Student Name: ______________________________ **Lab Section:** _______

/3
marks

Linux File System Permissions (modes) - Part 1

Due Date - Upload to Blackboard by 8:30am Monday March 12, 2012

Submit the completed lab to Blackboard following the Rules for submitting Online Labs and Assignments. You must upload **two** files for this submission. You must always upload **both** files when you submit. Both.

Commands, topics, and features covered

Use the on-line help (**man** command) for the commands listed below for more information.

- **`ls -lid`** – (list structure, **long** version, **inode**, **directory**) See the permissions of an inode
- **`chmod`** – (change mode) Change the permissions (mode) on an existing inode (file, directory, etc.)
- **`chown`** – (change owner) Change the owner and/or group of an existing inode (needs **root** privilege)
- **`umask [`*`value`*`]`** – (user mask - shell built in) Display or change the octal **umask** value for this shell
- **`su [-] [`*`user`*`]`** – (substitute user) Become another user (default **root**), with that user's permissions
- **`whoami`** – (who am I?) Display current account userid
- **`id [`*`user`*`]`** – (identity) Display account userid and all groups
- **`useradd`** ***`userid`*** – create a new login user account named **userid** with home directory

Correct user, command lines, and command output

- Parts of this lab are done as different **ordinary**, non-root users. Other parts are done as the **root** user. Pay attention to which part is done by which user. Your prompt will tell you if you are the **root** user.
- Some answer blanks require you to enter **command lines**. Do **not** include the shell **prompt** with your command lines. Give only the part of the command line that you would type yourself.
- Make sure you know the difference between a command **line** (which is what you type into the shell) and command **output** (which is what the command displays on your screen).

Viewing Permissions (modes) with the `ls -l` command

Permissions are stored in each inode. They control who is allowed to access and modify a file system object (inode) such as a file or directory. Another word for permissions is **mode**, and the command **`chmod`** that changes permissions is an abbreviation of "**change mode**". Only the **owner** of an inode can change its mode. We often casually say "file" permissions, but permissions apply to **each** inode whether file, directory, or other.

There are **nine** permissions altogether, **three** sets of **three** read/write/execute permissions: one set for the inode's **user/owner**, one set for the **group**, and a third set for all **other** users. When performing a long directory listing, **`ls -l`**, the inode's permissions (mode) appear as **nine** characters (three sets of read/write/execute) in the **first** field (column) of each output line, *after* the inode's **type** indicator character. The **second** field is a link count. The **third** field is the user/owner of the inode. The **fourth** field is the group to which the inode belongs. The **fifth** field is the date/time the inode was modified. The **last** field is a name for the inode. (Inodes may have multiple names.) If you use the **`-i`** option, the inode numbers appear at the start (left) of the output lines:

```
[user@host ]$ ls -il
555 -rw-r----- 3 user1 group1  123 Nov 12 14:14 fileone
928 drwxrwxr-x 2 user1 group1 4096 Nov 12 14:14 directoryone
382 lrwxrwxrwx 1  root   root   30 Oct 13 12:39 symlink -> ../some/place
```

Above, inode `555` is a plain **file** named **`fileone`** owned by **`user1`** and in group **`group1`** with size `123` and link count of `3`. Inode `928` is a directory named **`directoryone`** and a inode `382` symbolic link named

`symlink`. The permissions and owners of symbolic links are **ignored**; all that matters are the permissions on the **inode being linked to**. Symbolic links allow **directories** to appear to have multiple names.

The "inode type" character

The **first character** before the nine permission characters identifies the **type** of the inode that this name is attached to. The three most common inode types are:

- **`-`** (a hyphen/minus/dash) for a **regular** file inode
- **`d`** for a **directory** inode
- **`l`** (lower-case L) for a **soft** or **symbolic link** (soft link) linking to a **pathname** (not to an inode!)

In the example above, **`fileone`** is typed as a regular file (the type character is a leading **`'-'`**); **`directoryone`** is a directory (a leading **`'d'`**); **`symlink`** is a symbolic link (a leading **`'l'`**) that points to pathname **`../some/place`** The permissions and owners of symbolic links are **ignored**; all that matters are the permissions on the **inode being linked to**.

