudo su -thenewuseryouadded
thenewuseryouadded@machine$ pwd
# wherever
thenewuseryouadded@machine$ cd
# now you are in ~
# for this user, ~ means /home/thenewuseryouadded
# remember that ~ means the home directory for the currently logged in user.
thenewuseryouadded@machine$ exit
$cd
# now I'm back at my home dir.
```

2.2 useradd

```
man useradd
```

Note that there is also a useradd command. This is a lowlevel command that creates a user. But this command should be avoided. More details can be found in the man page.

2.3 Deleting Users

To delete a user you use the deluser command. This can only be run as a superuser.

```
user@machine$ sudo deluser thenewuseryouadded
```

You may want to remove their home directory as well in one fell swoop. If you run the command above, you'll have to

```
user@machine$ sudo rm -r /home/thenewuseryouadded
```

To do everything at once:

```
user@machine$ sudo deluser --remove-home thenewuseryouadded
```

There are other options for deluser, you can see them all with 'man deluser'.

2.4 userdel

Note that there is also a userdel command. this command is lowlevel and should be avoided unless you know exactly why to use it. Read the man page.

2.5 The root user and the sudo command

There is a special user in Linux/Unix called root / the superuser. The superuser is all powerful on the machine and can do anything he wants. You can delete whatever files you want, install software, modify system configuration settings, tamper with the operating system - anything. As such, it is important to limit access to this user profile. If you are a sysadmin at a company you will have root access to the company machines. Other employees typically do not, to limit the chances that non professionals will ruin the software on the machine. For example, go on the internet on these njcu machine and try to download and install a program. It will ask for administrator access and prohibit you from installing software - it is the same on Linux.

There are a few ways to gain root access to your machine.

You can run:

1. ‘ sudo su -‘ to change your user to root. 2. You can run an individual command with the ‘sudo‘ prefix. e.g. ‘sudo apt-get install somesoftware‘.

Not everyone is granted root access. Log in to the user account of the new user your created.

```
user@machine$ sudo su - thenewuseryouadded
thenewuseryouadded@machine$ sudo apt-get install software
#Asks for password
#You enter password
thenewuseryouadded is not in the sudoers file. This incident will be reported.
```

Whereas you do not get this error with the user given to you by default on the cloud machine. The new user is not a privileged user.

There are several ways to make a user a privileged user. One way is to run the following command from the account of a privileged user:

```
user@machine$ sudo adduser thenewuseryouadded sudo
```

This adds the new user to the sudo group. More about groups later. Sorry, there is no perfect order to teach all these concepts. So keep in mind that there is a sudo command and a sudo group. I’ll tell you what a group is in a minute.

2.6 Changing passwords

You can change a user’s password. To change your own password, run ‘passwd‘ and follow the prompts.

To change any user’s password, type ‘passwd USER‘ from a privileged setting. Then follow the prompts.

3 NO TIME TO DO THIS IN CLASS - Files that are property of the superuser are not safe from hackers!!!

You’ve seen that some things can only be done by the root user, right? Only the root user / a privileged user can

- install software
- change passwords
- add users
- delete users
- and more

Here is an interesting thing about Linux that I think you might think is cool. The root user is the same on all Linux machines! So if you take a harddrive from your computer and plug it into another computer as an external harddrive, the root user of that computer will be able to see all of your root user’s files (probably) . Every Linux machine I’ve seen interprets the root user to be the same entity. This deals with the way the operating system handles user data and that’s kind of technical. Today’s lecture is already quite technical (but we’re just presenting the concepts of adding/deleting users) so I’ll spare you the details of that until we’ve had time to experiment with adding/deleting users.

To protect you credit card info, personal photos, documents, etc. you need to encrypt your data. Encryption of files and disks is a subject of a later lecture.

4 Exercise

1. Add a user to your machine
2. sudo su - username
3. Change the user's password
4. sudo su - username again using new password
5. Delete the user and the user's homedir.
6. Verify that the user's homedir is gone
7. Try to sudo su - username and verify that you cannot because you already deleted the user.

5 Groups

Users can be binned together into groups to allow common security measures, privileges, etc. to be applied to all the users in a particular group. This is a pretty logical thing to do - for example, if I were to configure a server for my students to logon to (log on to? logonto?), I would add all my students to a group called 'students' and then I would restrict that group in certain ways that wouldn't allow you all to break the machine.

5.1 Adding Groups

Create a group

