The File System Project
(The Segmentation Faults)

Github Link:
https://github.com/CSC415-Fall2020/group-term-assignment-file-system-julia-ramos.git

Class: CSC-415-03

Name	Student ID
Julia Beatriz Ramos	920638040
Danish Siddiqui	920258388
Zachary Ma	916750281
Michael Morales	920237211

Description

A file system is a way a computing system controls the flow of data. File systems generally manage how data is both stored and retrieved, using a variety of different methods. Data is typically broken up into blocks, and the block size can vary from system to system. Thus making it difficult to narrow down what kind of system we had wanted to be implemented.

The File System created by Segmentation Faults uses a shell script with a bitmap allocation system to handle data storage and retrieval. The system has the ability to create, read, write, delete, and copy files to and from the file system to a Linux system.

Development Process

To split up our work, we decided that Danish would work on directory functions, Michael would work on buffered i/o functions, Julia would work on free space functions, and Zach would work on integrating fsshell. In the beginning, it was very difficult to start on our tasks because we all needed the basic foundational knowledge about how our file system was going to work. After a couple of weeks of trying to format the disk together, we were starting to unravel the onion and understand the basic functionality of our file system.

In the initial stages of design, we started with a linked list of free space allocation. The reason why it was initially chosen was that we believed that each file created in the system would be part of a linked list of disk blocks. These blocks could then be grouped into clusters which reduce the risk of pointer loss, and data fragmentation. It was also the one allocation type everyone had some familiarity with. Implementing the linked list allocation proved to be too difficult due to search and manipulation complexities, so we switched to a bitmap free space allocation system instead. By using a bitmap allocation method over the linked list, it allowed us to read and write blocks contiguously. This means that instead of having to search the whole file system for a free block, you would only have to find the first free block. This made reading and writing to them much easier.

Implementing the functions in mfs.c was the biggest struggle at first. We needed to know which functions were being called for which shell command. Using printfs in each function helped us prioritize which functions were being depended on, which gave us an order to our development process. After implementing our currentDirectoryPtr to keep track of where we were in our file system, this made the process of implementing the rest of the functions (like fs_getcwd and fs_setcwd) much easier.

After implementing a functioning directory system, we moved on to file functions and buffered reading/writing. It was helpful to refer back to Assignment 2 and Assignment 5 to borrow our buffered reading/writing logic but was very difficult to integrate it to work with our file system. Specifically, we needed to create our own fs_open() that would create our files and return their positions in our LBA.

Lastly, we needed to integrate all of the functions to work together in the fsshell and test all of the commands again. We could not debug all of our errors, but we got our basic shell commands to work and we are proud of the final product.

Issues

There were many challenges when creating our file system. One of the major ones that appeared was that most of the group had little true experience creating and implementing a root directory. This made knowing where to start coding difficult as we were unsure of where exactly we should have started.

Another issue we found was after creating the root directory and initializing our project was the difficulty of some of the directory functions. One of the first major hurdles was the implementation of the make directory. The core issue was due to the various functions that our function had to be aware of. It had to make sure that the directory it created was on a free block, that the directory was properly writing to the volume and that the bitmap was keeping track of the new directory.

Another directory function that proved challenging was the change directory function. What made this function difficult to implement was the fact that we had issues keeping track of our current and temporary directory pointers. Which makes changing our directory to an existing one difficult when it wouldn't be able to find the created directory. By fixing the bitmap and directory pointers, we were able to get it to work. Another minor bug we encountered was that our change directory function could only move into one directory at a time.

We also had some issues with copying files to and from the Linux system to our own. While initially, the functions worked fine on anything less than 200 bytes, anything over 200 bytes would cause our program to loop endlessly. To fix the issue, we had to return exactly how many bytes were returned instead of a whole 512 block that was being returned by LBAwrite/LBAread. We also had to deal with a minor bug where our copy function would write more bytes than it was given.

