File Access Permissions

Under a multi-user operating system like Linux, 'access permission' is the security mechanism for protecting system and user data. Depending on the filetype (for example regular files or directories), permissions may vary. Users can manage access permissions on their data to share it with collaborators on Gadi.

File Ownership

File access permissions control the access to the file separately for the file owner, the group owner of the file and other users on the system.

Files, by default, are owned by the user who creates them, and their group ownership is set to the project of the user owner when creating the files. Users who are members of this project can have different access permissions than other users on the system.

Owner

The posix user who currently owns the file. When the file is first created, it is owned by the user who created it. But since this ownership can be transferred, a current owner may not necessarily be its creator.

On Gadi, users need to request NCI Helpdesk for getting their data ownership transferred to another user.

Group

The project of the posix user when creating the data. Storage usage is debited from the project that owns the data no matter which project directory it is located in. The group ownership is not determined by the project directory in which the file was created.

On Gadi, users can change the group ownership of their own data as long as they are members of the project that about to own the data.

Others

Posix users who are neither the file owner nor members of the project who owns the file. On Gadi, it is not recommended to share data with other users outside the project.

Look up File Ownership and Change Group Owner

Here is an example long listing `ls -la` of a directory:

```
$ ls -la
drwxrws---  16 aaa777 a00 16384 Jul 17 10:51 .
drwxrws---  72 root   a00 16384 Oct  6 13:52 ..
drwxrwsr-x   2 aaa777 a00 16384 May 18 11:37 share_within_group
drwxr-sr-x 178 aaa777 a00 16384 Oct  4  2020 aaa777
-rw-r--r--   1 bbb777 a00 24362 Apr  1 14:59 usage_report.csv
```

The directory `share_within_group` is owned by the posix user aaa777 and its group ownership is set to the project `a00`.

File group ownership can be changed using `chgrp` utility:

```
$ touch test.log
$ ls -l test.log
-rw-r--r-- 1 aaa777 b11 0 Oct  5 21:39 test.log
$ chgrp a00 test.log
$ ls -l test.log
-rw-r--r-- 1 aaa777 a00 0 Oct  5 21:39 test.log
```

Posix File Access Permissions

In above example, first column shows the file access permissions set for their corresponding entries. It always contains 11 characters, the first one indicates the file type and the last one indicates whether extended access control attributes are set. The nine characters in between are three permission triads for the aforementioned three types of user(s), the file owner, the group owner, and others, left to right. Within each triad, the first character suggests whether read permission is granted to the corresponding type of user(s), shown as `r` when granted, the second one is for write permission, shown as `w` when granted, and the third one mainly for execute permission, shown as `x` when granted. When a permission is not granted, it is shown as `-`. Please see more details in the table below.

Permission Char	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
		Owner			Group			Others			

Attributes	File Type	Read	Write	Execute	Read	Write	Execute	Read	Write	Execute	Extended Attributes
Valid values	d | l | -	r | -	w | -	x | s | -	r | -	w | -	x | s | -	r | -	w | -	x | t | -	. | +
What they mean	d - directory l - symbolic link - is regular file	r - owner has read access	w - owner can modify /remove file	x - owner has execute permission	r - project group members have read access	w - project group members can modify /remove file	x - project group members have execute permission on file or dir	r - others have read access	w - others have modify / remove access	x - others have execute permission on file or dir	. - no special attributes set + - indicates presence of extended access control attributes on the file
modification		s - if setuid bit is set on an executable file, it is run under owner privileges (caution!)			s - if setgid bit is set, executable is run under owner's group privileges. if its a dir, new content is created under that group ownership			t - if sticky bit is set on a dir, only owners or root can remove the contents. typically set on shared dir to avoid possible loss of all user data by a rogue command			
Commands to modify permissions					To enable setgid bit on directory `chmod g+s dirname`			To enable sticky bit `chmod o+t dirname`			getfacl / setfacl

Permissions mean exactly what their names suggest when applied to standard files; for directories the effect would be slightly different.

Read

The permission to read a file.

When set for a directory, only names of files in the directory can be listed, other details like file type, size, ownership, permissions etc. are not visible.

Write

The permission to modify or remove a file.

When set for a directory, it grants the ability to modify entries in the directory, including creating, deleting and renaming files. Note that, without *execute* per mission on a directory, the write permission does not take any effect.

Execute

The permission to execute a file.

When set for a directory, it is interpreted as the *search* permission: it grants the ability to access file contents and meta-information if its name is known, but does not list files inside the directory, unless *read* is set.

Setuid, Setgid and Sticky Bit

When executing a binary that has the setuid bit enabled, it inherits the permissions set for the file owner. It is considered as a security hazard and will have no effect when setting the execute bit to `s` in the user owner triad on Gadi.

The setgid bit is similar to the setuid bit, but set the permissions for its group owner. It enables a command to be run as the project that owns the file rather than the user's default project. On Gadi, again, for security concerns, it is ignored when set on standard files but it can be used on directories to force new files and directories created inside them to inherit their group ownership.

The sticky bit sets extra permissions for others. It replaces the execute permission with t. On Gadi, it is can be used to do 'deletion protection' for public directories that you have to allow others to write. For example, when exchange data in the /scratch/public directory with users outside your project, files inside it can be deleted or renamed only by the file owner, directory owner and the root user because the sticky bit is set to `drwxrwxrwt.` on the /scratch /public directory. Similarly, when you want to share data with others inside a directory in /scratch/public, set the sticky bit to apply the deletion protection follow the last example in the next section.

Change File Access Permissions

To change file permissions, run `chmod`. For example, the following command searches for directories under /scratch/a00/aaa777(inclusive) recursively, grants read and execute permissions to project members of the owner group, enables the setgid bit, and removes all rwx permissions for others for all the directories in the search result.

```
$ find /scratch/a00/aaa777 -type d | xargs chmod g=rxs,o-rwx
```

The permissions of these directories set for the file owner remain the same but can be modified. For example, to remove the owner's write permission, run:

```
$ chmod u-w test
```

To set the deletion protection on the directory in which you have to allow others to write, run

```
$ mkdir -p /scratch/public/$USER
$ chmod o=rwxt /scratch/public/$USER
```

so that it reserves the permission of deleting a file for the file owner and the owner of the parent directory only.

Default File Access Permissions on Gadi

The default permission for newly created files is set in the umask value. On Gadi, `umask 022` is the default configuration in the system .bashrc file. User can overwrite it in their own file ~/.bashrc.

umask

The command `umask` sets the inverse mask of the default file access permissions. The OS restricts the permissions of newly created files to no more than the defined permissions by the umask value.

It is normally the octal number used in umask argument to present the permissions in a triad. Here is a lookup table for symbolic permissions, its octal number and the inverse of the octal number.

symbolic	octal	inverse
---	0	7
--x	1	6
-w-	2	5
-wx	3	4
r--	4	3
r-x	5	2
rw-	6	1
rwx	7	0

For example the umask value of 022 grants the user owner all permissions `rwx` while giving the group owner and others `rx` permissions.

Access Permissions on Project Directories

Default access permission on project directories on /scratch and /g/data is set to `rwxrws---` so that non-members of that project cannot see the project dir and its contents. All files created inside these directories inherit the group ownership attributes.