FEATURES

The commonly used commands in Linux that are implemented in the custom shell are as follows:

List: List and generate statistics for files so that the user can view the statistics of the file to check for the available functions that can be performed on the file.

Change mode: Change file mode to provide the list of functions on the file. For the future preferences of the file.

Show environment: Show the environment variable of PATH in which the user binaries are stored. Then it will append the path variable value to the command so that it can be executed.

Search binary location: It will search for the command that the user typed in the /usr/bin folder of the environment variable to execute the given command.

Change directory: Change the current directory of the Linux to the provided directory by the user so that future functions would be conducted in the changed current directory.

IMPLEMENTATION

In this project, a custom Linux shell was implemented that supports the features of the bash shell of the Linux Operating System. The shell includes support for keeping track of the environment variable so that the user can search for the specified commands in the user binary folder and then execute the user-typed command by searching in the binary files directory for the user. To implement the custom shell, the first thing that was needed was the environment variables so I used the char** env array provided in the main function of the program.

The environment variables array contains all the environment variables of the user who is executing the shell. After getting the environment variables for the user who is executing the shell, the shell searches for the PATH variable to get the PATH environment values from the environment strings. After successfully getting the PATH variable, it searches for the /usr/bin directory and then store it in a variable to append the command with it later.

After getting the environment variable the program takes input from the user. The program will continue to ask the user for input of command till the user enters “exit”. Upon entering the “exit” command, the program will end the execution and return control to the user.

After getting the command as input from the user, the command is tokenized so that the command and the arguments can be separated and stored in the command array to be passed onto the exec function to execute the given command. After tokenizing the command, the PATH environment variable value is appended to the command to get the complete path of the command that the user wants to execute.

After appending the environment variable to the command, the program creates a child process to call the exec function to execute the command. After creating the child process the command array is passed to the exec function along with the complete arguments that are provided with the command to search for the command in the user binary files location to get a binary executable of the command. Then after searching for the command in the user binary file location, the exec function runs the command along with the arguments provided.

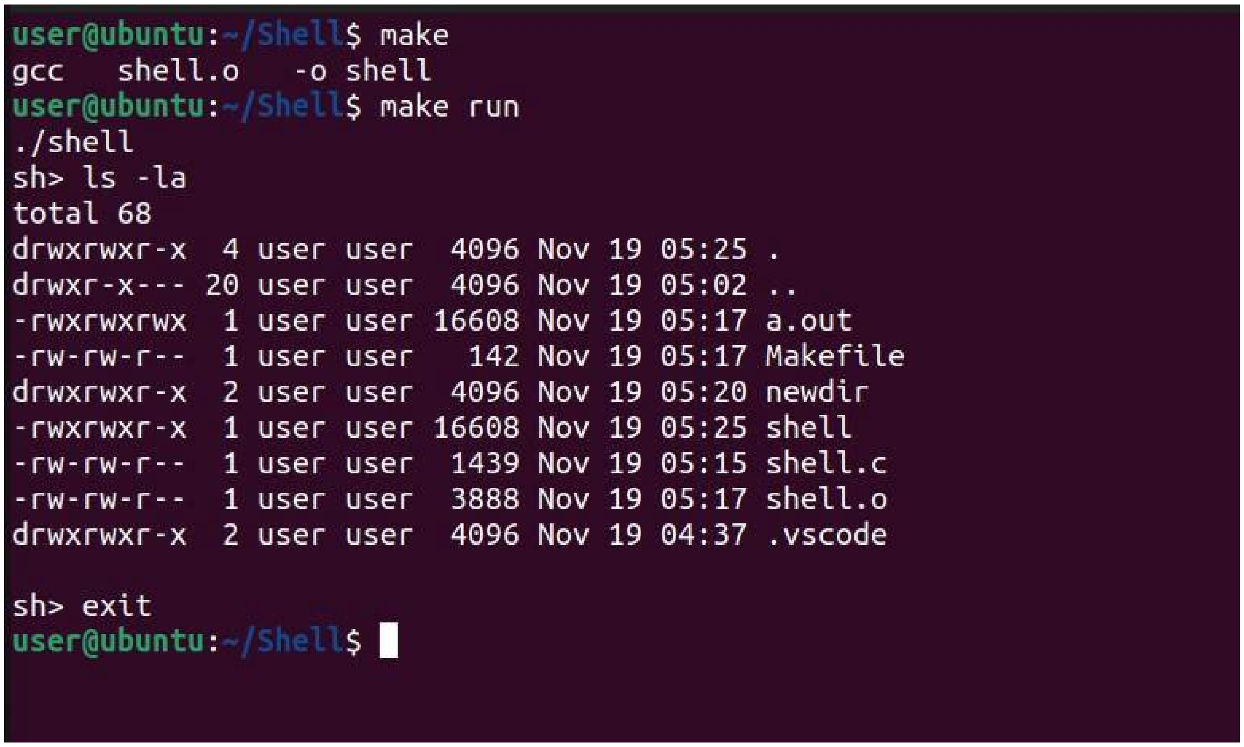
While the execution of the exec function, the command is invalid, then it shows the invalid command error on the screen so that the user may know that the command is not correct and the binary of the command cannot be found in the user binary file location. If the command execution is successful, the exec function shows the output of the command on the standard output and returns it to the parent process. And in the same way, the complete cycle of the shell execution continues till the user enters the exit command.

