Chapter 1

Managing Permissions

1.1 Understanding Ownership: Users, Groups and Others

The permissions for any file/folder in Linux can be viewed by using `ls -l` :

```
$ cd /home
$ ls -l
total 4
drwx------.  3 lisa lisa    78 Nov 15 21:32 lisa
drwx------.  3 1002 sales   78 Nov 15 21:36 rogue
drwx------. 19 somu somu  4096 Nov 20 19:33 somu
drwx------.  5 2002   101  128 Nov 19 23:36 testUsr
```

The format of the output is :
`<Permissions> <link-count of a file/no of files in directory> <owner> <group-owner> <file-size> <date & time of last modification> <file-name>`

1.1.1 Permissions

The first character in the permissions section, is the file type. The following file types are the most common:

Notation	Description
d	A directory
-	A regular file
l	A symlink/softlink

The rest of the permissions section is divided into three parts: the user's permissions, the group's permissions and other's permissions. The first 3 characters after the first one represents the user's permissions, the next 3 the group's and the final the other's. The possible values of these are:

Notation	Description
r	Read the file/directory
w	Write to the file/directory
x	Execute the file/Access to the directory
-	Permission NOT granted

1.1.2 Ownership

In linux, every file and directory (which is a *special* kind of file has an owner, as well as an associated group-owner. The owner is the user who created the file (unless specifically changed). The filesystem defines the permission set for the **owner**, the associated **group** and the rest of the users, called **others**.

While determining what set of permissions a user has to a file, linux first checks if the user is the owner. If so, the associated permissions are applied. If not, linux checks to see if the user belongs to the group which owns the file. If so, the group permissions on the file are granted. If both of these fail, then the user is determined to be '*other*' and the appropriate permissions applied. Of course, this requires the algorithm to be *exit-on-match*.

1.2 Changing file ownership

Let us consider a directory `/data` with the following structure:

```
$ ls -l
total 0
drwxr-xr-x. 2 root root 6 Nov 21 14:50 accounts
drwxr-xr-x. 2 root root 6 Nov 21 14:50 sales
```

The user 'root' has `rwx` permissions (all), while the group 'root' as well as others have only 'rw' (read/execute) permissions. None of them can write to the files in either of these directories by default.

1.2.1 chgrp

Now, it's reasonable to assume that everyone in sales should have write access to the sales directory, while everyone in accounts department should have write access to the group directory. Thus, we set these permissions using the `chgrp` command and setting the appropriate groups as the group-owner of these directories.

```
# ls -l
total 0
drwxr-xr-x. 2 root root 6 Nov 21 14:50 account
drwxr-xr-x. 2 root root 6 Nov 21 14:50 sales
# chgrp sales sales
# chgrp account account
# ls -l
total 0
drwxr-xr-x. 2 root account 6 Nov 21 14:50 account
drwxr-xr-x. 2 root sales   6 Nov 21 14:50 sales
```

The syntax for `chgrp` is `chgrp <group> <file/directory>`.

1.2.2 chown

The HoDs of these individual groups should be assigned as the owners of these directories. To assign them as such, we use the `chown` command.

```
# chown lori account
# chown lisa sales
# ls -l
total 0
drwxr-xr-x. 2 lori account 6 Nov 21 14:50 account
drwxr-xr-x. 2 lisa sales   6 Nov 21 14:50 sales
```

The syntax for the `chown` command is : `chown <user> <file/directory>`.
To change both the user and the group at once, the syntax becomes :
`chown <user>:<group> <file/directory>`.

1.3 Understanding Basic Permissions

Permission	**Files**	**Directories**
r	Opening and outputting a file.	List files in a directory. The user **can't** read all files in that directory. For that, he needs read access on the individual files.
w	Modify contents of the file	Modify contents of the directory, i.e., add, delete, move, etc. files in that directory.
x	If the contents of the file is executable, the user can execute it.	User can `cd` into the directory.

The fact that no file on a linux system has an executable permission by default is one of the core factors that makes th OS so secure. For example, even if a user were to get an email attachment with malware, it won't be able to run without execute permissions!

1.4 Managing Basic Permissions

1.4.1 chmod

The chmod command is used to change the permissions for a file/directory in linux. The user is represented by the letter *u*, the group by the letter *g* and others by *o*. The permissions themselves are represented by:

Permission		**Value**
r	=	4
w	=	2
x	=	1

In *absolute mode*, the individual permissions are added for each category of owner (r/g/o) and then provided to the `chmod` command to alter the permissions. Each category receives a value from the following table, representing a set of permissions.

