ITT420: System and Network Administration
Lab Exercise 4

Name: **MOHD AZWAN BIN SHAMSUDDIN** Student No: **2013549263**

LAB 4 – Advanced File Permissions and umask

At the end of this lab exercise students should be able to:

- Modify the permission for a file/directory in unix/linux system
- Modify the permission using octal modes
- Modify the permission using symbolic modes
- Modify the default umask for a unix/linux system
- Show understanding on special permission, setuid and sticky bit mode
- Understand command used to secure a file

Instruction: Complete this lab exercise and submit by the end of the week. Use on-line manual if you have trouble with some commands.

File and Directory Permissions

Open up a terminal and type : `ls -l`

The output are all directories and files with three sets of read/write/execute permissions: one set for the user or owner of the file, one set for the group of the file, and one set for everyone else. These permissions are determined by nine bits in the inode information, and are represented by the characters `rwxrwxrwx`. E.g:

```
-rwxrw-r-- 1 rozita pensyarah 2450 Jan 01 11:52 file1
```

Access Right on Files:

- r (or -), indicates read permission (or otherwise), that is, the presence or absence of permission to read and copy the file
- w (or -), indicates write permission (or otherwise), that is, the permission (or otherwise) to change a file
- x (or -), indicates execution permission (or otherwise), that is, the permission to execute a file, where appropriate

Access Right on Directories:

- r allows users to list files in the directory;
- w means that users may delete files from the directory or move files into it;
- x means the right to access files in the directory. This implies that you may read files in the directory provided you have read permission on the individual files.

Command to Change File/Directory permission:

There are two methods in which you can change permission of a file.

Method 1: Using chmod with octal numbers

Syntax

```
% chmod permission file/dir-name
```

Exercise:

1. Open up a terminal
2. Create a file called fileperm.txt
3. Type `ls -l fileperm.txt` and note the permission for the file fileperm.txt
4. Modify the permission of fileperm.txt so that the permission becomes:

```
-r--r--r--
```

Write down the command that you have used here:

```
apoigaga@Apoigaga:~$ chmod 444 fileperm.txt
apoigaga@Apoigaga:~$ ls -l fileperm.txt
-r--r--r-- 1 apoigaga apoigaga 0 Nov  23 23:19 fileperm.txt
```

5. Modify again the permission for fileperm.txt so that the permission becomes:

1. `-rw-rwxr--`

```
apoigaga@Apoigaga:~$ chmod 674 fileperm.txt
apoigaga@Apoigaga:~$ ls -l fileperm.txt
-rw-rwxr-- 1 apoigaga_apoigaga 0 Nov  23 23:19 fileperm.txt
```

2. `-------rwx`

```
apoigaga@Apoigaga:~$ chmod 007 fileperm.txt
apoigaga@Apoigaga:~$ ls -l fileperm.txt
-------rwx 1 apoigaga_apoigaga 0 Nov  23 23:19 fileperm.txt
```

3. `-rwxrw-r—-`

```
apoigaga@Apoigaga:~$ chmod 764 fileperm2.txt
apoigaga@Apoigaga:~$ ls -l fileperm2.txt
-rwxrw-r-- 1 apoigaga_apoigaga 0 Nov  23 23:35 fileperm2.txt
```

For each of the above permission, write down the octal numbers.

- **674**
- **007**
- **764**

Method 2: Using symbolic modes

For symbolic modes, this is the syntax:

```
chmod [references][operator][modes] file1 ...
```

References are used to distinguish the users to whom the permissions apply.

Reference	Class	Description
u	user	the owner of the file
g	group	users who are members of the file's group
o	others	users who are not the owner of the file or members of the group
a	all	all three of the above, is the same as ugo

Operator refers to how the mode of the file should be adjusted, whether a new permission is added or removed.

Operator	Description
+	adds the specified modes to the specified classes
-	removes the specified modes from the specified classes
=	the modes specified are to be made the exact modes for the specified classes

Exercise:

1. Open up a terminal
2. Create a file called fileperm2.txt
3. Type `ls –l` fileperm2.txt and note the permission for the file fileperm2.txt
4. Execute the following command:
   `chmod go-rwx fileperm2.txt`

5. Write down the new file permission for fileperm2.txt
6. What is the equivalent octal numbers for the above symbolic modes?

   600

7. Modify again the permission for fileperm2.txt using symbolic modes so that the permission becomes:
   a. `–rw-rwxr--`
   b. `-------rwx`
   c. `–rwxrw-r—-`

For each of the above permission, write down the command that you have used.

a) chmod u+rw,g+rwx,o+r fileperm2.txt
b) chmod u-rw,g-rwx,o+wx fileperm2.txt
c) chmod u+rwx,g+rw,o-wx fileperm2.txt

Exercise:

In file permission, there are two special modes called:
1. setuid/gid
2. sticky bit

Explain the function for each special modes and provide examples where necessary.

