##### **File system manager project 4**

Grand Canyon University

CST-315-WF900A

CST-315 Introduction to Operating Systems

Ethan Gladen, Gabriel Aracena and Tyler Allen Hall

Ricardo Citro

4/10/22

##### **File system manager project 4**

For this assignment we improved the shell that we developed for *Project 1 - Improved UNIX/Linux Command Line Interpreter* and implemented Project 3: Short-Term Process Scheduler, and added the final piece for the shell, Project 4: A File System Manager. Furthermore, we added custom commands that will meet the requirements for the commands specified in the assignment. For our code we have multiple commands that implement what a file system manager would do similar to the linux commands. There are at least 13 custom commands we had to implement which are listed in further detail, with an image to describe the correct use of it, as well as an explanation of what each one does and our implementation for each command. However to make each command, function work with the shell properly we needed to make a int function of each command like our exit custom command so when the string is received from the user to do our custom command, it will store the string of command from user in args[0] in the array and using strcmp() function it will compare the string of the user string and if the string is like our custom command, we return that custom command function that will execute their goal. The issues that we ran into had to do with deleting the directory which will be discussed in the test and validated.

1. For our first command is making the directories which uses the custom command we made “crdir” . The design is when the command is called it goes to return “dash\_create\_dir” which is where the function is made to create a new directory and our implementation is when we char name and have it to where whatever directory name is in args[1] then in args[2] will be the new name that the directory is changed to. We use 0755 filename command to allow the user to change the name.

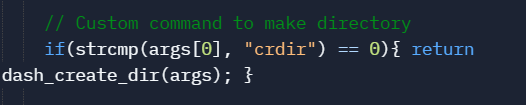


Figure 1.1: Depicts the “crdir” command in our program that creates a new directory.

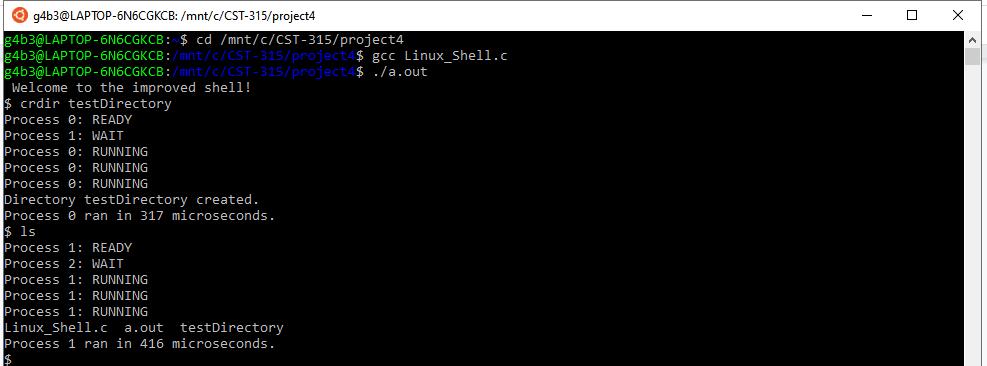


Figure 1.2: Screenshot showing the correct execution of the crdir command on Linux.

1. Next we also created a custom command that will change the directory of the current path, we named it “chgdir” The design is when the command is called it goes to return “dash\_change\_directory” which is when it goes to execute our change direction function and how we implemented it is having char target = to args[1] which is the name of directory from the user and then changes the pathing using the chdir() function.

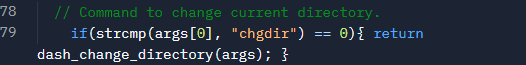


Figure 2.1: Depicts the “chgdir” command in our program that changes the current path directory.

