Linux/Unix System Programming

CSCI 2153

David L. Sylvester, Sr., Professor

WildCards

The * wildcard

The character ***** is called a wildcard and will match against none or more character(s) in a file (or directory) name. For example, typing in your **current** directory, type

```
$ ls list*
```

This will list all files in the current directory starting with **list....**

Try typing

```
$ ls *list
```

This will list all files in the current directory ending with **....list**

The ? wildcard

The character **?** will match exactly one character.

So **?ouse** will match files like **house** and **mouse**, but not **grouse**.

Try typing

```
$ ls ?list
```

Getting Help

<u>On-line Manuals</u>

There are on-line manuals which gives information about most commands. The manual pages tell you which options a particular command can take, and how each option modifies the behavior of the command. Type **man *command*** to read the manual page for a particular command.

For example, to find out more about the **wc** (word count) command, type

```
$ man wc
```

Alternatively

```
$ whatis wc
```

gives a one-line description of the command, but omits any information about options etc.

File Permissions (chmod)

chmod

You can control who can access files, search directories, and run scripts using the Linux's **chmod** command. This command modifies Linux file permissions, which look complicated at first glance but are actually pretty simple once you know how they work.

The permissions control the actions that can be performed on the file or directory. They either permit, or prevent, a file from being read, modified or, if it is a script or program, executed. For a directory, the permissions govern who can cd into the directory and who can create or modify files within the directory.

You use the **chmod** command to set each of these permissions. To see what permissions have been set on a file or directory, use the **ls** command.

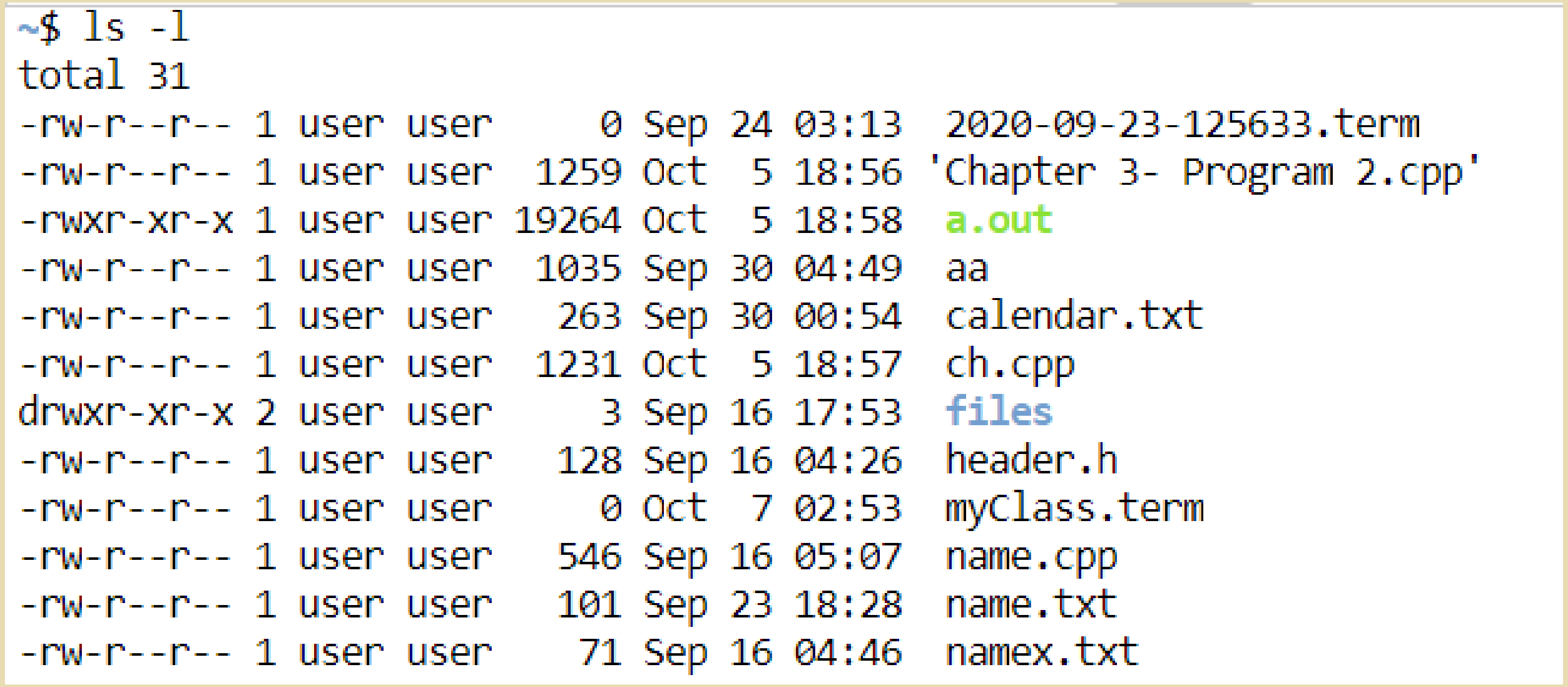
File Permissions (chmod)