Three sets of three permissions: 3 user/owner, 3 group, and 3 other

The **nine** characters following the type character show the **three** sets of **three** read/write/execute access permissions that apply to **user/owner**, **group**, and **other**. Each of the **three** sets contains **three** characters indicating which of these **three** permissions is allowed for each set:

- **`r`** means **read** permission (can access the content of the inode)
- **`w`** means **write** permission (can change the content of the inode)
- **`x`** means **execute** permission for files and **search** permission for directories

The three characters are always written as three in **`rwx`** order. In a set of three permission characters, a hyphen/minus/dash character **`'-'`** replaces a letter if the corresponding permission is **not** granted, e.g. **`rw-`**

```
555 -rw-r----- 1 user1 group1  123 Nov 12 14:14 fileone
```

After the inode **type** character (a dash means a plain file), the **first** three characters of the nine-character mode are the **`r,w,x`** permissions that apply to the **user (owner)** of the inode. Above, **`fileone`** has mode **`rw-`** (read, write, NO execute) for **`user1`**. The **second** three characters **`r--`** are the **`r,w,x`** permissions that apply to users who are not the owner but are in the same **group** as the inode; the **last** three characters **`---`** are the **`r,w,x`** permissions that apply to everyone else (people who are not the user/owner and are not in the group). A hyphen/minus/dash in any of the three positions means **NO** permission, so "**`---`**" means that **others** have **no** read, **no** write, and **no** execute (**no** permissions at all) on this file inode.

Symbolic (letter) and numeric (octal) permissions (mode)

Permissions (mode) can be represented in two ways: **symbolic** (three letters) or **numeric** (one octal digit):

- Symbolic mode (letters), e.g. **`rwx`** or **`r-x`** or **`r--`**
  - **`r (read)`**
  - **`w (write)`**
  - **`x (execute/search)`**
- Numeric, absolute, or octal mode, e.g. **`7`** or **`5`** or **`4`**
  - **`r (read)`** **`r--`** is $100_{(base\ 2)} = 2^2 = 4_{(octal)}$
  - **`w (write)`** **`-w-`** is $010_{(base\ 2)} = 2^1 = 2_{(octal)}$
  - **`x (execute)`** **`--x`** is $001_{(base\ 2)} = 2^0 = 1_{(octal)}$

Each of the three sets of symbolic permissions (user/owner, group, other) can be summarized by a single octal digit by adding up the three numeric **`rwx`** values using the three weights (4,2,1) given above:

• Example 1: **`rwx`**	corresponds to digit **`7`** because **`r`** is **`4`**, **`w`** is **`2`**, and **`x`** is **`1`** so		**`4+2+1=7`**
• Example 2: **`r-x`**	corresponds to digit **`5`** because **`r`** is **`4`** and **`x`** is **`1`** so		**`4+0+1=5`**
• Example 3: **`-wx`**	corresponds to digit **`3`** because **`w`** is **`2`** and **`x`** is **`1`** so		**`0+2+1=3`**
• Example 4: **`---`**	corresponds to digit **`0`** because no permissions are set so		**`0+0+0=0`**

The full set of **nine** permission characters can then be grouped and summarized as **three** octal digits:

• Example 5: **`rwxr-x-wx`**	is	**`rwx|r-x|-wx`**	and corresponds to the three digits	**`753`**
• Example 6: **`---r----x`**	is	**`---|r--|--x`**	and corresponds to the three digits	**`041`**
• Example 7: **`---------`**	is	**`---|---|---`**	and corresponds to the three digits	**`000`**
• Example 8: **`rwxrwxrwx`**	is	**`rwx|rwx|rwx`**	and corresponds to the three digits	**`777`**

Make sure you always write exactly **nine** characters when writing symbolic permissions. Exactly **nine**. Do **not** include the leading "**inode type**" character when listing the **nine** characters of symbolic permissions.