The concept of persistence we discovered was also a somewhat minor issue we solved. It was resolved by adding a flag to our MBR function that would check if our volume existed, and that any read or writes made by the directory functions were actually written to the volume.

Setup

git clone the assignment from
(https://github.com/CSC415-Fall2020/group-term-assignment-file-system-julia-ramos.git)

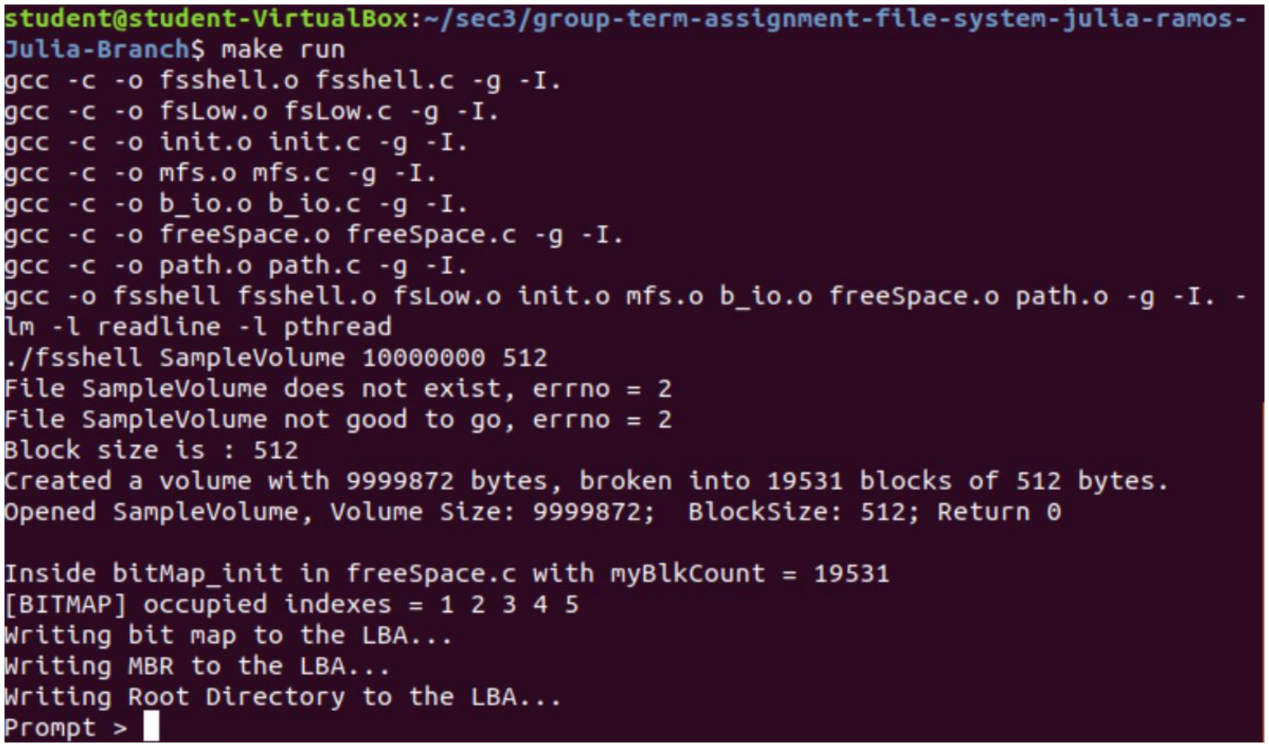
Type make clean in the command line.
Type make run in the command line to start and initialize our file system.

The following are commands that you can type to the shell.
File system directory commands:
ls - Lists the file in a directory
cp - Copies a file - source [dest]
mv - Moves a file - source dest
md - Make a new directory
rm - Removes a file or directory
cp2l - Copies a file from the test file system to the linux file system
cp2fs - Copies a file from the Linux file system to the test file system
cd - Changes directory
pwd - Prints the working directory
history - Prints out the history
help - Prints out help
exit - Exits the

Here are some additionally implemented commands you can type in the shell:
ls -la - Prints values from the fs_stat function.
cd ~ - Returns to root directory
cd .. - Returns to parent directory.