RESULTS AND ANALYSIS

The features proposed to be implemented in the project are implemented successfully and the desired output was achieved. A makefile for the program has also been created to easily compile and execute the shell. The makefile compiles the program using the GNU compiler collection command and then stores the compiled program in an object file. After that, it links the object file with the required libraries and generated a binary executable file using the -o flag. For the required features the output is shown below:

LIST STATISTICS

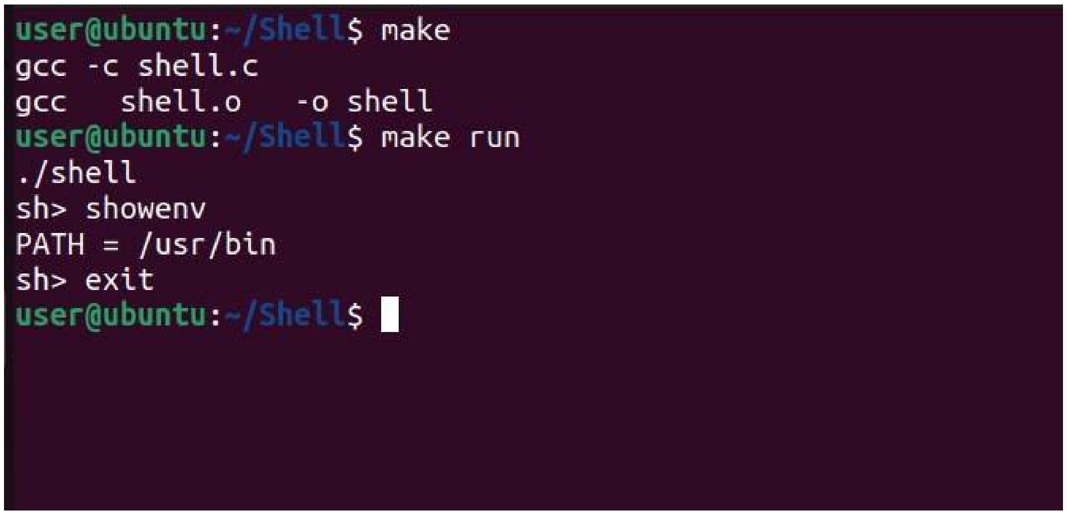
List and generate statistics for files so that the user can view the statistics of the file to check for the available functions that can be performed on the file.



```
user@ubuntu:~/Shell$ make
gcc   shell.o   -o shell
user@ubuntu:~/Shell$ make run
./shell
sh> ls -la
total 68
drwxrwxr-x  4 user user  4096 Nov 19 05:25 .
drwxr-x--- 20 user user  4096 Nov 19 05:02 ..
-rwxrwxrwx  1 user user 16608 Nov 19 05:17 a.out
-rw-rw-r--  1 user user   142 Nov 19 05:17 Makefile
drwxrwxr-x  2 user user  4096 Nov 19 05:20 newdir
-rwxrwxr-x  1 user user 16608 Nov 19 05:25 shell
-rw-rw-r--  1 user user  1439 Nov 19 05:15 shell.c
-rw-rw-r--  1 user user  3888 Nov 19 05:17 shell.o
drwxrwxr-x  2 user user  4096 Nov 19 04:37 .vscode

sh> exit
user@ubuntu:~/Shell$ 
```