Changing permissions with the chmod command

Change permissions of an inode using the **`chmod`** (change mode) command:

➢ **`chmod`** ***`mode`*** ***`pathnames...`***

Supply at least two arguments:

1. one permission ("mode") argument (that may contain multiple permissions) and
2. one or more pathnames of inodes for which the access permissions are to be set or modified

Only the user/owner of an inode (and the super-user) can change its permissions. Examples:

• **`chmod u=rwx,g=rx,o=wx pathname`**	(set to **`rwxr-x-wx`** or **`753`** octal)
• **`chmod 753 pathame`**	(set to **`753`** or **`rwxr-x-wx`** symbolic)
• **`chmod u+r pathame`**	(*add* user/owner read permissions)
• **`chmod go-rwx pathame`**	(*remove* all permissions for group and others)

Note that *adding* and *removing* permissions only work for **symbolic** permissions and only affect the *given* permission and don't change any of the other permissions. Octal permissions always affect *all* the permissions.

1 Exercise: Conversion between symbolic mode and octal mode

For each **nine**-character symbolic permission, give the equivalent **three**-digit **octal** permission and the **symbolic** permissions that apply to each of **User/Owner**, **Group**, and **Other**:

Row	Symbolic Mode	Octal Mode	User/Owner	Group	Other
1.	**`rwxrw-r-x`**	_	_	_	_
2.	**`r---wx-w-`**	_	_	_	_
3.	**`--x------`**	_	_	_	_

2 Exercise: Conversion between octal mode and symbolic mode

For each **three**-digit octal permission, give the equivalent **nine**-character **symbolic** permission and the **symbolic** permissions that apply to each of **User/Owner**, **Group**, and **Other**:

Row	Octal Mode	Symbolic Mode	User/Owner	Group	Other
1.	**`000`**	_	_	_	_
2.	**`001`**	_	_	_	_

3.	`020`	_	_	_	_
4.	`300`	_	_	_	_
5.	`004`	_	_	_	_
6.	`050`	_	_	_	_
7.	`600`	_	_	_	_
8.	`007`	_	_	_	_
9.	`715`	_	_	_	_

Did you read **all the words** before completing the above exercise? Symbolic or numeric? Read all the words.

3 Exercise: Testing permissions

1. While logged in as a regular user, execute the following three commands to create a testing directory:

```
[user@host ]$ cd ; mkdir -p lab06/top ; cd lab06
```

For each row of the table below, repeat these next two steps (Read All The Words):

2. Go to the **`lab06`** directory and change the permission of the **`top`** directory using the **`chmod`** command given in the second column of the table. Execute this **`chmod`** command in the **`lab06`** directory.
3. Next, for the row you are working on, execute the commands listed across the top of the table for that permission level. For each command enter in the table whether the command line executes successfully or not. Enter **PD** for Permission Denied; **OK** for success. The commands to try are:
   a) **`cd top`** followed immediately by "**`cd ..`**" if it worked (gave no error messages)
   b) **`touch top/file`** followed immediately by "**`rm top/file`**" if it worked
   c) **`mkdir top/dir`** followed immediately by "**`rmdir top/dir`**" if it worked
   d) **`ls -l top`**

- You will carry out Steps 2 and 3 for **each** of the rows of the table below.
- Before you run each **`chmod`** command in the table below, ensure your current directory is **`lab06`**.
- If the **`cd`** into **`top`** works (no error), follow it immediately with "**`cd ..`**" to return up to the **`lab06`** directory again, otherwise you will be in the wrong directory for the next command in the table.
- If the **`touch`** works (no error), immediately remove the file you just created.
- If the **`mkdir`** works (no error), immediately remove the directory you just created.

