

File Permission, Ownership and Attributes

LinuXpert Systems

No.5, Dr. Ambedkar St, Rathinapuram,
Nanganallur, Chennai – 600 061.
Phone: 98841 65649, 90808 01382
E-mail: baskar@linuxpert.in

TABLE OF CONTENTS

1. chmod 1

2. chown 8

3. chgrp 13

4. umask 18

5. getfacl 25

6. setfacl 31

1. chmod

The `chmod` (change mode) command is used to **change the permissions** of files and directories. It determines who can **read**, **write**, or **execute** a file.

Syntax

```
chmod [OPTIONS] MODE FILE
```

or

```
chmod [OPTIONS] PERMISSIONS FILE
```

File Permissions

Linux has three types of permissions:

Permission	Symbol	Value	Meaning
Read	`r`	4	View the contents of a file
Write	`w`	2	Modify the file
Execute	`x`	1	Run the file as a program

Permissions are assigned to three categories of users:

User	Symbol	Meaning
Owner	`u`	The file owner
Group	`g`	Users belonging to the file's group
Others	`o`	Everyone else
All	`a`	Owner, group, and others

Viewing Current Permissions

Use:

```
ls -l filename
```

Example:

```
$ ls -l script.sh
-rw-r--r-- 1 user user 245 Jul 24 10:00 script.sh
```

Breakdown:

```
-rw-r--r--
│└┘ └┘ └┘
│ │  │  └── Others: Read
│ │  └───── Group: Read
│ └──────── Owner: Read + Write
└────────── File type
```

Using Symbolic Mode

Add permission

```
chmod u+x file.sh
```

Adds execute permission for the owner.

Example:

Before

```
-rw-r--r--
```

After

```
-rwxr--r--
```

Remove permission

```
chmod g-w file.txt
```

Removes write permission from the group.

Set exact permissions

```
chmod u=rwx,g=rx,o=r file.txt
```

Result:

```
-rwxr-xr--
```

Give everyone execute permission

```
chmod a+x script.sh
```

Remove execute permission from others

```
chmod o-x script.sh
```

Numeric (Octal) Mode

Permissions are represented using numbers.

Permission	Value
Read	4
Write	2
Execute	1

Add the values together:

Number	Permission
0	---
1	--x
2	-w-
3	-wx
4	r--
5	r-x
6	rw-
7	rwx

Example 1

```
chmod 755 file.sh
```

Meaning:

```
Owner : 7 = rwx
Group : 5 = r-x
Others: 5 = r-x
```

Result:

```
-rwxr-xr-x
```

Commonly used for executable programs and scripts.

Example 2

```
chmod 644 document.txt
```

Meaning:

```
Owner : rw-
Group : r--
Others: r--
```

Result:

```
-rw-r--r--
```

Commonly used for text files.

Example 3

```
chmod 600 secret.txt
```

Permissions:

```
Owner : rw-
Group : ---
Others: ---
```

Result:

```
-rw-------
```

Only the owner can read and modify the file.

Example 4

```
chmod 777 file.txt
```

Permissions:

```
rwxrwxrwx
```

Everyone has full access.

This is generally **not recommended** because it allows anyone to read, modify, and execute the file.

Recursive Permission Change

Use `-R` to change permissions for a directory and all its contents.

```
chmod -R 755 project/
```

Example structure:

```
project/
├── main.c
├── script.sh
└── docs/
```

All files and directories under `project` receive the specified permissions.

Commonly Used Permission Values

Number	Permissions	Typical Use
777	rwxrwxrwx	Full access to everyone (avoid unless necessary)
755	rwxr-xr-x	Executable programs and directories
750	rwxr-x---	Private directories shared with a group
700	rwx------	Owner only
666	rw-rw-rw-	Read/write for everyone (generally avoid)
644	rw-r--r--	Regular text files
640	rw-r-----	Private files shared with a group
600	rw-------	Confidential files

Useful Options

Option	Description
`-R`	Change permissions recursively
`-v`	Display a message for every processed file
`-c`	Display changes only when permissions are modified
`--reference=FILE`	Copy permissions from another file

Example:

```
chmod --reference=file1 file2
```

Makes `file2` use the same permissions as `file1`.

