Linux Essentials – Module 3

FILE OPERATIONS

MODULE OBJECTIVE

- Use file operation utilities to copy, rename, delete, and print files; and to manage the directory structure; and do so in regards to access permissions

LESSON OBJECTIVES

- Manipulate files with the commands "**cp**", "**mv**", "**rm**"
- Manage the directory structure with the commands "**mv**", "**mkdir**", "**rmdir**"
- Understand file/directory access permissions, and be able to modify them.

NOTE: Many of the basic concepts related to file operations to be discussed in the next few pages are pretty straightforward and thus most of the learning mettle of these topics will be presented in lab rather in the textual discussion.

Copy, move and remove files

The **cp** command will copy a file to another location while changing the filename, resulting in an original and a copy of that file. The **mv** command will rename the file, leaving it in the same location or a new location, resulting in only one version of the file. The "**rm**" command will delete (remove) the command from the system.

Coping Files with cp

Copy a file when you want to make a new version of it while retaining the old Version. The file listed first is the original file followed by the name of the new file being created (cp {from} {to}).

$ **cp file1 file2**
$ **cp file3 /tmp/filex**
$ **cp ../fileA fileB**
$ **cp /etc/passwd myfile-passwd**
$ **cp /etc/passwd .**

As always we can subsitiute relative addressing for absolute anywhere we desire such as using the "." (dot) to indicate to copy to the current directory as in the last example above:

Renaming Files with mv.

This works like cp, except the new copy takes the place of the old. And thus the command can have the affect of moving a file to a wholly new location, or renaming the file, or a combination thereof. (mv {from oldfile} {to newfile})

$ **mv myfile-passwd passwd.myfile**

Removing Files with rm

$ **rm passwd.myfile**
$ rm /tmp/filex

Note: Using the "**-i**" option with any of these commands on the previous page will prompt you before overwriting an existing file.

$ **rm –i somefile**
(you'll be asked if you are sure)

Creating a new (sub)directory

$ **mkdir mydir**
$ mkdir ../myotherdir
$ mkdir /tmp/someotherdir

Removing a directory

$ **rmdir /tmp/someotherdir**

If a directory being removed is non-empty the system will require you to empty it of all content (files, sub-directories, etc…) before removing it. You can get around this by using a version of the '*rm*' command the 'f' (force) and 'r' (recursive) switches to force removal of all file and recursively all sub-directories.

$ **rm –rf mydir**

Remote Coping of Files with scp

The are a number of network utilities which allow a user to copy files between connected system. The "*ftp*" utility has been the workhorse in this for many many years. *ftp* is an interactive command, meaning that it takes several steps and answering prompts and supplying commands to complete the process. Though perhaps still the most commonly used command for a machine to machine file copy, we won't go into here for one simple reason – it is insecure. The NWS has proclaimed that sites should migrate away from using ftp to file copy routines that are apart of the Open-ssh (secure shell) suite. *scp* is it. It is secure in that unlike ftp the transmission of the data and all keystrokes is encrypted. Furthermore *scp* is actually easier to use, as its syntax and operation is very much like *cp*, except that we are transferring files between machines and thus at a minimum the remote machine needs identified as part of the command input:

$ s**cp file1 machineZ:/tmp/file1**

The above command copies file1 from the current working directory on the local machine to the /tmp directory on a machine named machineZ. Below is an example using ip-address instead of the machines hostname.

$ **scp ../fileA 192.168.21.242:/home/data**

The direction or the copy can be reversed. As you may have gathered the syntax is : scp {filefrom} {fileto} – remote location specification can occur for either the from or the to.

$ s**cp ntc242.nwstc.noaa.gov:/home/student1/Afile /tmp**

In the above example we are coping the remote file Afile to the local /tmp directory. As always we can subsitiute relative addressing for absolute anywhere we desire such as using the "." (dot) to indicate to copy to the current directory:

$ s**cp ntc242.nwstc.noaa.gov:/home/student1/Afile /tmp**

******* The above discussion only show examples of how the scp command can be used in practice; Understand for any network operation to succeed the system administrator has to ensure that network connectivity is in place between the two systems, and the appropriate network services are up and running (the discussion of which is beyond the scope of this course).