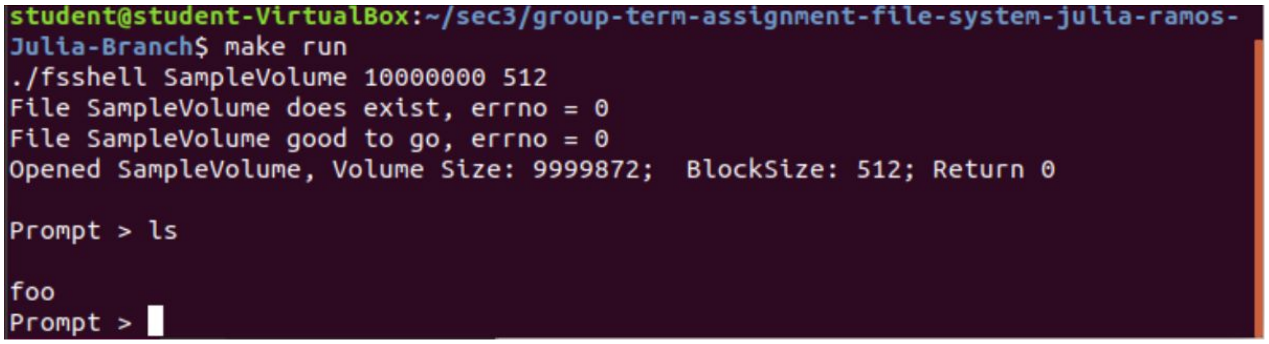
Screenshots

Initialize Volume



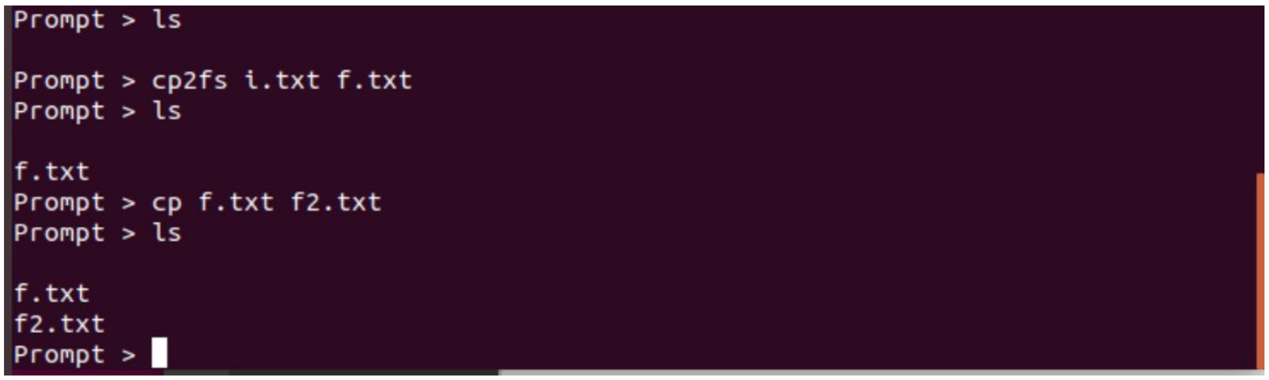
```
student@student-VirtualBox:~/sec3/group-term-assignment-file-system-julia-ramos-
Julia-Branch$ make run
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o fsLow.o fsLow.c -g -I.
gcc -c -o init.o init.c -g -I.
gcc -c -o mfs.o mfs.c -g -I.
gcc -c -o b_io.o b_io.c -g -I.
gcc -c -o freeSpace.o freeSpace.c -g -I.
gcc -c -o path.o path.c -g -I.
gcc -o fsshell fsshell.o fsLow.o init.o mfs.o b_io.o freeSpace.o path.o -g -I. -
lm -l readline -l pthread
./fsshell SampleVolume 10000000 512
File SampleVolume does not exist, errno = 2
File SampleVolume not good to go, errno = 2
Block size is : 512
Created a volume with 9999872 bytes, broken into 19531 blocks of 512 bytes.
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0

Inside bitMap_init in freeSpace.c with myBlkCount = 19531
[BITMAP] occupied indexes = 1 2 3 4 5
Writing bit map to the LBA...
Writing MBR to the LBA...
Writing Root Directory to the LBA...
Prompt >
```