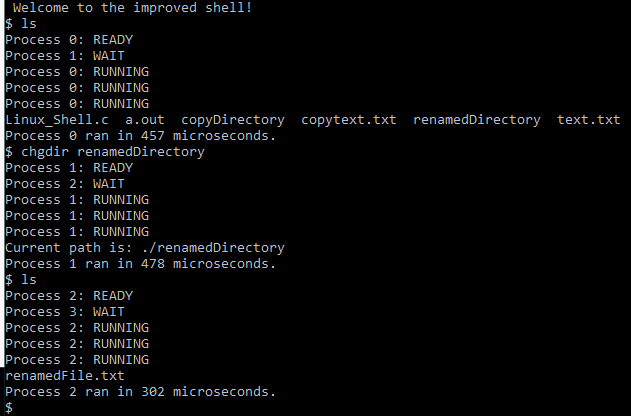


Figure 2.2: Screenshot showing the correct execution of the chgdir command on Linux.

1. Third command created is to delete a directories, having special handling with non-empty directories, we created the random command called “dltdir” for it. The design is when the command is called it goes to return “dash\_delete\_directory” , and for the implementation, it goes through each file within the directory and deletes the file. When it encounters a directory, it calls the command again, opening up and deleting everything within that directory, before deleting itself. It does this until there are no files or directories left within the original directory, where it then deletes itself.

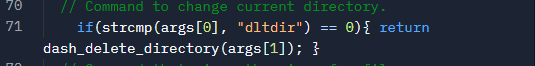


Figure 3.1: Depicts the “dltdir” command in our program that deletes and existing directory.

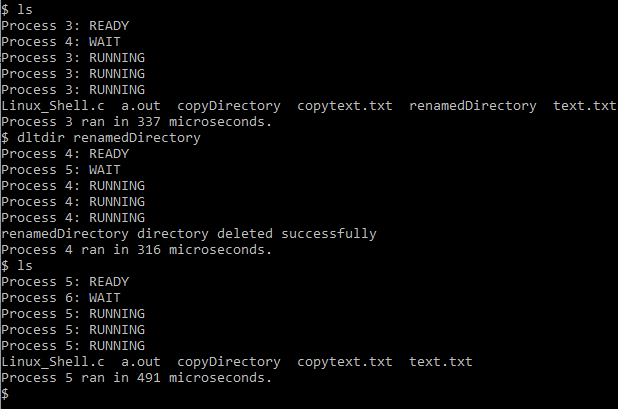


Figure 3.2:Screenshot showing the correct execution of the dltdir command on Linux

1. Fourth command created is to rename files or directories, we called our custom command “rename” to fulfill this function. The design for this function is that it renames files and directories to the new name the user wants. For the implementation we have in args[1] the name of the file or directory and it changes it to the new name from the user in args[2] of the array.

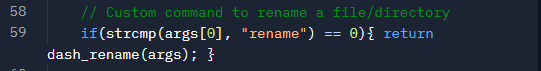


Figure 4.1: Depicts the “rename” command in our program that renames an existing file or directory.

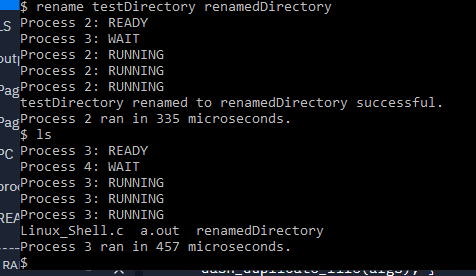


Figure 4.2: Screenshot showing the correct execution of the rename command on Linux.

1. Fifth command created is to delete files, for this purpose it works differently from deleting a directory, so we created another custom command called “dltf”. The design was to take the “dash\_delete\_file” and to return the function when it is called with the “dltf” custom command and for the implementation we get the file name from user and to put into args[1] in the array and to use the remove() function to delete all the contents and the file itself.

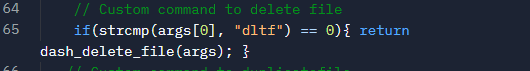


Figure 5.1: Depicts the “dltf” command in our program that deletes an existing file.