***** With all file operations commands the ability to perform the desired operation is dependent upon your user account having the authority to work with the specified files and/or directories in the ways in which you wish to work with them. You have to have 'permission' to work with files and directories, and so this seems a good time to go there.**

FILE PERMISSIONS

OVERVIEW

Part of the power of LINUX is in the access and security (**permissions)** for files and directories, allowing the owner of files the ability to work with and also grant or deny access to others to those files and directories. Permissions of files and directories are just another part of security on UNIX systems. Knowing how to read these permissions and what they mean are essential for the user to ensure who has access to their files, thus ensuring the integrity of the user's files and data. We will see how to use the **ls** command to list the file and directory permissions and the **chmod** command to change these permissions.

To have a good understanding of how these permissions work, the user must understand **Users and Groups**. A user is lie you of course - a single unique logon account. When you as a user create a file you are said to be the owner of a file. You saw (and will see again) this as one of the fields in the "ls –l" command (the 3rd field in the display). A group is just as it's sounds – a group of users - a set of user accounts that have something in common. For example you may have a group called "hydro" of which all the users who are actually hydrologists belong. Or perhaps a group called "mids" of which all the user accounts which belong to mid-shift workers are assigned. User accounts can be members of multiple groups and thus if there was a hydrologist that worked mids, his/her user account would likely be made a member of both groups. Why put user accounts in groups ? . Well, it stands to reason that there groups of users may need access to a common set of files that – e.g. the hydrologists will likely share files that relate to river level for example. They can thus get access to files, by virtue of being group members, even though some of them won't be the actual owner (e.g. creators) of the files. What group has access to a specific file or directory can also be viewed with the 'ls -l' command (the 4th field of the display)

Once the user understands what permissions are, and how they are used, they should know how to change these permissions to limit or grant file/directory access to those who need it. With this understanding, understanding how and why you can change the ownership of files and directories, will be simple with the **chown** command.

Listing Directory and File Permissions

Earlier in the course we discussed using the **ls** command to list filenames, and using **ls -l** command to get a long listing of file information including the file permissions. At this point we are only going to concern ourselves with the first part of the listing, the **File Type and Permissions**.

The **first 10 characters** are the **file type** and **permissions**. File types, the first character, can be a **dash (-)** for a file, or the **letter (d)** for a directory, or the **letter (l – ell)** for a link, or the letters (b, c, p, n, and s) can be further researched with the command "**man ls**".

At this point we want to discuss the user access permissions which are the remaining 9 characters, broken into groups of 3 (called octets for the octal base 8 numbering system).

```
$ ls -l
 Total 62
drwxrwxr-w  2   zarf   student      1024   Sep 30  10:38    front_porch
-rw-rw----    1   zarf   student    19987   Sep 30  10:38    grade_me.sh
-rw-rw----    1   zarf   student     4875   Sep 30  10:38    lab1.txt


  -   r w -  r w -  - - -     1       zarf    student   4875    Sep 30  10:38    lab1.txt
type  owner  group  other    links   owner    group      size       mod date            filename
```

So the access mode of a file consists of these 4 parts:

1) file type
2) access permissions **r**ead, **w**rite or e**x**ecute for owner, group, and other
3) the owner of the file (UID)
4) the group of the file (GID)

The file type is displayed in the first character of the first field of the output from the command **ls -l** . Common file types are: "d" = directory, "-" = regular file, and "l" = symbolic link. There may be other characters used "b,c,p,s", but our concern is the "d and -" characters.

The next nine characters of the first field are interpreted as three sets of three bits each which identify access permissions for the file owner, file group, and others (everyone else).

Three classes of users can access files and directories: owner, group, and other. For each of these classes of users, there are three types of access permissions: read, write, and execute. The access permissions on a file or directory specify how it can be accessed by the owner, group, and other users classes.