```
user@machine$ sudo groupadd njcu
```

To verify that the group was added, lets count the available groups on our system before and after the add. You can list all the groups on your machine using the 'getent group' command.

```
user@machine$ sudo getent group | wc -l
N
user@machine$ sudo groupadd njcu2
user@machine$ sudo getent group | wc -l
N+1
```

5.2 Deleting Groups

Just as easy as removing a user.

```
user@machine$ sudo groupdel njcu
```

5.3 Adding a user to a group

Now things get interesting.

```
user@machine$ sudo usermod -a -G GROUP USER
```

or use

```
user@machine$ sudo adduser USER GROUP
```

as we did before with adding the new user to the sudo group.

5.4 Deleting a user from a group

You can delete a user from a group just as you can add a user to a group.

```
user@machine$ sudo deluser USER GROUP
```

5.5 An example

In this example I'm going to create a group called NJCU, create a user called 'tux' (that's the Linux mascot), add tux to the NJCU group, and then verify that I added tux to the NJCU group. Then I'll do some housekeeping to clean up.

```
sudo groupadd njcu
sudo adduser tux
sudo usermod -a -G njcu tux
sudo su - tux
groups
exit
sudo deluser --remove-home tux
sudo groupdel njcu
```

5.6 Exercise

1. create a user
2. login to the user acct
3. type 'groups' to see what groups the user is a member of
4. try to 'sudo apt-get install build-essential' (it should fail)
5. exit user
6. add the user to the sudo group using *usermod -a -G sudo $USERNAME*
7. login to that user account using *sudo su - $USERNAME*
8. type 'groups' to verify the user is a sudoer now
9. try to install software with 'sudo apt-get install build-essential'.
10. exit and delete the user

In this example I had you install a package called 'build-essential'. This package has some useful compiler stuff for writing software. That's why it's called build-essential! This is relevant yet, we'll write software another day in this class, for now I'm just using it as an example. Only sudoers can install software.

6 Summary

The nixes are multiuser operating systems. Not all operating systems in the world are multiuser. The nixes have a concept of 'groups'. Not all operating systems in the world use groups. The most interesting group you know about so far is the 'sudo' group. We'll learn about why you would want to create a group in the next section. For now you just need to know that users can be pooled together in groups if you want the users to have similar rights while operating the computer.

7 Permissions

All linux files have permissions, an owner, and a group. You can see these characteristics when you type 'ls -l'.

```
user@machine$ touch sampleFile.txt
user@machine$ ls -l
...
-rw-rw-r-- 1 melvyn melvyn 0 Feb 4 07:30 sampleFile.txt
...
```

The owner is 'melvyn', the group is 'melvyn' and the permissions are 'rw-rw-r–'. There is a preceding dash, we'll discuss that a bit later, it has very fascinating properties, but at this point in the class we aren't ready to appreciate what it does yet.

Linux/Unix do not support editing permissions for files on a per user basis. Instead, the permissions can be set for the owner, the group, and everyone else. In the permission bits for sampleFile.txt we see 'rw-rw-r–'. This means that the file owner can read and write ('rw-.....'), the other group members can read and write ('...rw-...') and everyone else can only read ('.....r-0').

7.1 chmod

The chmod command is for changing permissions. Linux files have permission bits to specify whether they are readable, writable or executable. There are two ways to run the command, you can pass it numeric arguments or you can pass it character arguments. For the numeric syntax, you need to remember that

