Unix and Shell Programming Lab Course

Objectives:

1. Learn UNIX Filters related to text processing, communication and search utilities
2. Learn programming filters and interactive shell scripting
3. Learn shell programming constructs writing advanced scripts
4. Learn kernel programming on file operations and managing processes

Course Outcomes:

CO1: **Develop** chained commands to extract the required information from the system **(apply)**

CO2: **Navigate** in any Unix-flavoured operating system **(Apply)**

CO3: **Develop** scripts compatible with different shells available under UNIX environment **(Create)**

CO4: **Develop** scripts for automating the tasks of programmer during deployment and maintenance **(Create)**

CO5: **Develop** scripts to automate task using programmable filters **(Create)**

List of Shell Scripts:

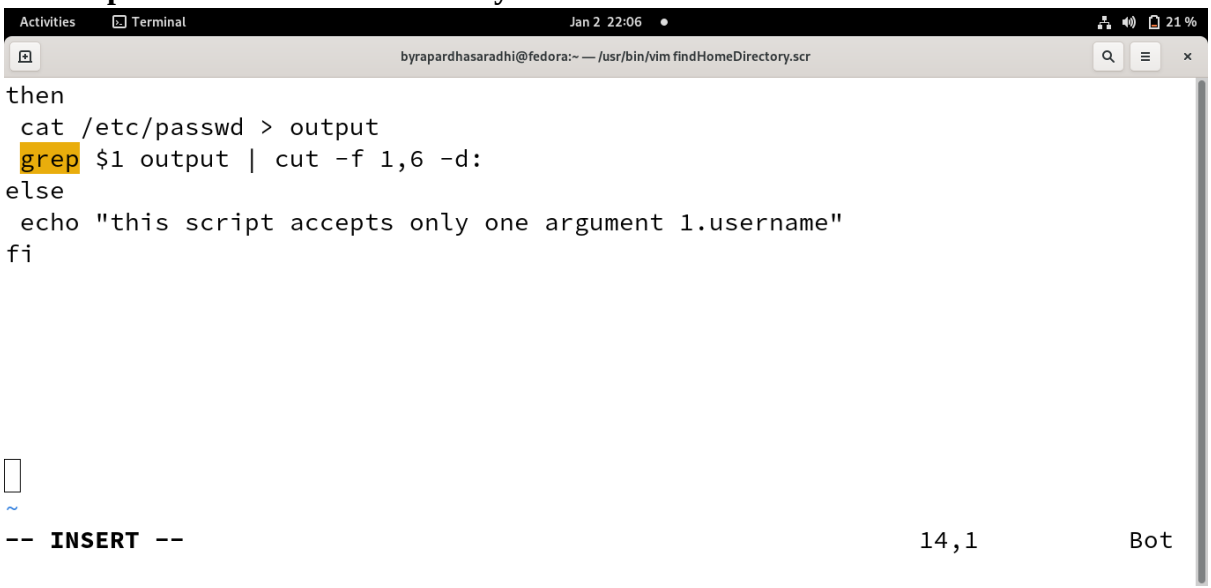
1. Create a script that, given a user name, finds the home directory of the user using the /etc/passwd file.

   Preparation:

   - None

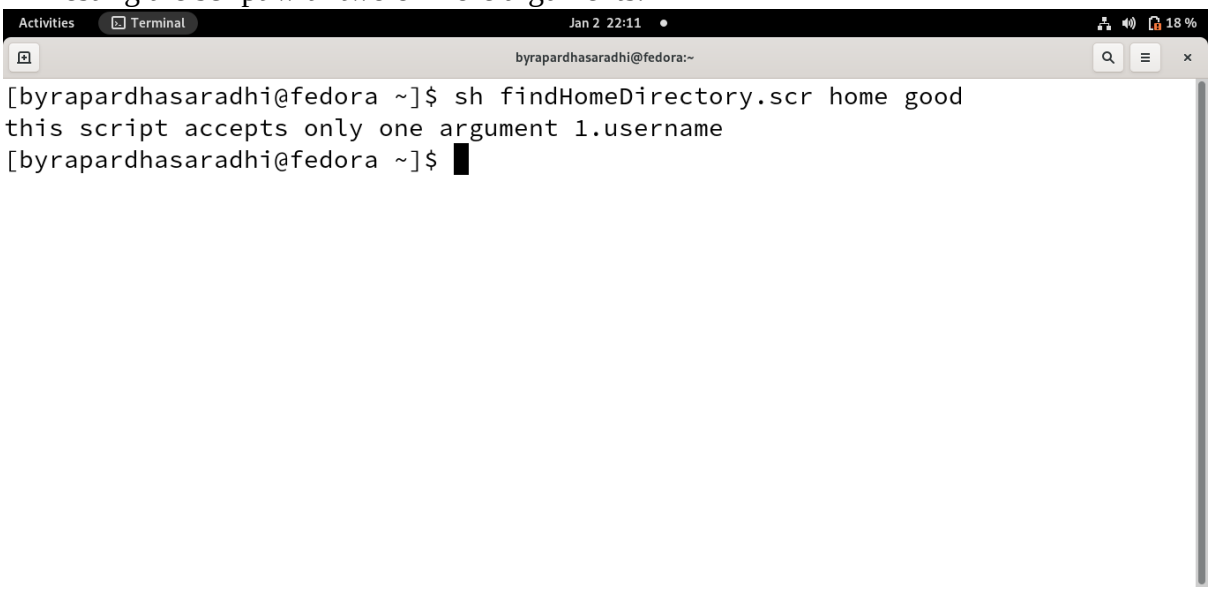
   Script:

   - **Script Name: findHomeDirectory.scr**

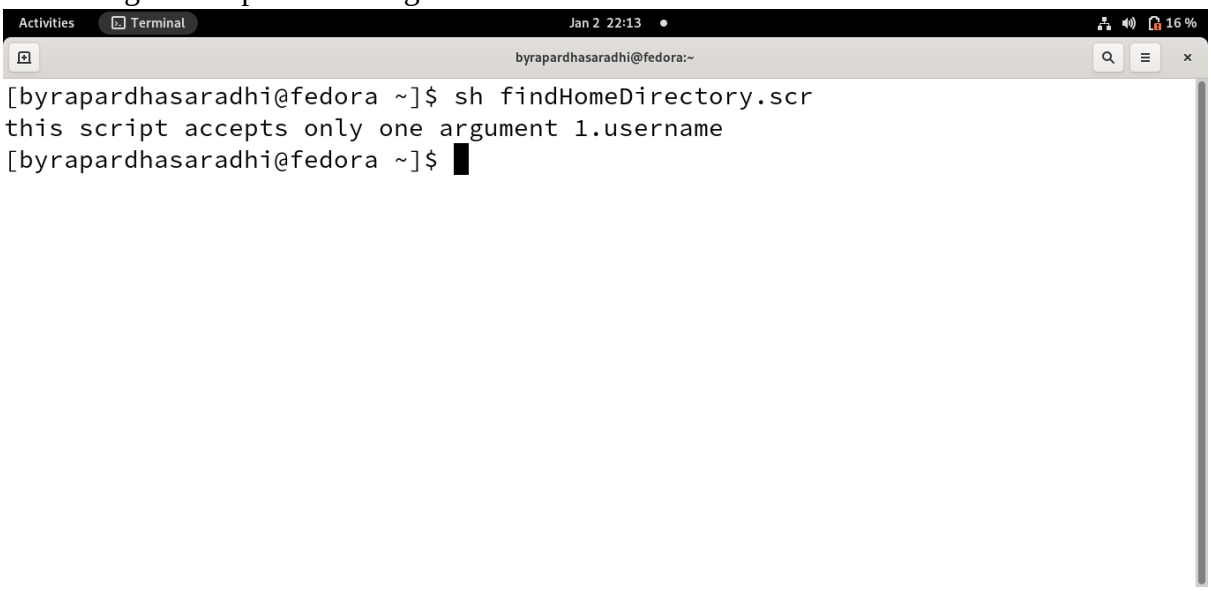


   - **Arguments:** One, The user name.
   - **Validation:** The minimum validation requirements are :
     i. Ensure that there is only one argument.
   - **Body Section:** Create a script that, given the name of a user (as the only argument), prints the absolute pathname of the user's home directory

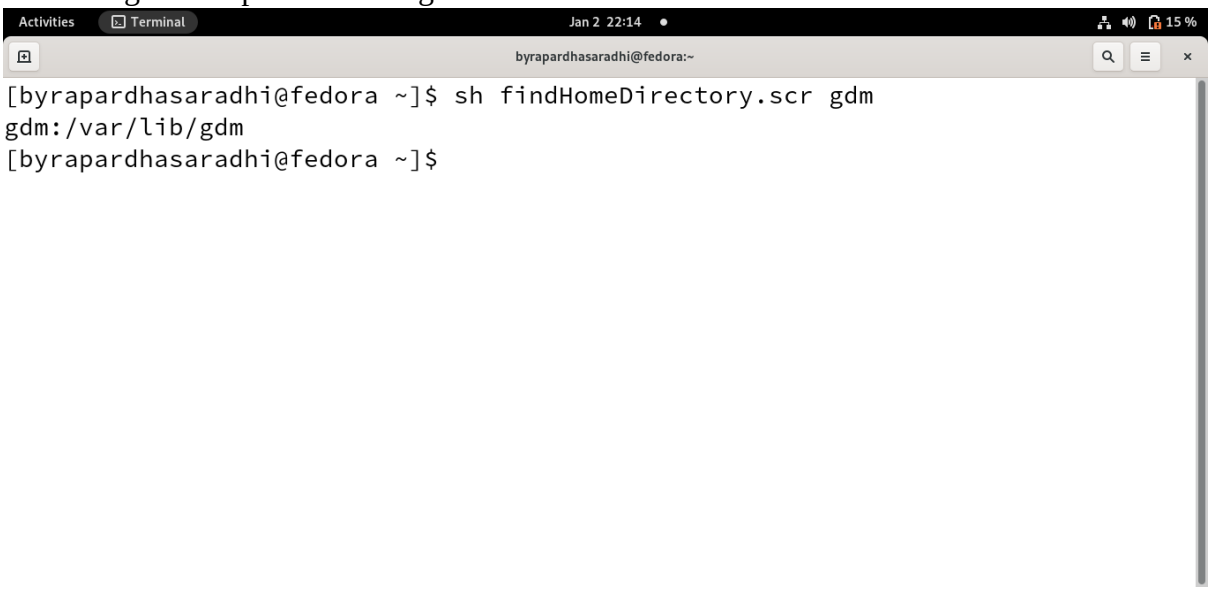
- **Testing the Script:**
- Testing the script with two or more arguments.



- Testing the script with no arguments.

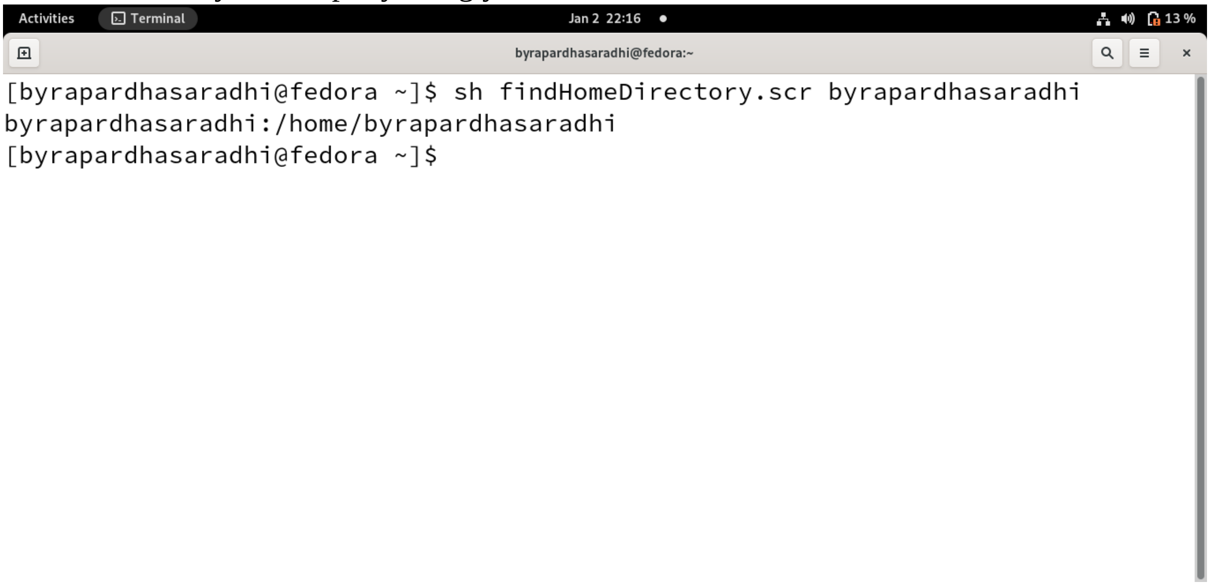


- Testing the script with one argument.



Testing the Effect of the Script:

- Verify the script by using your user name.



