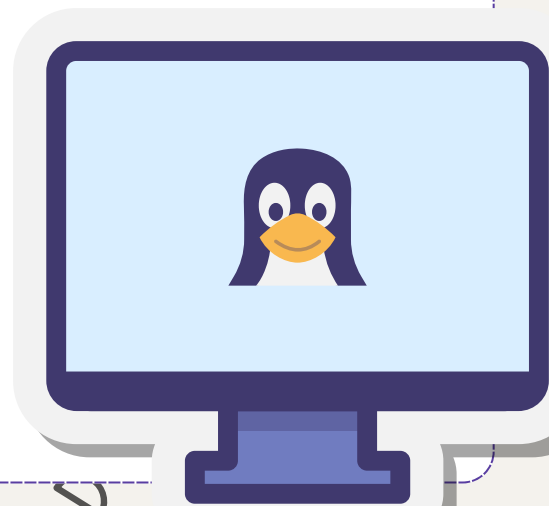


# ACCESS CONTROL LIST IN LINUX

*BY - Gauri Yadav*



# LIST OF TOPICS

01

WHAT ARE ACLS?



02

TYPES OF ACL MANAGENT  
COMMANDS ?



03

HOW TO WORK WITH GETFACL



04

HOW TO WORK WITH SETFACL?



05

TYPES OF ACLS?



06

HOW TO IDENTIFY,APPLY  
AND ERASE ACESS ACLS



# LIST OF TOPICS

07

HOW TO IDENTIFY, APPLY AND  
ERASE DEFAULT ACLS



# What are ACLs?

- The Access Control Lists (ACLs) provide an extended set of permissions that can be applied on files and directories. These permissions are in addition to the standard ugo/rwx permissions and the setuid, setgid, and sticky bit settings. The ACLs define permissions for named users and named groups using either octal or symbolic representation of permission allocation. The named users may or may not be part of the same group. ACLs are configured and treated the same way on both files and directories.
- ACLs are categorized into two groups based on their type and are referred to as access ACLs and default ACLs. Access ACLs are set on individual files and directories, whereas default ACLs can only be applied at the directory level with files and subdirectories inheriting them automatically. The directory to be applied the default ACLs needs to have the execute bit set at the public level

# Types of ACL management commands ?

Types of ACL management commands ?

getfacl

setfacl

# How to work with GETFACL?

- The getfacl The getfacl command has several options to see the output as desired; however, it reveals all necessary information without furnishing any flags with it. The example below creates an empty file ac/file1 in /tmp and then displays the ACLs on it:

```
[root@localhost ~]# cd /tmp
[root@localhost tmp]# touch ac/file1
[root@localhost tmp]# getfacl ac/file1
# file: ac/file1
# owner: root
# group: root
user::rw-
group::r--
other::r--

[root@localhost tmp]#
```

- The output returns the names of the file, owner, and owning group. It then exhibits the existing permissions placed on the file read and write for the owner, and read-only for everyone else. Notice the pair of colon character (:) in the permission rows, There is a space between them where a named user, a named group, or their corresponding UID or GID is inserted when extended permissions are set.

