COMP 311 Linux Operating System Lab

Lab 4: *File Systems – Part 2*

Ahmed Tamrawi

atamrawi atamrawi.github.io ahmedtamrawi@gmail.com

جامعة بيرزيت
BIRZEIT UNIVERSITY

Computer Science Department

Linux OS Laboratory Manual (V 1.2)

COMP311

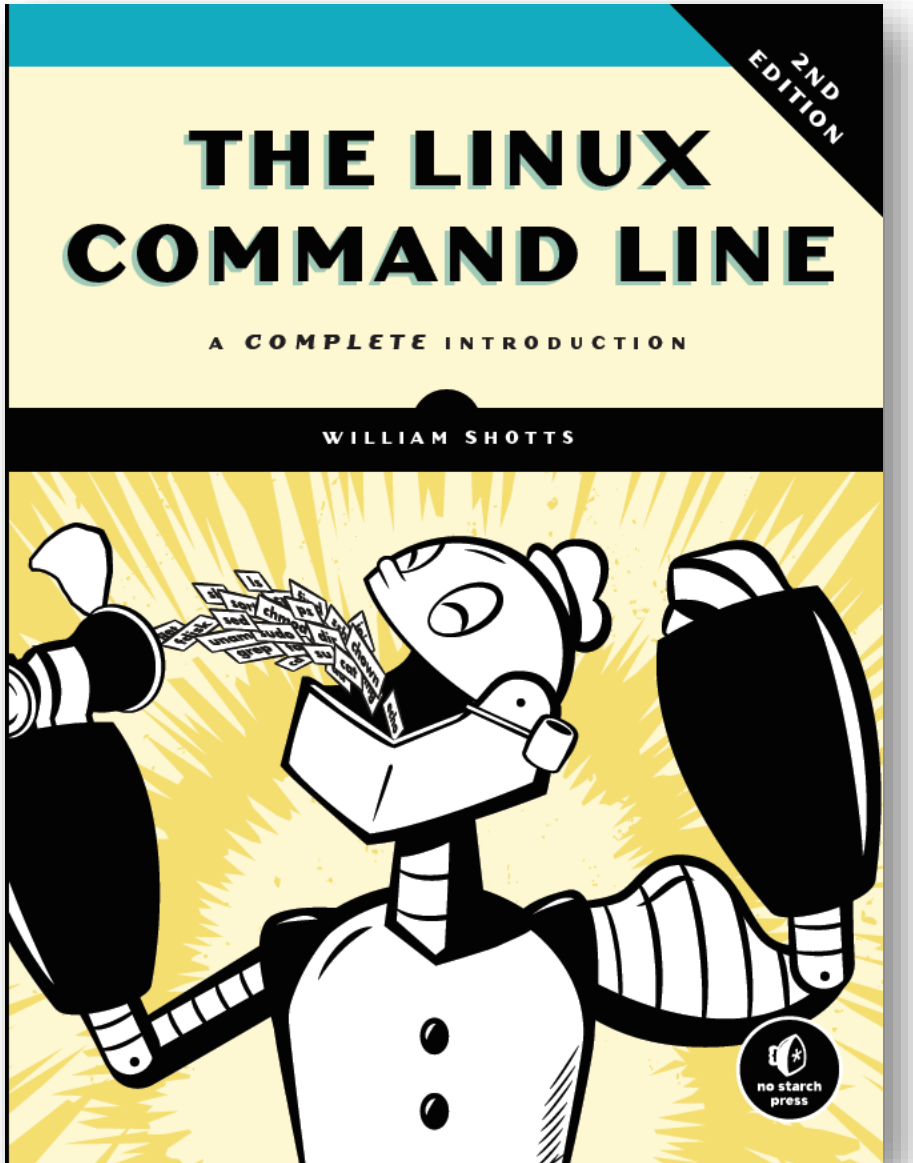
Nael I. Qaraeen

Approved By:

Computer Science department

May 2015

+



Multitasking and Multiusers

- More than one person (multiuser) can be using the computer at the same time. While a typical computer will likely have only one keyboard and monitor, it can still be used by more than one user.
  - For example, if a computer is attached to a network or the Internet, remote users can log in via ssh (secure shell) and operate the computer.
- In fact, remote users can execute graphical applications and have the graphical output appear on a remote display.

Permissions

- To make this practical, a method had to be devised to protect the users from each other.
- After all, the actions of one user could not be allowed to crash the computer, nor could one user interfere with the files belonging to another user.

