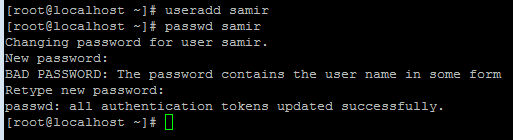
**Class 5**

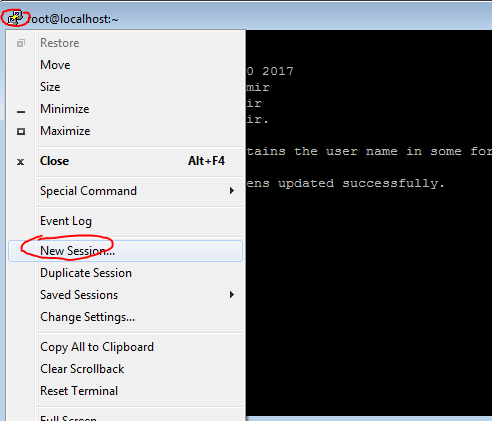
**Users and permissions**

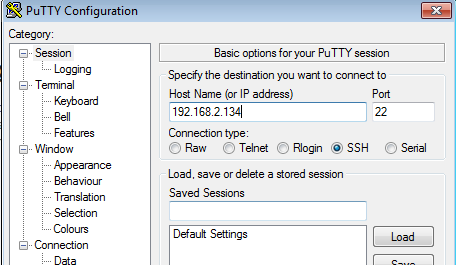
**useradd** **username** - adds a user

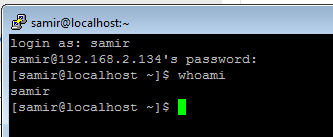
**passwd username** - changes the password of a user

In the following screenshots, you will see that the user samir was added, the password was set, and a new Putty session was used to connect as that user. The whoami command showed the user samir was logged in.



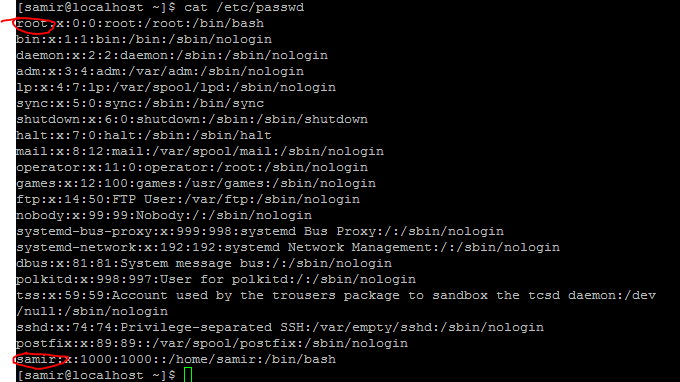






The **whoami** command showed that I am logged in as that user.

The /etc/passwd file contains a list of users. I can use the cat command (cat is used to read a file’s content, more on this later):



I circled root and samir

Users

A UNIX system serves many users. Users are an abstraction that denotes a logical entity for assignment of ownership and operation privileges over the system. A user may correspond to a real-world person, but also a type of system operation. For example, a system could have user 'nick' that corresponds to the server admin, but the system could also have user 'www' which corresponds to the privileges necessary to operate the local webserver. UNIX does not care about what the user means in terms of people. It just knows what belongs to any given user and what each user is allowed to do with any given thing (file, program, device, etc) on the system. UNIX identifies each user by a User ID (UID) and the username (or login) such as 'nick' and 'www' is just an alias to the UID that makes humans more comfortable.

Groups

Users can be organized in groups. A user may belong to one or more groups of users. The concept of groups serves the purpose of assigning sets of privileges for a given resource and sharing them among many users that need to have them. (perhaps because they are all members of a project working team and they all need access to some common project files). For example, a system’s users named “nick” and “www” both belong to the group “web”. This way, they can have some shared privileges over the files for this site. User 'nick' needs them to edit the site, and user 'www' needs them to manage the webserver that will be publishing the site. Groups also have a GUID (Group user ID).