# How to work with SETFACL?

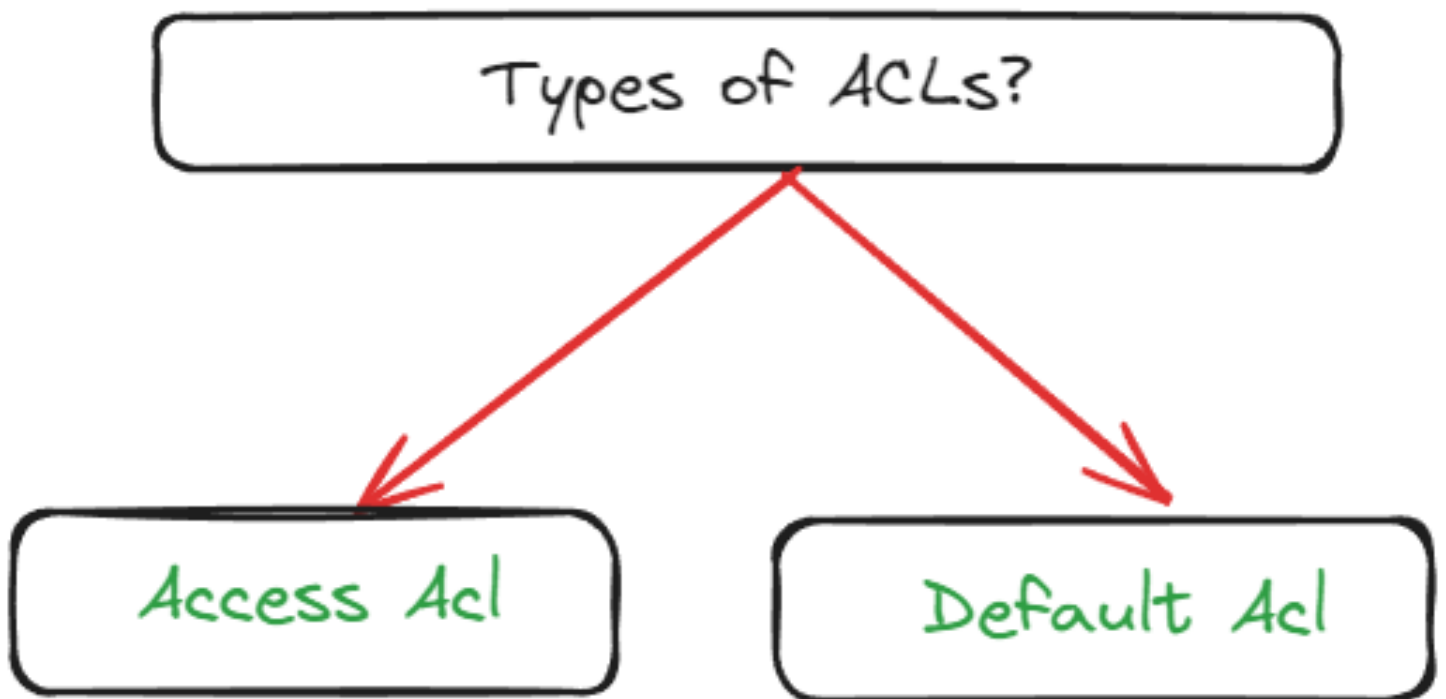
The `setfacl` command in Linux is used to set file access control lists (ACLs). ACLs provide a more fine-grained permission mechanism compared to the traditional Unix file permission system. While the standard file permissions allow you to set different permissions for the file owner, group, and others (everyone else), ACLs enable you to specify permissions for any user or group. Here's a brief overview of how to use the `setfacl` command:

```
[root@localhost ~]# setfacl -m u:username:permission file name
```

- `-m` stands for modify. It's used to modify the ACL of a file or directory.
- `u:username:permissions` specifies the user and the permissions you want to set. For groups, use `g:groupname:permissions` instead.
- `filename` is the name of the file or directory you're modifying the ACL for.



# Types of ACLs?



# How to identify, apply and erase access acls

Lets understand this by a real time example

Scenario: you will create a file acluser1 as user1 in /tmp and check to see if there are any ACL settings on the file. You will apply access ACLs on the file for a named user, user100, for read and write access. You will observe the change in the mask value. You will then add another named user, user200, to the file for full permissions. You will observe the update in the mask value. You will delete the settings for user200 and then the rest of the access ACLs from the file.

**Step 1 :** *Switch or log in as user1 and create a file acluser1 in /tmp:*

```
[root@localhost ~]# su - user1
[user1@localhost ~]$ cd /tmp
[user1@localhost tmp]$ touch acluser1
[user1@localhost tmp]$
```

**Step 2 :** *Use the ls and getfacl commands and check for the existence of any ACL entries:*

```
[user1@localhost tmp]$ getfacl acluser1
# file: acluser1
# owner: user1
# group: user1
user::rw-
group::r--
other::r--
```

*The output discloses an absence of ACLS on the file. The owner and group members have read and write permissions and everyone else has the read-only permission.*