SHOW ENVIRONMENT

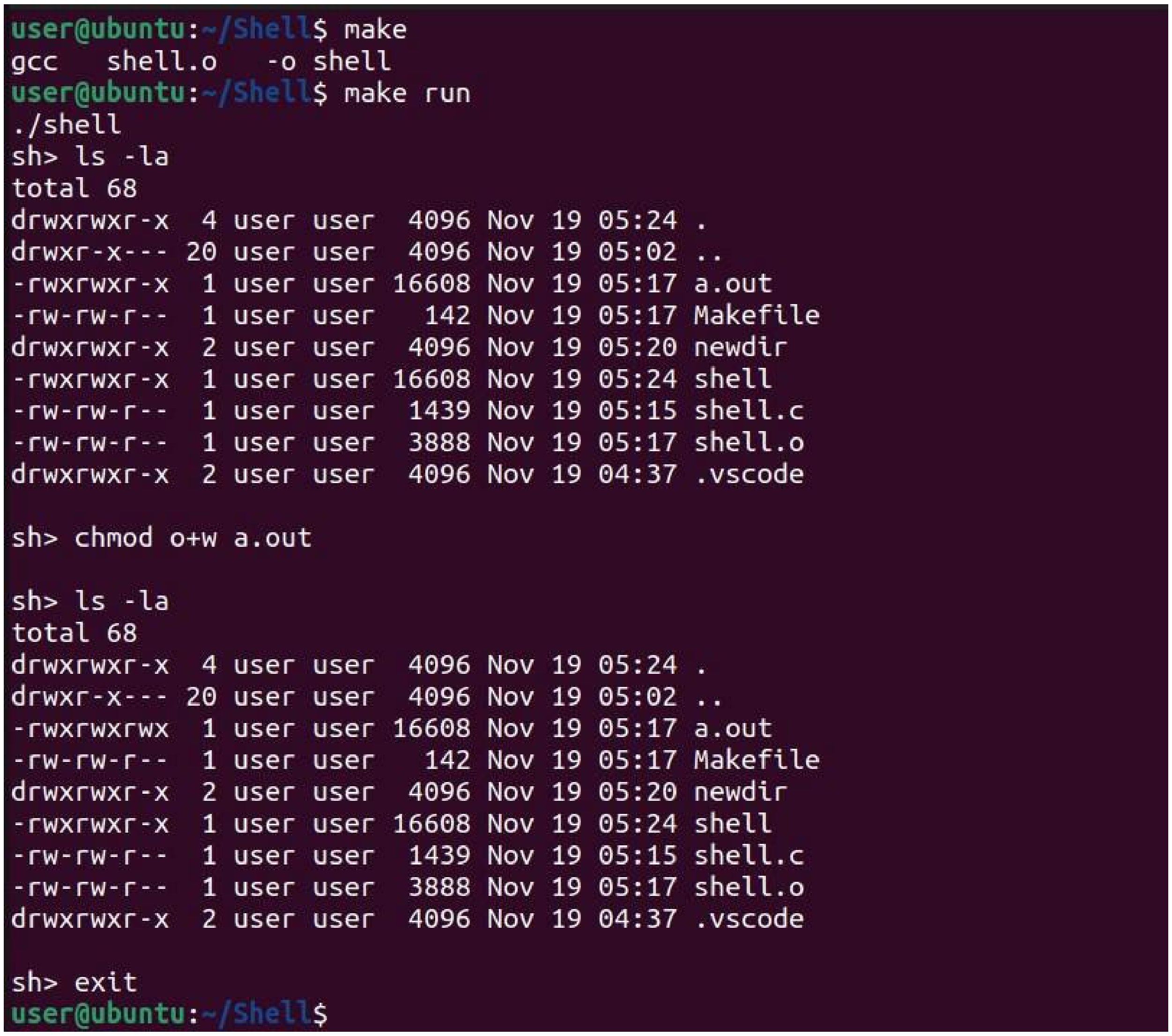
Show the environment variable of PATH in which the user binaries are stored. Then it will append the path variable value to the command so that it can be executed.



```
user@ubuntu:~/Shell$ make
gcc -c shell.c
gcc   shell.o   -o shell
user@ubuntu:~/Shell$ make run
./shell
sh> showenv
PATH = /usr/bin
sh> exit
user@ubuntu:~/Shell$ 
```