Viewing and Understanding File Permissions

We can use the **-l** (long format) option to have **ls** list the file permissions for files and directories.

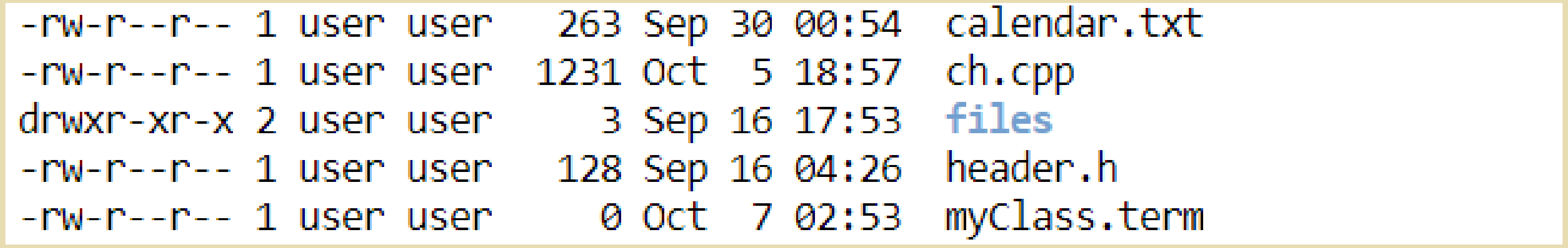
$ **ls -l**

output from ls -l in a terminal window



```
~$ ls -l
total 31
-rw-r--r-- 1 user user     0 Sep 24 03:13  2020-09-23-125633.term
-rw-r--r-- 1 user user  1259 Oct  5 18:56 'Chapter 3- Program 2.cpp'
-rwxr-xr-x 1 user user 19264 Oct  5 18:58  a.out
-rw-r--r-- 1 user user  1035 Sep 30 04:49  aa
-rw-r--r-- 1 user user   263 Sep 30 00:54  calendar.txt
-rw-r--r-- 1 user user  1231 Oct  5 18:57  ch.cpp
drwxr-xr-x 2 user user     3 Sep 16 17:53  files
-rw-r--r-- 1 user user   128 Sep 16 04:26  header.h
-rw-r--r-- 1 user user     0 Oct  7 02:53  myClass.term
-rw-r--r-- 1 user user   546 Sep 16 05:07  name.cpp
-rw-r--r-- 1 user user   101 Sep 23 18:28  name.txt
-rw-r--r-- 1 user user    71 Sep 16 04:46  namex.txt
```

File Permissions (chmod)



```
-rw-r--r-- 1 user user   263 Sep 30 00:54  calendar.txt
-rw-r--r-- 1 user user  1231 Oct  5 18:57  ch.cpp
drwxr-xr-x 2 user user     3 Sep 16 17:53  files
-rw-r--r-- 1 user user   128 Sep 16 04:26  header.h
-rw-r--r-- 1 user user     0 Oct  7 02:53  myClass.term
```

On each line, the first character identifies the type of entry that is being listed. If it is a dash (-) it is a file. If it is the letter d it is a directory.

The next nine characters represent the settings for the three sets of permissions.

The first three characters show the permissions for the **user** who owns the file (**user permissions**).

The middle three characters show the permissions for members of the file's **group** (**group permissions**).

The last three characters show the permissions **for anyone not in the first two categories** (**other permissions**).

File Permissions (chmod)

There are three characters in each set of permissions. The characters are indicators for the presence or absence of one of the permissions. They are either a dash (-) or a letter. If the character is a dash, it means that permission is not granted. If the character is an r, w, or an x, that permission has been granted.