**Step 3 :** *Allocate read and write permissions to user100 with the setfaci command using the octal form:*

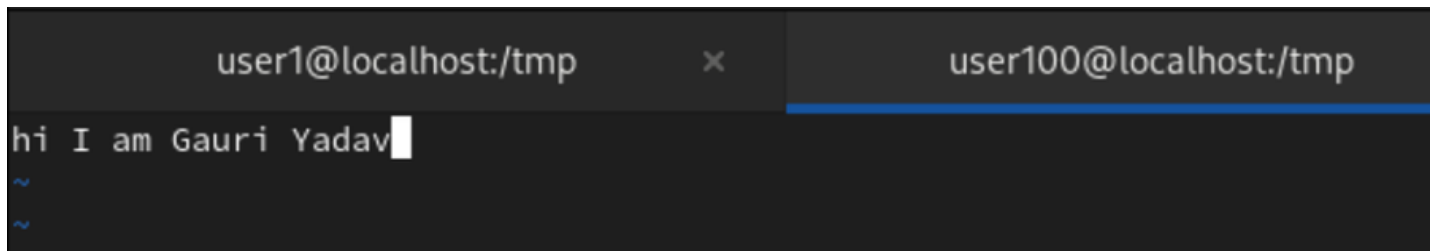
```
[user1@localhost tmp]$ setfacl -m user:user100:6 acluser1
```

**Step 4 :** *Run the ls command to check if the plus sign (+) has appeared next to the first column, then run the getfacl command to check the new access ACLs and the mask:*

```
[user1@localhost tmp]$ ls -l acluser1
-rw-rw-r--+ 1 user1 user1 0 Apr  1 11:28 acluser1
[user1@localhost tmp]$ getfacl acluser1
# file: acluser1
# owner: user1
# group: user1
user::rw-
user:user100:rw-
group::r--
mask::rw-
other::r--
```

*A row is added for the named user showing read/write permissions. Another row with the mask is also added and it is set to read/write as well. The mask value is auto-calculated based on the current maximum permissions that a named user of group has. This is reflected in the above output.*

**Step 5 :** *At this point, you can open another terminal session, switch into user100, change directory into /tmp, and open the file acluser1 in the vim editor. You should be able to modify the file and save it. full me permissions to acluser using*



**Step 6 :** *Add user200 with full rwx permissions to the symbolic notation and then show the updated ACL settings:*

```
[user1@localhost tmp]$ setfacl -m user:user200:rwx acluser1
[user1@localhost tmp]$ getfacl acluser1
# file: acluser1
# owner: user1
# group: user1
user::rw-
user:user100:rw-
user:user200:rwx
group::r--
mask::rwx
other::r--
```

*Notice the updated value for the mask, which is increased to rwx to reflect the maximum permission the named user, user200. has. If this file were to be an executable command, you would have been able to run it as user200 based on the assigned ACLs.*

## Step 7 : *Delete the Acl entry*

```
[user1@localhost tmp]$ setfacl -x user:user200 acluser1
[user1@localhost tmp]$ getfacl acluser1
# file: acluser1
# owner: user1
# group: user1
user::rw-
user:user100:rw-
group::r--
mask::rw-
other::r--
```

*Notice the reduction in the mask value to rw-, which now reflects the new current maximum permissions placed on the named user, user100.*

## Step 8 : *Delete the rest of the ACLS:*

```
[user1@localhost tmp]$ setfacl -b acluser1
[user1@localhost tmp]$ getfacl acluser1
# file: acluser1
# owner: user1
# group: user1
user::rw-
group::r--
other::r--
```

# How to identify, apply and erase Default acls

## Q1 What is Default Acls?

- A group collaboration on a shared directory gives members of the group identical access on files and subdirectories in the directory. Access ACLs may be applied to the shared directory to give non- group members certain rights. Furthermore, default ACLs can also be set on the shared directory to ensure new files and subdirectories created under the shared directory always have a consistent set of access rights for group and non-group members. This way the users do not have to adjust permissions on each new file and subdirectory they will create. The inheritance works slightly different for files and subdirectories, as indicated below:
- Files receive the shared directory's default ACLs as their access ACLS
- Subdirectories receive both default ACLs and access ACLS as they are
- The default ACLs can be described as the maximum discretionary permissions that can be allocated on a directory. Let's perform the following exercise and see how default ACLs are applied, viewed, and erased.

Lets understand this by a real time example

Scenario: this exercise, you will create a directory projects as user1 under/tmp. You will set default ACLs on the directory for named users, user100 and user200, to give them full permissions. You will create a subdirectory pridirl and a file prifile1 under projects and observe the effects of default ACLs on them. You will delete all the default entries at the end of the exercise.