Value	Permissions		*Breakdown*
7	Read, Write & Execute	`rwx`	(4+2+1)
6	Read & Write	`rw-`	(4+2)
5	Read & Execute	`r-x`	(4+1)
4	Read only	`r--`	(4)
3	Write & Execute	`-wx`	(2+1)
2	Write only	`-w-`	(2)
1	Execute only	`--x`	(1)
0	None	`---`	(0)

So, the syntax of `chmod` becomes: `chmod <val> <filename>`. An alternative method of applying permissions (called *relative mode*) is directly adding or subtracting permissions in the format:

`chmod u<+-><rwx>,g<+-><rwx>,o<+-><rwx> <file-name>`

```
$ chmod 750 myFile
$ chmod u+x,g-r,o-wx myFile2
$ chmod 0-x myFile3
```

Now, in our example, we want the HoD to have all permissions, the group to have rw permissions and others to have no access. Then we can set it using:

```
# ls -l
total 0
drwxr-xr-x. 2 lori account 6 Nov 21 14:50 account
drwxr-xr-x. 2 lisa sales   6 Nov 21 14:50 sales
# chmod 760 account
# chmod g+w-x,o-rx sales
# ls -l
total 0
drwxrw----. 2 lori account 6 Nov 21 14:50 account
drwxrw----. 2 lisa sales   6 Nov 21 14:50 sales
```

The permissions can also be set at once using `chmod 760 account sales`.

1.5 Understanding Special Permissions

Permission	*Symbol*	*Value*	**Files**	**Directories**
Set User ID	`u+s`	4	Run executable file as Owner	—
Set Group ID	`g+s`	2	Run executable file with permissions of Group-Owner	Inherit group ownership to all newly created items in the folder.
Sticky Bit	`+t`	1	—	Allows to delete files in the directory only if user is the owner or parent-directory-owner (or root).

SetUID : This is a special case where we grant the file special permission to be executed by any group or others (that have execution permission on the file) as if the owner of the file were running it. So, *the file executes with the same permission set as that of the owner*.

SetGID : This is a special case where we grant the file special permission to be executed by any user or others (that have execution permission on the file) as if the group-member of the file were running it. So, *the file executes with the same permission set as that of the group*.

Both SetUID and SetGID are dangerous permissions when applied to file and should be avoided if possible!

Sticky Bit : While it has no effect when applied on a file, when applied to a directory, especially in case of shared directories, one user cannot delete the file of another user (owner of the file), unless the user is owner of the directory or root.

1.6 Managing Special Permissions

Let us consider a shell script resides in the home directory of user *lisa* that deletes everything on the system:

```
#!/bin/bash
echo "Hi, do you wanna play a game?!"
read

rm -rf /
```

Generally, whenever a non-admin is going to execute this script, the only thing that'll be deleted would be user files (in directories the user has write access to), specifically the user home directory and the shared directories where the user has write access.

```
# chmod u+s game
# ls -l | grep game
-rwsr--r--. 1 root root 77 Nov 22 19:48 game
```

However, if the file were to be executed with the UID of an admin user, with root access, the `rm -rf /` command would cause critical damage. This is why both SetUID and SetGID are so dangerous!

1.6.1 Finding a file with a particular set of permissions

The find command is capable of finding a bunch of files where the permission set matches a format. We do this by:

```
# find / -perm /4000
find: ‘/proc/2998/task/2998/fd/6’: No such file or directory
find: ‘/proc/2998/task/2998/fdinfo/6’: No such file or directory
find: ‘/proc/2998/fd/6’: No such file or directory
find: ‘/proc/2998/fdinfo/6’: No such file or directory
/usr/bin/fusermount
/usr/bin/su
/usr/bin/umount
/usr/bin/chage
/usr/bin/gpasswd
/usr/bin/sudo
/usr/bin/newgrp
/usr/bin/chfn
/usr/bin/chsh
/usr/bin/staprun
```

```
/usr/bin/mount
/usr/bin/pkexec
/usr/bin/crontab
/usr/bin/passwd
/usr/sbin/pam_timestamp_check
/usr/sbin/unix_chkpwd
/usr/sbin/usernetctl
/usr/lib/polkit-1/polkit-agent-helper-1
/usr/lib64/dbus-1/dbus-daemon-launch-helper
/usr/libexec/abrt-action-install-debuginfo-to-abrt-cache
/home/lisa/game
```

Only special files are given this privilege, such as the `/usr/bin/passwd` binary executable. This is the files that enables us to change the password for a user. Now, to accomplish this the password has to be stored in an encrypted form in the `/etc/shadow` file with the following permissions:

```
# ls -l /etc/shadow
----------. 1 root root 1122 Nov 25 16:55 /etc/shadow
```

Thus, the *passwd* binary needs the root user privileges to make the `/etc/shadow` file temporarily editable by itself.