Result:- The Home directory of the given user displayed on the screen.

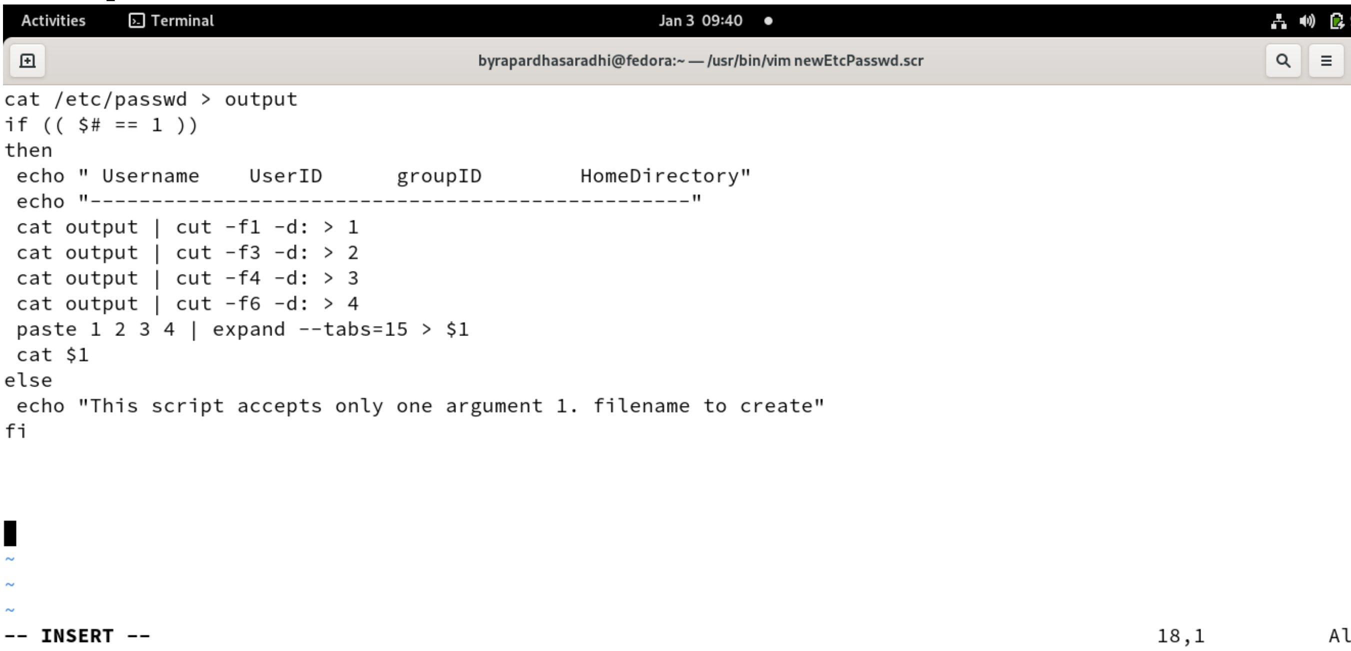
2. Write a script that creates a file out of the /etc/passwd file.

Preparation:

- None

Script:

- **Script Name: newEtcPasswd.scr**



- **Arguments:** One, The name of the file.
- **Validation:** The minimum validation requirements are:
    i. Ensure that there is only one argument.
- **Body Section:** Create a script that makes a file out of the information in the /etc/passwd file using the following format.

User Name User Id Group ID Home Directory

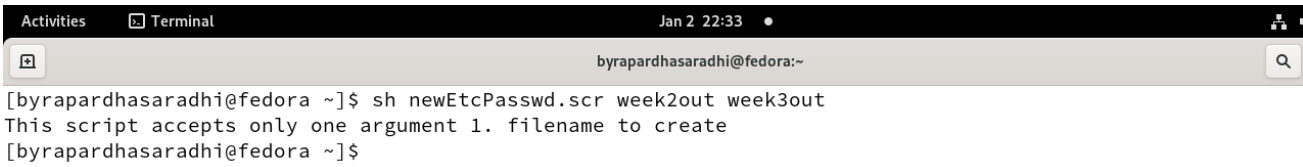
- - - - - - - -   - - - - - - - - - - - - - - - - - - - - - - - - -

ram 234 23 /etc/usr/student/ram

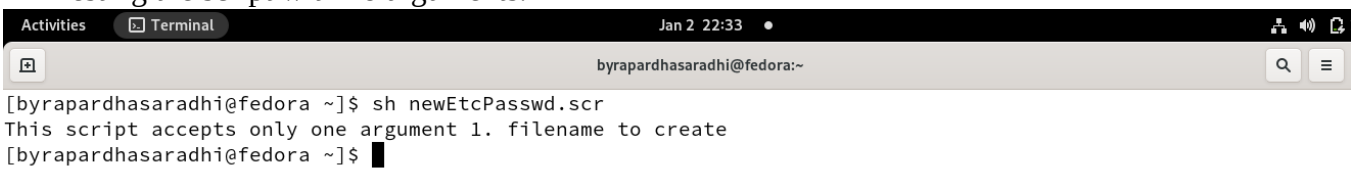
- - - -

Testing the Script:

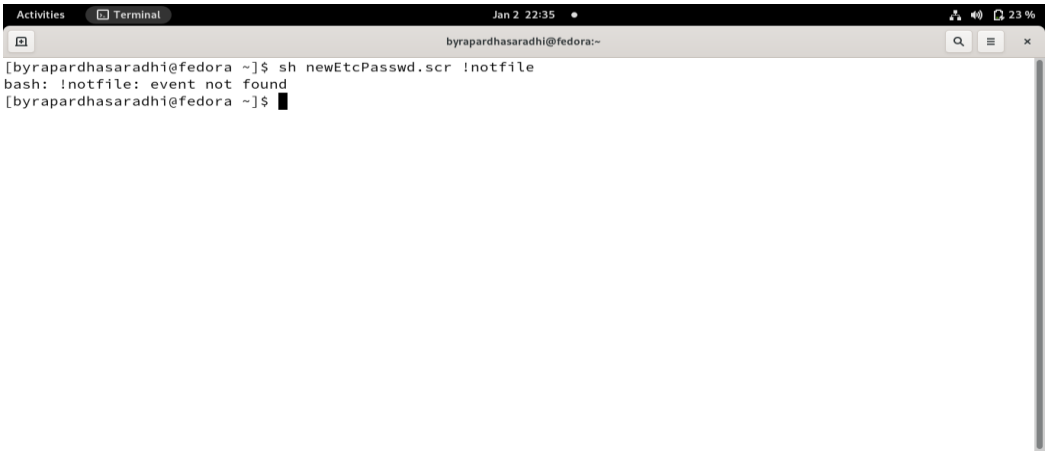
- Testing the script with two or more arguments.



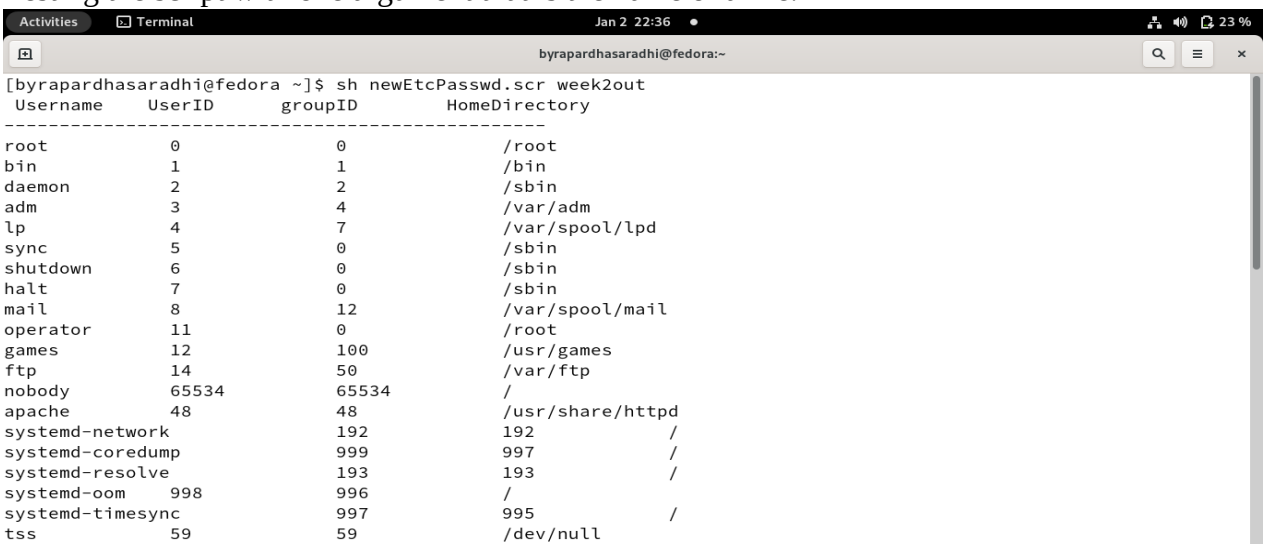
- Testing the script with no arguments.



- Testing the script with one argument that is not the name of a file.

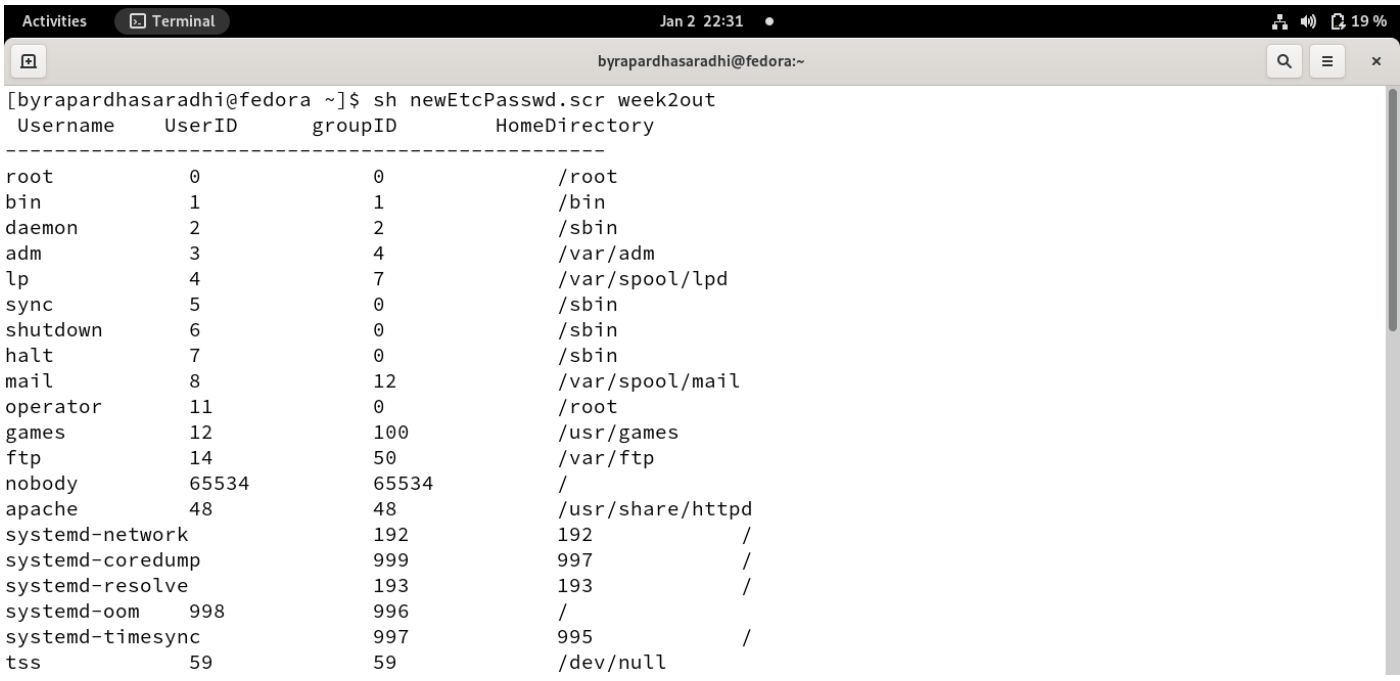


- Testing the script with one argument that is the name of a file.



Testing the Effect of the Script:

- Verify the file was created and contains the correct information and format.



Result:- The Required file was created and the result is as shown above.

3. In a C Program, there is only one comment format. All comments must start with an open comment token, /*, and end with a close comment token, */. C++ programs use the C tokens for comments

that span several lines. Single-line comments start with two slashes (//). In either case, the start token can be anywhere on the line.

Write a script to change every single-line comment in a C++ source file that uses C program start and end comment tokens to a single-line comment starting with a C++ single-line token. The comment itself is to be unchanged.

Preparation:

- Create at least five C++ source files in your home directory. The files do not have to be real C++ source files; they can contain only a few lines of comments, some with C program tokens and some with C++ single-line tokens. Each program should have at least one multiple comment and at least one single-line comment that uses the C program tokens. Use one or more blank lines between comments. The name of the files should have C++ extension (.c++), such as file1.c++.