**Step 1 :** *Switch or log in as user1 and create a directory project in /tmp:*

```
[root@localhost ~]# su - user1
[user1@localhost ~]$ cd /tmp
[user1@localhost tmp]$ mkdir projects
[user1@localhost tmp]$
```



**Step 2 :** *Use the ls and getfacl commands and check for the existence of any ACL entries:*

```
[user1@localhost tmp]$ getfacl projects/  
# file: projects/  
# owner: user1  
# group: user1  
user::rwx  
group::r-x  
other::r-x
```

*The output discloses that the owner and group members have full permissions on the directory and everyone else has read and execute permission.*

**Step 3 :** *Allocate default read, write, and execute permissions to user100 and user200 on the directory. Use both octal and symbolic notations and the -d (default) option with the setfacl command*

```
[user1@localhost tmp]$ setfacl -dm user:user100:7,u:user200:rwx projects
[user1@localhost tmp]$ getfacl projects/
# file: projects/
# owner: user1
# group: user1
user::rwx
group::r-x
other::r-x
default:user::rwx
default:user:user100:rwx
default:user:user200:rwx
default:group::r-x
default:mask::rwx
default:other::r-x
```

*The named users have the default ACLs. The rest of the default CL entries are for the directory owner, owning group, and public with the mask reflecting the maximum permissions.*

**Step 4 :** *Create a subdirectory prjdir1 under projects and observe the ACL inheritance:*

```
[user1@localhost projects]$ mkdir prjdir1
[user1@localhost projects]$ getfacl prjdir1/
# file: prjdir1/
# owner: user1
# group: user1
user::rwx
user:user100:rwx
user:user200:rwx
group::r-x
mask::rwx
other::r-x
default:user::rwx
default:user:user100:rwx
default:user:user200:rwx
default:group::r-x
default:mask::rwx
default:other::r-x
```

*As stated earlier, the subdirectory prjdir1 inherited both default and access ACLs. This is reflected in the above output.*

**Step 5** : *Create a file prjfile1 under projects and observe the ACL inheritance:*

```
[user1@localhost projects]$ touch prjfile1
[user1@localhost projects]$ getfacl prjfile1
# file: prjfile1
# owner: user1
# group: user1
user::rw-
user:user100:rw-          #effective:rw-
user:user200:rw-          #effective:rw-
group::r-x                #effective:r--
mask::rw-
other::r--
```

*As stated earlier, the file prjfile1 inherited the parent directory's default ACLs as its access ACLs minus what the mask limits. The maximum effective access for the named users is rw.*

**Step 6 :** *At this point, you can log in as one of the named users, change directory into / tmp /projects , and edit prjfile1 (add some random text). Then change into the prjdir1 and create file file100. You should be able to perform both tasks successfully.*

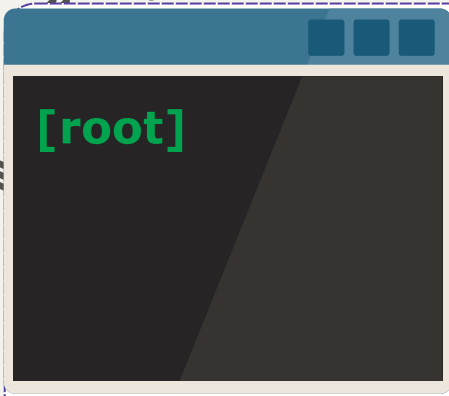
```
[root@localhost ~]# su - user100
[user100@localhost ~]$ cd /tmp/projects/
[user100@localhost projects]$ vim prjfile1
[user100@localhost projects]$
```

```
[user100@localhost projects]$ ls -l prjfile1
-rw-rw-r--+ 1 user1 user1 0 Apr  1 15:05 prjfile1
```

```
[user100@localhost projects]$ cd prjdir1/
[user100@localhost prjdir1]$ touch file100
[user100@localhost prjdir1]$ pwd
/tmp/projects/prjdir1
[user100@localhost prjdir1]$
```

**Step 7 :** *Delete all the default ACLs from the projects directory as user1 and confirm:*

```
[user1@localhost tmp]$ setfacl -k projects/  
[user1@localhost tmp]$ getfacl projects/  
# file: projects/  
# owner: user1  
# group: user1  
user::rwx  
group::r-x  
other::r-x
```



# THANK YOU

*Follow - @Gauri Yadav*