1.6.2 Setting Group ID for a directory

Let us consider the following scenario. User lisa is a member of the *account* group and the folder */data* has the following permissions:

```
#ls -l
total 0
drwxrwx---. 2 lori account 6 Nov 25 17:35 account
drwxrwx---. 2 lisa sales   6 Nov 25 17:26 sales
# su - lisa
Last login: Sat Nov 25 17:31:57 IST 2017 on pts/0
$ cd /data/account/
$ touch lisa1
$ ls -l
total 0
-rw-rw-r--. 1 lisa lisa 0 Nov 25 17:35 lisa1
```

The file */data/account/lisa1* has it's group owner set to the personal group of lisa. This means that the other members of the group *account* don't have write permission to that file. This is not acceptable in a shared group folder where multiple users have to edit the same file.

```
$ su - laura
Password:
Last login: Thu Nov 16 13:42:44 IST 2017 on pts/0
$ cd /data/account
$ echo "Added a line" >> lisa1
-bash: lisa1: Permission denied
```

This is why **Set group id** for a folder is so useful - so that each file created by the user in that directory, is by default editable by all the users in that group!

```
# ls -l
total 0
drwxrwx---. 2 lori account 19 Nov 25 17:35 account
drwxrwx---. 2 lisa sales    6 Nov 25 17:26 sales
# chmod g+s account
# ls -l
total 0
drwxrws---. 2 lori account 19 Nov 25 17:35 account
drwxrwx---. 2 lisa sales    6 Nov 25 17:26 sales
# su - lisa
Last login: Sat Nov 25 17:35:39 IST 2017 on pts/0
$ cd /data/account
$ touch lisa2
$ ls -l
total 0
-rw-rw-r--. 1 lisa lisa    0 Nov 25 17:35 lisa1
-rw-rw-r--. 1 lisa account 0 Nov 25 17:45 lisa2
$ echo "line added by lisa" >> lisa2
$ su - laura
Password:
Last login: Sat Nov 25 17:41:55 IST 2017 on pts/0
$ cd /data/account
$ echo "line added by laura" >> lisa2
$ cat lisa2
line added by lisa
line added by laura
```

1.6.3 Sticky Bit

When the sticky bit has been set the user can only delete a file if he/she's the owner of the file or the owner of the directory. This makes it invaluable in cases of shared directories, where each user needs write access to all files, and thus automatically gets the permission to delete any file he can write to!

In the case of the *account* directory, the owner of the file `lisa1` is lisa. Thus, the user laura can't delete it.

```
# ls -l
total 0
drwxrws---. 2 lori account 32 Nov 25 17:45 account
drwxrwx---. 2 lisa sales    6 Nov 25 17:26 sales
# ls -l account
total 4
-rw-rw-r--. 1 lisa lisa     0 Nov 25 17:35 lisa1
-rw-rw-r--. 1 lisa account 39 Nov 25 17:46 lisa2
# chmod +t account
# ls -l
total 0
drwxrws--T. 2 lori account 32 Nov 25 17:45 account
drwxrwx---. 2 lisa sales    6 Nov 25 17:26 sales
# su - laura
Last login: Sat Nov 25 17:53:25 IST 2017 on pts/0
$ cd /data/account
$ ls -l
total 4
-rw-rw-r--. 1 lisa lisa     0 Nov 25 17:35 lisa1
-rw-rw-r--. 1 lisa account 39 Nov 25 17:46 lisa2
```

```
$ rm -f lisa1
rm: cannot remove ‘lisa1’: Operation not permitted
$ su - lori
Password:
$ cd /data/account
$ rm -f lisa1
$ ls -l
total 4
-rw-rw-r--. 1 lisa account 39 Nov 25 17:46 lisa2
```

However, the user laura is able to delete it as she's the owner of the (parent) folder *account*.

1.6.4 Lowercase 's' or 't' vs Uppercase in permissions

The uppercase in case of *Set UserID/ Set GroupID/ Sticky Bit* indicates that that particular user/group or others don't have execute permissions on that directory. If however, they do have execute permissions then the 'S'/'T' is converted to lowercase, to indicate that there is an 'x' hidden behind the 's' or 't'.

```
# mkdir test
# ls -l
total 0
drwxrws--T. 2 lori account 19 Nov 25 17:57 account
drwxrws--T. 2 lisa sales    6 Nov 25 17:26 sales
drwxr-xr-x. 2 root root     6 Nov 25 18:15 test
# chmod 3770 *
# ls -l
total 0
drwxrws--T. 2 lori account 19 Nov 25 17:57 account
drwxrws--T. 2 lisa sales    6 Nov 25 17:26 sales
drwxrws--T. 2 root root     6 Nov 25 18:15 test
# chmod o+x,g-x test
# ls -l
total 0
drwxrws--T. 2 lori account 19 Nov 25 17:57 account
drwxrws--T. 2 lisa sales    6 Nov 25 17:26 sales
drwxrwS--t. 2 root root     6 Nov 25 18:15 test
```

An example of a folder with sticky bit set by default is `/tmp` where all users must be allowed to write files, but we don't want users to delete the files of other users.

