DT021/4 OperatingSystems

Lab 2: More LINUX Commands

1. File Permissions

Every file has a set of permissions associated with it. File permissions are an inherent part of the security of any computer system and are used to prevent user accidentally or maliciously deleting files and directories. The permissions in your home directory are usually set that the owner (you) can read, execute and write to all files in the directory, but other users can only read the files.

There are three types of access you can have to a file or directory, read, write, and execute. In the case of a file, read access allows you to look at the file's
contents, write access allows you to edit or remove the file, and execute access allows you to run the file (applicable only if the file is a program). In the case of a
directory, the types of permissions are the same, but the meaning for each is a little different: read access allows you to use "ls" to list the files in the directory, write
access allows you to create new files or directories within the directory, and execute access allows you to "cd" into the directory.

To see the permission on a file (or directory), use the command ls -l:

```
bash-2.03$ ls -l test
-rw-r--r--   1 mohanlo  others       967 Nov 22 10:07 test
```

The fields displayed are, from left to right,
The first character in file permission field is the type of file; "-" indicates the file is a "plain" file (e.g. text or a program), and "d" indicates that the file is actually a
directory.
the file permissions (-rw-r--r--) (explained in the next paragraph),
the number of "links" to the file (1) (usually not of interest),
the login name of the file owner (mohanlo),
the group the file belongs to (others),
the file size in bytes (967),
the date and time the file was last changed (Nov22 10:27), and

the file's name (test).

The nine permission fields are divided into sets of three and represent the owner's, the group's, and everyone else's permissions on the file, respectively. The "owner" is generally the person who created the file, the "group" is other people in the same group as the owner , and "everyone else" includes all people using the system. You can check what group you belong to by using the command: groups
The field in each set of three indicate whether read, write, and execute permissions (respectively) are turned on or off. In all cases, "-" indicates that the permission is turned off, and the appropriate letter (r, w, or x) indicates that the permission is turned on. For example, the first set of three fiels (rw- in the example) indicate that the owner has both read and write permission to the file. The next two sets of three (r--r--) indicate that other users in the owner's group, as well as all all other users on the system, have read but not write access to the file.

Changing file permissions

File permissions are changed using the command *chmod*. The letters "u", "g", "o" represent the owner ("user"), the owner's group ("group"), and everybody else ("other"). You grant or revoke privileges to these, using "+" or "-". For example, to revoke the read permissions on file test in mohanlo directory from group and others you would use the following command:

chmod go-r test

Task 1

Create a directory in your home drive called lab3. Copy the /etc/passwd file to lab3. Get a long listing of the contents of the lab3 directory
You should see the permissions for the passwd file listed as :

```
- r w-r--r--
```

What do these permissions mean?

Remove the permission w from the owner (user) of the file and verify that you can no longer write to it by executing the command:

```
$ ls –l>> passwd   (what does this command attempt to do?)
```

Write down the message you get from this command.
Change back the permissions for this file to its original condition. Now the permissions on test should be :

```
-rw-------   1 mohanlo  others       967 Nov 22 10:07 test
```

Task 2

Create a file in your current directory called *dateinfo* and use the date command to redirect the output from date command to this file.
Verify the information in the file by using the command cat.
Remove any permissions that others may have for this file.
Logout of the system and login as a different user (student2) and see can you write to this file.
Note you will have to navigate to the directory where the file is.

Task 3

From your home directory change the privileges for the lab3 directory you created earlier so that another user cannot view the files in this directory. Verify this by logging in as a different user and seeing can you view the files in the lab3 directory.

Shell Programming

Command languages provide the means for writing programs using a sequence of commands, the same commands that you type at the prompt. Most of today's command languages provide more than just the execution of a list of commands. They have features that you find in traditional high-level languages, such as looping constructs and decision making statements. This gives you a choice of programming in high-level languages or command languages.

Example 1

Use the text editor to create a file called Users:

```
# this is a comment
# program displays the number of users currently logged in.
who | wc –l
```

There are two ways to make a shell program executable: you can use the sh command, or you can make the file executable.

%***sh Users*** using the sh command to execute the file. This invokes another copy of the shell to execute it.
Alternatively you can make the program executable:

chmod +x Users using chmod to make file executable then it can be run simply by typing shell2

Example 2

```
# This script displays the date, time, username and
   # current directory.
 clear
 echo "Date and time is:"
   date
   echo
   echo "Your username is: `who am i` \n"
   echo "Your current directory is: \c"
  pwd
```

The first two lines beginning with a hash (#) are comments and are not interpreted by the shell. Clear simply clears the screen. The backquotes (`) around the command *who am i* illustrate the use of command substitution. The \n is an option of the echo command that tells the shell to add an extra carriage return at the end of the line. The \c tells the shell to stay on the same line.

Command Line Arguments

If you follow the shell script name by one or more words, the script will relate them to the embedded symbols $0, $1, $2 ... in that order. The symbols will be replaced by the respective arguments.

Example create a script called family:

```
echo $1 is married to $2
echo they have $3 children
```

Note the argument $0 stands for the name of the script “family”

% ***sh family John Mary three***

Task 4

Write a shell script called *home* that will print out a users home directory when they use the login name as a command line argument i.e+. when a user types in

```
home mohanlon
```

the output is as follows.

```
your home directory is /export/home
```

Dr. Ray Lynch