`id`	Display user identity
`chmod`	Change a file's mode
`umask`	Set the default file permissions
`su`	Run a shell as another user
`sudo`	Execute a command as another user
`chown`	Change a file's owner
`chgrp`	Change a file's group ownership
`passwd`	Change a user's password

Owners, Group Members, and Everybody Else

- We may have encountered a problem when trying to examine a file such as /etc/shadow.

```
[me@linuxbox ~]$ file /etc/shadow
/etc/shadow: regular file, no read permission
[me@linuxbox ~]$ less /etc/shadow
/etc/shadow: Permission denied
```

- The reason for this error message is that, as regular users, we do not have permission to read this file.

Owners, Group Members, and Everybody Else

- In the **Unix security model**, a user may **own** files and directories.
  - When a user **owns** a file or directory, **the user has control over its access**.
- Users can, in turn, belong to a **group** consisting of one or more users who are **given access to files and directories** by their **owners**.
- In addition to granting access to a group, an **owner** may also grant some set of access rights to **everybody**, which in Unix terms is referred to as the **world/other**.
- We have three important terms: **owner (u)**, **group (g)**, and **everybody else (other) (o)**.

Owners, Group Members, and Everybody Else

- To find out information about your identity, use the `id` command.

```
[me@linuxbox ~]$ id
uid=500(me) gid=500(me) groups=500(me)
```

- When user accounts are **created**, **users** are assigned a number called a **user ID** (uid), think of it as a **username**.
- The user is assigned a **group ID** (gid) and may belong to additional groups.

Owners, Group Members, and Everybody Else

- Where does this information come from?
- Like so many things in Linux, it comes from a couple of text files. User accounts are defined in the **/etc/passwd** file, and groups are defined in the **/etc/group** file.
  - When user accounts and groups are created, these files are modified along with **etc/shadow**, which holds information about the user's password.
- For each user account, the **/etc/passwd** file defines the user (login) name, uid, gid, account's real name, home directory, and login shell.
- If we examine the contents of /etc/passwd and /etc/group, we notice that besides the regular user accounts, there are accounts for the **superuser** (**uid** 0) and various other system users.

Reading, Writing, and Executing

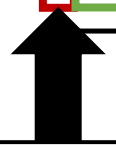
- Access rights to files and directories are defined in terms of **read** access, **write** access, and **execution** access.
- If we look at the output of the `ls` command, we can get some clue as to how this is implemented.

```
[me@linuxbox ~]$ > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-r-- 1 me    me    0 2018-03-06 14:52 foo.txt
```

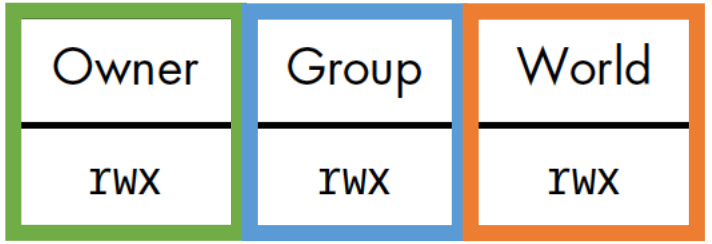
10 characters

File Attributes

```
[me@linuxbox ~]$ > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-r-- 1 me    me    0 2018-03-06 14:52 foo.txt
```



Attribute	File type
-	A regular file.
d	A directory.
l	A symbolic link. Notice that with symbolic links, the remaining file attributes are always rwxrwxrwx and are dummy values. The real file attributes are those of the file the symbolic link points to.
c	A *character special file*. This file type refers to a device that handles data as a stream of bytes, such as a terminal or /dev/null.
b	A *block special file*. This file type refers to a device that handles data in blocks, such as a hard drive or DVD drive.



Attribute	Files	Directories
r	Allows a file to be opened and read.	Allows a directory's contents to be listed if the execute attribute is also set.
w	Allows a file to be written to or truncated; however, this attribute does not allow files to be renamed or deleted. The ability to delete or rename files is determined by directory attributes.	Allows files within a directory to be created, deleted, and renamed if the execute attribute is also set.
x	Allows a file to be treated as a program and executed. Program files written in scripting languages must also be set as readable to be executed.	Allows a directory to be entered, e.g., cd *directory*.

