File ownership and access permissions on Linux

Robert Bukowski
Bioinformatics Facility

BioHPC Users' meeting 2/19/2019

Outline

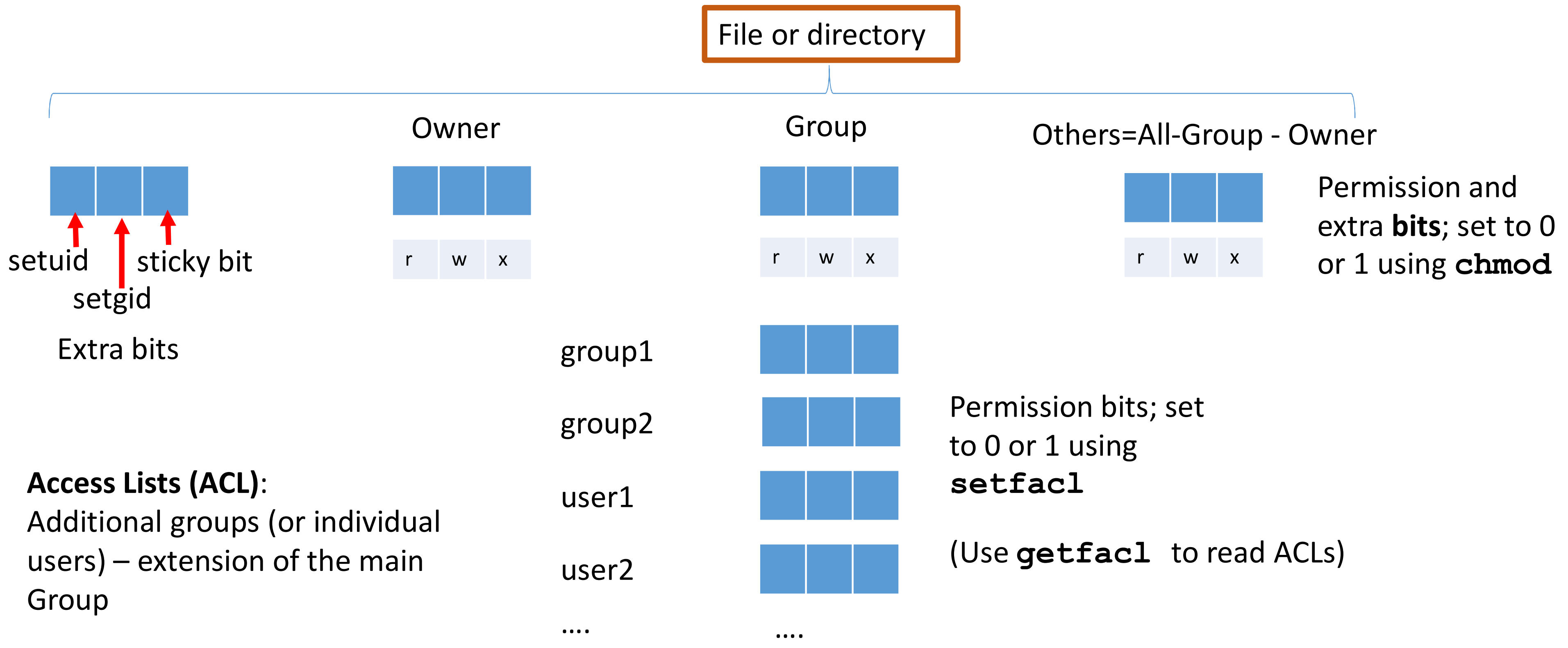
File ownership, representation and meaning of access permissions

Adjusting ownership on permissions of existing files and directories

Ownership and access permissions of **new** files – how to set default behavior

Complications

File ownership and permissions - overview



Permission and extra **bits**; set to 0 or 1 using **`chmod`**

Access Lists (ACL): Additional groups (or individual users) – extension of the main Group

Permission bits; set to 0 or 1 using **`setfacl`**

(Use **`getfacl`** to read ACLs)

Objects with ACLs have a "**+**" after permission string

```
drwxrws---+  4 bukowski panzea      4096 Feb 14 17:32 ttt1
```

Meaning of permission bits

Bit	Effect on file if set	Effect on dir if set
r	File can be read	Directory content (file and subdir names) can be shown by **ls**
x	File can be executed	One can cd into the directory (**x** required for all subdirs in the path)
w	File can be modified (**x** required for all subdirs in the path) File can be renamed, moved, or removed **only** if **x** is set for all subdirs in the path and **w** is set for parent directory	Files and subdirs can be created, renamed, or removed in the directory [even if there is no **w** on these files themselves (!!)]; **x** also required for all subdirs in the path