Script:

- **Script Name: commentType.scr**

- **Arguments:** None
- **Validation:** The minimum validation requirements are:
  i. Ensure that there is no argument.
- **Body Section:** Create a script that finds all files with extension (.c++) under your directory and change only the lines with comments. The name of the files should be preserved. If a file has the name file1.c++, the name still should be file1.c++ after the change.

Testing the Script:

- Testing the script with one or two arguments.

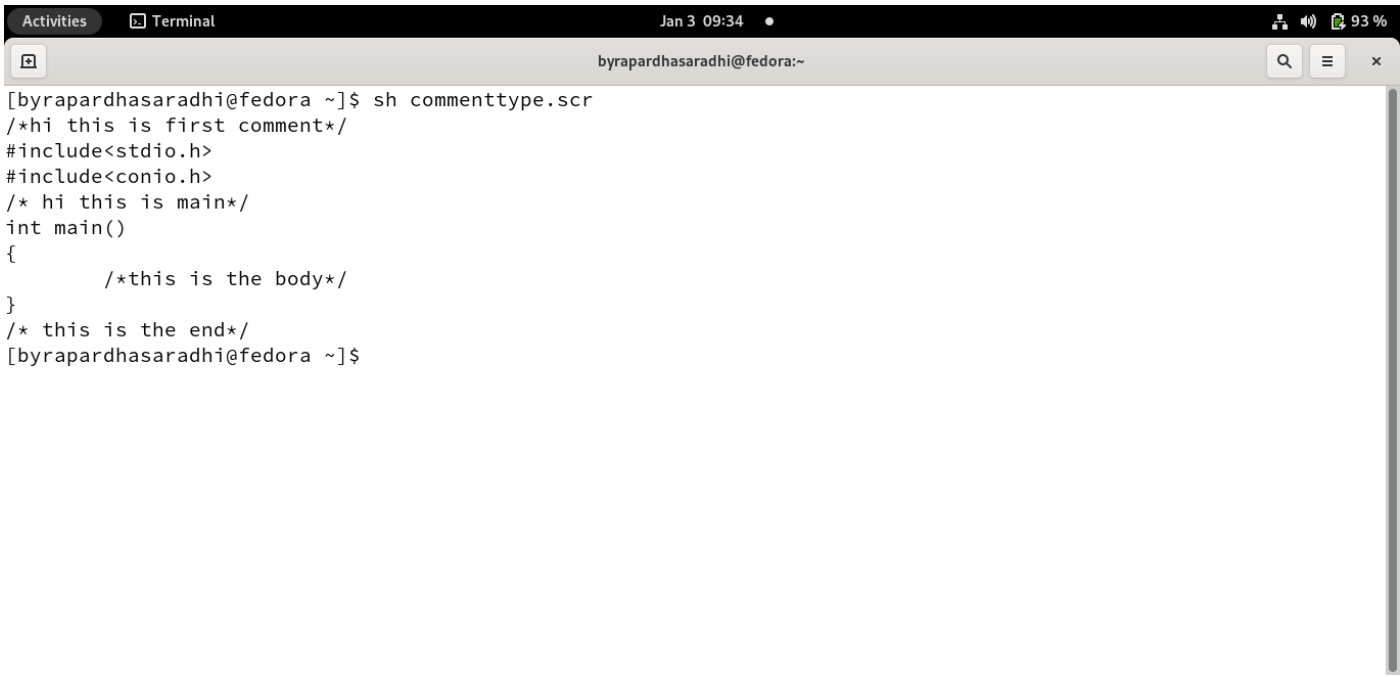
```
Activities  Terminal                    Jan 3 09:33
byrapardhasaradhi@fedora:~
[byrapardhasaradhi@fedora ~]$ sh commenttype.scr myfile.c
script needs zero args only
[byrapardhasaradhi@fedora ~]$ sh commenttype.scr myfile.cc file.c
script needs zero args only
[byrapardhasaradhi@fedora ~]$
```

- Testing the script with no arguments.

```
Activities  Terminal                    Jan 3 09:34
byrapardhasaradhi@fedora:~
[byrapardhasaradhi@fedora ~]$ sh commenttype.scr
/*hi this is first comment*/
#include<stdio.h>
#include<conio.h>
/* hi this is main*/
int main()
{
        /*this is the body*/
}
/* this is the end*/
[byrapardhasaradhi@fedora ~]$
```

Testing the Effect of the Script:

- Check to see if the comments are changed in the files.

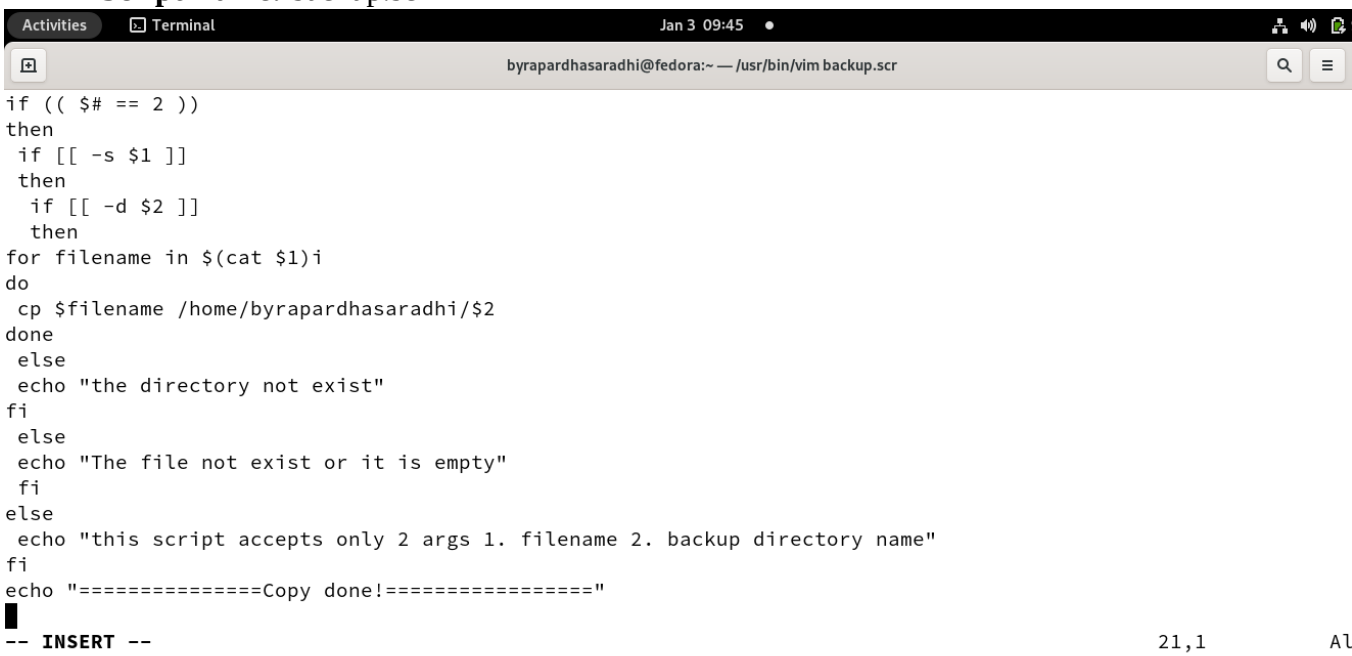


Result: It is observed that the comments in c++ source file has been changed to C source file comments.

4. Write a script to backup and archive a list of files.
   Preparation:
   - Create a file and type in it the list of files (in your home directory) that you want to back and archive
   - Create a directory in which you will store the backed-up files and archive file.

Script

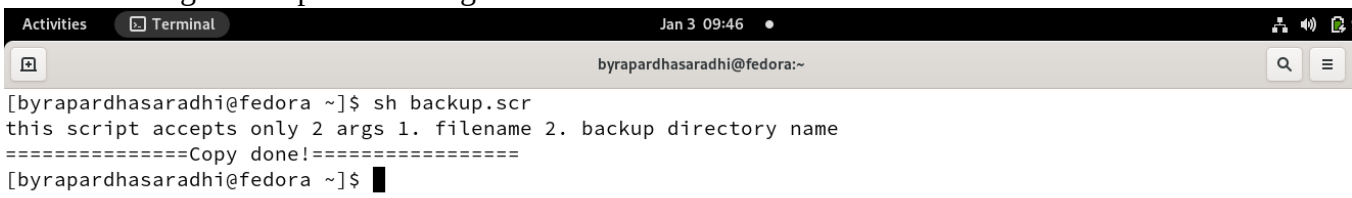
- **Script Name:** backup.scr



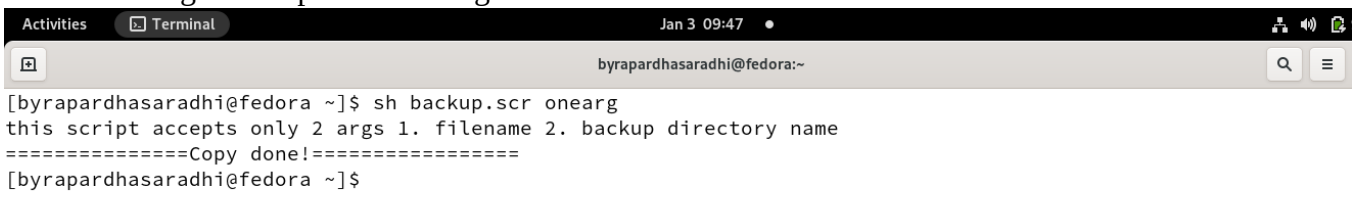
- **Arguments:** A filename and a directory. The filename holds the list of the files that should be backed-up. The directory is where the backed-up files should be stored.
- **Validation:** The minimum validation requirements are:
  i. Ensure that exactly two arguments are entered. ii. Check that the first argument is the name of a file exists
  iii. Check that the second argument is the name of the directory that exists
- **Body Section:** Create backup files for files listed in the first argument. The backup files should have the same name as the original file with the extension bak. They should be copied to the directory given as the second argument.

Testing the Script:

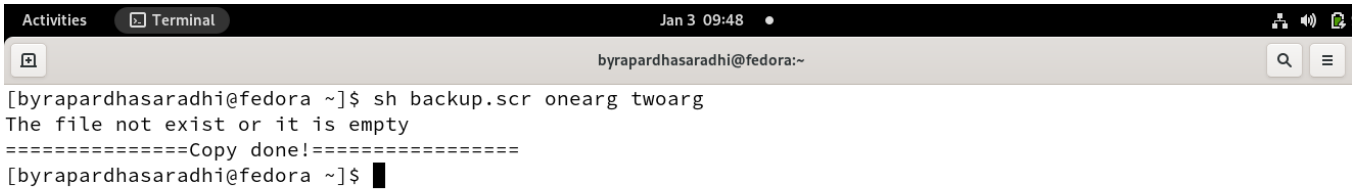
- Testing the script with no arguments



- Testing the script with one argument

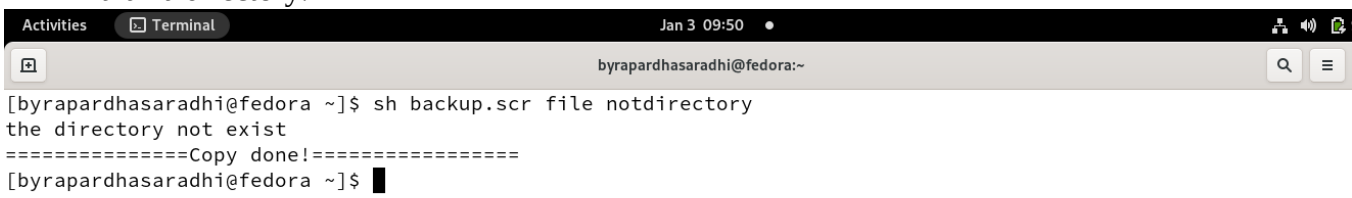


- Testing the script with three arguments

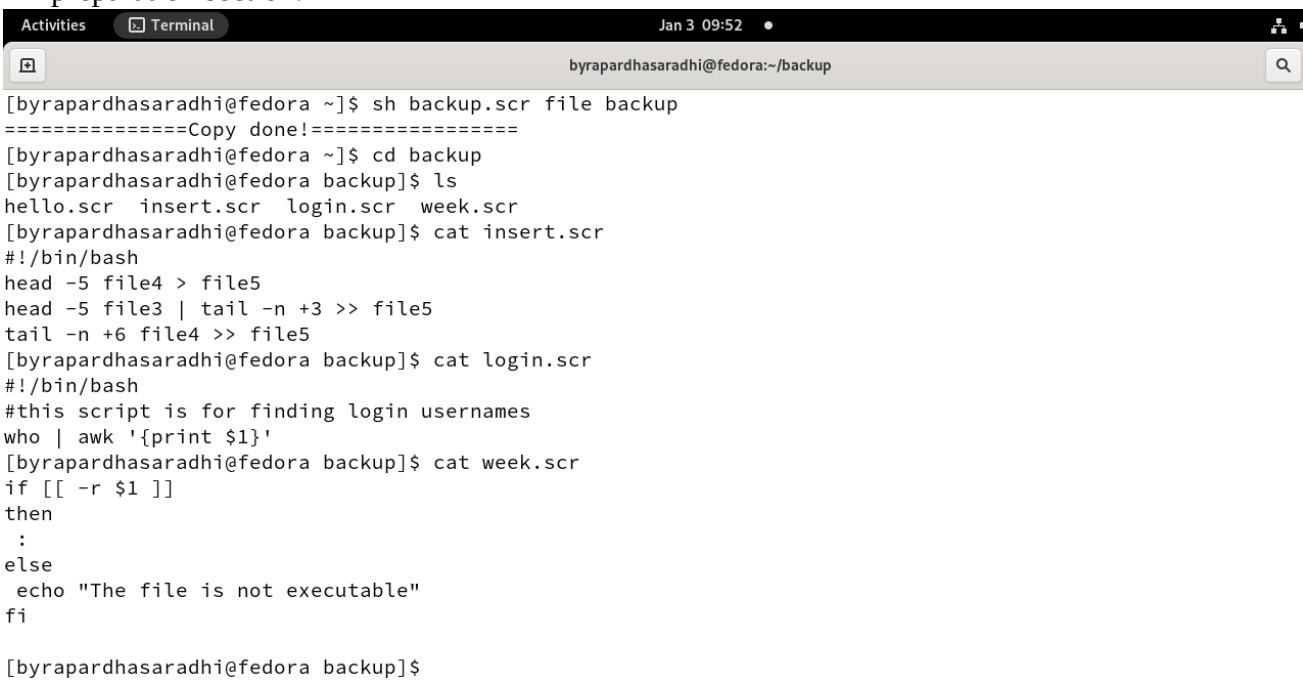


- Testing the script with two arguments in which the first one is not the name of the file.