```
2^0 = 1 <-> executable
2^1 = 2 <-> write
2^2 = 4 <-> read
```

You can remember which one is which by remembering that r is to the left in the permission triplet and 2 squared is to the left in the binary representation of the number. Not sure if this helps, you can think about it to find a memorization technique that works for you. I was never taught such a technique and devised this one on my own.

To illustrate:

```
4   2    1
r   w    x
```

So to change the permissions of a file to only readable for all users we say

```
user@machine$ chmod 444 sampleFile.txt
```

To change permissions on the file to read and writable for the owner only, we run

```
user@mchine$ chmod 644 sampleFile.txt
```

To make the file read and writable by the owner, but deny everyone else all rights, we run

```
user@machine$ chmod 600 sampleFile.txt
```

To verify that no one but the owner has rights to this file you can log in as the other user you created earlier and try to 'cat' the file now.

```
user@machine$ sudo su - thenewuseryouadded
thenewuseryouadded@machine$ cat sampleFile.txt
blah blah Permission Denied: blah blah
```

7.2 Exercise

Create a dummy directory. Create files in it with all different permissions.

7.3 Mnemonix syntax

The mnemonic syntax for chmod uses characters instead of these permission numbers built of sums of 1s, 2s and 4s.

There is another 10 minutes that can be spent discussing this syntax, but I'll just show you really quick. 'u' means 'user', 'g' means 'group', and 'o' means 'other' in what follows.

```
user@machine$ chmod u+x sampleText.txt
user@machine$ chmod g=rw sampleText.txt
user@machine$ ls -l
...
-rwxrw---- other_information_here sampleFile.txt
```

'chmod' also has a recursive option that can be used to recursively apply permissions.

```
user@machine$ mkdir a/b
user@machine$ touch a/foo.txt
user@machine$ touch a/b/bar.txt
user@machine$ chmod -R 700 a
```

Then you can have a look inside a at the files and directories there.

We haven't written executable scripts yet, we'll take a minute to do that now.

```
user@machine$ vim firstScript.sh
user@machine$ cat firstScript.sh
#!/bin/bash

echo "hello world"
```

The first line is called a shabang line, and that specifies which interpreter to use when executing the script. We want to run a bash script. Then we can make it executable and run it like this:

```
user@machine$ chmod u+x firstScript.sh
user@machine$ ./firstScript.sh
hello world
```

7.4 Permissions and directories

It is pretty clear what the permissions mean for a file. read means you can read it, write means you can write, execute means you can run it as an executable program. What do permissions mean for a directory?

1. read means the contents of the directory can be listed
2. write means files can be created, destroyed, and renamed in the directory
3. execute means the directory can be entered.

```
user@machine$ mkdir a
user@machine$ chmod 700 a
user@machine$ chmod u-r a
user@machine$ ls a
Permission Denied
user@machine$ chmod u=rx a
user@machine$ cd a
user@machine$ touch foo.txt
Permission Denied
user@machine$ cd ..
user@machine$ chmod 600 a
user@machine$ cd a
Permission Denied
```

7.5 chown

The chown command is for changing ownership.

You run

```
user@machine$ chown USER FILE
```

If time, do an exercise here where we set permission to 600, then try to read. Then change owner, and try to read.

7.6 chgrp

Like chown, you can change the group.

```
chgrp GROUP FILE
```

Try it in class with a file.
Use 'ls -l' to inspect the file before and after the change.

7.7 TODO!!

MAKE SURE TO ADD SOME EXAM QUESTIONS ABOUT HOW TO SET FILE/DIRECTORY PERMISSIONS/OWNERSHIP/GROUPMEMBERSHIP/ETC FOR CERTAIN USE CASES.

E.G. EXAM QUESTION: SHOW TWO COMMANDS TO SET PERMISSIONS ON FILE X.SH SO THAT THE OWNER CAN RWX, THE OTHER GROUP MEMBERS CAN RW, AND ALL OTHER SYSTEM USERS CAN ONLY R?

E.G. EXAM QUESTION2: WHY CAN'T /HOME/USER/FILE.SH BE RUN BY USER2? LS -L /HOME -> RW——- /HOM/USER LS -L /HOME/USER RWXRWXRWX /HOME/USER/FILE.SH ETC.:

8 A Few More Command Line Tools for Your Pleasure

A couple more useful commands:

- date
- watch
- alias

8.1 watch & date