Practical Examples

Make a script executable

```
chmod +x install.sh
```

Run it:

```
./install.sh
```

Remove write permission from everyone

```
chmod a-w report.txt
```

Allow only the owner full access

```
chmod 700 private/
```

Make a file read-only for everyone

```
chmod 444 notes.txt
```

Make all shell scripts executable

```
chmod +x *.sh
```

Summary

Command	Description
`chmod +x file`	Add execute permission
`chmod -x file`	Remove execute permission
`chmod 755 file`	Owner: rwx, Group/Others: r-x
`chmod 644 file`	Owner: rw, Group/Others: r
`chmod 600 file`	Owner only can read/write
`chmod 700 dir`	Owner has full access
`chmod -R 755 dir`	Recursively change permissions
`chmod u+x file`	Add execute permission for owner
`chmod g-w file`	Remove group write permission
`chmod o+r file`	Give others read permission
`chmod a+x file`	Give execute permission to everyone

`chmod` is one of the core Linux permission management commands. It works alongside `chown` (change file owner) and `chgrp` (change file group) to control access to files and directories.

2. chown

The `chown` command in Linux is used to **change the ownership of files and directories**. It allows you to change the **owner (user)**, the **group**, or both.

Syntax

```
chown [OPTIONS] OWNER[:GROUP] FILE...
```

- **OWNER** – New owner (username)
- **GROUP** – New group (optional)
- **FILE** – File or directory whose ownership is to be changed

Basic Examples

1. Change the owner of a file

```
chown alice report.txt
```

Result:

- Owner becomes `alice`
- Group remains unchanged

---

2. Change owner and group

```
chown alice:developers report.txt
```

Result:

- Owner → `alice`
- Group → `developers`

---

3. Change only the group

```
chown :developers report.txt
```

Result:

- Owner remains the same
- Group becomes `developers`

---

4. Change ownership recursively

```
chown -R alice:developers /home/project
```

Result:

Changes the owner and group of the directory and **all files and subdirectories** inside it.

---

5. Use numeric UID and GID

```
chown 1001:1001 file.txt
```

Instead of usernames, you can specify:

- UID (User ID)
- GID (Group ID)

Common Options

Option	Description
`-R`	Change ownership recursively
`-v`	Verbose; show every change made
`-c`	Show only when a change is made
`-f`	Suppress most error messages
`--from=OLDOWNER:OLDGROUP`	Change only if current owner/group matches
`-h`	Change ownership of symbolic links themselves (not their targets)
`--reference=FILE`	Copy owner and group from another file

Practical Examples

Change owner

```
sudo chown john document.pdf
```

Change owner and group recursively

```
sudo chown -R john:staff /var/www/html
```

Copy ownership from another file

```
sudo chown --reference=file1.txt file2.txt
```

Now `file2.txt` has the same owner and group as `file1.txt`.

Verbose output

```
sudo chown -Rv john:staff project/
```

Example output:

```
changed ownership of 'project/file1'
changed ownership of 'project/file2'
```

Check Ownership

Before changing ownership:

```
ls -l report.txt
```

Example:

```
-rw-r--r-- 1 root root 1024 Jul 24 report.txt
```

After:

```
sudo chown alice:developers report.txt
```

```
ls -l report.txt
```

Output:

```
-rw-r--r-- 1 alice developers 1024 Jul 24 report.txt
```

Common Use Cases

Change ownership of a downloaded file

```
sudo chown bharath myfile.txt
```

Give ownership of a web directory

```
sudo chown -R www-data:www-data /var/www/html
```

Give ownership of a home directory

```
sudo chown -R alice:alice /home/alice
```

Important Notes

- Only the **root** user (or a user with `sudo` privileges) can change the owner of a file to another user.
- A regular user may be able to change the group of a file to a group they belong to, depending on system permissions and configuration.
- Be careful with recursive operations (`-R`), especially in system directories.

Difference Between `chown` and `chmod`

Feature	`chown`	`chmod`
Changes owner	Yes	No
Changes group	Yes	No
Changes permissions	No	Yes
Recursive option	`-R`	`-R`

Example:

```
sudo chown alice:developers file.txt
chmod 755 file.txt
```

- `chown` changes **who owns** the file.
- `chmod` changes **what permissions** the file has.