- Testing the script with two arguments in which the second one is the name of a file rather than a directory.

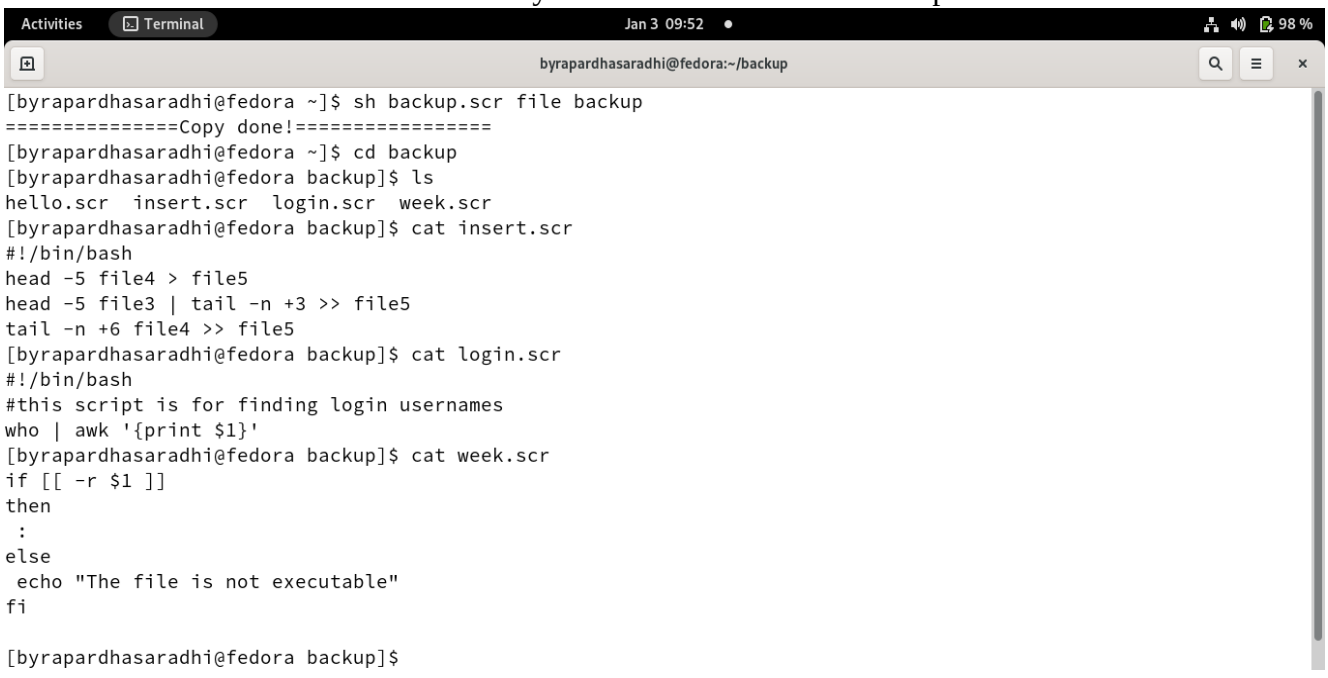


- Testing the script with name of the file and the name of the directory you created in the preparation section.



Testing the Effect of the Script:

- Check the contents of the directory to be sure that the files are copied



Result: The backup of list of files given in “file” was created in the given directory.

5. Write a script that finds all soft links to a specific file.

Preparation:

- Create a file and type some junk in it.
- Make at least five soft links to this file using completely arbitrary names.

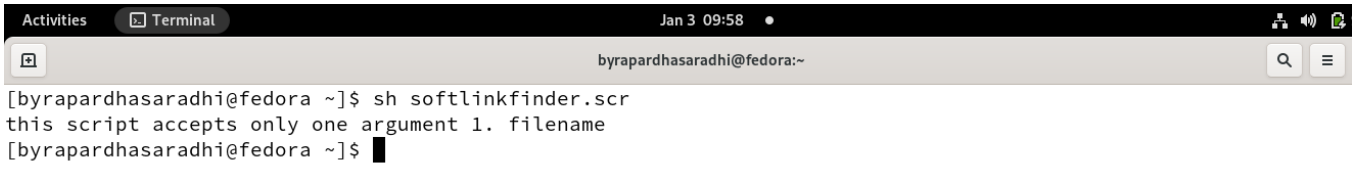
Script:

- **Script Name: softLinkFinder.scr**

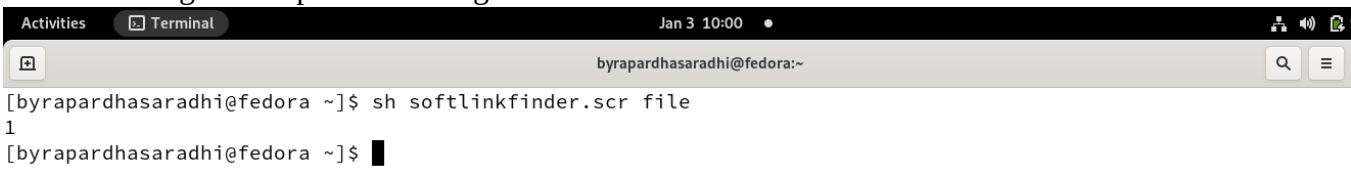
- **Arguments:** A filename. The file for which we want to find the soft links.
- **Validation:** The minimum validation requirements are:
  i. Ensure that exactly one argument is entered. ii. Check that only argument is the name of a file and that the specified file exists.
- **Body Section:** Use ls -l and grep command to find all the soft links attached to $1 positional parameter. Note that a file of type soft link is distinguished by lower case l. Be sure to find the soft links to the file defined in $1 and not other files.

Testing the Script:

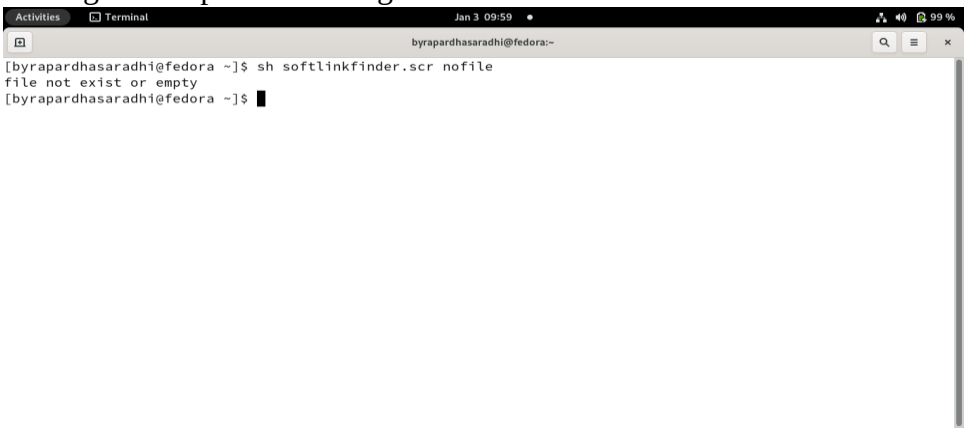
- Testing the script with no arguments



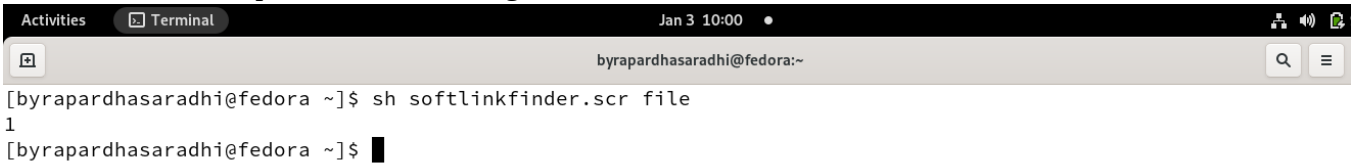
- Testing the script with one argument



- Testing the script with one argument that is not a file

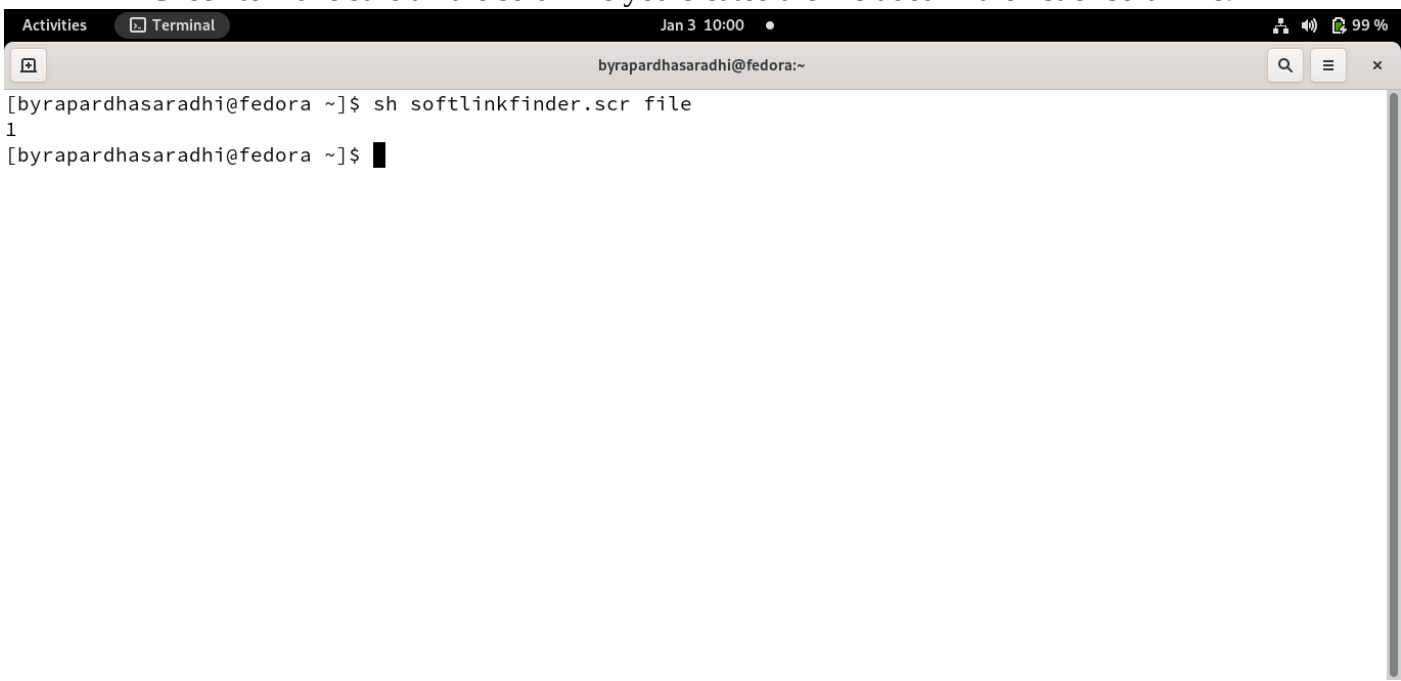


- Test the script with one valid argument.



Testing the Effect of the Script:

- Check to make sure all the soft links you created are included in the list of soft links.



Result: The Softlinks for the given file displayed.