Row	**Command line**	**`cd top`**	**`touch top/file`**	**`mkdir top/dir`**	**`ls -l top`**
1.	**`chmod 000 top`**	_	_	_	_
2.	**`chmod 100 top`**	_	_	_	_
3.	**`chmod 200 top`**	_	_	_	_
4.	**`chmod 300 top`**	_	_	_	_
5.	**`chmod 400 top`**	_	_	_	_
6.	**`chmod 500 top`**	_	_	_	_
7.	**`chmod 600 top`**	_	_	_	_
8.	**`chmod 700 top`**	_	_	_	_

Did you read **all the words** before completing the above exercise? Your table must contain only **OK** or **PD**.

4 Exercise: Minimum Permissions

In the table below, give in **symbolic `rwx`** form the **minimum** permissions a non-root user requires to successfully complete the actions listed in the left column below. "**Minimum**" means the *least* amount of **`rwx`** permissions needed. How many of **`rwx`** can you *take away* and still perform the given action *successfully*? Test your guess to make sure you are correct. Recall that directories store only names and inode numbers, not data.

```
1. [user@host]$ cp       srcdir/srcfile  targetdir/
2. [user@host]$ mv       srcdir/srcfile  targetdir/
3. [user@host]$ ln       srcdir/srcfile  targetdir/
4. [user@host]$ rm       srcdir/srcfile
5. [user@host]$ cat      srcdir/srcfile
6. [user@host]$ date  >> srcdir/srcfile
```

Action	on the source directory	on the source file	on the target directory
1. copy a file	_	_	_
2. move a file	_	_	_
3. link to a file	_	_	_
4. delete a file	_	_	**N/A**
5. read a file	_	_	**N/A**
6. modify an existing file	_	_	**N/A**

Now go back and read all the words of this question, including the word "**symbolic**" in the first line.

5 Exercise: setting permissions using chmod

Create the following **structure** containing files and directories with the following **permissions**. All files and directories should be **owned by you**, not by the **root** user. **Numeric** permissions required below will be **three-digit** octal numbers. When you are applying permissions to directories as a non-**root** user, be careful not to lock yourself out of directories by using **`chmod`** on them too soon - first create all the file and directory structure from the *top* down and then apply restrictive permissions afterward from the *bottom* up.

1. Create directory **`lab06/top06`** under your home directory.
   a) Directory **top06** has no permissions for *others*.
   b) The *group* can use **ls** to see the content, but cannot create files or **cd** into the directory.
   c) The *user* has full permissions.
   Give the **numeric** permission (see above) for directory **top06**: ______________________________
2. Underneath directory **top06** create *three* single-letter **directories** named: **`u g o`**
   a) Directory **u** has no permissions for *group* or *other*. The *user* can **cd** into it, but cannot create any new content nor use **ls** to see any files in it.
   Give the **numeric** permission for directory **u**: ______________________________
   b) Directory **g** has no permissions for *user* or *other*. The *group* can **cd** into it and use **ls** in it, but cannot create any new content.
   Give the **numeric** permission for directory **g**: ______________________________
   c) Directory **o** has no permissions for *user* or *group*. *Others* have full permissions.
   Give the **numeric** permission for directory **o**: ______________________________
3. Underneath directory **u** create *three* single-letter **files** named: **`r w x`**
   a) None of the files have any permissions for *group* or *other*.
   b) File **r** has read permission (only) for the *user*.
   Give the **numeric** permission for file **r:** ______________________________

c) File **w** has write permission (only) for the *user*.
   Give the **numeric** permission for file **w:** ______________________________
d) File **x** has execute permission (only) for the *user*.
   Give the **numeric** permission for file **x:** ______________________________

4. Underneath directory **g** create *three* two-letter files named: `rw` `wx` `rx`
   a) None of the files have any permissions for *user* or *other*.
   b) File **rw** has read and write permission (only) for the *group*.
      Give the **numeric** permission for file **rw:** ______________________________
   c) File **wx** has write and execute permission (only) for the *group.*
      Give the **numeric** permission for file **wx:** ______________________________
   d) File **rx** has read and execute permission (only) for the *group*.
      Give the **numeric** permission for file **rx:** ______________________________
5. Underneath directory **o** create *three* three-character files named: `rwx` `???` `***`
   (Some characters are meta-characters that are special to the shell unless you quote them!)
   a) None of the files have any permissions for *user* or *group*.
   b) File **rwx** has full permissions for *other.*
      Give the **numeric** permission for file **rwx:** ______________________________
   c) File **???** has no permissions for *other*.
      Give the **numeric** permission for file **???:** ______________________________
   d) File *** has only *read* permission for *other.*
      Give the **numeric** permission for file *****:** ______________________________