Summary

Task	Command
Change owner	`chown user file`
Change owner and group	`chown user:group file`
Change only group	`chown :group file`
Recursive change	`chown -R user:group directory`
Verbose	`chown -v user file`
Copy ownership	`chown --reference=file1 file2`

`chown` is one of the fundamental Linux file management commands and is commonly used by system administrators to assign the correct ownership to files and directories after creating users, deploying applications, restoring backups, or moving data.

3. chgrp

The `chgrp` command is used to **change the group ownership** of a file or directory. Only the file owner or the root user can change a file's group (to a group they belong to).

Syntax

```
chgrp [OPTIONS] GROUP FILE...
```

- **GROUP** – The new group name or Group ID (GID).
- **FILE** – One or more files or directories.

Basic Examples

1. Change the group of a file

```
chgrp developers report.txt
```

Changes the group owner of `report.txt` to `developers`.

---

2. Change the group using GID

```
chgrp 1002 report.txt
```

Changes the group to the group whose GID is `1002`.

---

3. Change the group of multiple files

```
chgrp developers file1.txt file2.txt file3.txt
```

All three files will belong to the `developers` group.

---

4. Change the group of a directory

```
chgrp developers /project
```

Only the directory's group changes—not the files inside it.

---

5. Change the group recursively

```
chgrp -R developers /project
```

Changes the group of the directory and all files and subdirectories.

Common Options

Option	Description
`-R`	Change group recursively
`-v`	Verbose (show every change)
`-c`	Show only files whose group changed
`-f`	Suppress most error messages
`--reference=FILE`	Copy the group ownership from another file
`-h`	Change the group of symbolic links instead of their targets (where supported)

Examples of Options

Verbose output

```
chgrp -v developers report.txt
```

Output:

```
changed group of 'report.txt' from 'bharath' to 'developers'
```

Recursive with verbose

```
chgrp -Rv developers /project
```

Displays every file whose group is changed.

Copy group from another file

Suppose:

```
file1 -> group: developers
file2 -> group: users
```

Run:

```
chgrp --reference=file1 file2
```

Now `file2` also belongs to the `developers` group.

Check the Result

Using `ls -l`

```
ls -l report.txt
```

Example output:

```
-rw-r--r-- 1 bharath developers 2500 Jul 24 10:15 report.txt
```

The columns mean:

```
Permissions  Owner    Group
-rw-r--r--   bharath  developers
```

Difference Between `chgrp` and `chown`

Command	Changes
`chgrp`	Only the group owner
`chown`	Owner, or owner and group together

Example:

```
chgrp developers report.txt
```

Only the group changes.

```
chown alice report.txt
```

Only the owner changes.

```
chown alice:developers report.txt
```

Both owner and group change.

Practical Examples

Project collaboration

```
sudo groupadd projectteam
sudo chgrp -R projectteam /srv/project
```

All files in `/srv/project` are assigned to the `projectteam` group.

Shared directory

```
mkdir shared
sudo chgrp developers shared
chmod 775 shared
```

Members of the `developers` group can collaborate in the `shared` directory.

Common Errors

Error: Operation not permitted

```
chgrp: changing group of 'file.txt': Operation not permitted
```

Cause: You are not the file owner or root, or you're trying to change the group to one you are not a member of.

Solution:

```
sudo chgrp developers file.txt
```

Error: Invalid group

```
chgrp: invalid group: 'abc'
```

Cause: The specified group does not exist.

Check available groups:

```
getent group
```

or

```
cat /etc/group
```

Summary

- **Purpose:** Change the group ownership of files and directories.
- **Basic syntax:**

  ```
  chgrp GROUP FILE
  ```

- **Recursive:**

  ```
  chgrp -R GROUP DIRECTORY
  ```

- **View ownership:**

  ```
  ls -l
  ```

- **Related commands:**
  - `chown` – Change file owner and/or group.
  - `groups` – Display the groups a user belongs to.
  - `groupadd` – Create a new group.
  - `groupmod` – Modify a group.
  - `groupdel` – Delete a group.

4. umask

The `umask` command is used to **set or display the default permissions** assigned to newly created files and directories in Linux.

Instead of granting permissions, `umask` **removes (masks)** permissions from the system's default values.

Syntax

```
umask [options] [mask]
```

How `umask` Works

When a file or directory is created, Linux starts with these default permissions:

Object	Default Permission
File	`666` (`rw-rw-rw-`)
Directory	`777` (`rwxrwxrwx`)

The `umask` value subtracts permissions from these defaults.