The letters represent:

r: Read permissions. The file can be opened, and its content viewed.

w: Write permissions. The file can be edited, modified, and deleted.

x: Execute permissions. If the file is a script or a program, it can be run (executed).

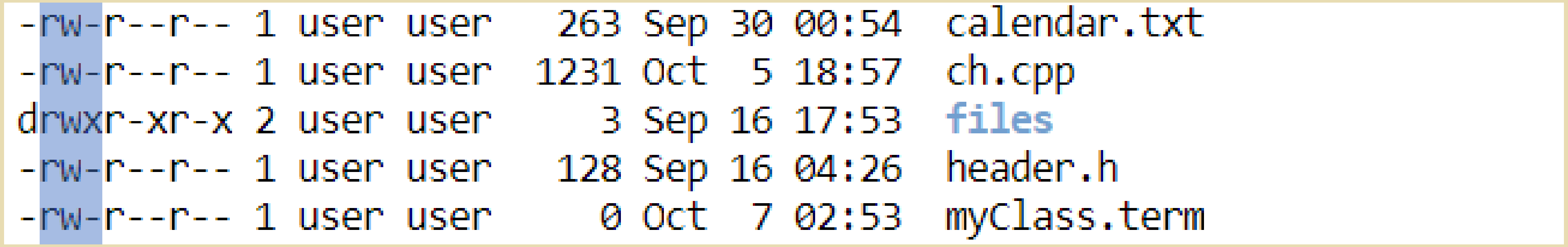
For example:

--- means no permissions have been granted at all.

rwx means full permissions have been granted. The read, write, and execute indicators are all present.

File Permissions (chmod)

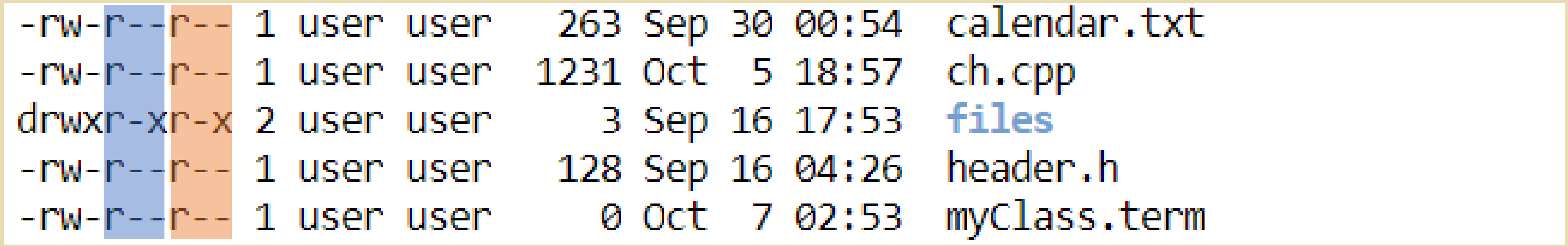
In the picture below, the third line starts with a **d**. This line refers to a directory called “files.” The owner of the directory is “user,” and the name of the group that the directory belongs to is also called “user.”



```
-rw-r--r-- 1 user user   263 Sep 30 00:54  calendar.txt
-rw-r--r-- 1 user user  1231 Oct  5 18:57  ch.cpp
drwxr-xr-x 2 user user     3 Sep 16 17:53  files
-rw-r--r-- 1 user user   128 Sep 16 04:26  header.h
-rw-r--r-- 1 user user     0 Oct  7 02:53  myClass.term
```

The next three characters are the user permissions for this directory. These show that the owner has full permissions. The r, w, and x characters are all present. This means the user “user” has read, write and execute permissions for that directory.

File Permissions (chmod)



```
-rw-r--r-- 1 user user   263 Sep 30 00:54  calendar.txt
-rw-r--r-- 1 user user  1231 Oct  5 18:57  ch.cpp
drwxr-xr-x 2 user user     3 Sep 16 17:53  files
-rw-r--r-- 1 user user   128 Sep 16 04:26  header.h
-rw-r--r-- 1 user user     0 Oct  7 02:53  myClass.term
```

The second set of three characters on the directory line are the group permissions, these are r-x. These show that the members of the user group have read and execute permissions for this directory. That means they can list the files and their contents in the directory, and they can cd (execute) into that directory. They do not have write permissions, so they cannot create, edit, or delete files.

The final set of three characters are also r-x. These permissions apply to people who are not governed by the first two sets of permissions. These people (called ”others”) have read and execute permissions on this directory.