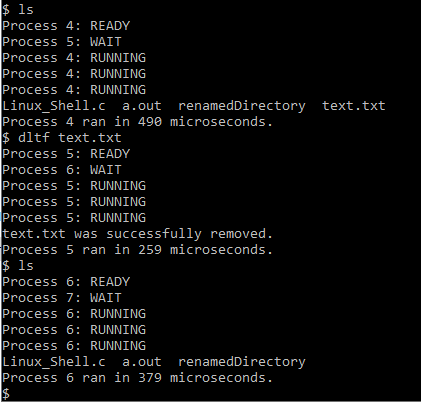


Figure 5.2: Screenshot showing the correct execution of the dltf comman on Linux.

1. We then created a command to move files across different directories, we called this custom command ”movef”. The design was to take the “dash\_move\_file” and to return the function when it is called with the “movef” custom command, the implementation was to have “source\_file” = args[1] and “target\_dir” = args[2] which like in the shell typing the command as [0] the file[1] and target directory[2]. Then we have FILE of fp1 and fp2, with fp1 being the file with the original content, and fp2 being the empty file that will take all the content of the original file.

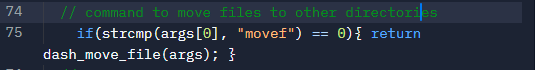


Figure 6.1: Depicts the “movef” command in our program that moves a file to another directory.

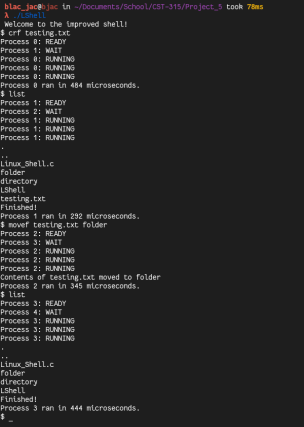
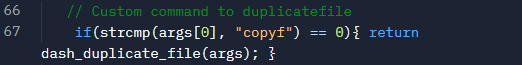


Figure 6.2: Screenshot showing the correct execution of the movef command on Linux.

1. Next command created is to duplicate any file, we created a custom command for that purpose and called “copyf”. The design for this command is to have “dash\_duplicate\_file” and to return the function when the custom command “copf” is called. For the implementation is to have FILE source and target and having source\_file =args[1] and target\_file = args[2], first is the file that has the contents you want to duplicate and second is the target file that you want to duplicate those contents to. Which

First the code opens the file and reads it and then goes to the target file to write the contents which then duplicates the file.

Figure 7.1: Depicts the “copyf” command in our program that copies an existing file.

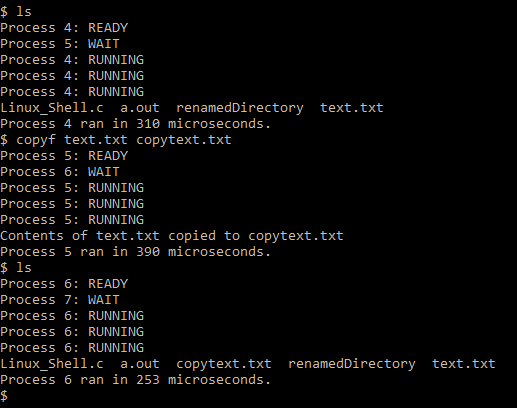


Figure 7.2: Screenshot showing the correct execution of the copyf command on Linux.

1. Following the same logic, we also created a custom command to duplicate an existing directory. The design is to take dash\_copy\_directory and to return the function when it is called with the “copydir” custom command. The implementation is to have a source[1] and target[2] and to have source\_path and target\_path using the getcwd() function which is to duplicate the inside of the folder and to move that content to the target folder.

How it works is that it runs through each within the directory the user wants to copy, creating an identical file. When it runs into another directory, it calls the command again, using that directory as the one to be copied, making it a recursive function.

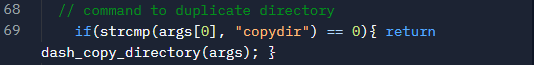


Figure 8.1: Depicts the “copydir” command in our program that duplicates an existing directory.