NOTE:

To delete a file it is sufficient to have **`wx`** permission on the parent directory
`w` permission on the file itself is not needed to delete it

Meaning of extra bits

Bit	As shown by `ls –al` (example)	Effect on file	Effect on directory
setuid (implies **x**)	`-rwsr-xr-x 1 jarekp   cbsuguest1 45583 Feb 12 12:22 some_script.sh`	File will execute as owner (here: jarekp), no matter who runs it	None
setgid (implies **x**)	`drwxr-s--- 4 bukowski cbsuguest1 4096 Feb 12 11:57 my_dir`	File will execute as owning group (here: `cbsuguest1`), no matter who runs it	New files and directories created inside `my_dir` will inherit group (here: `cbsuguest1`); new dirs will have **setgid** set as well
sticky	`-rw-rwxr-t 1 bukowski panzea 172092320 Feb 22  2011 flygenome.fa`	None	File can be deleted or renamed only by the owner, even if **w** on directory allows others to delete/remove files

Adjusting ownership and permissions for <u>existing</u> files: examples

Recursively change owner to **`user1`** and group to **`group1`** for **`/local/storage/some_dir`** and all its content (only **`root`** or **`user1`** can do it like this)

```
        chown –R user1.group1 /local/storage/some_dir
```

Recursively change group to **`group1`** for **`/local/storage/some_dir`** and all its content (owner has to belong to **`group1`**)

```
        chgroup –R group1 /local/storage/some_dir
```

Set permissions for group and change permissions for “others” for a single file

```
        chmod  g=rwx,o-w /local/storage/some_dir/my_file
```

Recursively set permissions for group and revoke all permissions for “others” for a directory and its content. Group permission for all files will be **`rw-`** and for directories **`rwx`**

```
        chmod  -R g=rwX,o=  /local/storage/some_dir
```

ACLs:
Add (or modify if already there) the ACL for user1 on one file

```
        setfacl  -m u:user1:rwx  /local/storage/some_dir/my_file
```

Recursively add (or modify if already there) the ACL for group group2 on a directory and all its content

```
        setfacl -R -m g:group2:rX   /local/storage/some_dir
```

How to recognize objects with ACL attached

```
ls -al
drwxrws---+  4 bukowski panzea         4096 Feb 14 17:32 ttt1
```

Objects with ACLs have a **+** after permission string

Permissions displayed in Group triad represent the **mask** – typically the union (logical OR) of permissions for the Group and all ACLs

Use **getfacl** to check the details:

```
getfacl ttt1
# file: ttt1
# owner: bukowski
# group: panzea
# flags: -s-
user::rwx
user:jarekp:rwx
group::r-x
mask::rwx
other::---
```

New files (directories):

Who owns it?

Owner: the user who created the file (directory)
Group: the primary group of the owner

- Exception**:** if **setgid** bit is set on the parent directory – then the new object inherits the group of the parent directory

What are the permissions?

Permissions = (Default permissions) AND (~ **umask**)
umask is user-dependent; **default umask =** (0022) = (000 000 010 010) ← says which permissions to turn **off**

Assuming default mask, permissions for new objects are

New files:	`rw- r-- r --`
New directories:	`rwx r-x r-x`

Example of user-defined **umask**:

```
    umask 0027
```

in `.bashrc` will turn off all permissions for "others" on new files

- Exception: if parent directory has **default ACLs** attached to it, permissions on new object will obey these ACLs

NOTE: unless "exceptions" are used, default permissions depend on **user (owner) rather than location** – not good for file sharing!

Example: Setting inheritable ownership/permissions on a directory tree

Set desired main group owner and permissions (including ACLs) on the existing files in the directory tree, e.g.,

```
chgroup –R mylabgroup /local/storage/ourdir
chmod –R g+rwX /local/storage/ourdir
setfacl –R –m u:user1:rwX /local/storage/ourdir
setfacl –R –m g:group1:rX /local/storage/ourdir
```

Make **`mylabgroup`** the default for all new object within the directory tree (i.e., set **setgid** bit for all exisiting directories) – will override the primary group of the owner