File Permissions (chmod)

Understanding The Permission Syntax

To use **chmod** to set permissions, we need to tell it:

- Who: Who we are setting permissions for.
- What: What change are we making? Are we adding or removing the permission?
- Which: Which of the permissions are we setting?

We use indicators to represent these values and form short “permissions statements” such as u+x, where “u” means ” user” (who), “+” means add (what), and “x” means the execute permission (which).

File Permissions (chmod)

The “**who**” values we can use are:

u: User, meaning the owner of the file.

g: Group, meaning members of the group the file belongs to.

o: Others, meaning people not governed by the u and g permissions.

a: All, meaning all of the above.

If none of these are used, chmod behaves as if “a” had been used.

The “**what**” values we can use are:

–: Minus sign. Removes the permission.

+: Plus sign. Grants the permission. The permission is added to the existing permissions. If you want to have this permission and only this permission set, use the = option, described below.

=: Equals sign. Set a permission and remove others.

File Permissions (chmod)

The “which ” values we can use are:

r: The read permission.

w: The write permission.

x: The execute permission.

File Permissions (chmod)

Setting And Modifying Permissions

Let’s say you have a file where everyone has full permissions on it.

```
$ ls -l namex.txt
```

If you want the user to have read and write permissions and the group and other users to have read permissions only. You can do using the following command:

```
$ chmod u=rw,og=r namex.txt
```

Using the “=” operator means we wipe out any existing permissions and then set the ones specified.

Let’s check the new permission on this file:

```
$ ls -l namex.txt
```

The existing permissions have been removed, and the new permissions have been set, as expected.

File Permissions (chmod)

Adding a permission without removing the existing permissions settings

If we have a script file that we have finished editing. We need to make it executable for all users. Its current permissions look like this:

```
$ ls -l new_script.sh
```

We can add the execute permission for everyone with the following command:

```
$ chmod a+x new_script.sh
```

If we take a look at the permissions, we'll see that the execute permission is now granted to everyone, and the existing permissions are still in place.

```
$ ls -l new_script.sh
```

We could have achieved the same thing without the "a" in the "a+x" statement. The following command would have worked just as well.

```
$ chmod +x new_script.sh
```

File Permissions (chmod)

Setting Permissions for Multiple Files

We can apply permissions to multiple files all at once.

```
$ ls -l
```

To remove the write permissions for the “other” users from files that have a “.cpp” extension. We can do this with the following command:

```
$ chmod o-r *.cpp
```

Let’s check what effect that has had:

```
$ ls -l
```

As we can see, the read permission has been removed from the “.cpp” files for the “other” category of users. No other files have been affected.

The **-R** (recursive) option is used to include files in subdirectories..

```
$ chmod -R o-r *.cpp
```

File Permissions (chmod)

Numerical Shorthand

Another way to use chmod is to provide the permissions you wish to give to the owner, group, and others as a three-digit number. The leftmost digit represents the permissions for the owner. The middle digit represents the permissions for the group members. The rightmost digit represents the permissions for the others.

The digits you can use and what they represent are listed here:

0: (000) No permission.

1: (001) Execute permission.

2: (010) Write permission.

3: (011) Write and execute permissions.

4: (100) Read permission.

5: (101) Read and execute permissions.

6: (110) Read and write permissions.

7: (111) Read, write, and execute permissions.

File Permissions (chmod)

Each of the three permissions is represented by one of the bits in the binary equivalent of the decimal number. So 5, which is 101 in binary, means read and execute. 2, which is 010 in binary, would mean the write permission.

Using this method, you set the permissions that you wish to have; you do not add these permissions to the existing permissions. So if read and write permissions were already in place you would have to use 7 (111) to add execute permissions. Using 1 (001) would remove the read and write permissions and add the execute permission.

File Permissions (chmod)

To add the read permission back on the ".txt" files for the others category of users. We must set the user and group permissions as well, so we need to set them to what they are already. These users already have read and write permissions, which is 6 (110). We want the "others" to have read permissions, so they need to be set to 4 (100).

The following command will accomplish this:

```
$ chmod 664 *.page
```

This sets the permissions we require for the user, group members, and others to what we require.

```
$ ls -l
```

Inclass Assignment

Using CoCalc terminal, try the following on a .cpp file:

- *Modify the permissions for the user to read and execute and group to execute.*
- *Modify the user, group and other permissions to no access*
- *Modify the user, group and other permissions to write and execute*

File Permissions Assignment

Using the **vi editor** in your **CoCalc terminal**, create a file bash file named your first initial, lastname .sh, capitalizing the first letter of your first name and last name.