In the /etc/passwd example above, the samir user has the UID 1000 and GUID 1000. Notice how the root user has the 0 UID and 0 GUID. The shutdown user (as well as others) also has the 0 GUID – these are part of the operating system and they share the permission of root to do certain administrative tasks.

Others

The third party in permissions is the “others” group which means “everyone else” or public Internet users or anyone that is not in a given user or group.

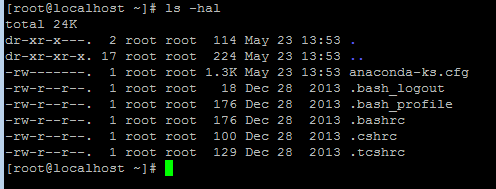
There are three types of access permissions on Linux: read, write, and execute. These permissions are defined separately for the file's owner, group, and all other users.

**Read permission.** On a regular file, the read permission bit means the file can be opened and read. On a directory, the read permission means you can list the contents of the directory.

**Write permission.** On a regular file, this means you can modify the file, aka write new data to the file. In the case of a directory, the write permission means you can add, remove, and rename files in the directory. This means that if a file has the write permission bit, you are allowed to modify the file's contents, but you are allowed to rename or delete the file only if the permissions of the file's directory allow you to do so.

**Execute permission.** In the case of a regular file, this means you can execute the file as a program or a shell script. On a directory, the execute permission (also called the "search bit") allows you to access files in the directory and enter it, with the cd command, for example. However, note that although the execute bit lets you enter the directory, you're not allowed to list its contents, unless you also have the read permissions to that directory.

To check the permissions for all files & directories in a given folder, type **ls –hal** (used to list files). Here is an example (run this with your root user):



The first group consists of only one character, and it shows the file's type. For example, **d** means a directory and **-** means a normal file and **l** is used for symbolic link.

The meaning of r, w, x, -:

**r** = read permission = 4  
**w** = write permission = 2  
**x** = execute permission = 1  
**-** = no permission

7 would mean read, write, execute (4+2+1)

6 would mean read & write (4+2)

5 would mean read, execute (4+1)

4 would mean read only (4)

3 would mean write & execute only (2+1)

2 would mean write only (2)

1 would mean execute only (1)

Access rights on directories:

Example of permission for “file6”:

-rw-r--r-- 1 root root 583 Nov 1 00:45 file6

rw- = the user

r-- = the group

r-- = all others

In this case, the permission for this file is:

644

6 (4 for read 2 for w) 4 (4 for read) 4 (4 for read)

If a file has rwxrwxrwx means that everyone can read, write, and execute

This would also be 777 permission. Note that you would never want to have 777 permission on any file or directory, **\*EVER\***  do not listen to people on forums that advise to have 777 permission to fix an issue – this opens a big security hole (a public Internet user could replace your files with malware).

Every file in UNIX has an owner user and an owner group. So, for any file in the system, user 'nick' may have one of the following ownership relations:

nick owns the file, i.e. the file's owner is 'nick'.

nick is a member of the group that owns the file, i.e. the file's owner group is 'www'.

nick is neither the owner, nor belonging to the group that owns the file

Also remember that everything in UNIX is a file, **EVERYTHING**. So you can really customize and control all permissions by just manipulating file & directory permissions.

**chmod -** this command is used to change the permission of a file or directory

examples

chmod a+x filename

would add an execute permission to all (a = all, so it would add +x permission to the user, group, and others)

chmod g+w filename

would add a write permission to the group

chmod o+r filename

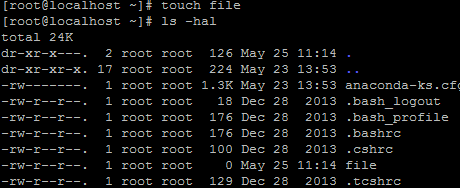
would add read permission to the other member (meaning the Internet)

chmod u+x filename

would add permission to the owner of the file

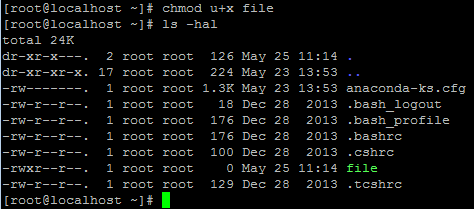
Here are some screen-shots to show this in practice:

I used the **touch file** command to create a blank file named file and then I used ls -hal to show the permission on it. Since I created the file using the root user, the permissions are -rw-r—r—(the first block is for root, the owner, has read and write, the second block is group and it’s just read, and the third block is for others and it’s just read).