A Comparison of Permissions for Directories and Files

Permissions	**Means This For a Directory**	**Means This For a File**
read (r)	Users can view names of files and directories in that directory.	Users can view the contents of the file.
write (w)	Users can create, rename, or remove files or directories contained in that directory.	Users can change the contents of the file.
execute (x)	Users can position themselves within that directory with the 'cd' command.	Users can execute (run) the file (if it is an executable file or script) by typing the filename at the command line prompt.

You should always be aware of the permissions assigned to your files and directories. Check your files and directories periodically to make sure appropriate permissions are assigned. If you find any unfamiliar files in your directories, report them to the system administrator or security officer.

Some examples may help:

All of the following examples assume that user student1 and user student2 are in the group unixintro. The directory sample_dir is in /home/student1.

Example: drwx rwx --- 2 student1 unixintro 1024 Jun 11 14:36 sample_dir
student2 can: **cd**, **ls -l** , **cp**, **rm**, **touch**, **vi** and execute files

Example: drwx r-x --- 2 student1 unixintro 1024 Jun 11 14:36 sample_dir
student2 can: **cd**, **ls -l** , **cp**, and execute files
student2 can not: **rm**, **touch** or **vi** files

Example: drwx rw- --- 2 student1 unixintro 1024 Jun 11 14:36 sample_dir
student2 can: **ls** files
student2 can not: **cd**, **ls -l**, **cp**, **rm**, **touch** or **vi** files

Example: drwx -wx --- 2 student1 unixintro 1024 Jun 11 14:36 sample_dir
student2 can: **cd**, **rm**, **touch**, **vi**, **cp** (without wildcards) and execute files
student2 can not: **ls**, **ls -l**, **cp** files

Example: drwx r-- --- 2 student1 unixintro 1024 Jun 11 14:36 sample_dir
student2 can: **ls**
student2 can not: **cd**, **ls -l**, **cp**, **rm**, **touch**, **vi** or execute files

Example: drwx -w- --- 2 student1 unixintro 1024 Jun 11 14:36 sample_dir
student2 can not: **cd**, **ls**, **ls -l**, **cp**, **rm**, **touch**, **vi** or execute files (no permissions)

Example: drwx --x --- 2 student1 unixintro 1024 Jun 11 14:36 sample_dir
student2 can: **cd**, **cp**, **touch** existing files, **vi** existing files or execute files
student2 can not: **ls**, **ls -l**, **rm**, **touch** new files or **vi** new files

*** Be sure to look over the above examples carefully, as understanding of them is quite important.

Changing File Permissions

File permissions can only be changed by the file's owner or the Superuser (root). Changing file permissions is done by using the **chmod** command.

The **chmod** command allows you to specify permissions in two different ways: **symbolic** or **numeric**.

SYMBOLIC PERMISSIONS

Syntax: chmod [who] operator [permissions] filename

who **u**ser, **g**roup, **o**ther, **a**ll
operator + (add) - (remove) = (set equal to)
permission **r**ead, **w**rite, e**x**ecute

Original permissions:	mode	user	group	other
	-rw-r--r--	rw-	r--	r--

chmod u+x,g+x,o+x filename OR chmod a+x filename

Both of these commands will add the permission x to **u**ser, **g**roup, and **o**ther

Result permissions:	mode	user	group	other
	-rwxr-xr-x	rwx	r-x	r-x

Example: chmod g+w,o-x filename

adds w to **g**roup, and removes x from **o**ther

Result permission:	mode	user	group	other
	-rwxrwxr--	rwx	rwx	r--

Example: chmod u=rwx,g=rx,o=r filename

Overlays the current permissions with the equals (=) permissions

Result permission:	mode	user	group	other
	-rwx r-x r--	rwx	r-x	r--

Numeric Method:

(This next idea may come across as a big “huh” . But don’t fret too much – if you get it great – if not, the literal method hopefully will serve you well.)

The second form of input that **chmod** accepts is absolute numeric values for permissions. Before you can learn how to use this notation, you have to understand the BINARY (Base 2) and OCTAL (Base 8) numbering system.