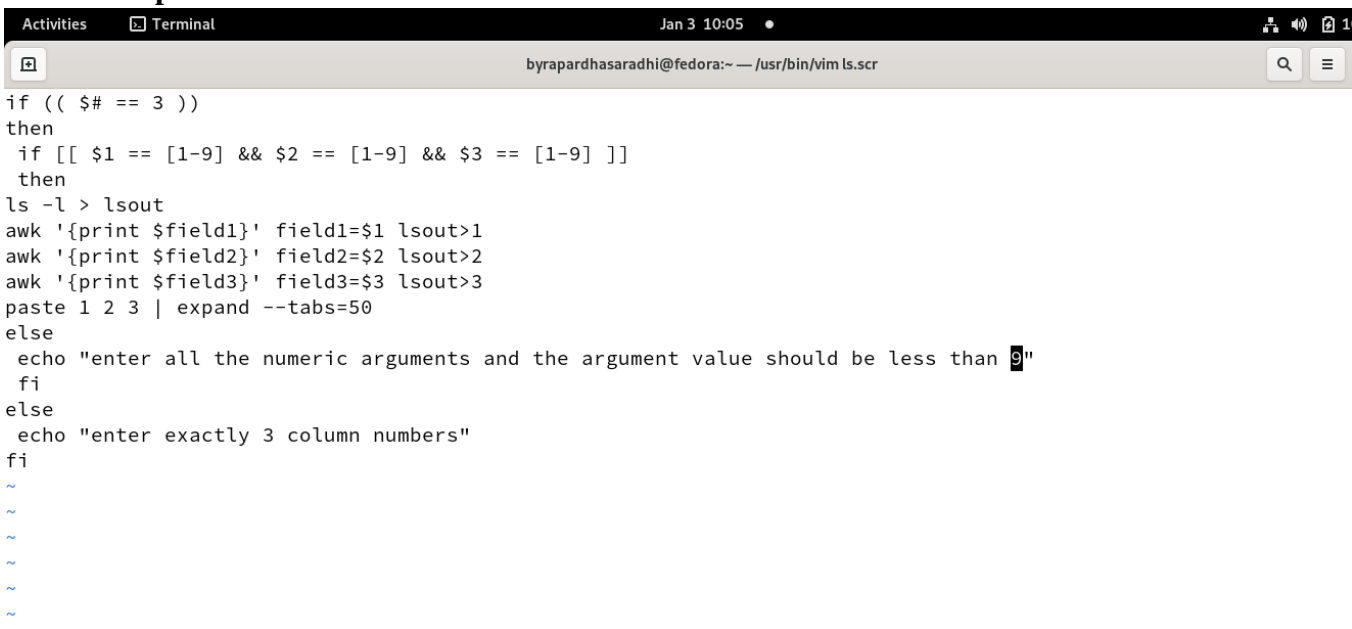
6. Create a script that simulates the ls -l command but prints only three columns of our choice.

Preparation:

- None

Script:

- **Script Name: ls.scr**

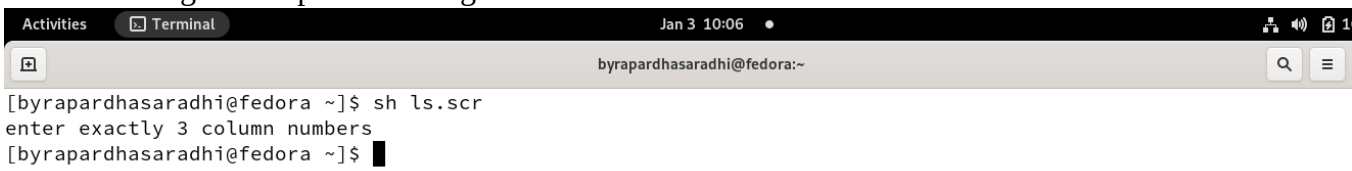


- **Arguments:** Three numeric arguments defining the column number of the ls -l output to be printed in the order we specify.
- **Validation:** The minimum validation requirements are :
  i. Ensure that exactly three arguments are entered.
  ii. Ensure that all three arguments are numeric
  iii. Ensure that each argument is less than or equal to the actual number of columns in the ls -l command output.

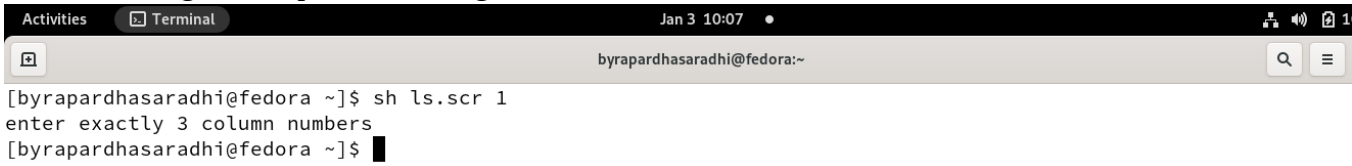
- **Body Section:** Creates a new command that shows the output of the ls -l command to be printed in three columns in the order we like.

Testing the Script:

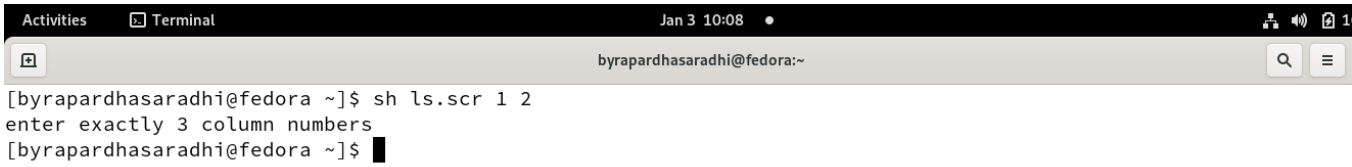
- Testing the script with no arguments.



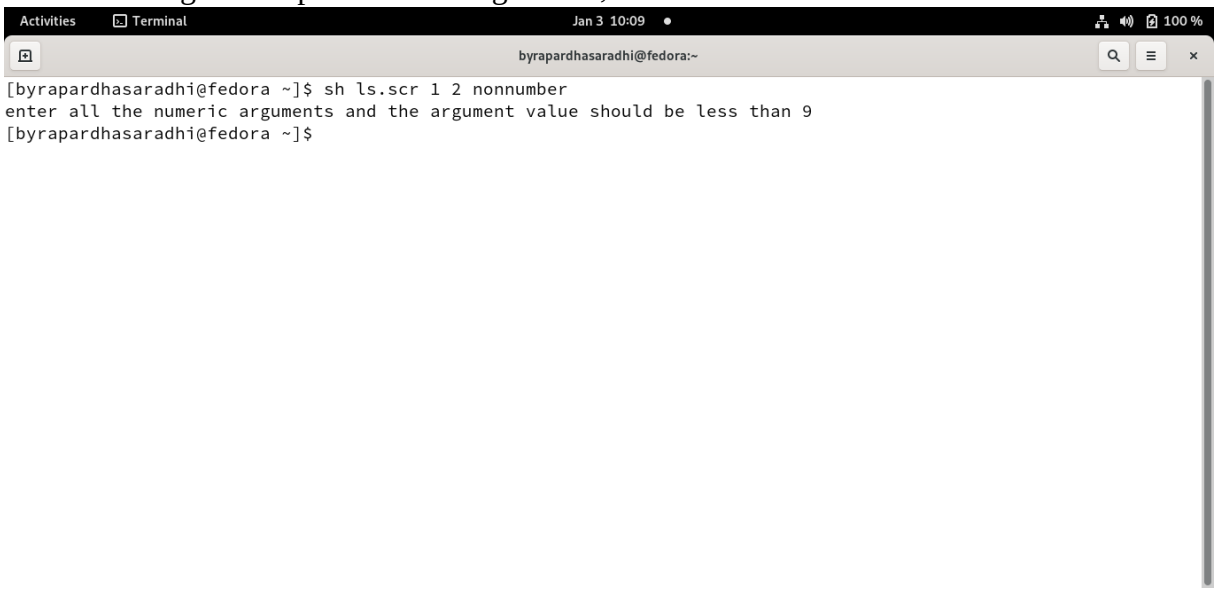
- Testing the script with one argument.



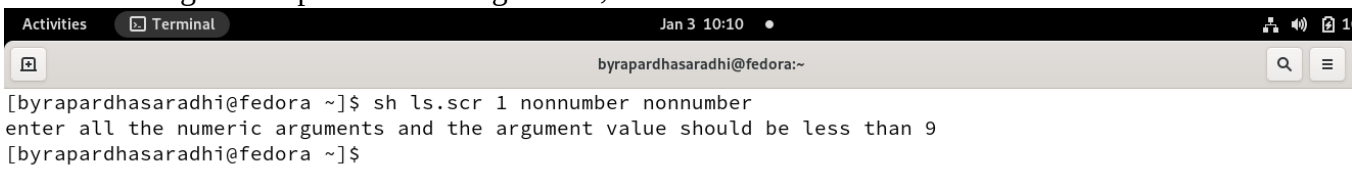
- Testing the script with two arguments.



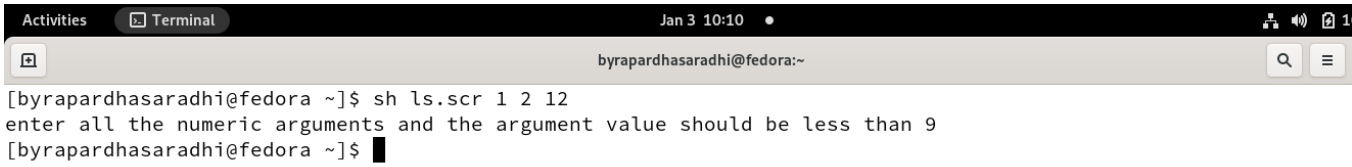
- Testing the script with three arguments, one of them nonnumeric.



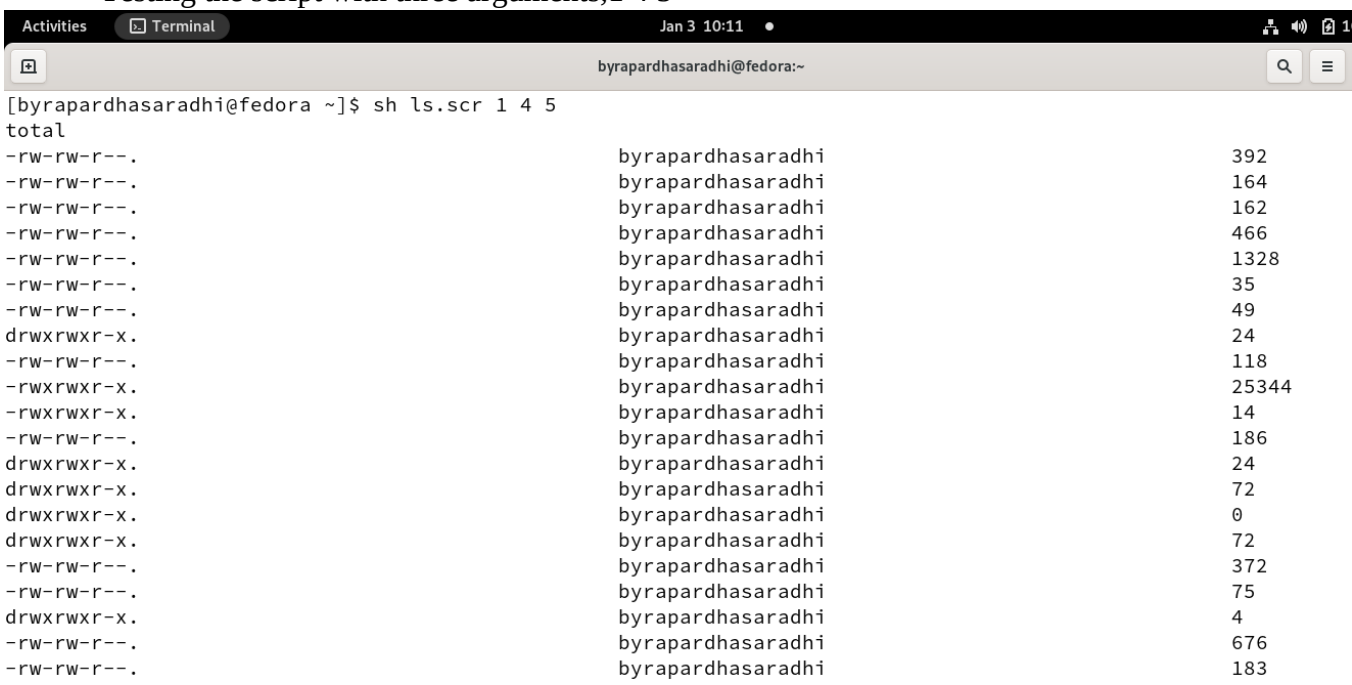
- Testing the script with three arguments, two of them nonnumeric.