ls - Lists the file in a directory



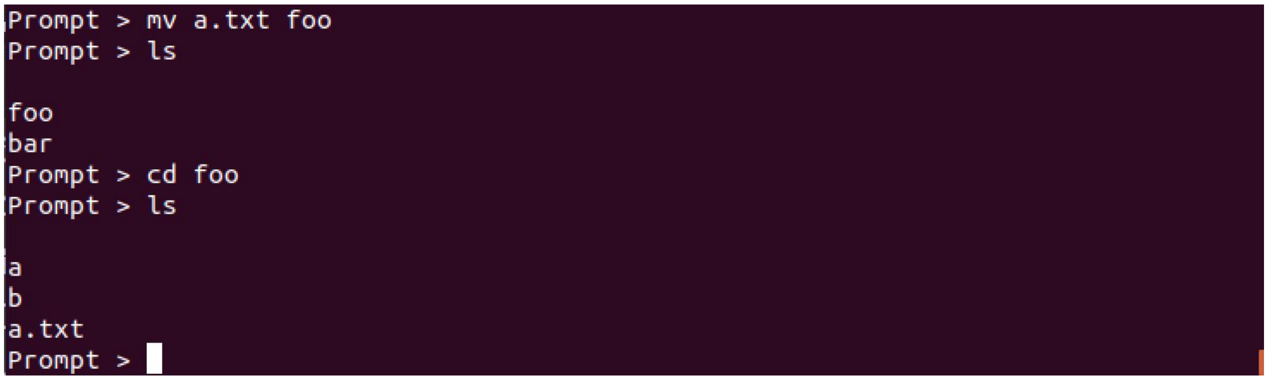
```
student@student-VirtualBox:~/sec3/group-term-assignment-file-system-julia-ramos-
Julia-Branch$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0

Prompt > ls

foo
Prompt >
```

cp - Copies a file - source [dest]



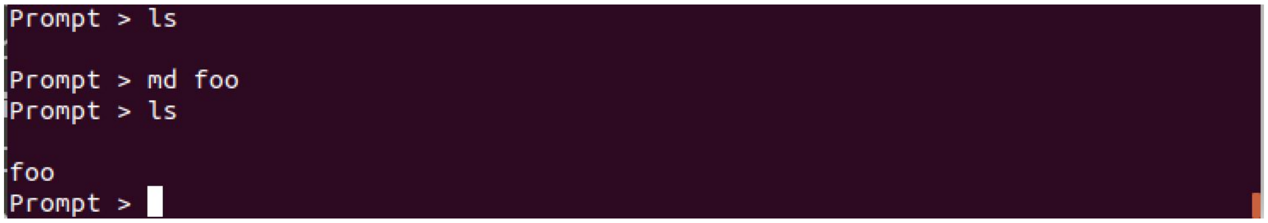
```
Prompt > ls

Prompt > cp2fs i.txt f.txt
Prompt > ls

f.txt
Prompt > cp f.txt f2.txt
Prompt > ls

f.txt
f2.txt
Prompt >
```