In the decimal system, what most are used to, use the digits 0 thru 9. The far right digit is the least significant digit equaling 1, the next place is 10's, next 100's and so on moving to the left.

100	**10**	**1**	
1	6	9	100 + 60 + 9 = 169

The binary system uses only two digits, 0 and 1. The place values from right to left are 1's, 2's, 4's, 8's. For our purposes, we will only concern ourselves with the first three places 1,2, and 4 having the values to add up to a total of 7.

4	**2**	**1**	
0	1	1	0 + 2 + 1 = 3
1	0	1	4 + 0 + 1 = 5

Remember, the permissions are divided into three groups of three. These can be converted to decimal notation and used with the **chmod** command. If you want a permission set, it's a 1 and if not it will be a 0.

So … the following is a list of permission, binary, and decimal conversions:

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

It is the Decimal number we are looking for ultimately as this number will be used when setting permission using the numeric method. It may be hard for this to stick to begin with, but if you work with this a number of time, you will get to a point that you will know for example that 6 means read-write.

The trick of working with **chmod** in numeric mode, is that you specify the file access mode for the user, group and other as a three-digit number. The system then sets the permissions of the file or directory appropriately.

NUMERIC PERMISSIONS

Syntax: chmod (###) filename

Original permissions: r-xr--r-- r-x r-- r--
5 4 4

chmod 755 filename

Resulting permissions: rwxr-xr-x rwx r-x r-x
7 5 5

chmod 744 filename

Resulting permissions: rwx r-- r-- rwx r-- r--
7 4 4

Note: A file with -wx, can't be executed if it can't be read.

The following is a symbolic to numeric conversion chart for file permissions.

Permissions	Numeric	Used with/for
---------	000	all types (no access)
r--------	400	files (read for owner only)
r--r--r--	444	files (read-write for all)
rw-------	600	files (read-write owner only)
rw-r--r--	644	files (read-write owner only; read everyone else)
rw-rw-r--	664	files (read-write owner and group members; read everyone else)
rw-rw-rw-	666	files (read-write everyone)
rwx------	700	programs (read-write-execute owner only)
		directories (read-write-execute owner only – list files, create/remove files, and use cd)
rwxr-x---	750	programs (read-write-execute owner only; for group read-execute)
		directories(read-write-execute owner only – list files, create/remove files, and use cd; for group read-execute – list and cd)
rwxr-xr-x	755	programs (read-write-execute owner only; for everyone else read-execute)
		directories(read-write-execute owner only – list files, create/remove files, and use cd; for everyone else read-execute, list and cd)

Changing Ownership and Groups on a file or directory

You may find you want to change the ownership of a file to another user. This is accomplished with the **chown** command.

CAUTION!! Only the owner and the root can change the ownership of a file. So, once the ownership of a file is changed to another user, it cannot be changed back, except by the new owner.

The group field in the long listing identifies what user & group has access to this file. To change the group access of a file you would use the **chgrp** command.

The following example will copy a file to another users home directory and change the group access to this file. Then change the ownership of that file to the new user.

chgrp newgroup filename
&
chown owner [:group] filename Changing the group at this time in a single command instead of separately is also possible.

```
$ id
 UID=612(user3) GID=504(class) groups=504(class)
$ pwd
 /home/user3
$ cp file1 /home/user2/file1
$ ls -l /home/user2/file1
-rw-r--r--   1   user3  class   3697  JAN  24  13:13   file1

$ chgrp class2 /home/user2/file1
$ ls -l /home/user2/f1
-rw-r--r--   1   user3  class2  3697 JAN  24  13:13   file1

$ chown user2 /home/user2/file1
$ ls -l /home/user2/file1
-rw-r--r--   1   user2  class2  3697  JAN  24  13:13   file1
```

Using the chmod Command to Set File and Directory Permissions

Setting or Changing File Permissions

You can specify permissions for **chmod,** using the letters **u**, **g**, and **o**, as **symbolic code** for the owner (u), group (g), and other (o). This symbolic mode is easy to remember, since the symbols r, w, and x are used directly as arguments in the command. The chmod syntax uses the **+**, **-**, and **=** signs and where the "**u,g,o**" are the class and the "**r,w,x**" are the modes.

