# ACLs and Intro to Special Permissions in Linux

These notes explain how to manage permissions in Linux beyond the basic User, Group, and Other model. They include **Access Control Lists (ACLs)** and **Special Permissions**, which solve the limits of basic permissions.

## 1. Basic Permissions and Their Limits

Linux files and directories normally have three permission sets: **User (owner)**, **Group**, and **Other**, each with r (read), w (write), and x (execute) permissions. This means access is controlled in only three fixed blocks.

Example:

ls -ld mumbai  
drwxr-x---. 2 rohit mi\_group 6 Aug 26 21:05 mumbai

|  |  |  |
| --- | --- | --- |
| Category | Permission | Meaning |
| User (rohit) | rwx | Full access |
| Group (mi\_group) | r-x | Read and execute only |
| Other | --- | No access |

This setup works for simple cases but cannot handle exceptions without changing everything.

### Problem Scenario

A user called **Neeta** is neither the owner (rohit) nor part of the group (mi\_group), so she cannot access the mumbai directory. This shows how rigid the default permissions model is when one person needs access but others shouldn’t.

### Common Workarounds and Their Drawbacks

* **Give “Other” full permissions:**
* chmod o+rwx mumbai
  + This grants access to everyone outside the group (for example, Virat). This can be a serious security risk.
  + In other words, you lose all control over who else gets in.
* **Add Neeta to the group:**
* usermod -aG mi\_group neeta
  + Not always appropriate if Neeta does not belong to that group’s role. Giving write access to the whole group would also let other members (like Hardik and Surya) modify contents, which may not be desired.
  + It turns a small exception into a big change for everyone in the group.
* **Change ownership:**
* chown neeta mumbai
  + The original owner (Rohit) would lose control of the directory, which is often impractical.
  + This swaps control completely instead of sharing it.

These limits are exactly what ACLs are designed to solve. They give a way to grant access precisely to who needs it without disrupting others.

## 2. Access Control Lists (ACLs)

**Purpose:** ACLs let you grant extra permissions to specific users or groups without altering the main User/Group/Other permissions. They work like an extra layer on top of the standard model.

### Checking ACL Permissions

getfacl mumbai

Shows the owner, group, and any extra ACL entries on the file or directory. This gives a clear picture of all effective permissions in one place.

### Adding or Modifying ACL Permissions

setfacl -m u:<user\_name>:<permissions> <file\_name>

* -m = modify or add
* u = user
* g = group

Example:

setfacl -m u:neeta:rwx mumbai

Grants full rwx access to user Neeta on mumbai. A + at the end of the permissions string (rwxr-x---+) indicates ACLs are present. This way, only Neeta gets the new rights without touching others.

### Changing or Removing ACL Permissions

* Change a user’s ACL:
* setfacl -m u:mukesh:r-x mumbai
* Updates Mukesh’s ACL entry to read and execute only. This lets you fine-tune each person’s access individually.
* Remove a specific ACL entry:
* setfacl -x u:mukesh mumbai
* (leave the permission part blank with -x) This completely removes Mukesh’s extra permissions but keeps everyone else’s.
* Remove all ACL entries:
* setfacl -b mumbai
* Clears all ACLs from the directory at once. It’s a quick reset back to standard permissions.

### Granting a Group Access with ACL

setfacl -m g:<group\_name>:<permissions> <file\_name>

Example:

setfacl -m g:mi\_owner:rwx mumbai

All members of mi\_owner (for example, Mukesh, Akash, Radhika, Neeta) get full access to mumbai. This saves time when a whole team needs the same access.

**Key idea:** ACLs act like an extra layer of permissions. They do not disturb the normal User/Group/Other settings and can be adjusted per user or group. They combine flexibility with security by targeting only the needed accounts.

## 3. Special Permissions

Special permissions give additional control over executable files and directories. They work alongside basic permissions and ACLs. They are used to control how programs run or how shared folders behave.

### a) Set User ID (SUID)

* Set: chmod u+s <file\_name>
* Unset: chmod u-s <file\_name>
* In ls -ld: an s appears in the owner’s execute position.

Lowercase s: owner has execute permission and SUID is set. Uppercase S: owner does not have execute permission but SUID is set. This symbol tells you at a glance if the file runs with elevated privileges.

**Effect:** When a file with SUID is run, it runs with the owner’s privileges, not the user’s. This is useful for programs that need temporary higher rights.

### b) Set Group ID (SGID)

* Set: chmod g+s <file\_name>
* Unset: chmod g-s <file\_name>
* In ls -ld: an s appears in the group’s execute position.

**Effect:** Files created in a directory with SGID inherit the group ownership of that directory. This keeps group collaboration smooth without manually changing ownership every time.

### c) Sticky Bit

* Set: chmod o+t <directory\_name>
* Unset: chmod o-t <directory\_name>
* In ls -ld: a t appears in the “other” permissions position.

Lowercase t: others have execute permission and sticky bit is set. Uppercase T: others do not have execute permission but sticky bit is set. This mark tells you the directory is protected against accidental file deletions by others.

**Example:**

chmod 777 data  
chmod o+t data

Makes data a public folder where everyone can create files (for example, Amir, Salman, Shahrukh). Without the sticky bit, one user (Salman) could delete another’s files (Amir’s files). With the sticky bit set, only the owner of each file can delete or rename it. This is very helpful for shared folders like /tmp where many people work.

**Effect:** Prevents users from deleting or renaming files they do not own in shared directories. It adds a safety net to open or collaborative areas.