CHANGE MODE

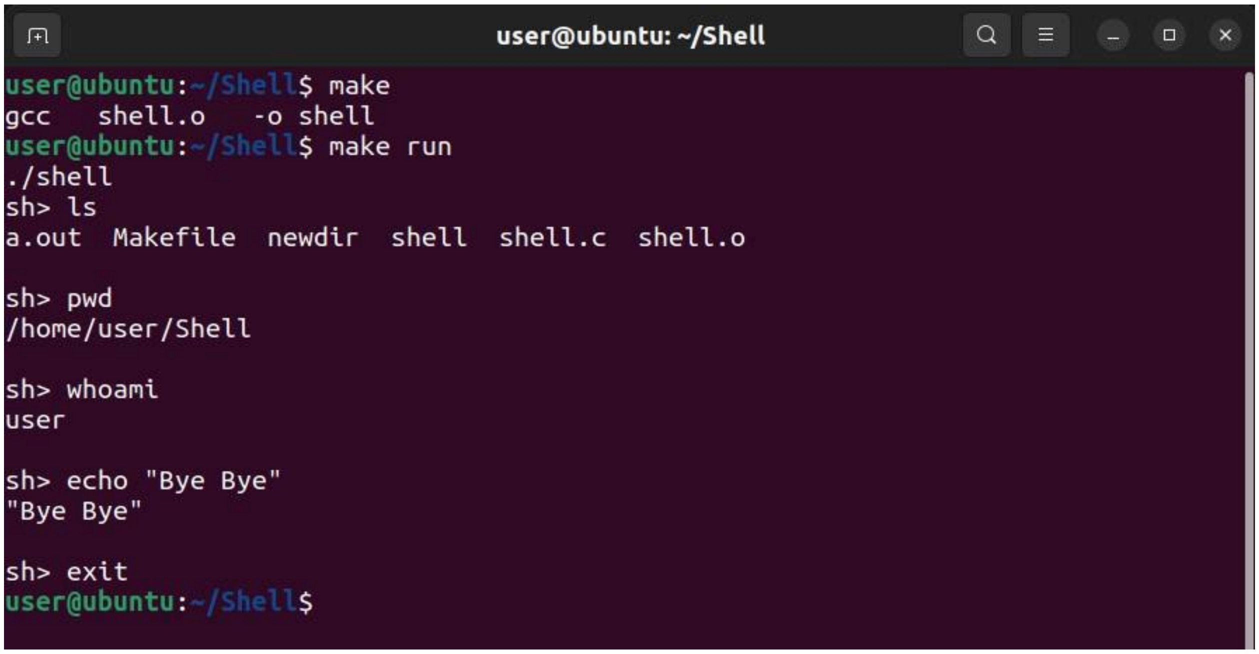
Change file mode to provide the list of functions on the file. For the future preferences of the file.



```
user@ubuntu:~/Shell$ make
gcc   shell.o   -o shell
user@ubuntu:~/Shell$ make run
./shell
sh> ls -la
total 68
drwxrwxr-x  4 user user  4096 Nov 19 05:24 .
drwxr-x--- 20 user user  4096 Nov 19 05:02 ..
-rwxrwxr-x  1 user user 16608 Nov 19 05:17 a.out
-rw-rw-r--  1 user user   142 Nov 19 05:17 Makefile
drwxrwxr-x  2 user user  4096 Nov 19 05:20 newdir
-rwxrwxr-x  1 user user 16608 Nov 19 05:24 shell
-rw-rw-r--  1 user user  1439 Nov 19 05:15 shell.c
-rw-rw-r--  1 user user  3888 Nov 19 05:17 shell.o
drwxrwxr-x  2 user user  4096 Nov 19 04:37 .vscode

sh> chmod o+w a.out

sh> ls -la
total 68
drwxrwxr-x  4 user user  4096 Nov 19 05:24 .
drwxr-x--- 20 user user  4096 Nov 19 05:02 ..
-rwxrwxrwx  1 user user 16608 Nov 19 05:17 a.out
-rw-rw-r--  1 user user   142 Nov 19 05:17 Makefile
drwxrwxr-x  2 user user  4096 Nov 19 05:20 newdir
-rwxrwxr-x  1 user user 16608 Nov 19 05:24 shell
-rw-rw-r--  1 user user  1439 Nov 19 05:15 shell.c
-rw-rw-r--  1 user user  3888 Nov 19 05:17 shell.o
drwxrwxr-x  2 user user  4096 Nov 19 04:37 .vscode

sh> exit
user@ubuntu:~/Shell$
```