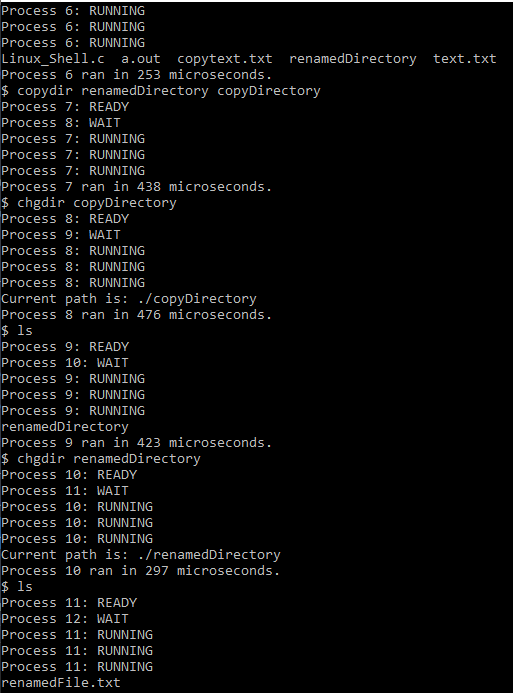


Figure 8.2: Screenshot showing the correct execution of the copydir command on Linux.

1. Ninth command created is a very useful one, this command is created to get information in the current directory and file, we named it “list”. The design is to list all the files in the directory with “dash\_list\_files” and to return that function when the name “list” is called from the user. For the implementation we also included a flag which can be used with a list called “list -d” which requires a space between list and -d, correct way to use command “list -d”, which lists all the detailed info that would be in a file and directory when in a linux shell, the Dev id, permissions the file has, the time of creation, links, etc. To get most of the permissions with easy implementation we used the POSIX library.

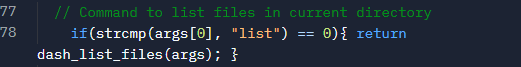


Figure 9.1: Depicts the “list” command in our program that displays all the files and directories in the current directory.

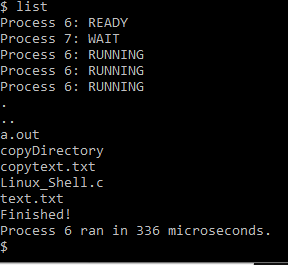


Figure 9.2: Screenshot showing the correct execution of the list command on Linux.

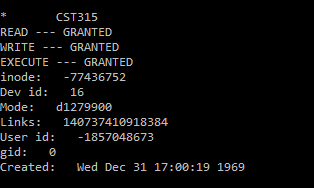


Figure 9.3 showing the execution list -d showing the detailed info of the 315 directory

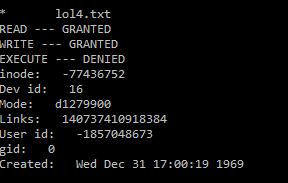


Figure 9.4 showing the execution of displaying detailed info of file using the flag -d

1. Following the same logic, we created a custom command that will show important information about a file, the “sif” command displays the size of a file. The design is to call the function “dash\_s\_file” when the custom command “sif” is entered from the user. The implementation is to display the accurate size of a file by bytes by opening the file and to then read the file and count each letter or character which is about the size of 1 byte. Which calculates all the characters giving the accurate size of a file.

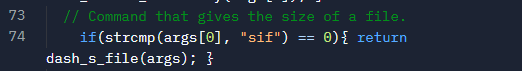


Figure 10.1: Depicts the “sif” command in our program that shows the size of an existing file.

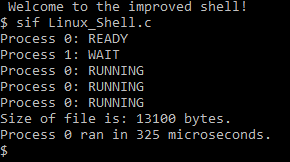


Figure 10.2: Screenshot showign the correct execution of the sif command on Linux.

1. After that, we had to create a custom command to let the user edit and already existing file, we named it “editf”. The design is to use the function “dash\_edit\_file” which returns when called “editf” by user. First we have source\_file= args[1] which then has a user\_string[1000] which allows the user to edit with that many chars and to then open the FILE ptr with the function “a” so the file can be edited by the user, and the user can add text and save it to a text file and when there done editing the file they can enter “DONE” to quit.