mv - Moves a file - source dest



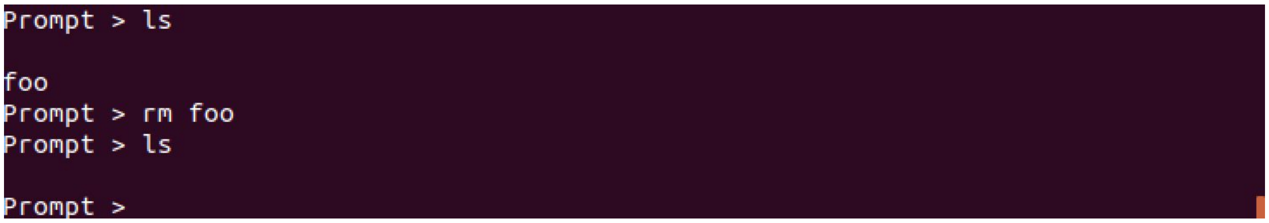
```
Prompt > mv a.txt foo
Prompt > ls

foo
bar
Prompt > cd foo
Prompt > ls

a
b
a.txt
Prompt >
```

md - Make a new directory



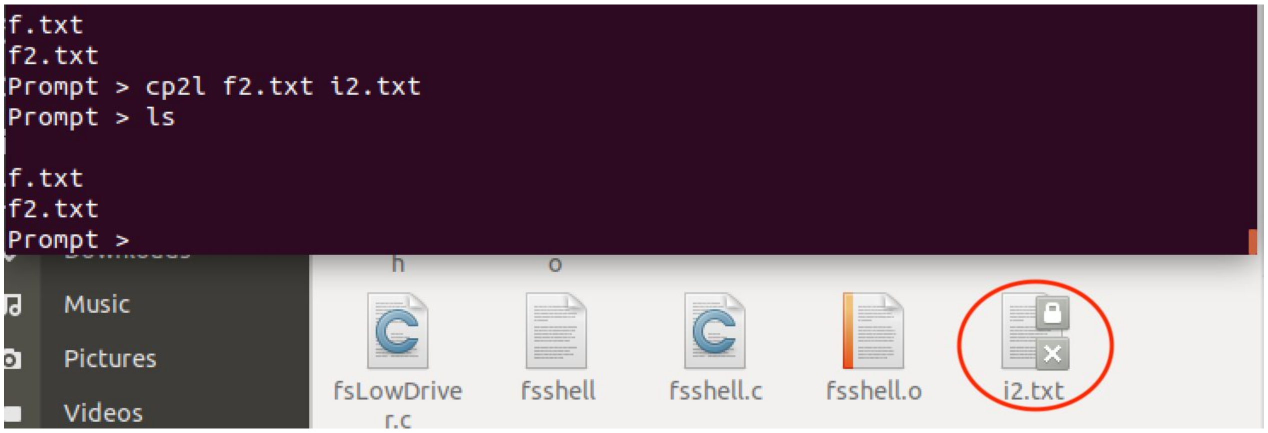
```
Prompt > ls

Prompt > md foo
Prompt > ls

foo
Prompt >
```

rm - Removes a file or directory



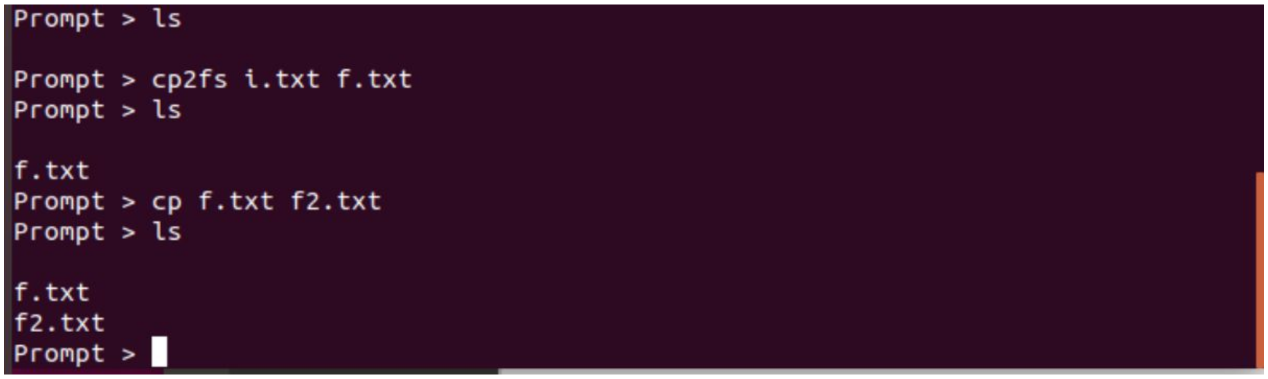
```
Prompt > ls

foo
Prompt > rm foo
Prompt > ls

Prompt >
```