View Current umask

```
umask
```

Example:

```
$ umask
0022
```

This means:

- Owner loses nothing
- Group loses write permission
- Others lose write permission

Symbolic View

```
umask -S
```

Output:

```
u=rwx,g=rx,o=rx
```

This is easier to understand.

Common umask Values

umask	Files	Directories	Meaning
0000	rw-rw-rw-	rwxrwxrwx	No restrictions
0002	rw-rw-r--	rwxrwxr-x	Group collaboration
0022	rw-r--r--	rwxr-xr-x	Default on many Linux systems
0077	rw-------	rwx------	Private files only

Example 1 – Default umask

Current umask:

```
umask
```

Output

```
0022
```

Create a file

```
touch test.txt
```

Check permissions

```
ls -l test.txt
```

Output

```
-rw-r--r-- 1 user user 0 Jul 24 test.txt
```

Calculation

```
666
-022
-----
644
```

Result

```
rw-r--r--
```

Example 2 – Directories

```
mkdir demo
```

```
777
-022
-----
755
```

Result

```
drwxr-xr-x
```

Example 3 – Set a New umask

```
umask 0077
```

Create a file

```
touch secret.txt
```

Permissions

```
ls -l secret.txt
```

Output

```
-rw------- 1 user user 0 Jul 24 secret.txt
```

Calculation

```
666
-077
-----
600
```

Only the owner can read and write the file.

Example 4 – Allow Group Collaboration

```
umask 0002
```

Create a directory

```
mkdir project
```

Permissions

```
drwxrwxr-x
```

This is commonly used on shared project servers.

Example 5 – Change umask for Current Shell

```
umask 027
```

Check it:

```
umask
```

Output

```
0027
```

This change affects only the current shell session unless added to a shell startup file.

Understanding the Calculation

Suppose:

```
Default file permission = 666
umask = 027
```

Calculation:

```
666
-027
-----
640
```

Result:

```
rw-r-----
```

Owner:

```
rw-
```

Group:

```
r--
```

Others:

```
---
```

Make umask Permanent

For a single user, add it to:

```
~/.bashrc
```

or

```
~/.profile
```

Example:

```
umask 022
```

Reload:

```
source ~/.bashrc
```

For all users (system-wide), an administrator can configure the default umask in system configuration files such as `/etc/profile` or through PAM (`pam_umask`), depending on the Linux distribution.

Options

Option	Description
`umask`	Display current mask
`umask -S`	Display mask in symbolic form
`umask 022`	Set mask using octal notation
`umask u=rwx,g=rx,o=rx`	Set mask using symbolic notation

Practical Uses

- Protect newly created files from unauthorized access.
- Configure default permissions for users and groups.
- Secure sensitive data by making new files private (`umask 077`).
- Enable group collaboration with shared write access (`umask 002`).

Interview Questions

1. What does the `umask` command do?

It displays or sets the default permission mask for newly created files and directories.

2. Does `umask` change permissions of existing files?

No. It only affects permissions of files and directories created **after** the mask is set.

3. Why do files start with `666` and not `777`?

Regular files do not receive execute (`x`) permission by default because most new files are data files, not executable programs.

4. What is the most common default `umask`?

`022`, which results in:

- Files: `644` (`rw-r--r--`)
- Directories: `755` (`rwxr-xr-x`)

5. What `umask` should be used for maximum privacy?

`077`, which results in:

- Files: `600` (`rw-------`)
- Directories: `700` (`rwx------`)

5.getfacl

`getfacl` (Get File Access Control List) is a Linux command used to **display the Access Control List (ACL)** of a file or directory. ACLs provide more fine-grained permissions than the standard `rwx` (read, write, execute) permissions.

Syntax

```
getfacl [OPTIONS] FILE...
```

Purpose

Normally, Linux permissions allow access control for:

- Owner (user)
- Group
- Others

ACLs allow you to assign permissions to **specific users and groups** without changing the file owner or group.