$ chmod class [+-=] mode , class [+-=] mode , class [+-=] mode filename

When permissions are being set the same, you can also combine the arguments as:

$ chmod ugo=r myfile2

Set the modes or permissions absolutely by using the = sign.

```
$ chmod u=rwx,g=rw,o=rw myfile2
$ ls -l myfile2
-rwxrw-rw-    1    student1    linuxclass   1024  Sep 26 09:21   myfile2
```

Modes or permissions are added with the + sign. Separate each "class - operator - mode" grouping with a comma and no space:

```
$ ls -l myfile2
-rw- - - - - - -  1    student1    unixintro   1024  Sep 26 09:21   myfile2
$ chmod u+x,g+rw,o+rw myfile2
$ ls -l myfile2
-rwxrw-rw-    1    student1    unixintro   1024  Sep 26 09:21   myfile2
```

Modes or permissions are deleted using the - sign.

```
$ ls -l myfile
-rwxrw-rw-    1    student1    unixintro   1024  Sep 26 09:21   myfile2
$  chmod g-w,o-w myfile2
$  ls -l myfile2
-rwxr- -r- -    1    student1    unixintro   1024  Sep 26 09:21   myfile2
```

Setting or Changing Directory Permissions

Changing the permissions on a directory is accomplished in the same manor as changing permissions on a file. The same commands using the u,g,o (class) and r,w,x (modes) are used, producing the same permission results.

The difference between file and directory permissions are in what they allow the different class users to do. Where the r,w,x on a file seems pretty simple, the r,w,x on a directory have a little different meaning. The previous table gave an overview of what the permissions on a file and directory will allow. The following are some examples of file and directory permissions.

Permissions Set on a File

Type This	**Permissions Set So That ...**
chmod u=r,g=,o= myfile1 -r- - - - - - - - -	The user can read from myfile1, and no one (including the user) can write to it.
chmod ugo=r myfile1 -r- - r- - r- -	Everyone can read from myfile1, but no one can write to it.
chmod u=rw,go=r myfile1 -rw-r- - r- -	Only the user can write to myfile1, but everyone can read it.
chmod ug=rw,o=r myfile1 -rw-rw-r- -	Only the user and members of your group can write to myfile1, but everyone can read it.
chmod ugo=rw myfile1 -rw-rw-rw-	Everyone can read or write to myfile1