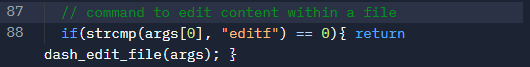


Figure 11.1: Depicts the “editf” command in our program that lets the user edit the content within a file.

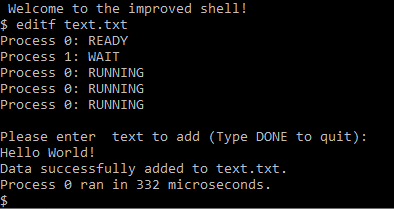


Figure 11.2: Screenshot showing the correct execution of the editf command on Linux.

1. We also created a custom command to display the info inside a file, we decided to name it “basicf”. The design is when the “dash\_basic\_file” function is returned, the user would call the command basicf. The implementation is to have source\_file =args[1], and FILE ptr, so using fopen the file is read and the content is printed using fgetc() showing the inside of the file which is the basic info.

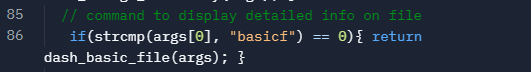


Figure 12.1: Depicts the “basicf” command in our program that displayes basic detailed information of a file.

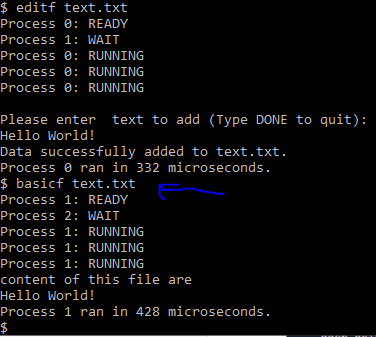


Figure 12.2: Screenshot showing the correct execution of the basicf command on Linux.

1. Here is our custom command to exit the like from project 1 improved shell, with the design is when “exit” is called then returns the function “dash\_exit” which is the implementation is return 0; which exits the shell.

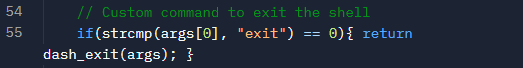


Figure 13.1: Depicts the “exit” command in our program that exits the shell.

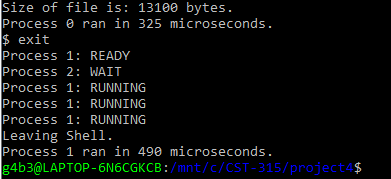
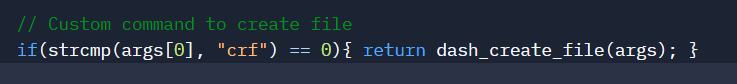
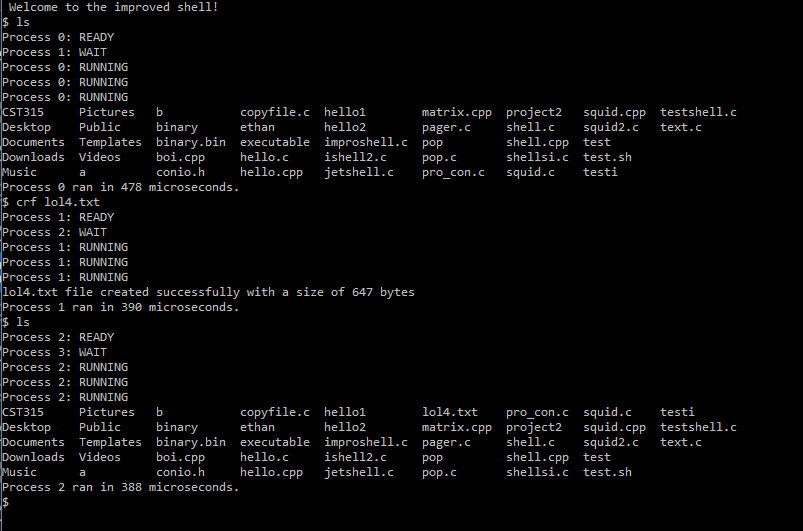
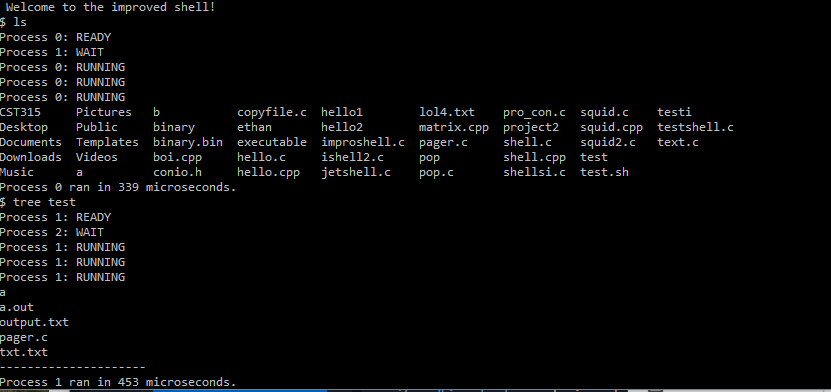