6 Upload: Check your work and submit the `lab06marks.txt` file

Into the **`lab06`** directory, download the **`lab06check`** program (from the Class Notes), make it executable, and run it as the **root** user with the **HOME** variable set to the **absolute path** of **your** own **HOME** directory:

```
[user@host lab06]$ su root
[root@host lab06]# chmod +x lab06check
[root@host lab06]# HOME=/home/user      (use your own HOME directory here)
[root@host lab06]# ./lab06check
```

This program will check your work, assign you a mark, and put the mark into file **`lab06marks.txt`** so that you can upload it to Blackboard as part of this assignment. You may run the **`lab06check`** program as many times as you wish, to correct mistakes and get the best mark, before you upload the final marks file.

Linux File System Permissions - Part 2

1 Create two new non-root user accounts

For this section you will require two more ordinary user (non-root) accounts. To create the two accounts follow these steps below (you need **root** privileges to create accounts - become the **root** user first):

1. `[root@host ]# useradd homer`
   The above creates a new "**`homer`**" login account and home directory. The account has no password yet.
2. `[root@host ]# passwd homer`
   The above sets **`homer`**'s password. If you do not type the username after the **`passwd`** command, you are changing the **root** password. Do **not** change your **root** password! Change **`homer`**'s password.
3. Repeat the above steps to create another account named **`flanders`** and give it the same password.
4. Record the account information for the two new accounts by typing: **`id homer ; id flanders`**

____

____

5. Give the absolute pathname of the **flanders** account home directory: ______________________
6. Give the **numeric** permissions of the above home directory: ___________________________

2 Creating a Public Directory in the system ROOT

We will create a **public** directory in which **any user** can create files. The directory will allow any user to create names in it (or remove names). Recall that the permissions on a directory are not the same as the permissions on the inodes named in the directory. Permission to change names does not grant permission to change content.

1. With **root** privileges create a directory called **public** under the **ROOT** directory: **`/public`** (***NOT /root/public***) and record the command line: ______________________________
2. Give a **command line** that will show the permissions of **only** the new **`/public`** directory:
____
3. What are the current **numeric** permissions for the **`/public`** directory: ______________
4. Record the **owner** and **group** of the **`/public`** directory: ______________
5. Give **`/public`** **full access permissions** for **everybody** and record the exact **command** line you used:
____
6. What are the resulting changed **numeric** permissions for **`/public`**: ______________

3 Using the Public Directory

In the next steps, where command lines are required, **do** each command and **record** the command line used:

1. What command line lets you become the **`flanders`** user: ______________
2. What command verifies that you are currently the **`flanders`** user: ______________
3. What command line creates a new file **`/public/flanfile`**: ______________
4. Record the **owner** and **group** of the new **`flanfile`** file: ______________
5. What are the current **numeric** permissions for **`flanfile`**: ______________
6. What command line removes (only) all **other** permissions from **`/public/flanfile`** and *does not change any existing **user** or **group** permissions*: ______________
7. What are the resulting **numeric** permissions for **`flanfile`**: ______________
8. As user **`flanders`**, append the **`date`** to the new **`flanfile`** file. Record the full **command line** here:
____
9. What **command line** shows that the size of **`flanfile`** is **29** bytes: ______________
10. As the **`homer`** user, try to display the contents of the **`flanfile`** file and record the **error message**:
____
11. As the **`homer`** user, rename the **`flanfile`** file owned by **`flanders`** to have the new name **`foo`**, and give the output of **`ls -il /public/foo`** showing that the renamed **`foo`** file is still owned by **`flanders`** :
____
12. As the **`homer`** user, remove the name **`foo`** for the file owned by **`flanders`**. Why can you both **rename** and then **delete** this file that you don't own and can't read? (*Hint*: Names store separately from content.)
____
____

