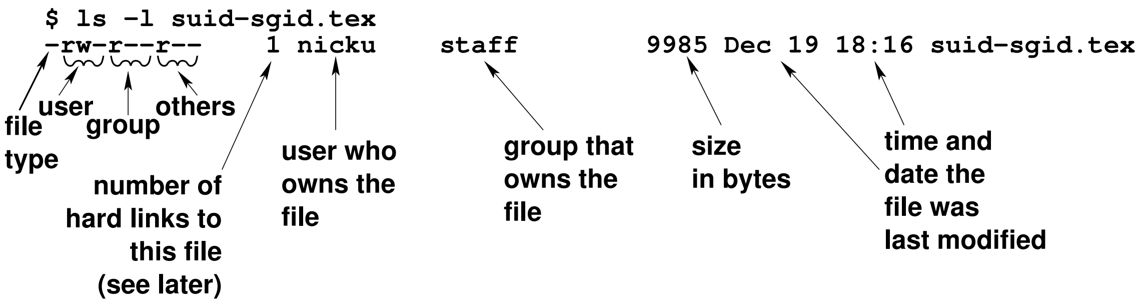
Access Control & Permissions: SUID, SGID, chmod, chown, and chgrp

1 Aim

This short exercise is a quick introduction to the use of the tools `chmod` (*change mode*), `chown` (*change owner*), and `chgrp` (*change group*). It also aims for you to understand the access control permissions, including *set user* ID (SUID) and *set group* ID (SGID) through some practical exercises.

2 Background

2.1 Access Control to Files: the Rules

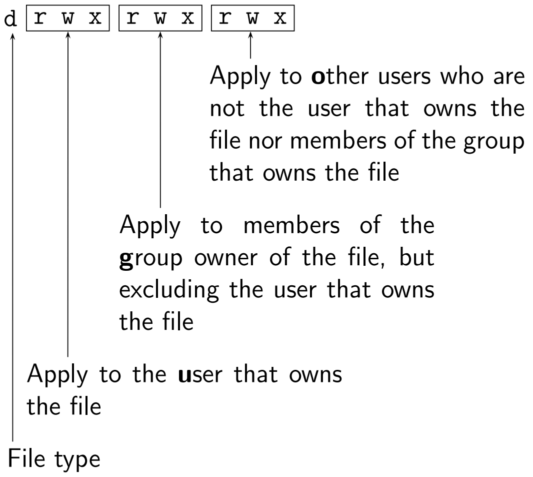


```
$ ls -l suid-sgid.tex
-rw-r--r--    1 nicku    staff        9985 Dec 19 18:16 suid-sgid.tex
```

file type
user
group
others
number of hard links to this file (see later)
user who owns the file
group that owns the file
size in bytes
time and date the file was last modified

Figure 1: The output of `ls -l`: what each field is.

File permissions are shown when you list a file's details with the command `ls -l`; see figure 1. Permissions are divided into three sets, as shown in figure 2.



d
r w x
r w x
r w x
Apply to other users who are not the user that owns the file nor members of the group that owns the file
Apply to members of the group owner of the file, but excluding the user that owns the file
Apply to the user that owns the file
File type

Figure 2: There are three sets of of permissions: one for the user that owns the file, one set for the group that owns the file, and the last set for all other users who are not the owner of the file, not members of the group that owns the file.

The rules that determine your access rights to a file are quite simple:

1. if your user ID is the same as the **u**ser that owns the file, then the left-most read, write, execute permissions on the file apply to you.
2. Otherwise, if you belong to the **g**roup that owns the file, then the middle three read, write, execute permissions on the file apply to you.
3. Otherwise, the right-most read, write and execute permissions on the file apply to you.