cp2l - Copies a file from the test file system to the linux file system



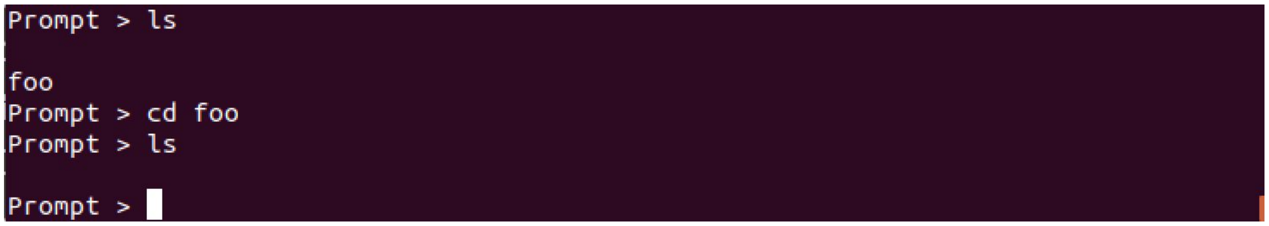
```
f.txt
f2.txt
Prompt > cp2l f2.txt i2.txt
Prompt > ls

f.txt
f2.txt
Prompt >
```

cp2fs - Copies a file from the Linux file system to the test file system



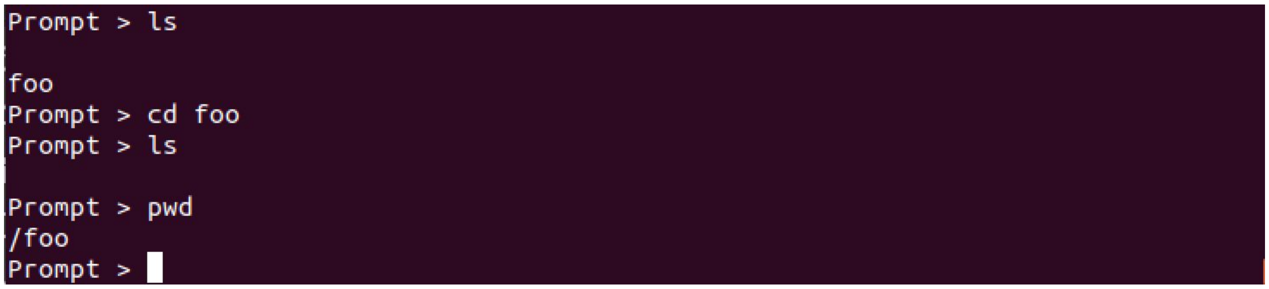
```
Prompt > ls

Prompt > cp2fs i.txt f.txt
Prompt > ls

f.txt
Prompt > cp f.txt f2.txt
Prompt > ls

f.txt
f2.txt
Prompt >
```