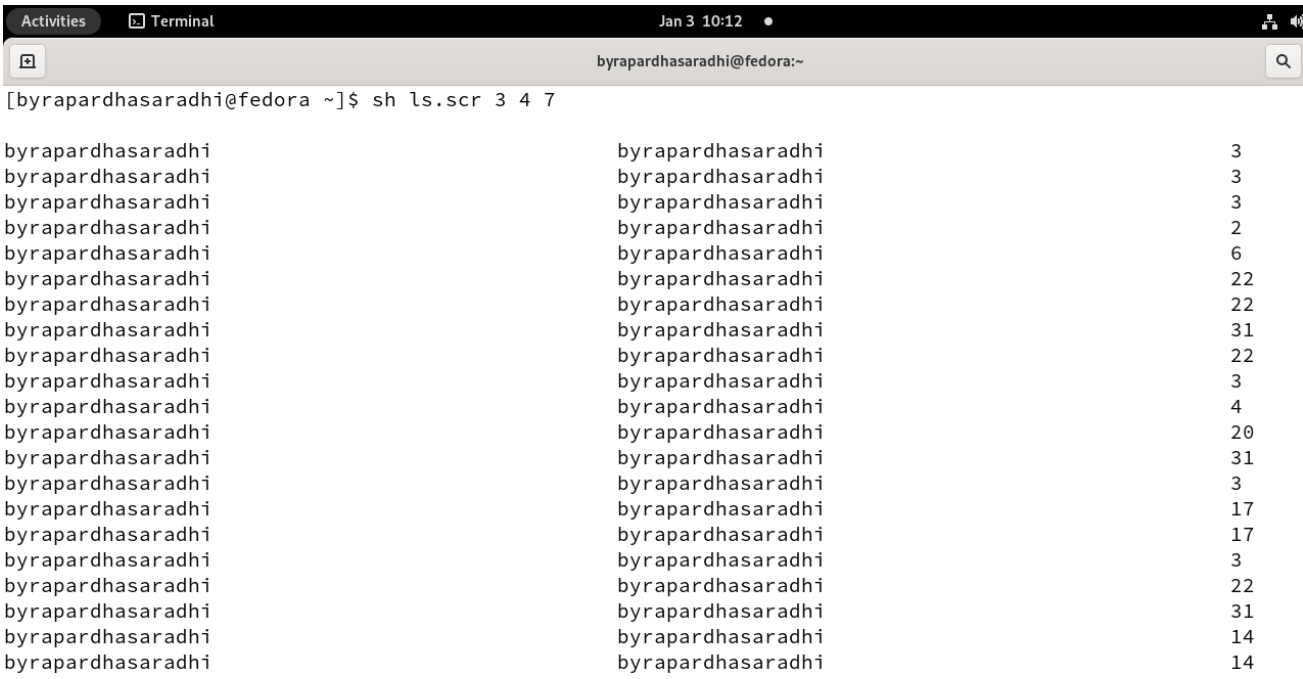
- Testing the script with three arguments, one of them too large.



- Testing the script with three arguments,1 4 5



- Test the script with three arguments,3 7 1



Testing the Effect of the Script:

- None

Result:- The required fields in the ls-l command were displayed.

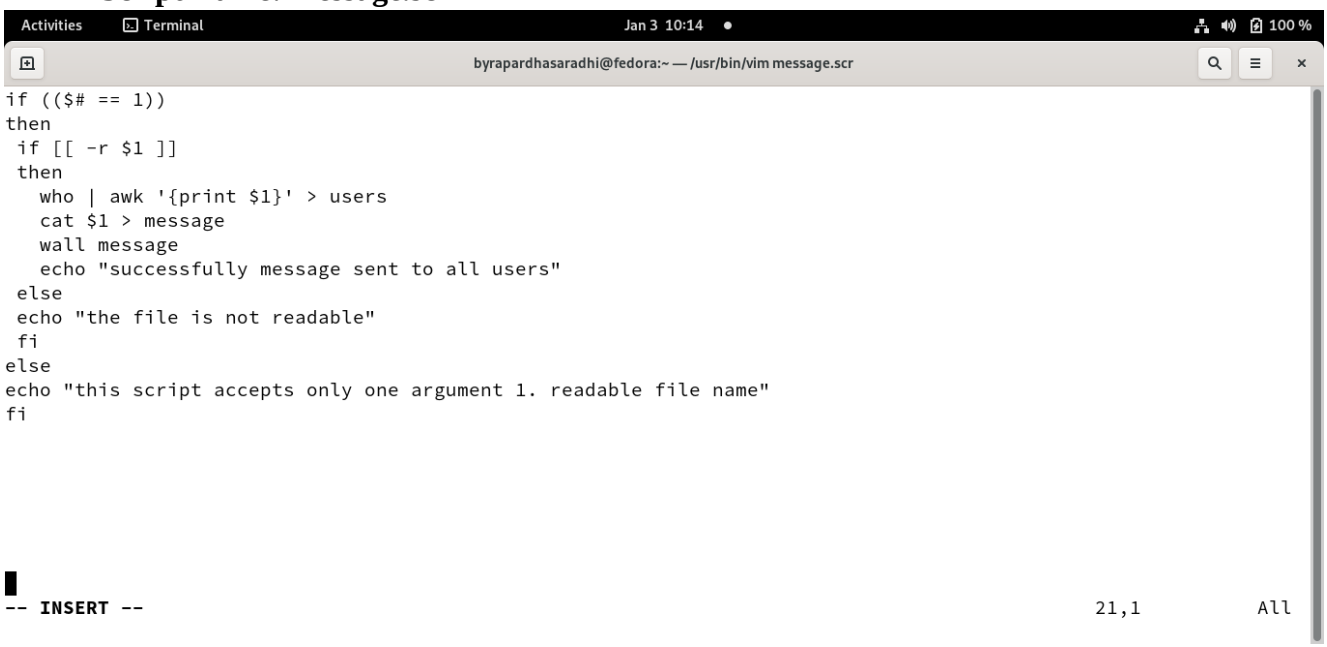
7. Create a script that sends contents of a message file to everybody who logged in.

Preparation:

- Create a file of a short friendly message and mention that this is a test message that should be discarded by the receiver

Script:

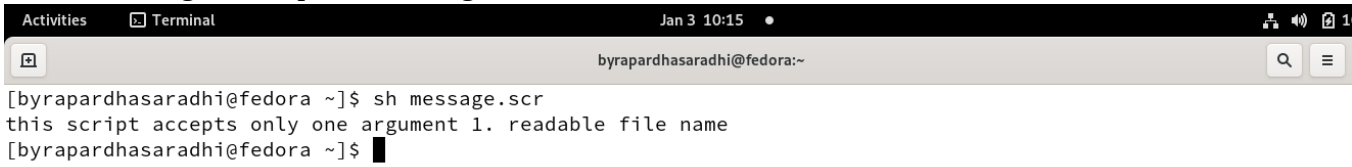
- **Script Name: message.scr**



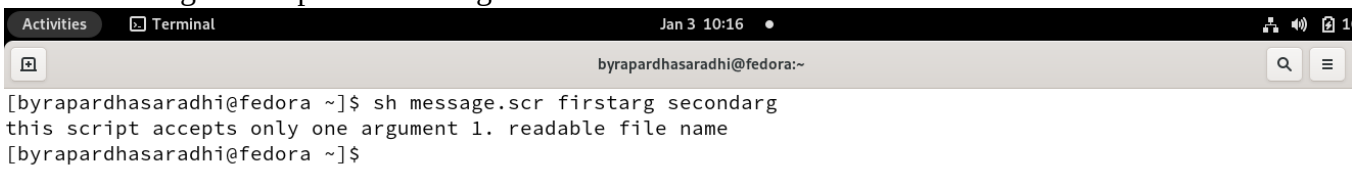
- **Arguments:** One argument, a message file.
- **Validation:** The minimum validation requirements are:
  i. Ensure that exactly one argument is entered.
  ii. Ensure that the argument is a readable filename.
- **Body Section:** Create a script that uses awk to create a temporary file containing the usernames of those users who are logged into the system at this moment. Then send the message contained in the first argument to every logged-in user. Note that a user who has logged in more than once should receive only one message.

Testing the Script:

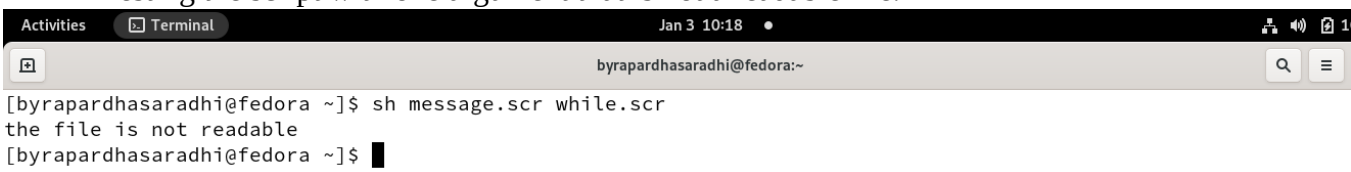
- Testing the script with no arguments.



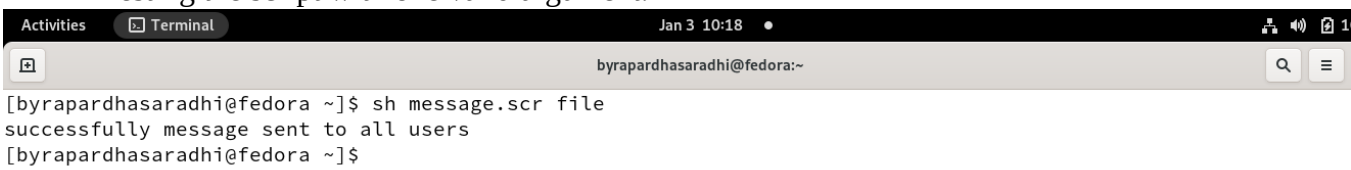
- Testing the script with two arguments.



- Testing the script with one argument that is not a readable file.

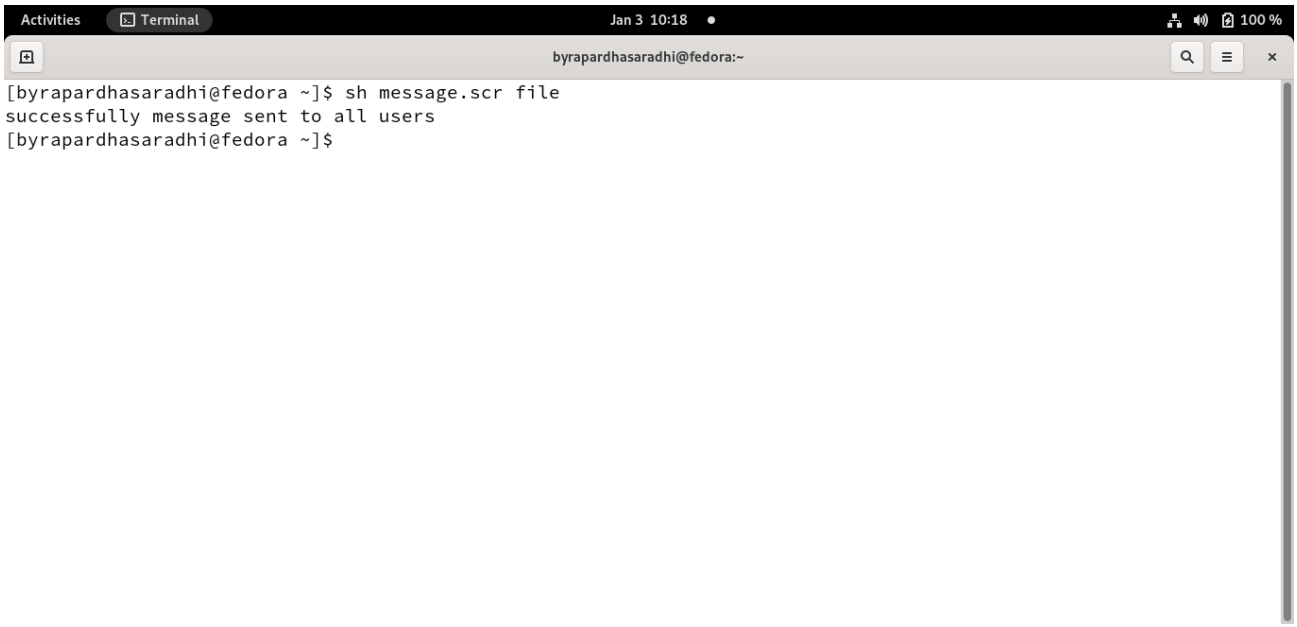


- Testing the script with one valid argument.



Testing the Effect of the Script:

- You should include yourself in the recipient list. Check to see if you have received the message.



8. Create a script that can be executed only from a specific terminal. This is done for security purposes. For example, a superuser may write scripts that can only be executed from his or her office and nowhere else.