Permission Attribute Examples

File Attributes	Meaning
`-rwx------`	A regular file that is readable, writable, and executable by the file's owner. No one else has any access.
`-rw-------`	A regular file that is readable and writable by the file's owner. No one else has any access.
`-rw-r--r--`	A regular file that is readable and writable by the file's owner. Members of the file's owner group may read the file. The file is world-readable.
`-rwxr-xr-x`	A regular file that is readable, writable, and executable by the file's owner. The file may be read and executed by everybody else.
`-rw-rw----`	A regular file that is readable and writable by the file's owner and members of the file's group owner only.
`lrwxrwxrwx`	A symbolic link. All symbolic links have "dummy" permissions. The real permissions are kept with the actual file pointed to by the symbolic link.
`drwxrwx---`	A directory. The owner and the members of the owner group may enter the directory and create, rename, and remove files within the directory.
`drwxr-x---`	A directory. The owner may enter the directory and create, rename, and delete files within the directory. Members of the owner group may enter the directory but cannot create, delete, or rename files.

chmod: Change File Mode

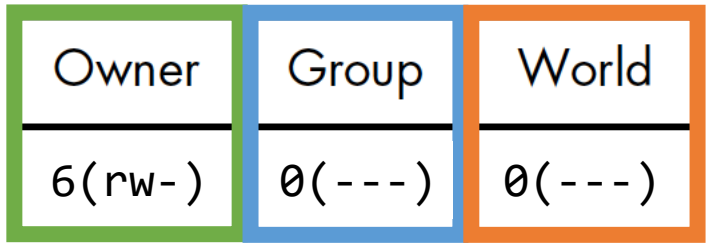
- To change the **mode** (**permissions**) of a file or directory, use the **chmod** command.
- **NOTE:** *Be aware that only the file's **owner** or the **superuser** can change the mode of a file or directory.*
- chmod supports two distinct ways of specifying mode changes:
  - **Octal number representation** or **absolute** method.
  - **Symbolic representation** or **relative** method.

File Modes in Binary and Octal

Octal	Binary	File mode
0	000	`---`
1	001	`--x`
2	010	`-w-`
3	011	`-wx`
4	100	`r--`
5	101	`r-x`
6	110	`rw-`
7	111	`rwx`

Change File Mode: *Octal Representation*

```
[me@linuxbox ~]$ > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-r-- 1 me    me    0 2018-03-06 14:52 foo.txt
[me@linuxbox ~]$ chmod 600 foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw------- 1 me    me    0 2018-03-06 14:52 foo.txt
```



Octal	Binary	File mode
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

Change File Mode: *Symbolic Representation*

- **Symbolic** notation (**relative** method) is divided into three parts.
  - **Who** the change will affect
  - **Which** operation will be performed
  - **What** permission will be set

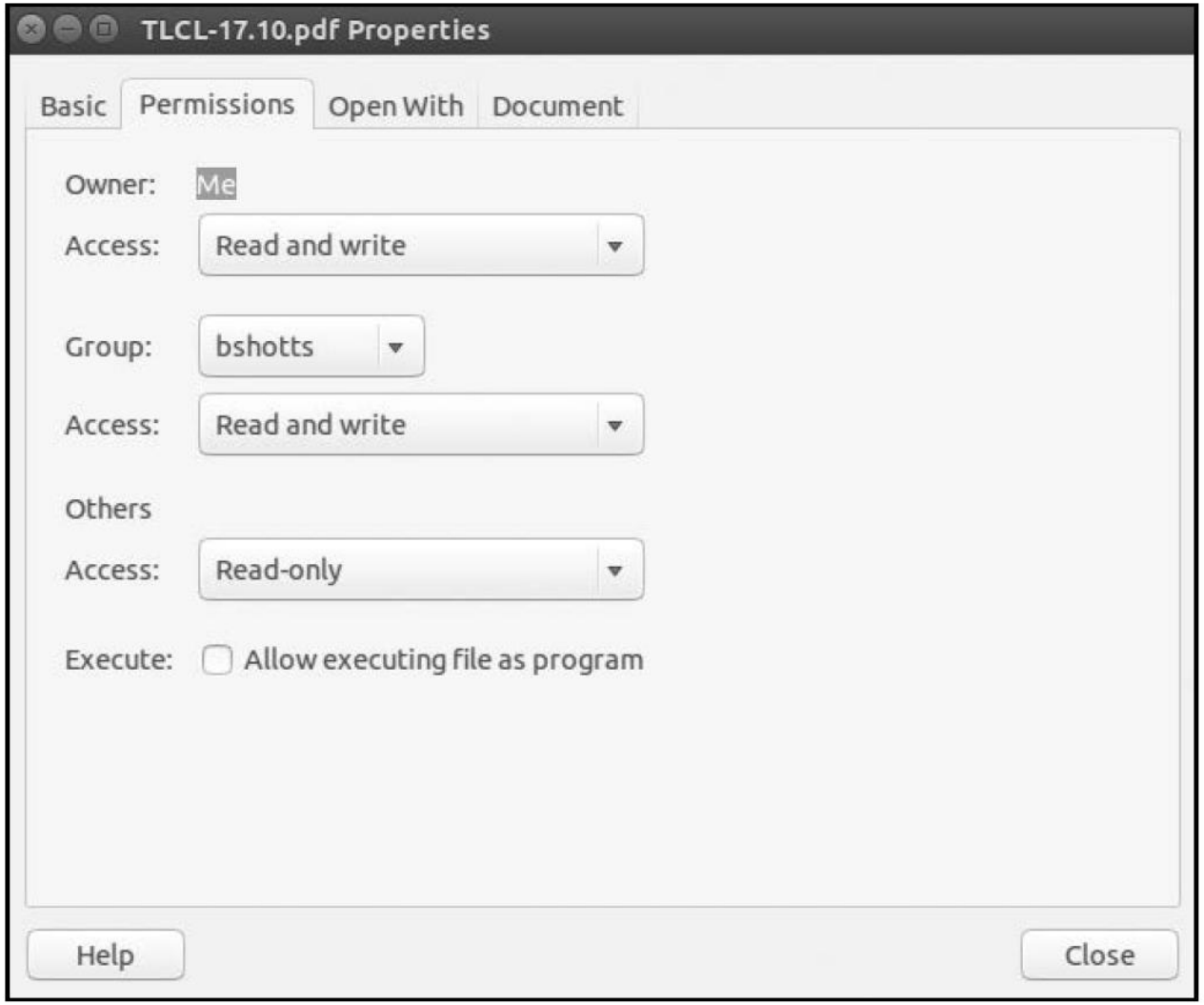
To specify who is affected, a combination of the characters **u**, **g**, **o**, and **a** is used