Example: Suppose we have three users as shown in this table:

Username	**Belongs to these groups**	
`nicku`	`nicku`	`staff`
`henryl`	`henryl`	`staff`
`a1`	`a1`	`students`

Suppose that the following files have these permissions and ownership:

```
$ ls -l
----r--r--    1 nicku    root         4022 Oct  9 22:05 wgetrc
drwxr-x---   17 root     henryl       4096 Aug 28 12:43 X11
-rw-r-----    1 root     staff         289 Nov 20 01:00 xinetd.conf
drwxr-x--x    2 root     root         4096 Dec  5 01:43 xinetd.d
drwxr-xr--    2 root     nicku        4096 Jul  3 01:07 xml
-rw-r--r--    1 nicku    nicku        4941 Mar 19  2002 xpdfrc
-rw-rw-rw-    1 root     root          361 Mar 26  2002 yp.conf
-rw-r-----    1 root     root         1626 Oct  8 01:56 ypserv.conf
```

Then for each file or directory, each user has the following access rights:

<table>
<tr><th>file</th><th>nicku</th><th>henryl</th><th>a1</th></tr>
<tr><td>wgetrc</td><td>no access</td><td>read</td><td>read</td></tr>
<tr><td>X11</td><td>no access</td><td>read, can change into the directory</td><td>no access</td></tr>
<tr><td>xinetd.conf</td><td>read</td><td>read</td><td>no access</td></tr>
<tr><td>xinetd.d</td><td colspan="3">can change into the directory, but not list it.</td></tr>
<tr><td>xml</td><td>list, change into</td><td colspan="2">list with ls, not ls -l</td></tr>
<tr><td>xpdfrc</td><td>read, write</td><td colspan="2">read</td></tr>
<tr><td>yp.conf</td><td colspan="3">read, write</td></tr>
<tr><td>ypserv.conf</td><td colspan="3">no access</td></tr>
</table>

2.2 Meaning of Read, Write, Execute Permissions on a Directory

Directories are files that contain a list of data for each file: filename, *inode number*. The directory does not contain other information about the file, such as the size, the time it was last modified, ... The *inode number* is just a number that uniquely identifies where on the disk partition the file contents are actually stored. We will discuss inode numbers in the module on filesystems in the workshop notes.

read permission in a directory means that you can list the file names in the directory with commands such as `ls`.

write permission on a directory means the right to delete a file from the directory. Note that the right to delete a file does not depend on the permissions on the file itself, only on the directory in which the file is located.

execute permission on a directory allows changing into that directory with a `cd` command.

2.3 Minimum Permissions Required for some Operations

Command	minimum access required	
	on the file	on the directory
`cd /var/project`	no file	`--x`
`ls /var/project`	`---`	`r--`
`ls -l /var/project`	`---`	`r-x`
`cat /var/project/user1.txt`	`r--`	`--x`
`echo "hello" >> /var/project/user1.txt`	`-w-`	`--x`
`/var/project/binary-program`	`--x`	`--x`
`/var/project/script-program`	`r-x`	`--x`
`rm /var/project/user1.txt`	`---`	`-wx`

2.4 The Set User ID and Set Group ID Permissions

Every process has a user ID that owns the process, and a group that owns the process.

When you execute a program, the owner of the process is equal to your user ID, and the group that owns the process is equal to your primary group ID.

If you execute a program file that has the *set user* ID permission, the process will probably execute with a different user ID from yours.

If you execute a program file that has the *set group* ID permission, the process will most likely execute with a different group ID from yours.

The aim of these exercises is to find out what determines the user ID of a process started from a SUID executable file, and what determines the group ID of a process started from a SGID file.

2.5 Who owns a process?

How can you tell who owns a process? The `ps` command can tell you: `ps -eo user,group,cmd`. This will show *all* processes. To filter out all but `/tmp/ash`, you could do: `ps -eo user,group,cmd | grep /tmp/[a]sh` Or, any file created by a process will be owned by the owner of the process. The file will have group ownership equal to the primary group owner of the process. You can check the ownership of the file using the `ls -l` command.

3 Procedure

3.1 Exercises with the Set User ID Permission

What we will do here is copy a shell program file to your `/tmp` directory, execute it, and use the command `whoami`, then exit, set the SUID permission on the shell executable file, run it again, and find out who you are. Next, you will change the ownership of the shell program, and run it again. Do this last step a number of times, until you see what is happening. Don't forget to delete this shell file, as it is a great security risk!

The `bash` shell has built in precautions against the danger of running as a different user, but the simple shell `ash` does not, so we will experiment with it.