Preparation:

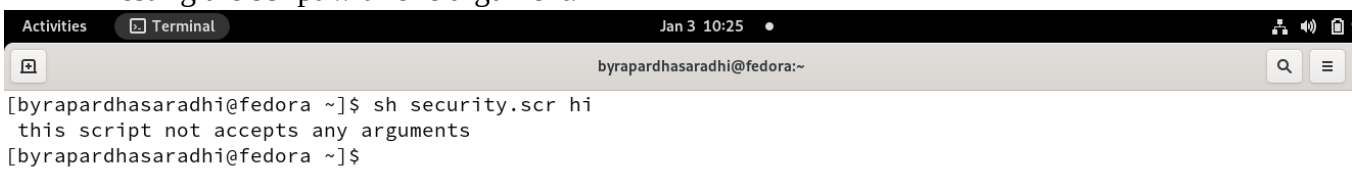
- None

Script:

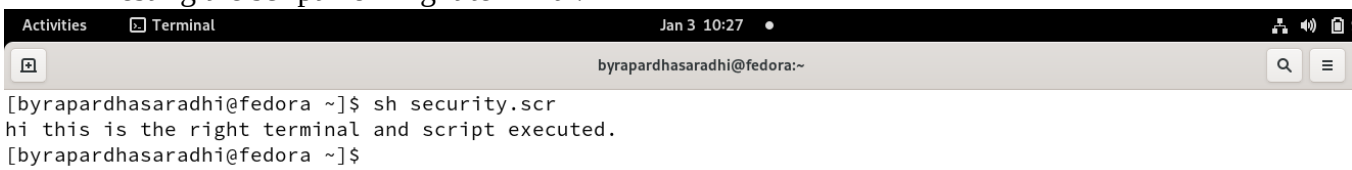
- **Script Name: security.scr**
- **Arguments:** None.
- **Validation:** The minimum validation requirements are:
  i. Ensure that no argument is entered.
- **Body Section:** Create a script that prints a friendly message. However, the script can be executed only for one terminal. You can use the name of the terminal you are using when you write the script. If somebody uses the script from a terminal that is not authorized, the script is to exit immediately. Hint: Use the tty command to show your current terminal.

Testing the Script:

- Testing the script with one argument.



- Testing the script from right terminal.



- Log into the system using another terminal and test the script.

Testing the Effect of the Script:

- None

Result:- Script tested in other terminals and is not executed and it's only executing in the right terminal only.

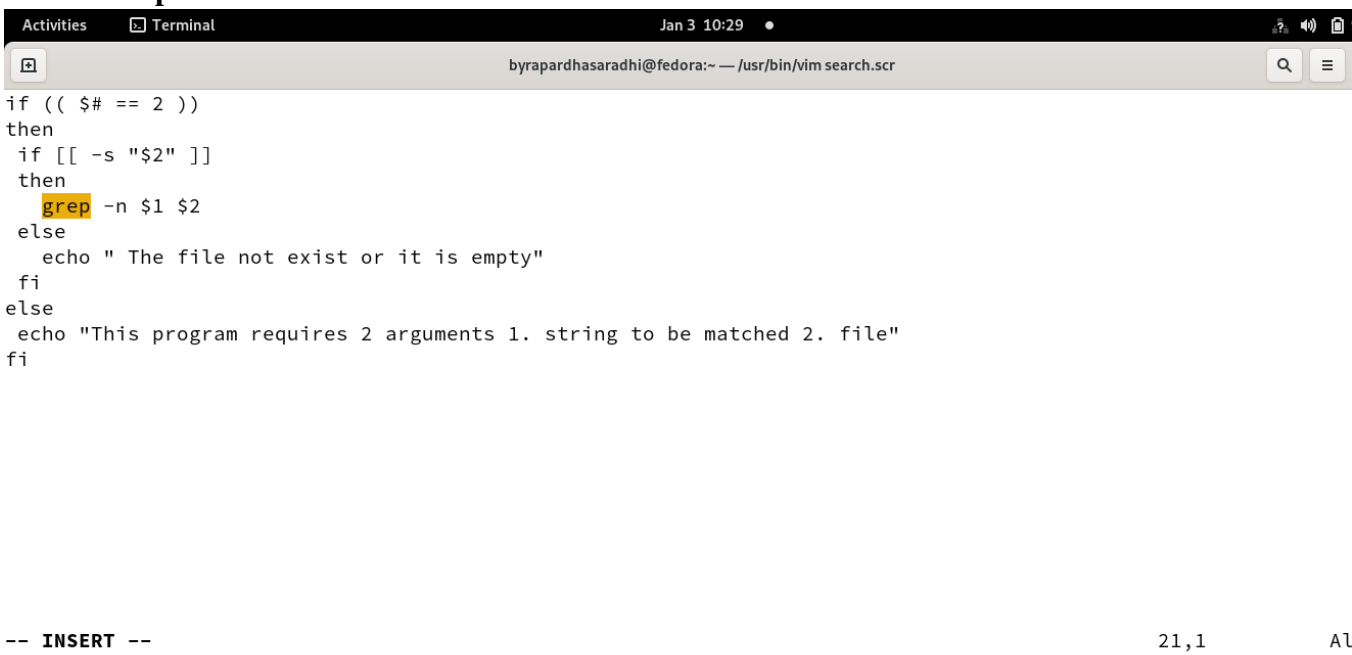
9. Create a script that finds each line in a file that contains a specified string.

Preparation:

- Create a file of at least 20 lines and insert a double quoted string, such as "hello," in several lines.

Script:

- **Script Name: search.scr**

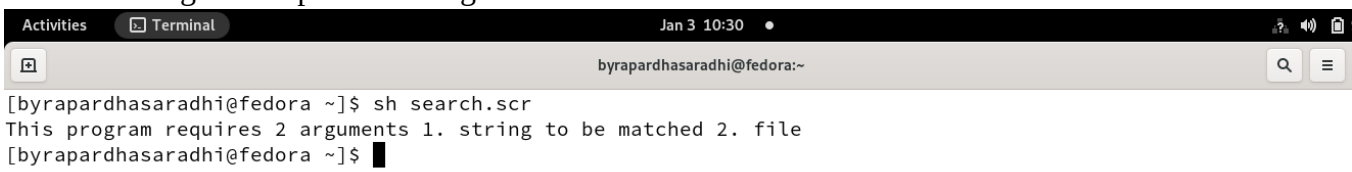


- **Arguments:** Two arguments, the first is the string to be found; the second is the name of the file.

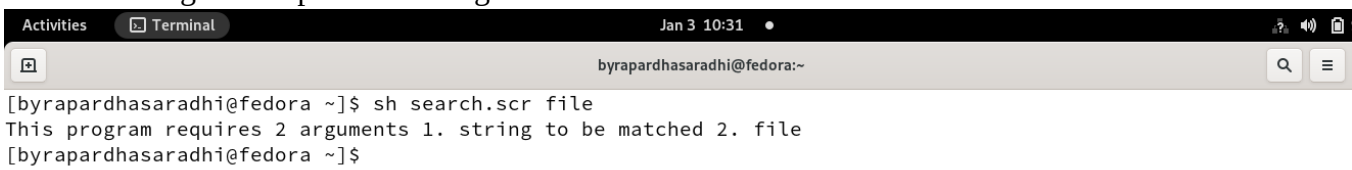
- **Validation:** The minimum validation requirements are:
  i. Ensure that exactly two arguments are entered. ii. Ensure that the second argument is the name of the file that exists and is not empty.
- **Body Section:** Create a script that uses grep and loops to find the line numbers in which the string is found. Note that grep should be applied to each line, not the whole file. The script should print the result in the following format:

Line Number: [Line contents] Testing the Script:

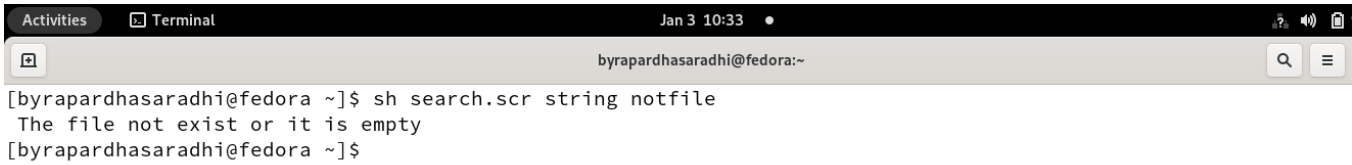
- Testing the script with no arguments.



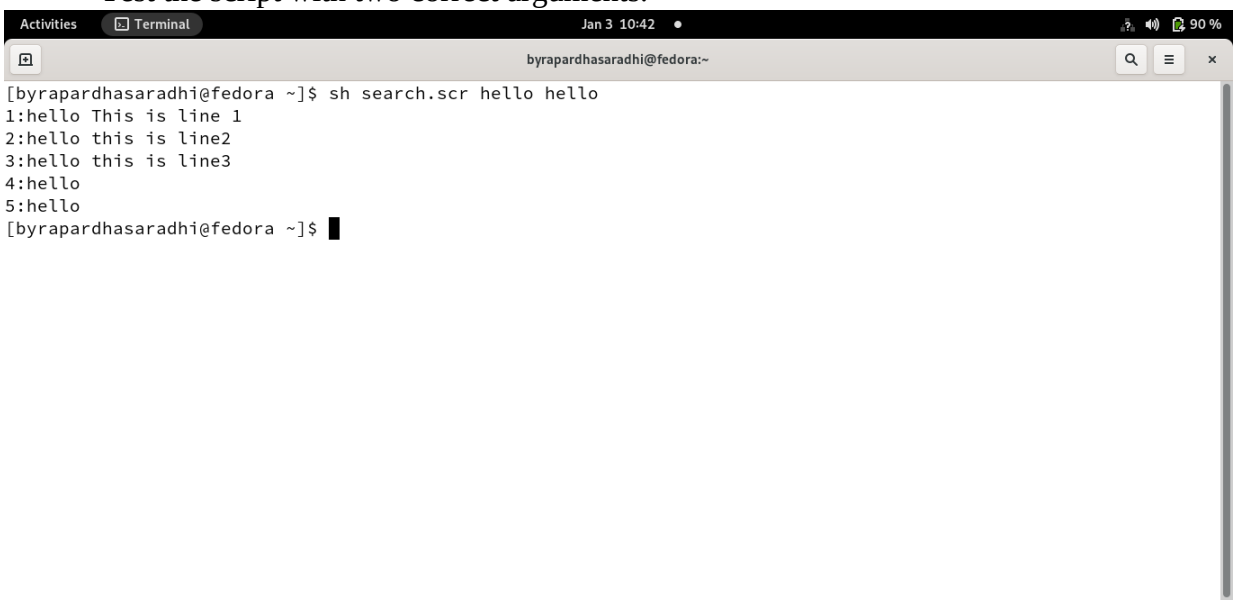
- Testing the script with one argument.



- Testing the script with two argument but the second one is not a file.



- Test the script with two correct arguments.



Testing the Effect of the Script:

- Compare the results of your script with a printout of the file.

Result:- Results were compared with printout of the file and the line numbers are the exact matched lines.

10. Create a script that compiles all C source files in your home directory and create executable files.

Preparation:

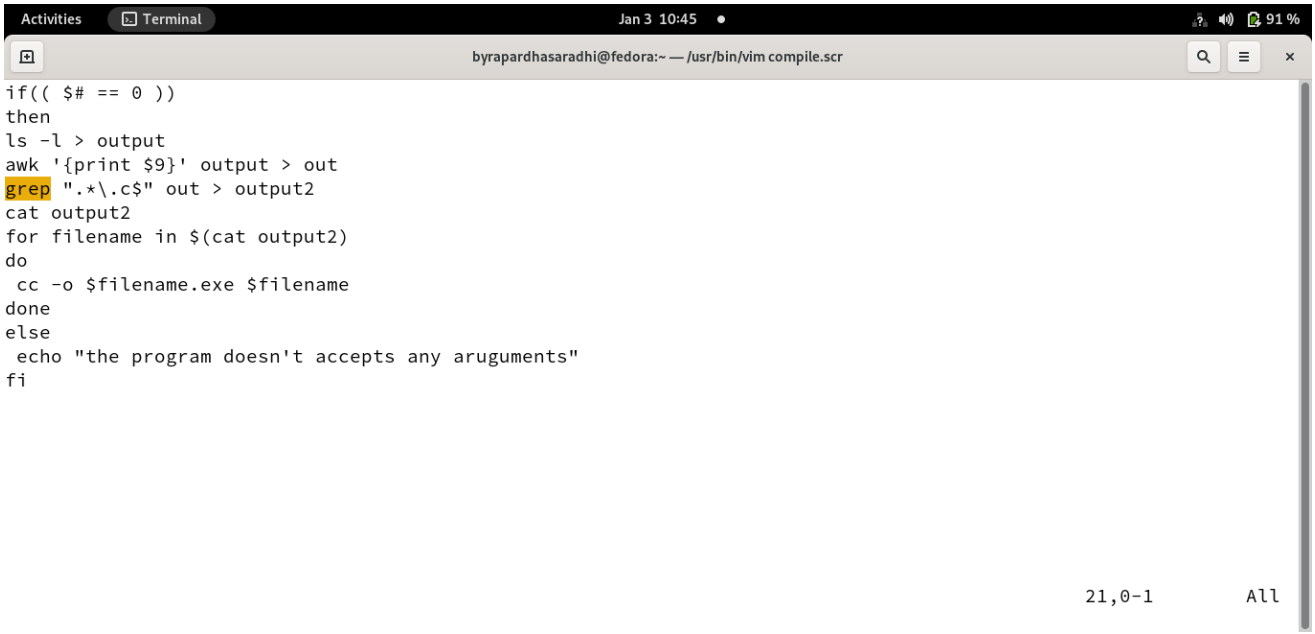
- Create at least five C source files in your home directory. The files do not have to be real C source files; at a minimum they should contain a comment line that contain a unique program name such as the following example:

  /* .....file1.c .....*/

  The name of the files should have a C source file extension (.c), such as file1.c.