`getfacl` displays:

- File owner
- File group
- Standard permissions
- ACL entries for additional users/groups
- Default ACLs (for directories)

Basic Example

```
getfacl myfile.txt
```

Example output:

```
# file: myfile.txt
# owner: bharath
# group: users
user::rw-
```

```
user:john:r--
group::r--
mask::r--
other::---
```

Explanation

Entry	Meaning
`user::rw-`	Owner has read and write permission
`user:john:r--`	User **john** has read permission
`group::r--`	File's group has read permission
`mask::r--`	Maximum effective permission for named users/groups
`other::---`	Others have no permission

Viewing ACL of Multiple Files

```
getfacl file1 file2 file3
```

Displays ACL information for all files.

Display ACL of a Directory

```
getfacl Documents
```

Example:

```
# file: Documents
# owner: bharath
# group: bharath
user::rwx
group::r-x
other::r-x
default:user::rwx
default:group::r-x
default:other::r-x
```

The `default:` entries are inherited by newly created files and subdirectories.

Display ACL Recursively

```
getfacl -R Documents
```

Shows ACLs for every file and subdirectory inside `Documents`.

Skip Base Permissions

```
getfacl -c myfile.txt
```

Suppresses the comment lines (`# file`, `# owner`, etc.) and shows only the ACL entries.

Example output:

```
user::rw-
group::r--
other::r--
```

Show Physical Files (Ignore Symbolic Links)

```
getfacl -P directory
```

If the directory contains symbolic links, `-P` prevents following them.

Follow Symbolic Links

```
getfacl -L directory
```

Follows symbolic links and displays the ACLs of the target files.

Numeric User and Group IDs

```
getfacl -n myfile.txt
```

Displays numeric UIDs and GIDs instead of names.

Example:

```
user::rw-
user:1001:rwx
group::r--
other::---
```

Display Effective Permissions

```
getfacl -e myfile.txt
```

Shows the effective permissions after the ACL mask is applied.

Example:

```
user:john:rwx        #effective:r--
```

Although `john` is granted `rwx`, the mask limits the effective permission to `r--`.

Restore ACL Backup

If ACLs were previously saved:

```
getfacl -R project > acl_backup.txt
```

They can later be restored using:

```
setfacl --restore=acl_backup.txt
```

Common Options

Option	Description
`-R`	Display ACLs recursively
`-c`	Omit comment lines
`-n`	Show numeric UIDs/GIDs
`-e`	Show effective permissions
`-L`	Follow symbolic links
`-P`	Do not follow symbolic links

Practical Example

Create a file:

```
touch report.txt
```

Grant user `john` read and write access:

```
setfacl -m u:john:rw report.txt
```

View the ACL:

```
getfacl report.txt
```

Output:

```
# file: report.txt
# owner: bharath
# group: bharath
user::rw-
user:john:rw-
group::r--
mask::rw-
other::r--
```

Here:

- The owner (`bharath`) has `rw-`.
- The additional user (`john`) has `rw-`.
- The group has `r--`.
- The ACL mask allows up to `rw-` for named users and groups.
- Others have `r--`.

When to Use `getfacl`

- Verify ACL permissions on files or directories.
- Troubleshoot unexpected permission issues.
- Audit file access for specific users or groups.
- Back up ACLs before migrating or copying files.
- Confirm inherited (default) ACLs on directories.

Summary

Feature	Description
Command	`getfacl`
Purpose	Display Access Control Lists (ACLs)
Works On	Files and directories
Companion Command	`setfacl` (used to create, modify, or remove ACLs)
Common Use	View extended permissions beyond the standard owner/group/others model

`getfacl` is an essential tool when working with **extended file permissions** on Linux systems, especially in environments where multiple users require different levels of access to the same files or directories.

6.setfacl

The `setfacl` command is used in Linux to **set or modify Access Control Lists (ACLs)** on files and directories.

ACLs provide **more flexible permissions** than the standard `rwx` (read, write, execute) permissions. They allow you to grant permissions to **multiple users and groups** without changing the file's owner or group.

Syntax

```
setfacl [OPTIONS] ACL_SPEC FILE...
```

or

```
setfacl [OPTIONS] -m ACL_SPEC FILE
```

Basic ACL Format

```
u:user:permissions
g:group:permissions
o::permissions
m::permissions
```

Where:

- `u` = User
- `g` = Group
- `o` = Others
- `m` = Mask (maximum effective permissions)

Permissions:

- `r` = Read
- `w` = Write
- `x` = Execute

Why use `setfacl`?

Normally Linux allows permissions for only:

- Owner
- Group
- Others

Suppose:

- Owner = Alice
- Group = Developers

Now you want **Bob** to have read access without changing the owner or group.