Symbol	Meaning
u	Short for "user" but means the file or directory owner.
g	Group owner.
o	Short for "others" but means world.
a	Short for "all." This is a combination of u, g, and o.

If no character is specified, "all" will be assumed.

Notation	Meaning
u+x	Add execute permission for the owner.
u-x	Remove execute permission from the owner.
+x	Add execute permission for the owner, group, and world. This is equivalent to a+x.
o-rw	Remove the read and write permissions from anyone besides the owner and group owner.
go=rw	Set the group owner and anyone besides the owner to have read and write permissions. If either the group owner or the world previously had execute permission, it is removed.
u+x,go=rx	Add execute permission for the owner and set the permissions for the group and others to read and execute. Multiple specifications may be separated by commas.

Setting File Mode with the GUI



`umask`: Set Default Permissions

- The `umask` command controls the **default permissions** given to a file when it is created.
- It uses octal notation to express a **mask of bits** to be **removed** from a **file's mode attributes**.

```
[me@linuxbox ~]$ umask
0002
[me@linuxbox ~]$ > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-r-- 1 me    me    0 2018-03-06 14:53 foo.txt
```

Original file mode	--- rw- rw- rw-
Mask	000 000 000 010
Result	--- rw- rw- r--

Octal	Binary	File mode
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

`umask`: Set Default Permissions

Octal digit in `umask` command	Permissions the mask will prohibit from being set during file creation
0	any permission may be set (read, write, execute)
1	setting of execute permission is prohibited (read and write)
2	setting of write permission is prohibited (read and execute)
3	setting of write and execute permission is prohibited (read only)
4	setting of read permission is prohibited (write and execute)
5	setting of read and execute permission is prohibited (write only)
6	setting of read and write permission is prohibited (execute only)
7	all permissions are prohibited from being set (no permissions)

Octal	Binary	File mode
0	000	`---`
1	001	`--x`
2	010	`-w-`
3	011	`-wx`
4	100	`r--`
5	101	`r-x`
6	110	`rw-`
7	111	`rwx`

Original file mode	--- rw- rw- rw-
Mask	000 000 000 010
Result	--- rw- rw- r--

```
[me@linuxbox ~]$ umask
0002
[me@linuxbox ~]$ > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-r-- 1 me    me    0 2018-03-06 14:53 foo.txt
```

```
[me@linuxbox ~]$ rm foo.txt
[me@linuxbox ~]$ umask 0000
[me@linuxbox ~]$ > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-rw- 1 me    me    0 2018-03-06 14:58 foo.txt
```

