Entitlements are a combination of three entitlements:

* r- read (4)
* w- writing (2)
* x- execute (1)

respectively for the owner, group and the rest of the users, e.g.:

**-rw-r–r–** 1 root users 0 Oct 24 18:04 file

The file has **644** rights, viz:

* read and write (rw- : 4+2=6 decimal notation) for the owner, in this case for the user “root”
* reading right for the group (r– : 4 decimal notation), in this case for the group “users”
* reading right for the rest (r– : 4 decimal notation)

The command to modify and list permissions:

ls -la <- listing  
chmod 755 <- change permissions  
chown owner:group <- change owner  
umask <- display the default privilege assignment

Default umask set in /etc/profile

**SetUID** - allows you to specify under which user ID the program is executed, for files this means that the process will have the rights of the user who owns the executable file, not the user running that file. You need to be very careful when setting this bit and not give administrator (root) rights by accident. Such an example is “passwd” or “ping”, set takes the value 4 which means binary 100. chmod 4755 file

**SetGID** - allows you to specify under which group identifier the program is executed, when used with a directory all subdirectories and files in it have the group of the parent directory, it is set as the second bit of the octal digit (binary 010).

chmod 2755 file  
chmod 2644 file

**Sticky bit** - allows the program to remain in memory after it finishes running. It is a leftover from the past, because there is no need for programs to remain in memory after termination. It is most often used with a directory. When used with a directory then users can only delete files for which they have explicit write permissions, even if they have write permissions to the directory, such as the /tmp directory, it is set as an extra 1, which is placed before the standard 3 bits.

chmod 1755 file

Setting on the group directory, give chmod g+w \* for directories. chmod g+s \* <- niebezpieczne bo nie zabezpiecza przed usunieciem chmod +t \* <- zabezpiecza przed usuwanie tylko /tmp to ma

for i in fin1 fin2 mark1 mark2; do id $i; done

**Uprawnienia nadane a efektywne** |Special Bit| User | Group | Others | | — | — | — | — | | | r w x | r w x | r w **t** r w **T**| | 0 0 1 | 1 1 1 | 1 1 1 | 1 1 1 1 1 0| | 1 | 7 | 7 | 7 6|

If this bit is NOT set it is denoted by “T”, if it IS set it is “t”.

t = T+x

s = S+x

## ACLs

* ACLs - (Access Control List), which provide the ability to set an extended group of permissions on files and directories
* To be able to use ACLs the file system must be mounted with ACL support enabled, the acl option must be added to the mounted file system in the /etc/fstab file. In XFS file system such support is enabled by default, in EXT file systems ACL support must be enabled
* ACLs are divided into two categories:
  + access ACLs - access ACLs, set on files and/or directories
  + default ACLs - default ACLs, set only at the directory level. Files and subdirectories included in a directory with default ACLs set inherit the default ACLs of the parent directory, that is, a default ACL set for a directory will cause all files created in it to have specific default permissions
* IMPORTANT!!! Assigning these permissions will not affect the permissions of existing files already in the directory, only those that are newly created!

mount -o remount,acl /home

getfacl - displays the ACL settings for the file and directory. setfacl - sets, modifies and deletes ACLs for a file or directory. chacl - changes the ACL settings on a file or directory. IRIX UNIX system command.

Options setfacl:

* -m: Sets or modifies the ACL
* -x: Deletes a specific ACL
* -d: Sets the default ACLs
* -k: Deletes all default ACLs
* -b: Deletes all ACLs
* -R: Sets ACLs recursively on all files and subdirectories
* -c: Skips the display of the permissions header

Scope:

* u[ser]:UID:perms: Permissions assigned to a specific user (username or UID). The user must be in the /etc/passwd file.
* g[roup]:GID:perms: Permissions assigned to a specific group (username or GID). The user must be in the /etc/group file.
* m[ask]:perms: Maximum permissions a particular user or group can have on a file or directory. E.g. rw- means that no user or group will have more permissions than read and write.
* o[ther]:perms: Permissions assigned to users who are not members of the owning group.

**Unusual ACLs (access)**.

If we grant the user student write and read rights and change the mask to read-only at the same time, the effective permissions for student will be read-only:

setfacl -m u:student:rw,m:r file1   
getfacl -c file1

That is, the user user1 will not be able to modify this file even though it appears to have write permissions.

After changing the mask as below, the student user will be able to modify file1.

setfacl -m m:rw file1   
getfacl -c file1

getfacl /home/student/file1  
setfacl -m g:finances:rx marketing/ <- assign ACL for group   
setfacl -d -m g:finance:rx marketing/ <- assign a default ACL for the group  
setfacl -d -m g:marketing:rx finance/  
setfacl -m u:student:rx /home/student/file1 <- assign ACL for user  
setfacl -x u:student /home/student/file1 <- remove ACL for user

Removing all ACLs:

setfacl -b /path/to/file

\*\* Default ACLs\*\*.

Log in as user user1 and create a projects directory. Use the getfacl command to check the default permissions:

pwd  
mkdir projects  
getfacl projects

Let’s grant a default ACL for reading and writing to user user2 and user3 to the projects directory:

setfacl -m d:u:user2:6,d:u:user3:6 projects  
getfacl -c projects

In the projects directory, let’s create a subdirectory dev and check if it inherits the ACL settings from the parent directory.

cd projects  
mkdir dev  
getfacl -c dev

Let’s create a file file1 in the projects directory and see if it inherits the ACL from the parent directory.

touch file1  
getfacl -c file1

We should see that the maximum permissions for group members are read and write. The execute right is ineffective due to the mask settings.