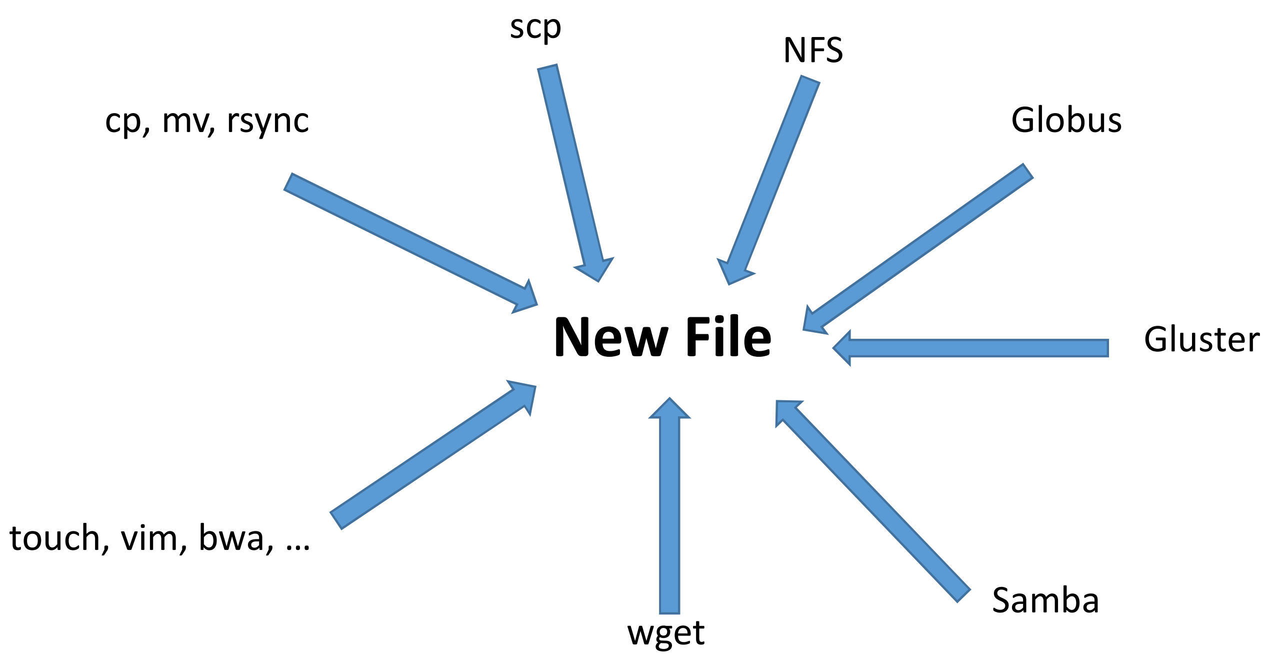
```
chmod g+s $(find /local/storage/ourdir –type d –print)
```

Set **default ACLs** (i.e., ACLs to be applied to new objects) – will override **umask** set by the owner

```
setfacl –R –dm g::rwX /local/storage/ourdir
setfacl –R –dm u:user1:rwX /local/storage/ourdir
setfacl –R –dm g:group1:rX /local/storage/ourdir
```

Complications

A “new file” can be created by many different tools processes) – each with its own “ideas” about ownership and permissions…



Tool	Obeys setgid	Applies default ACLs
cp	YES	YES
cp -p	YES	NO
rsync –a	YES	YES
mv	NO (also: preserves original owner, group)	NO
scp	YES	YES (but may change mask, which changes effective ACL permissions)
FileZilla	YES	YES
Samba	YES	YES
Gluster	YES	NO (ACL not supported)
NFS	YES	NOT supported by client ACLs set on server show up as modified group mask on client (possible security hole!) Files created on client get their ACLs applied on server, but with mask inherited from client…
chmod (applied to group)	YES	Modifies ACL mask (changing effective permissions)

<u>New files</u> (directories): what are the permissions?

Actual permissions = (**Default permissions**) AND (~ **umask**)

	(New) File	**(New) Directory**
Default permission	0666 = (000 110 110 110) = (rw- rw- rw-)	0777 = (000 111 111 111) = (rwx rwx rwx)
Default **umask**	0022 = (000 000 010 010)	0022 = (000 000 010 010)
Actual permissions	0644 (= 000 110 100 100) = (rw- r-- r--)	0755 = (000 111 101 101) = (rwx r-x r-x)

umask can be changed (from its default 0022) <u>by the user</u> (e.g., putting command **`umask 0027`** into **`.bashrc`** will change **umask** to 0027, making any new files and directories off-limits for “others”)