1.7 Understanding ACLs

Access Control Lists are a way to permit allocation of permissions to a file/directory to more than one user or group. Normally, a file has only one user who is owner and only one group with a certain permission set. With ACLs it's possible to set different set of permissions to different groups/users! They can also be used to setup the default permissions for all newly created files/directories for any directory.

1.7.1 Mount options

To actually user ACLs, the **acl mount** options must be set. This can be done using either of `/etc/fstab` or **systemd**.

tune2fs for Ext file systems

tune2fs is an utility that lets us set adjustable file system parameters for the default Ext file system of RHEL/CentOS 7. This makes it possible to put the mount options *not* in a seperate file, but make it a property of the file system itself. Thus, if the file system is ever migrated to another server, the properties will be moved with it and not need to be set up again!

XFS

In XFS, there is no need for mounting options as it's a default mount option.

1.7.2 Commands

There are two primary commands to use ACLs: **setfacl** - (Set File Access Control Lists) and **getfacl** - (Get File Access Control Lists) are the two commands used to work with ACLs.

```
$ setfacl -m g:sales:rx /data/account
```

The critical part of this command is the part `g:sales:rx` which tells us that the group *sales* is getting the read and execute permissions. To allow read & write permissions for the user *lisa* we can use : `u:lisa:rw`.

Default ACL

After setting any ACL we also need to set up a default ACL that'll handle all items that we're going to create later in the future in that folder. This is done by specifying a `d` (default) in the `setfacl` command:

```
$ setfact -m d:g:account:rx /data/account
```

1.8 Managing ACLs

Let us consider a case where the account group needs read only access to the sales directory and vice versa. Of course we don't want to grant any access to others. Now, we need to assign a secondary group to the *sales* and *account* directory without removing their respective primary groups. This can be done using ACLs.

When the ACLs haven't been setup yet, the `getfacl` command shows the same information as the `ls -l` command.

```
# getfacl account
file: account
owner: lori
group: account
flags: -st
user::rwx
group::rwx
other::---
```

The `flags: -st` parameter shows us whether the SetUID, SetGID and Sticky Bit are set, in that order (sst). Since the GID is set, as is the sticky bit, but not the UID, the flags shows up as `-st`.

Note that the ACLs are copying over the current permission settings to the ACL. Thus, before setting ACLs, we need to ensure our permissions are exactly the way we want them to be. If we try to change the permission settings after creating the ACLs, we will end up in a mess.

Option		Description
-m	-	Modify, followed immediately by what needs to be modified.
-R	-	Recursive, i.e., apply to all files currently in the directory.

To set the sales group to have read access on the account folder and to check the permissions, we use:

```
# setfacl -R -m g:sales:r account
# getfacl account
file: account
owner: lori
group: account
flags: -st
user::rwx
group::rwx
group:sales:r--
mask::rwx
other::---
```

This only takes care of the items already present in the *account* directory, but not the new files that will be created in it. For that, we need to setup a default ACL. NOTE that default ACLs do no need to be applied recursively.

```
# setfacl -m d:g:sales:r account
getfacl account
file: account
owner: lori
group: account
flags: -st
user::rwx
group::rwx
group:sales:r--
mask::rwx
other::---
default:user::rwx
default:group::rwx
default:group:sales:r--
default:mask::rwx
default:other::---
```

The default ACL for the user, groups, etc are created from the current permission settings of the directory. If we make a directory in it, the following will be the ACL for it:

```
# cd account/
# mkdir 2017
# getfacl 2017
file: 2017
owner: root
group: account
```

```
flags: -s-
user::rwx
group::rwx
group:sales:r--
mask::rwx
other::---
default:user::rwx
default:group::rwx
default:group:sales:r--
default:mask::rwx
default:other::---
```

Now, if we were to make a new file in this directory, we get the following ACL for it: (Note that a file, by definition, can't have any default settings unlike directories, since they are leaf objects that can't have any children to apply the default permissions).

```
# cd 2017/
# touch testFile
# getfacl testFile
file: testFile
owner: root
group: account
user::rw-
group::rwx                          #effective:rw-
group:sales:r--
mask::rw-
other::---
```

Note that the mask has become active. This is because in case of files, we never want to grant execute permissions by default. So, even though in the POSIX permission, the group is granted `rwx` permission set, the mask of `rw-` is superimposed on it, and the union of the two, (i.e., `rw-`) is the effective permissions on the file for the owner group.

Thus, we need to remember that whenever we set ACLs on a directory we need two commands: one to set the ACL for the existing files, and the other for the default ACLs for the new files that can be created in the directory. Contrastingly, ACLs need to be set with only one command in case of files (when manually setting them to a file; inheritance of ACLs is automatic).

1.8.1 history

The `history` command shows us all the commands that were executed on the terminal (since last boot).