1. Copy the `ash` shell to your `/tmp` directory (not your network directory, as only *you* have permission to read that):

```
$ sudo cp -a /bin/ash /tmp
```

Copy the shell using `sudo` to preserve the ownership of the shell; it should be owned by the `root` user:

```
$ ls -l /tmp/ash
-rwxr-xr-x    1 root     root       110048 Jul 18 07:50 /tmp/ash
```

2. Now execute it, and run the command `whoami`:

```
$ /tmp/ash
$ whoami
nicku
```

You may see some error messages saying "`function: not found`"; this is because `ash` does not understand everything in your ∼`/.bashrc` login script. Don't worry; this does not alter what we will learn here.

3. Then exit:

```
$ exit
```

4. and add the SUID permission to the executable:

```
$ sudo chmod u+s /tmp/ash
```

We are adding the special permission to the **u**ser who owns the file (which is `root`). The `chmod` (change mode) command changes the permissions on files. Here, we *add* the special permission to the existing permissions for the **u**ser who owns the file.

5. then list the permissions on the file:

```
$ ls -l /tmp/ash
-rwsr-xr-x    1 root     root       110048 Jul 18 07:50 /tmp/ash
```

Notice that the only change from before is that the first "`x`" (for **u**ser) has changed into an "`s`".

6. Now execute the shell and see who you are this time:

```
$ /tmp/ash
# whoami
```

✐ What user did you see? .... □

7. Create a file in the `/tmp` directory:

```
# touch /tmp/file
# ls -l /tmp/file
```

✐ Which user owns the file? Which group? .... □

8. Now change the ownership of the shell program to the user **apache**, run the shell, then see who you are:

```
# exit
$ sudo chown apache /tmp/ash
$ ls -l /tmp/ash
-rwsr-xr-x    1 apache   root       110048 Jul 18 07:50 /tmp/ash
$ /tmp/ash
$ whoami
```

✐ What user did you see? .... □

9. Open your file `/etc/passwd`, and select a number of other users. Repeat the last exercise for each user. Try creating files (perhaps with the `touch` command) and see who owns the files. See which group owns each file.

10. Fill in the following table:

SUID?	*SGID?*	*user that owns* `/tmp/ash`	*group that owns* `/tmp/ash`	*user that owns the process*	*group that owns the process*	*user that executes* `/tmp/ash`	*group that executes* `/tmp/ash`
no	no	root	root			me	my group
yes	no	root	root			me	my group
yes	no	apache	root			me	my group
yes	no					me	my group

Conclusion What general rule can you state about the effect of the SUID permission?

3.2 Exercises with Set Group ID Permission

Here you will do essentially the same exercise as before, but this time with the *set group* ID (SGID) permission turned on, and the SUID permission turned off.

1. Turn off the SUID permission and turn on the SGID permission:

```
$ sudo chmod u-s /tmp/ash
$ sudo chmod g+s /tmp/ash
```

   or, as one command,

```
$ sudo chmod u-s,g+s /tmp/ash
```

   and list the file to ensure that the permissions are correctly set:

```
$ ls -l /tmp/ash
-rwxr-sr-x    1 apache   root       110048 Jul 18 07:50 /tmp/ash
```

   Note the “s’ is now in the permissions that apply to the **g**roup owner of the executable file.

2. Now execute the shell, and see who you are:

```
$ /tmp/ash
$ whoami
```

3. Create a file, and list it to see who the user that owns the file is, and who the group that owns the file is:

```
$ touch /tmp/newfile
$ ls -l /tmp/newfile
```

4. Exit from the shell, then change the group owner of the executable shell program file to a number of other users, perhaps yourself, the user `apache`,...

```
$ sudo chgrp apache /tmp/ash
$ /tmp/ash
$ touch /tmp/newfile2
$ ls -l /tmp/newfile2
```

5. Fill in the following table:

SUID?	*SGID?*	*user that owns* `/tmp/ash`	*group that owns* `/tmp/ash`	*user that owns the process*	*group that owns the process*	*user that executes* `/tmp/ash`	*group that executes* `/tmp/ash`
no	no	root	root			me	my group
no	yes	root	root			me	my group
no	yes	root	apache			me	my group
no	yes					me	my group
yes	yes					me	my group

Conclusion

Explain what the effect of the SGID permission is.

Conclusion How is the SGID permission different from the SUID permission on an executable file?

✐ ____

____

3.3 Effect of Set Group ID Permission on a Directory

The effect of the SGID permission on a directory is that all files created in the directory have a group owner equal to the group owner of the directory. This is very useful for group projects. This is discussed in detail in the module on user management in the workshop notes.