cd - Changes directory



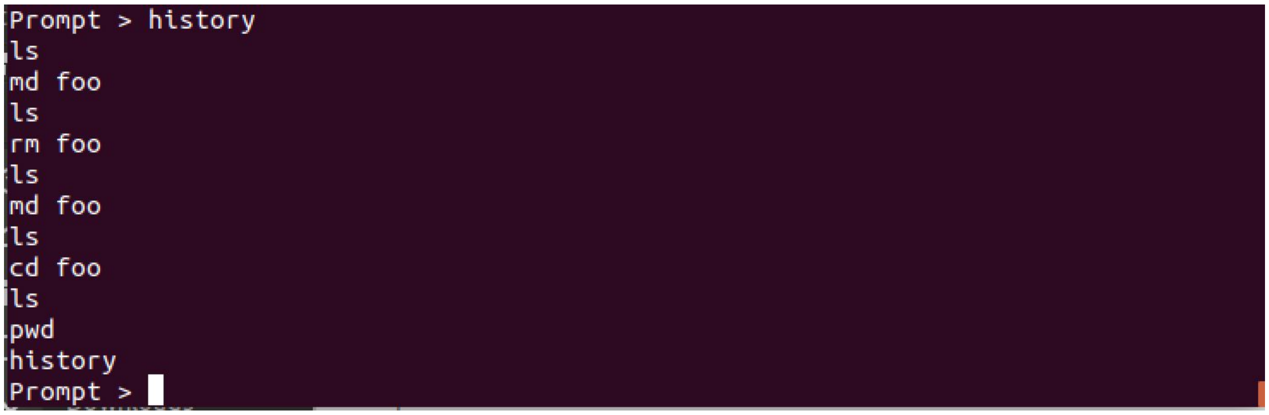
```
Prompt > ls

foo
Prompt > cd foo
Prompt > ls

Prompt >
```

pwd - Prints the working directory



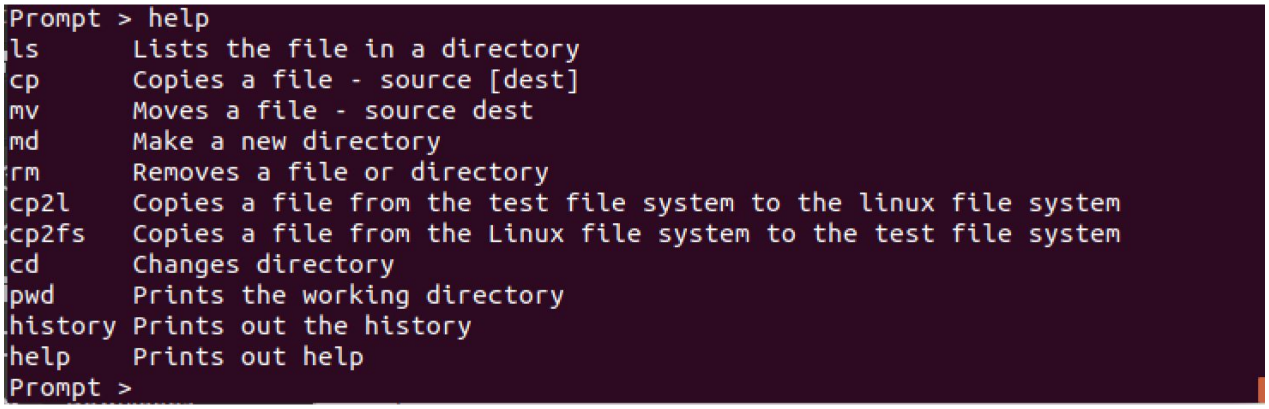
```
Prompt > ls

foo
Prompt > cd foo
Prompt > ls

Prompt > pwd
/foo
Prompt >
```

history - Prints out the history



```
Prompt > history
ls
md foo
ls
rm foo
ls
md foo
ls
cd foo
ls
pwd
history
Prompt >
```