If I wanted to add the execute permission on the file (+x) for the owner (root) I could use:

chmod u+x file

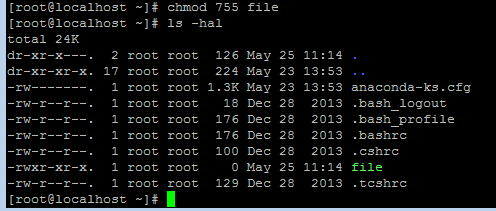


Before learning an easier way, go ahead and practice adding permissions and removing permissions.

Numbers can also be combined with chmod. This is a preferred method by most Linux users and it is quick to learn.

To change a file from 777 ownership to 755, you would just use:

chmod 755 filename

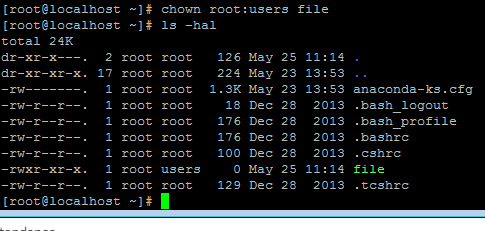


As you can see, the first block shows rwx (4 + 2 + 1 = 7) second block is rx (4 + 1) and third is rx (4 + 1).

**chown** – this command allows to set the name of the user/group for a file or directory (note that the “Others” is not part of the two, the “others” permission is dictated by the permission itself)

chown samir:www directoryname

would put samir as the user and www as the group



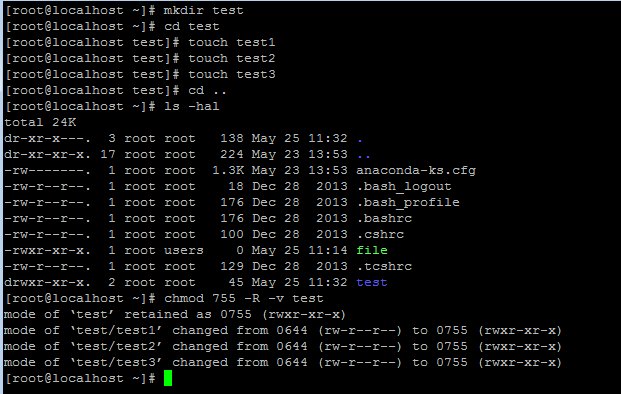
In this example, I put the file in the “users” group.

**-R –** the –R flag can be combined with chmod or chown to perform a **recursive** action meaning that everything below that directory (and sub directories & files) will also take the permission.

**-v –** this stands for verbose and it is used to see “what happens” as the permission or ownership change is done.

For example I could do:

chmod 755 –R / (this would set 755 permission on all files & folders on my server because / is the very top directory in Linux).



In this example I made a test directory, moved to that directory, created 3 files, moved up again one directory, then I did a bulk permission update with 755 to all directories and files in the test directory.

chown samir:www –R foldername (this would set the user as samir and group as www for foldername and every other folder and files inside of foldername)

For example, this could happen when you do it recursively:

$ **chown -Rv samir:www directoryname**

changed ownership of 'directoryname/' to username:www  
changed ownership of 'directoryname/subfile' to username:www  
changed ownership of 'directoryname/subdir/subfile’ to username:www

etc…

To manipulate users and groups (add users to group, create groups, etc) check:

(Look especially into adding groups to specific users from the get-go)

<http://www.cyberciti.biz/faq/howto-linux-add-user-to-group/>

<http://wtuto.com/redhat/ug.html>

To remove a user from a group, example if I want to remove the user **samir** from the **www** group, I would use:

usermod –G samir samir (this sets the group name for the user, so samir is the name for the samir user).

usermod –g 502(or the name such as the www group) samir (this sets the group ID for the user).

Give these commands a try, the more practice the better!