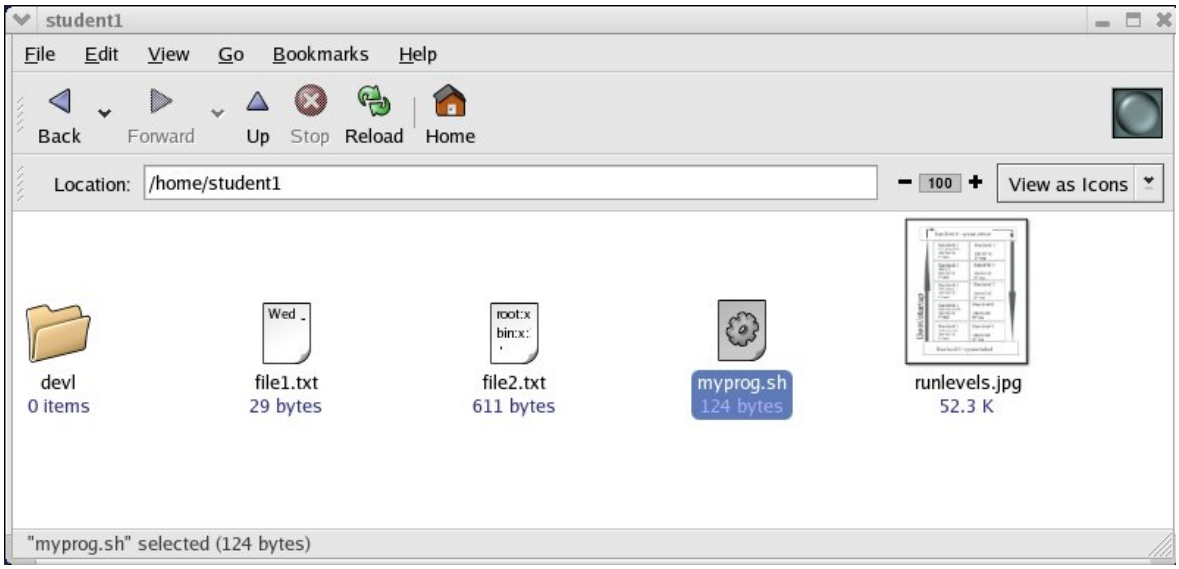
Permissions Set on a Directory

Type This	**Permissions Set So That ...**
chmod u=rwx,go=rx projects drwxr-xr-x	Allow other users and groups to list and access the files in projects, but not to create or remove files from it.
chmod ugo=rwx projects drwxrwxrwx	Allow all users to list, create, remove, and access files in projects.
chmod u=rwx,go=- drwx- - - - - -	Allow only yourself to list, create, remove, and access files in projects.

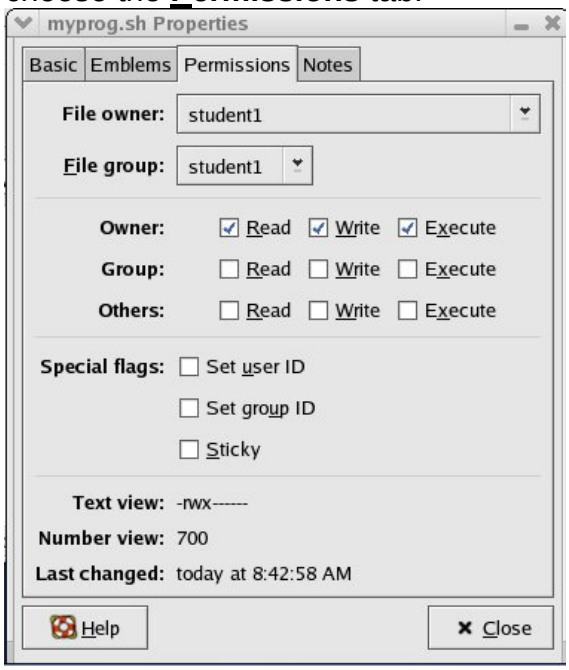
An again to review … Obsolete Method of Setting Modes on Files and Directories

$**chmod 755 my_dir**

All of the operations we have discussed so far can be accomplished via a GUI if you are in a desktop environment such as Gnome or KDE . Many of the file manipulation operations can be initiated either by right-clicking on a file, or by highlighting a file then selecting operations from the **File** pull-down menu.



As we've also note in module 1, even file permissions can be maintined through a GUI. From Gnome's Nautilus File-manger you right-click to get the properties window and the choose the **Permissions** tab:



PRINTING DOCUMENTS

Generating printouts is a common task that should be fairly easy. However, there are differences in this area between various UNIX and Linux varieties. Basic tasks include starting (**lp**), stopping (**cancel**), and finding the status of a users print job (**lpstat**).

Starting a Print Job

NOTE: In a **GUI environment**, lp requests can also be initiated by dragging a filename to the printer icon and dropping it. This sends a print request to the scheduler through lp, the same as if you had typed it at the command line. Recall from module 1:

Print Manager

Click on this panel icon and the **print manager** will automatically start. This application can be used to add or remove printers and printer queues, There are no graphical tools to start printing. You can start a printing from a command in a terminal window.

Printing text files is accomplished through the **lp** command.

$ lp filename	Sends the text file to the default printer
$ lp -d PRINTERNAME filename	Sends the text file to a specific printer

$ **lp myfile1**

$ **lp -d UNIXLAB1 myfile1**

In either case the lp command will return a status message indicating that your file was assigned a **request ID** and sent to the printer.