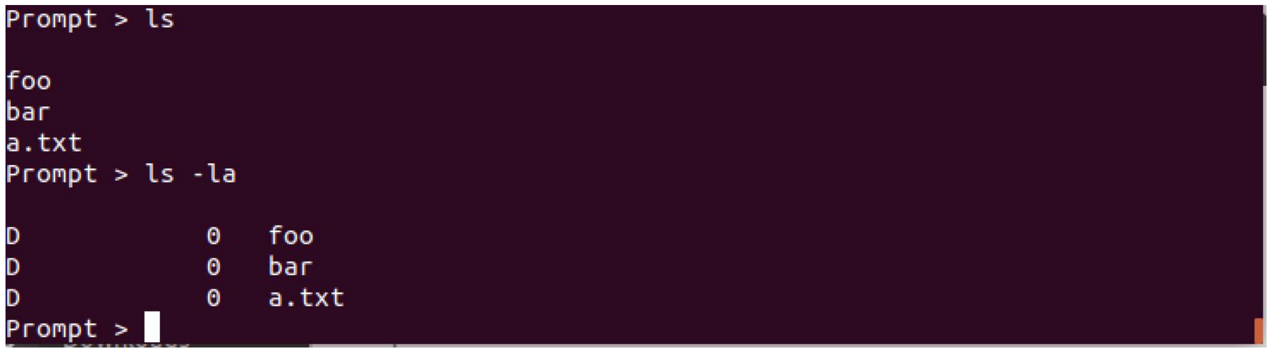
help - Prints out help



```
Prompt > help
ls      Lists the file in a directory
cp      Copies a file - source [dest]
mv      Moves a file - source dest
md      Make a new directory
rm      Removes a file or directory
cp2l    Copies a file from the test file system to the linux file system
cp2fs   Copies a file from the Linux file system to the test file system
cd      Changes directory
pwd     Prints the working directory
history Prints out the history
help    Prints out help
Prompt >
```

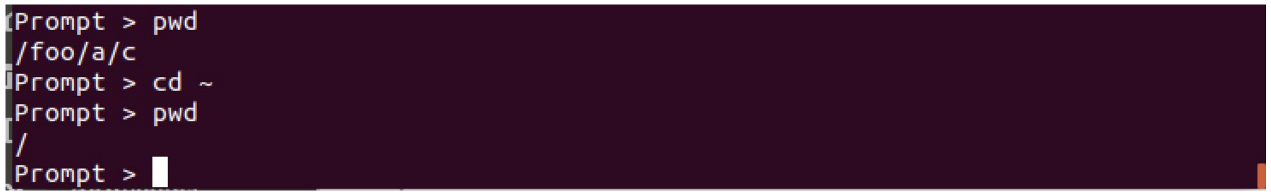
Exit - Exits out of shell

ls -la - Prints values from the fs_stat function.



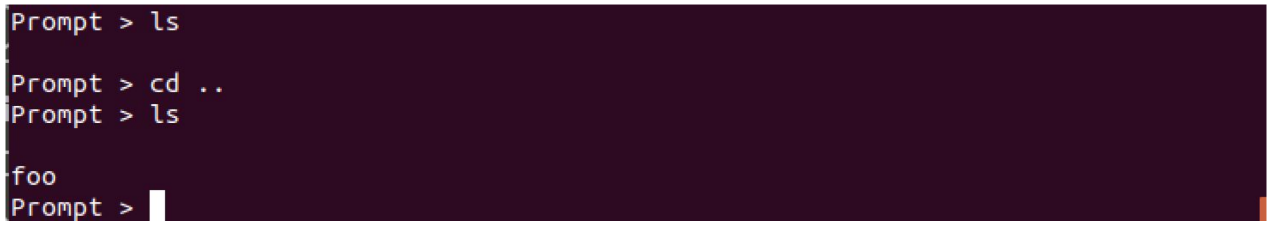
```
Prompt > ls

foo
bar
a.txt
Prompt > ls -la

D            0   foo
D            0   bar
D            0   a.txt
Prompt >
```

cd ~ - Returns to root directory



```
Prompt > pwd
/foo/a/c
Prompt > cd ~
Prompt > pwd
/
Prompt >
```

cd .. - Returns to parent directory.



```
Prompt > ls

Prompt > cd ..
Prompt > ls

foo
Prompt >
```