Script:

- **Script Name: compile.scr**

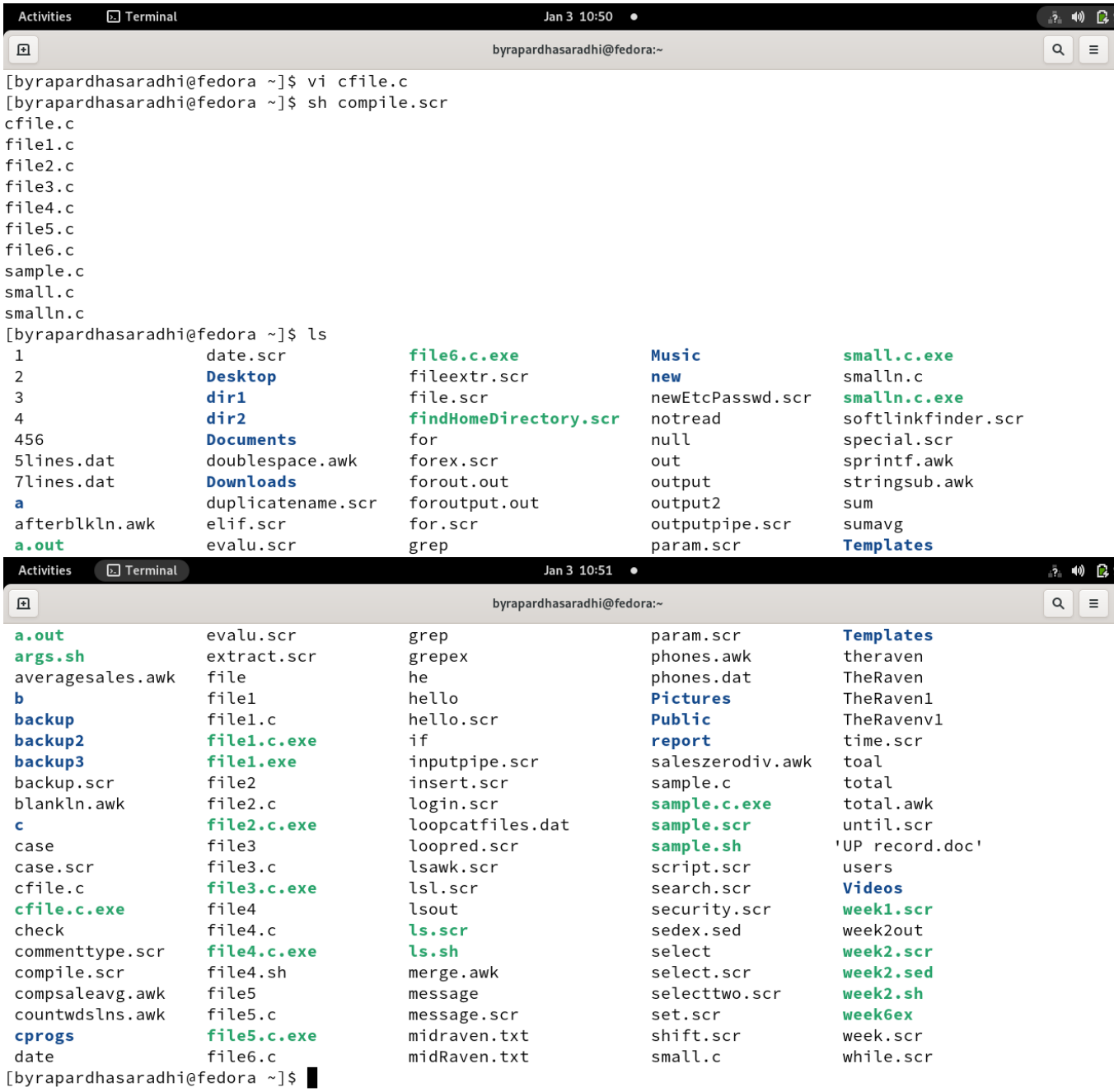


- **Arguments:** Two arguments, the first is the string to be found; the second is the name of the file.
- **Validation:** The minimum validation requirements are :
  i. Ensure that there is no argument
- **Body Section:** Create a script that finds all files with extension (**.**c) under your home directory and compiles them one by one. Each executable file should have the same name as the source file except that the extension should be (**.**exe). For example, if the source filename is file1.c, the executable filename should be file1.exe. Use the following command to compile: cc -o executable_filename source_filename

Testing the Script:

- Test the script with one or two arguments.

- Test the script with no arguments.



Testing the Effect of the Script:

- Verify that executable files were created under your home directory.

Result:- It is verified that the required executable files are created as shown above in snapshot.

11. Create a script that finds all files in subdirectories that have the same filename.

Preparation:

- Make several directories, at different levels, under your home directory. For example, make ~/A, ~/B, ~/C, ~/A/AA, ~/A/BB, ~/A/AA/AAA, and so on until you have at least 15 directories. Copy a small junk file named file1 under some of these directories; do not change its name. Copy another small junk file named file2 under some other directories. Copy a third junk file under several directories. Be sure that some directories get a combination of file1 and file2 or file1 and file3. In at least three of the directories, create a junk file with a unique name.

Script:

- **Script Name: duplicateName.scr**
- **Arguments:** None
- **Validation:** The minimum validation requirements are :
  i. Ensure that there is no argument.

- **Body Section:** Create a script that uses find and awk commands to create a list of files that are duplicated; use the full pathname for the duplicated filenames. Hint: Use a basename command and an array in awk. The output should look like the following example:

  file1: ~/A/file1 ~/A/AA/file1 ~/A/B/BB/BBB/file1
  file2: ~/B/file2 ~/C/file2
  . . .

Testing the Script:

- Test the script with one argument.
- Test the script with no arguments.

Testing the Effect of the Script:

- Use a recursive long list command to list the complete contents of your home directory. Verify the output of your script against the list command output.

12. Create a script that search for multiple occurrences of the specified string in each line. **Preparation:**
    - Create a file of at least 20 lines and insert a double quoted string, such as "hello," in several lines.
    - Include two or three occurrences of the string in some lines.

Script:

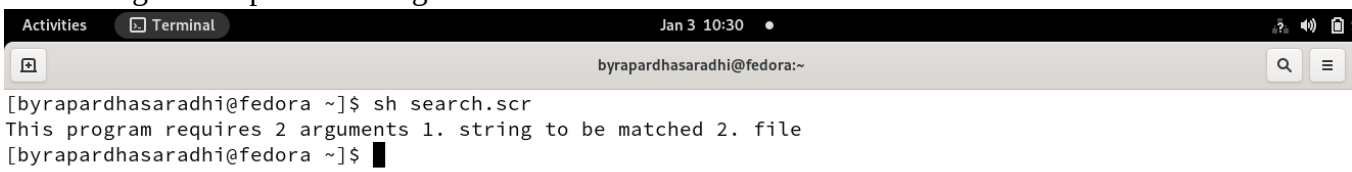
- **Script Name: search.scr**

- **Arguments:** Two arguments, the first is the string to be found; the second is the name of the file.
- **Validation:** The minimum validation requirements are :
  i. Ensure that exactly two arguments are entered.
  ii. Ensure that the second argument is the name of the file that exists and is not empty.
- **Body Section:** Create a script that uses grep and loops to find the line numbers in which the string is found. Note that grep should be applied to each line, not the whole file. The script should print the result in the following format:

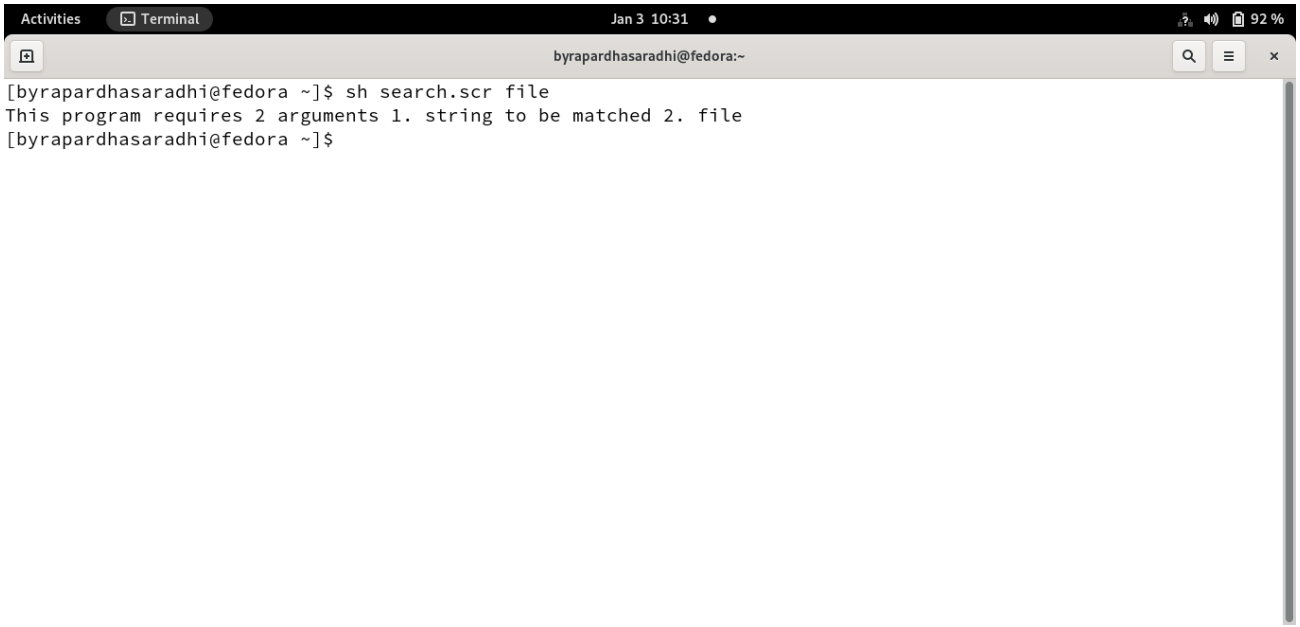
  Line Number: [Line contents]

Testing the Script:

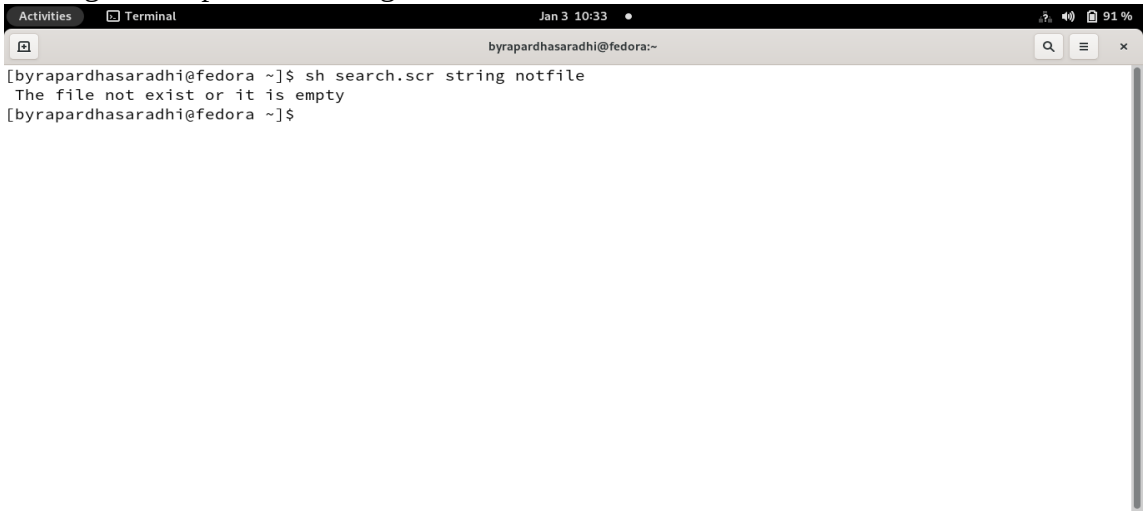
- Testing the script with no arguments.



- Testing the script with one argument.



- Testing the script with two argument but the second one is not a file.



- Test the script with two correct arguments.

Testing the Effect of the Script:

Compare the results of your script with a printout of the file.

Result:- Compared the results of the script with printout of the file and got exact matching results.