`umask`: Set Default Permissions

Octal digit in `umask` command	Permissions the mask will prohibit from being set during file creation
0	any permission may be set (read, write, execute)
1	setting of execute permission is prohibited (read and write)
2	setting of write permission is prohibited (read and execute)
3	setting of write and execute permission is prohibited (read only)
4	setting of read permission is prohibited (write and execute)
5	setting of read and execute permission is prohibited (write only)
6	setting of read and write permission is prohibited (execute only)
7	all permissions are prohibited from being set (no permissions)

Octal	Binary	File mode
0	000	`---`
1	001	`--x`
2	010	`-w-`
3	011	`-wx`
4	100	`r--`
5	101	`r-x`
6	110	`rw-`
7	111	`rwx`

Original file mode	`---` `rw-` `rw-` `rw-`
Mask	000 000 010 010
Result	`---` `rw-` `r--` `r--`

chown: Change File Owner and Group

- The chown command is used to change the **owner** and **group owner** of a file or directory.
- **NOTE:** Superuser privileges are required to use this command.
- The syntax of chown looks like this:

```
chown [owner][:[group]] file...
```

Argument	Results
bob	Changes the ownership of the file from its current owner to user bob.
bob:users	Changes the ownership of the file from its current owner to user bob and changes the file group owner to group users.
:admins	Changes the group owner to the group admins. The file owner is unchanged.
bob:	Changes the file owner from the current owner to user bob and changes the group owner to the login group of user bob.

`chgrp`: Change Group Ownership

- In older versions of Unix, the `chown` command changed only file ownership, not group ownership.
- For that purpose, a separate command, `chgrp`, was used.
  - It works much the same way as `chown`, except for being more limited.

- `chgrp newgroup file` - Changes the group owner of `file` to `newgroup`.

Hard and Symbolic Links

- Hard links follow the permission of their links as they represent the same `inode` object.
  - Changing the permissions on hard links changes the permissions on the original file as well.
- The permissions for the symbolic link are determined by the actual linked file, so permissions of symbolic links appears to be executable by everyone.
  - Changing the permissions on symbolic links will **not** change the permissions on the original file.

Normal File Sizes

- `ls` and `stat` command would show the number of bytes in a file.
- Size for normal text files depend on the editor. There is always some characters are appended at the end such as new lines and carriage returns.

Device File Size

- Devices are defined by type, such as '**block**' or '**character**', and '**major**' and '**minor**' number.
  - The major number is used to categorize a device.
  - The minor number is used to identify a specific device type.
- These are the two numbers precede the date in the following display:

```
brw-rw----    1 root     disk       3,   0 Mar 15  2002 /dev/hda
brw-rw----    1 root     disk       3,   1 Mar 15  2002 /dev/hda1
brw-rw----    1 root     disk       3,  10 Mar 15  2002 /dev/hda10
brw-rw----    1 root     disk       3,  11 Mar 15  2002 /dev/hda11
brw-rw----    1 root     disk       3,  12 Mar 15  2002 /dev/hda12
brw-rw----    1 root     disk       3,  64 Mar 15  2002 /dev/hdb
brw-rw----    1 root     disk       3,  65 Mar 15  2002 /dev/hdb1
brw-rw----    1 root     disk      22,   0 Mar 15  2002 /dev/hdc
brw-rw----    1 root     disk      22,  64 Mar 15  2002 /dev/hdd
```

Symbolic and Hard link File Sizes

- Hard link files have the same size as the original file.
- Symbolic link files size correspond to the size of the symbolic link – not the content of the original file.

Time Stamps

- A file has several **time stamps:**
  - **Last modification time**: which is the time the file was **last modified and saved**. This is the default time displayed by `ls -al` command.
  - **Last access time**: which is the time the file was **last accessed or viewed**. Use the `ls -alu` command.

Hidden File Names

- A Linux file name can be up to **255 characters long** and is made of any characters.
- A **dot** has no special meaning in a file name except if it is the first character then the file is a **hidden file**.

Important Notes for Lab Solution Submission

- **What do you need to submit for each lab?**
  - Reply to the lab message on Ritaj with the following two items:
    - **[OPTIONAL]** *Your lab solution on a separate document.*
    - **[MUST]** *Script log file*; the `script` log file will prove that you have literally solved the lab tasks yourself as it contains timestamps as well as your username and machine name.
  - If you do not send your `script` log file, then you will be rewarded with **zero**.
- **How to generate the script log file?**
  - While solving lab tasks, perform the following per each task part:
    - `script –a UniversityID_FirstnameLastName.log`
    - solve the lab task using proper command, scripts, etc.
    - `exit`