Figure 13.2: Screenshot showing the correct execution of the exit command on Linux.

1. Another command to implement is to create a file which we named “crf” . The design is when “dash\_create\_file” is returned, its when the name “crf” is called by the user. The implementation is to randomize the amount of bites when creating a file, so file\_name[1] is where the name is put by user and to simulate what running a file is like when a file is made rand() % 5000 bytes when a file is made.



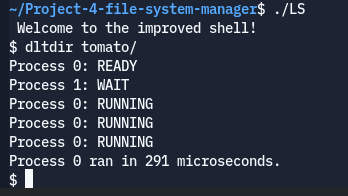
1. The final command that we implemented is one that displays a directory tree is called as a custom command the “tree”. For the design when the user calls the custom command “tree” it returns the function “dash\_display\_tree”. The implementation is exactly what list does, it just when it runs into a directory it will list all the contents within it. We used d\_name from the POSIX c library and to have DIR dir so it can open the directory from the user.

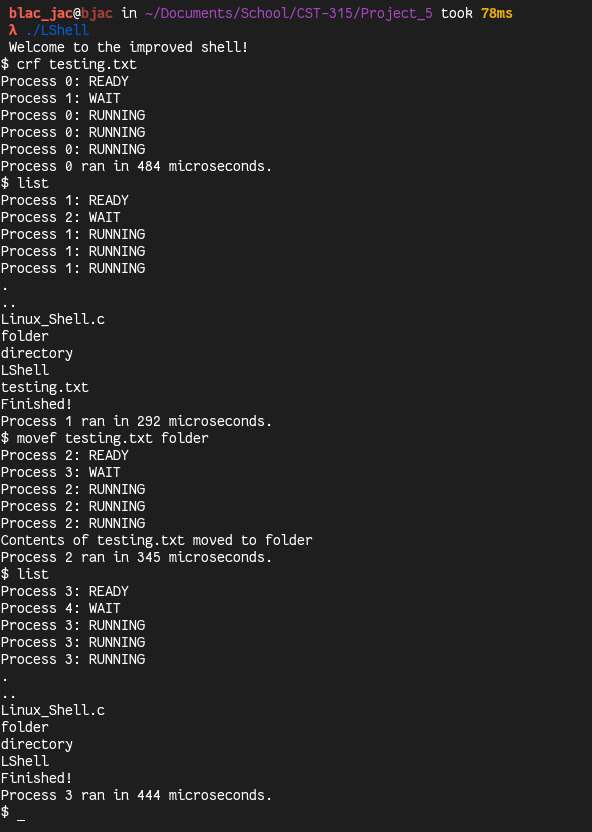




Test and Validate

There is a problem as we try to implement the delete dir function, there is no result and the directory tomato is still there not being deleted. It would go through each file, but not actually delete the directory at the end. How we fixed this was by changing the command that is called at the end from rm to rm -r, as it had to be recursive in order to cycle through each directory within the one we want to delete. Once that was changed, it deleted the directory properly.



Correct result

References

C Library Function - strcmp(). (n.d.). Retrieved April 10, 2022, from https://www.tutorialspoint.com/c\_standard\_library/c\_function\_strcmp.htm