Stopping a Print Job Using the “cancel” Command

The cancel command is used to stop print jobs, even if they are currently printing. It is commonly used to stop a single print job, but it can be used to stop all jobs on a particular printer, and all jobs owned by a particular user.

$ cancel PRINTERNAME-number	Cancels the print job with that number only
$ cancel -e PRINTERNAME	Cancels the print jobs for that user, on the specified printer

To **cancel** a print request, enter the cancel command with the “**request ID**” number.

$ **lp /etc/passwd**
request id is UNIXLAB1-4 (1 file)
$ **cancel UNIXLAB1-4**
request "UNIXLAB1-4" cancelled

To cancel all print requests sent to a printer, use the “**-e**” option.
NOTE: The “**-e**” option is used by the superuser, ALL print jobs cancel

$ **lp /etc/passwd ; lp /etc/group**
request id is UNIXLAB1-5 (1 file)
request id is UNIXLAB1-6 (1 file)
$ **cancel -e UNIXLAB1**
cancel: You must have root capability to use this option
root:ntc231# **cancel -e unixlab1**
request "UNIXLAB1-5" cancelled
request "UNIXLAB1-6" cancelled

Displaying Printer Status Information

The **lpstat** command is used to display printer status. The **lpstat -t** option will display all status information.

Finding Printer Information Using the "**lpstat**" command

$ lpstat -t Lists all print status information.

The following example displays all status information for the printers in the Linux lab.

$ **lpstat -t**
scheduler is running
system default destination: UNIXLAB1
device for UNIXLAB1: socket://192.168.21.245:9100
UNIXLAB1 accepting requests since Aug 29 14:45
printer UNIXLAB1 is idle. enabled since Aug 29 14:45
$

LAB

Your practical exercises for this module:

Again log onto the NWSTC student server (204.227.127.133) **and practice more Linux commands.** If you need the instructions again they can be found at the following link:

http://webdev.nwstc.noaa.gov/d.train/linuxinstr.html

Remember:

1. You are encouraged to **EXPERIMENT** in this course and try various commands, so that you **SUCCEED** in the field and subsequent training.

2. DO NOT enter the commands robotically without trying to understand them in the process. Your success at further Linux training and actual work in the field is wholly dependant upon grasping the subject matter in this course.

EXERCISE 1

- Use UNIX commands **mkdir**, **touch**, **cp**, **mv**, **rm**, and **rmdir** to create and delete sub-directories and files and copy and move files. This lab is designed to challenge you a bit in that you will not be given the exact commands to type.

Let's play house …

```
(The following questions require the directory "front_porch" to be present in
your home directory. If it is not present, contact Jim Kaplafka or
Dave Rowell or at the NWSTC.)
```

Oops - your fridge is on the fritz.

1. Use the "mkdir" command to make two new directories in your kitchen - "trash_can" and "cooler"

2. Use the "mv" command to move unspoiled food from "fridge" to "cooler".

3. Use the "mv" command to move spoiled food to the "trash_can".

4. Use the "rmdir" command to remove the fridge.

5. Sell something to buy a new fridge (suggested items may be a lamp, a rug, or a bicycle. Find one of them and remove it with the 'rm' command.

6. All this certainly caused a headache. Pills are the answers. Find them and bring some of them(copy "cp" them) to the kitchen

7. By the way pills are not a good thing to keep laying about the house when you have many children in the house (as you do), so let's secure them. Let's make both copies so that only you (the owner) can open them (write) and ingest them (execute). It may be good however that everyone can read them for caution's sake (so use the 'chmod' command to set the file permissions to rwxr—r--).

END EXERCISE

A little further background information via the 'man' pages or an internet search on the Linux file permission mechanism may be of interest to you, and will become more pertinent when performing systems administration: /etc/passwd, /etc/group, UID, GID, umask

End

This is the end of this module. At this time you should proceed to module 4.