4 Changing ownership

With **root** privileges, create an empty file and then change the **owner** and **group** to **`homer`** and **`homer:`**

```
[root@host]# touch /public/foo ; chown homer:homer /public/foo
```

a) Give the output of **`ls -il /public/foo`** showing the **`homer homer`** owner and group:
____
b) Become the **`flanders`** user and try to append the **`date`** to **`/public/foo`**. Can you do it? _______
c) Become the **`homer`** user and try to append the **`date`** to **`/public/foo`**. Can you do it? _________
d) As **root**, set (only) the **group** and group **permissions** so that *both* **`homer`** *and* **`flanders`** can read and write **`foo`** but **others** cannot. Test it as *both* users. Give the output of **`ls -il /public/foo`**:
____

Linux File System Permissions - Part 3

Shell umask and permissions for new files and directories

The shell **umask** value restricts the permissions given to **new** files and directories when they are **first created**. Without the **umask** value, or with a zero value, a new file would be created with full permissions **666** and a new directory with full permissions **777**. The **umask** *masks* out permissions - each bit of the **umask** turns **off** the corresponding permission in the newly created file or directory. A **umask** value of of 777 turns off **all** permissions on new files and directories; a **umask** of **000** doesn't turn off any permissions (not recommended!).

Note that **masking** is *not* the same as **subtracting**, e.g. **`666`** masked with **`001`** is still **`666`** and **`666`** masked with **`003`** is **`664`**. The mask turns **off** permission bits. If they are already off, the **umask** makes no change:

- **`rw-`** (**6**) ***masked*** with **`--x`** (**1**) is **`rw-`** (**6**) because the **`x`** bit in the mask does not change any permissions
- **`rw-`** (**6**) ***masked*** with **`-wx`** (**3**) is **`r--`** (**4**) because only the **`w`** bit is changed (turned off) by the mask

Each process, and thus each shell, has its own **umask** value. Different Linux distributions set different default (at login) **umask** values. The values in your particular distribution of Linux may not be the same as other distributions. The values set by the system administrator may differ from the distribution defaults. Do not rely on the **umask** having any standard value.

a) What built-in shell command displays or sets the **umask** value: ____________________
b) What option to the **`su`** command does a full "**login shell**" login: ____________________
c) Give the default (at full login) **umask** for an **ordinary** user account in Fedora 12: ________________
d) Give the default (at full login) **umask** for the **root** account in Fedora 12: _________________________

For each row of the following table **set** the given **umask** and then fill in the four new permission columns:

e) **Set** and use the **given umask** in column two before filling in the permissions in the rest of the row.
f) Based on the **umask** in column two, write down the **permissions** that would be **given** to a new **file** and a new **directory**. Give both the **symbolic** and **numeric** forms. You can create a new file and a new directory to verify your answer, but you should calculate and know the answer without needing to create anything. Using the given **umask**, what would be the permissions set on a **new** file and a **new** directory?

Row	**Set your umask to this value and then fill in the fields on the right:**	**create *new* file permissions**		**create *new* directory permissions**	
		symbolic	**numeric**	**symbolic**	**numeric**
1.	**`umask 0001`**	_	_	_	_
2.	**`umask 0002`**	_	_	_	_
3.	**`umask 0003`**	_	_	_	_
4.	**`umask 0022`**	_	_	_	_
5.	**`umask 0077`**	_	_	_	_
6.	**`umask 0777`**	_	_	_	_

After you are finished the above exercise, exit your shell or reset your **umask** back to the normal **umask**. Do **not** continue to use a shell that has a **umask** of **0777**.

7. **True or False** - the **umask** can change the permissions of **existing** files and directories? ____________
8. What command name changes the permissions of **existing** files and directories: ______________
9. What value **umask** gives the **owner** and **group** full permissions on new files and directories: _______
10. What value **umask** gives only the **owner** full permissions on new files and directories: __________