- **setuid**
  When set-user identification (setuid) permission is set on an executable file, a process that runs this file is granted access based on the owner of the file (usually root), rather than the user who is running the executable file. This special permission allows a user to access files and directories that are normally only available to the owner. For example, the setuid permission on the passwd command makes it possible for a user to change passwords, assuming the permissions of the root ID:

  ```
  -r-sr-sr-x   3 root     sys       104580 Sep 16 12:02 /usr/bin/passwd
  ```

- **setgid**
  The set-group identification (setgid) permission is similar to setuid, except that the process's effective group ID (GID) is changed to the group owner of the file, and a user is granted access based on permissions granted to that group. The /usr/bin/mail command hassetgid permissions:

  ```
  -r-x--s--x   1 root     mail       63628 Sep 16 12:01 /usr/bin/mail
  ```

When setgid permission is applied to a directory, files that were created in this directory belong to the group to which the directory belongs, not the group to which the creating process belongs. Any user who has write and execute permissions in the directory can create a file there. However, the file belongs to the group that owns the directory, not to the user's group ownership.

- **sticky bit**
  The sticky bit is a permission bit that protects the files within a directory. If the directory has the sticky bit set, a file can be deleted only by the owner of the file, the owner of the directory, or by root. This special permission prevents a user from deleting other users' files from public directories such as /tmp:

  ```
  drwxrwxrwt 7  root  sys   400 Sep  3 13:37 tmp
  ```

Setting an Exact umask

You can use the *umask* command to set the default mode for newly created files. Its argument is a three-digit numeric mode that represents the access to be *inhibited* - masked out - when a file is created. Thus, the value it wants is the octal complement of the numeric file mode you want. To determine this, you simply figure out the numeric equivalent for the file mode you want and then subtract it from 777. For example, to get the mode 751 by default, compute 777-751 = 026; this is the value you give to *umask*:

```
% umask 026
```

Once this command is executed, all future files created will be given this protection automatically. System administrators can put a *umask* command in the system initialization file to set a default for all users. You can set your own *umask* in your shell setup files to make it permanent.

Exercise:

1. Open up a terminal
2. Execute the following command:

   ```
   umask
   ```

   write down the current value for umask

   - **0002**

   write down the default permission for newly created file with this value of umask

   - **-rw-rw-r-- (664)**

3. Execute the following command:

   ```
   umask 026
   ```

4. Create a new file called fileperm3.txt
5. Write down the file permission for the newly created file. Does it follow the new value set by umask.

   - **-rw-r----- . yes it follow the new value set by umask.**

6. Modify again the value for umask so that every new file will have the following permission:
   a. `-rw-rwxr--`
   b. `-------rwx`
   c. `-rwxrw-r—-`

   For each of the above permission, write down the umask value that you have used.

- **umask 103**
- **umask 770**
- **umask 013**

Exercise:

Describe how you can set a global umask in such a way that when you create a user, the permission of the newly created file by the user will be based on the umask set.

- **By set umask 0002**

Security to Files Using ACL

1. Check the ACL value for file `fileperm2` by using `getfacl` command.. Next, change the ACL by giving a user in your Linux system permission to write this file (his/her username must be correctly identified).

   Command: `setfacl -m group:`*`group-name`*`:`*`perm-mode`*`, user:`*`user-name`*`:`*`perm-mode filename`*

```
apoigaga@Apoigaga:~$ getfacl fileperm2.txt
# file: fileperm2.txt
# owner: apoigaga
# group: apoigaga
user::rwx
group::rw-
other::r--

apoigaga@Apoigaga:~$ setfacl -m group:apoigaga:rw- -m user:apoigaga:rw- fileperm
2.txt
```

2. Check the value of ACL and switch user to the one you have used in (1.) and test the permission
   Command used :_____.

```
apoigaga@Apoigaga:~$ getfacl fileperm2.txt
# file: fileperm2.txt
# owner: apoigaga
# group: apoigaga
user::rwx
user:apoigaga:rw-
group::rw-
group:apoigaga:rw-
mask::rw-
other::r--

apoigaga@Apoigaga:~$ su apoigaga
Password:
apoigaga@Apoigaga:~$ nano fileperm2.txt
apoigaga@Apoigaga:~$ cat fileperm2.txt
hi this is apoigaga
apoigaga@Apoigaga:~$ nano fileperm3.txt
```

Get error:
Error writing fileperm3.txt: Permission denied