Ex: A person named John Doe would create a file named **JDoe.sh**

This file should output

- Two blank lines, then your
- Name (**First and Last**)
- Course name: **Linux/Unix System Programming**
- Course ID/Number: **CSCI 2153**
- Instructor: **David L. Sylvester**
- Due Date: **10/14/2020**
- Two blank lines

Execute the file:

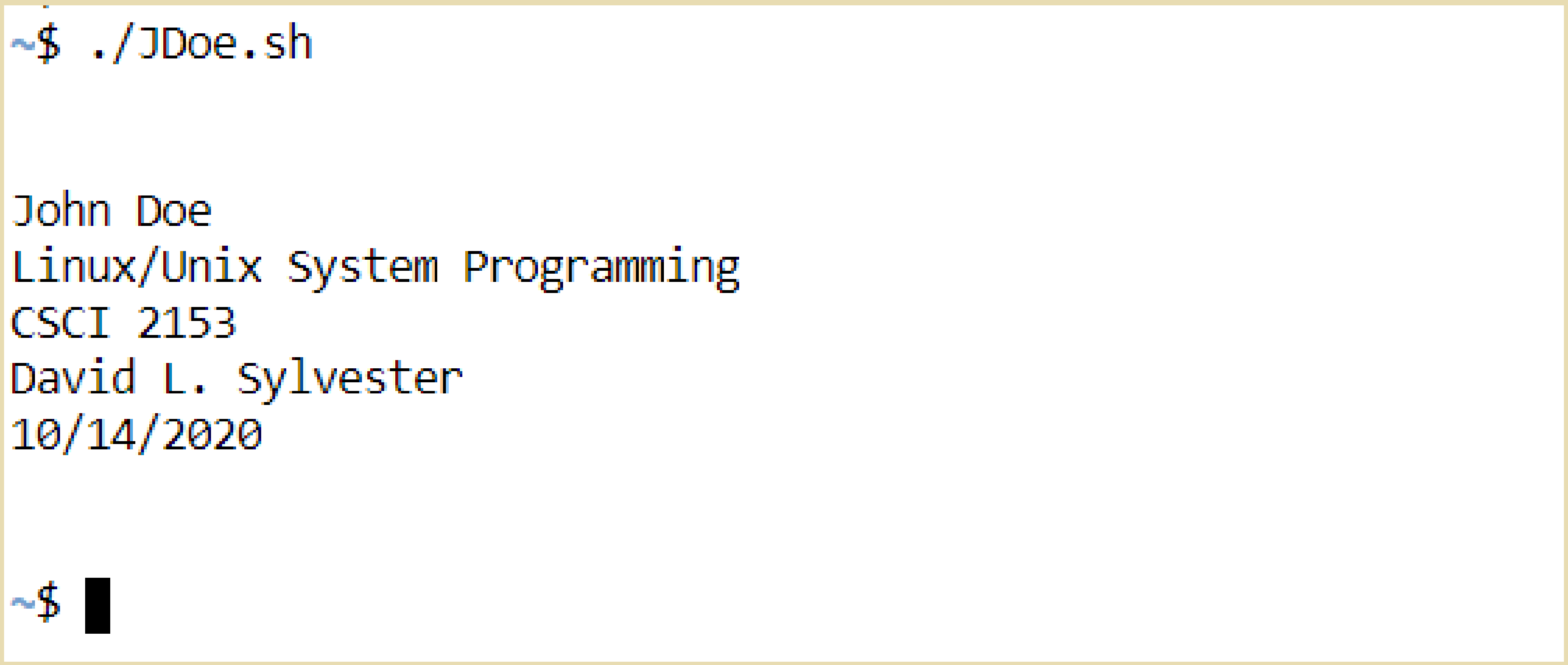
```
$  ./JDoe.sh
```

Then screenshot your output and submit to CANVAS.

Now do the assignment in your Linux Virtual Box loaded on your PC and submit a snapshot of your PC's desktop to CANVAS.

File Permissions Assignment

Sample output:



```
~$ ./JDoe.sh



John Doe
Linux/Unix System Programming
CSCI 2153
David L. Sylvester
10/14/2020



~$
```