ACL makes this possible.

Common Options

Option	Description
`-m`	Modify or add an ACL entry
`-x`	Remove an ACL entry
`-b`	Remove all ACL entries
`-k`	Remove default ACL
`-R`	Apply recursively
`-d`	Set default ACL for directories
`--restore`	Restore ACLs from backup
`--test`	Test changes without applying

Example 1: Give a User Read Permission

```
setfacl -m u:bob:r file.txt
```

Meaning:

- `-m` → modify ACL
- `u:bob:r`
    - user = bob
    - permission = read

Result:

Bob can read `file.txt`.

Example 2: Give Read and Write Permission

```
setfacl -m u:bob:rw file.txt
```

Bob gets:

```
Read
Write
Execute ✘
```

Example 3: Give Execute Permission

```
setfacl -m u:bob:x script.sh
```

Bob can execute the script.

Example 4: Give Full Permission

```
setfacl -m u:bob:rwx file.txt
```

Bob receives

- Read
- Write
- Execute

Example 5: Give Group Permission

```
setfacl -m g:developers:rw project.txt
```

The `developers` group gets:

- Read
- Write

Example 6: Multiple ACL Entries

```
setfacl -m u:bob:rw,u:alice:r,g:admins:rwx file.txt
```

Adds:

```
Bob → Read/Write
```

```
Alice → Read
```

```
Admins → Read/Write/Execute
```

Example 7: Remove an ACL Entry

```
setfacl -x u:bob file.txt
```

Removes Bob's ACL entry.

Example 8: Remove All ACLs

```
setfacl -b file.txt
```

Deletes every ACL from the file.

Only normal Linux permissions remain.

Example 9: Recursive ACL

```
setfacl -R -m u:bob:rwx /project
```

Applies ACLs to:

- `/project`
- every subdirectory
- every file

Example 10: Default ACL on a Directory

```
setfacl -d -m u:bob:rwx /shared
```

Now every **new file and directory** created inside `/shared` automatically inherits Bob's `rwx` permissions.

Example 11: Default Group ACL

```
setfacl -d -m g:developers:rw /shared
```

Every newly created file under `/shared` grants the `developers` group read/write access.

Example 12: Set a Mask

```
setfacl -m m:r-- file.txt
```

The **mask** limits the maximum permissions that named users and groups can actually use.

For example, even if Bob has `rwx`, if the mask is `r--`, his effective permission becomes only `r--`.

View ACLs

Use `getfacl`:

```
getfacl file.txt
```

Example output:

```
# file: file.txt
# owner: alice
# group: users
user::rw-
user:bob:rwx
group::r--
mask::rwx
other::---
```

Backup ACLs

```
getfacl project > acl_backup.txt
```

Restore later:

```
setfacl --restore=acl_backup.txt
```

Real-World Example

Suppose you have:

`/project`

Owner:

`alice`

Group:

`developers`

You want:

- Bob → Read only
- Charlie → Read/Write
- QA group → Read
- Others → No access

Commands:

```
setfacl -m u:bob:r /project
setfacl -m u:charlie:rw /project
setfacl -m g:qa:r /project
chmod o-rwx /project
```

Now each user has exactly the required permissions without changing the owner or primary group.

Filesystem Support

ACLs require filesystem support (such as ext4, XFS, or Btrfs with ACLs enabled). Most modern Linux distributions enable ACL support by default.

Summary

Command	Purpose
`setfacl -m u:bob:r file`	Give Bob read permission
`setfacl -m u:bob:rw file`	Give Bob read/write permission
`setfacl -m g:admins:rwx file`	Give a group full access
`setfacl -x u:bob file`	Remove Bob's ACL
`setfacl -b file`	Remove all ACLs
`setfacl -R -m u:bob:rwx dir`	Apply ACLs recursively
`setfacl -d -m u:bob:rwx dir`	Set default ACLs for new files/directories
`getfacl file`	Display current ACLs
`setfacl --restore=backup.txt`	Restore ACLs from a backup

Key Difference from `chmod`

- **`chmod`** manages permissions only for the **owner, one group, and others**.
- **`setfacl`** lets you assign permissions to **specific additional users and groups**, making it ideal for shared projects and collaborative environments where standard permissions are too limited.