```
melvyn@thinkpad$ watch -n1 date
```

8.2 alias

You use the alias command to change the names of system commands, implement system commands, etc.

For example

```
touch filea
mv filea fileb
move fileb filec
# error
alias move=mv
move fileb filec
# now fileb is called filec
```

8.3 .bashrc, /etc/profile.rc

There are many bash profile configuration files on your machine we need to mention now.

These files save custom system configurations. There are several of these files on you system, and the availability of these files may vary from system to system. We'll focus on the .bashrc file, which will work for 99% of your usage needs.

If you change a user's bashrc file, you'll be able to change the system configuration for that user. We can alias "mv" to "move" so that it persists every time we log on to our system.

Open a new subshell after aliasing move as you did above and try it again. Notice that the alias hasn't persisted to the new subshell - you can't use the alias, the system doesnt know about it.

```
alias move=mv
touch a
move a b
# works
bash
move b a
# error
```

now, that alias command will only persist until we start a new shell. What if we want a more persistent change? The folks in my Java class will know that we've already done this - we had to create a file called .bashrc and modify the PATH variable so that our computer would always know where the java compiler (javac) and the java virtual machine (java) were.

```
# start a new terminal
user@machine$ move file1 file2
# error
# append 'alias move=mv' to the end of the bashrc file.
user@machine$ source ~/.bashrc
user@machine$ move file1 file2
# works!
# close the terminal and start a new one
user@machine$ move file2 file2
# works still. Because we modded our bashrc file.
```

9 Exercise

Follow steps carefully!!

9.1 Step 1

Add this line to your .bashrc file. make sure no typos or it won't work.

```
alias rm="echo 'I refuse to delete: '"
```

9.2 Step 2

```
$touch b.txt
$ls
# b.txt is there
$rm b.txt
$ls
# b.txt is not there
$ source ~/.bashrc
$ touch b.txt
$ rm b.txt
I refuse to delete: b.txt
$ ls
# b.txt is there still
```

9.3 Step 3

Remove the line from .bashrc

9.4 Step 4

```
$ rm b.txt
I refuse to delete b.txt
$ source ~/.bashrc
$ rm b.txt
I refuse to delete b.txt
$ bash
$ rm b.txt
$ ls
# b is gone
```

9.5 Step 5

quick class discussion, then move on to how to sync a fork.

10 Syncing Fork

Last week we didn't get to cover how to sync your fork when it gets out of date. Do that at the end of class, because it could be a time sink. See the sections about syncing forks and then also mention Gitlab, bitbucket, github from last week's notes.

10.1 Uh oh my fork is out of sync

When you fork the class repo, you get a snapshot of it at a point in time. You are adding to this old snapshot and making pull requests that, for very technical reasons which we are all too tired to think about at the moment, everything works nicely. I've forcing you to submit your homework in a very specific way, with a very particular directory structure - this is so that everything works nice and easy for you and for me.

But as I was saying, you forked off old code. If you fork today, you have this weeks code. But between now and next week I'm going to update class materials for next week. You might not be able ot submit next weeks homework because of these changes I'm going to make. For this, you need to periodically sync your fork with my repo.

To do this look in the Images/SyncYourFork directory.

You need to make sure you have an 'upstream' remote, then 'fetch' it, then it will merge with yours. Then you are good to go.

Do this with class. Have them all fork, then make a change to the repo, and have them sync their forks with the repo. This should take some time. Minimize discussion of branches for now.

11 Oh No Lecture finished early

If this happens, there is some material in the Readme.md file in this directory to go over, a bit more about if statements, etc. Best might be to go over the syncing fork business over and over and over and over though.

Maybe have students practice pull requests

Do a bit more grep stuff.