SEARCH BINARY LOCATION

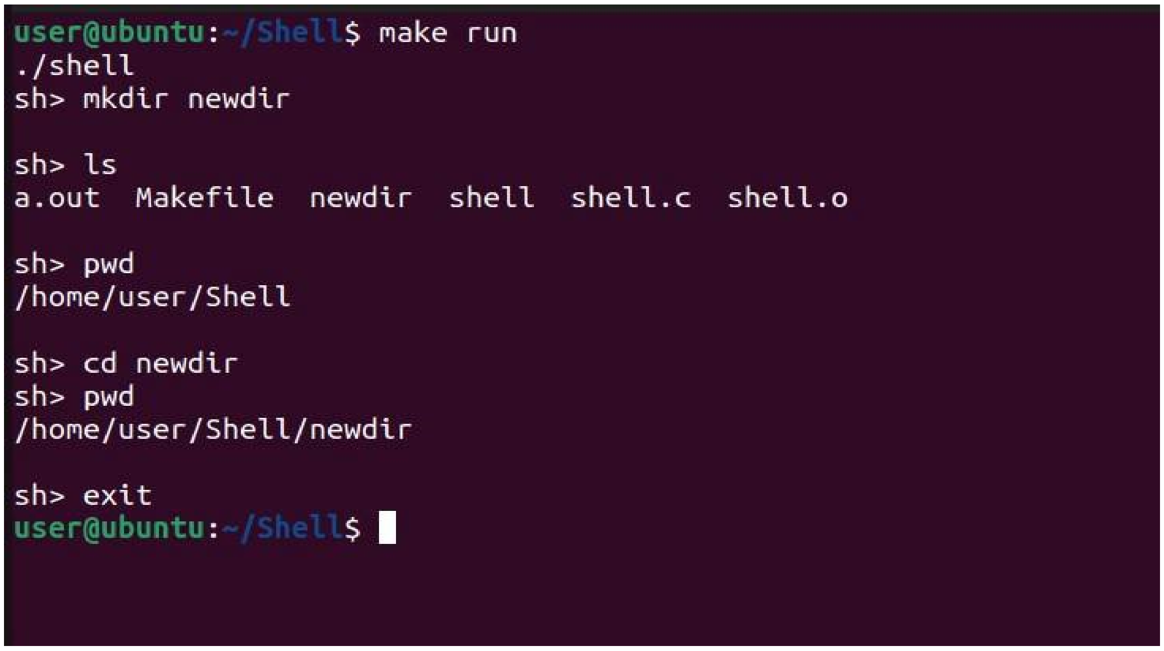
It will search for the command that the user types in the /usr/bin folder of the environment variable to execute the given command.



user@ubuntu: ~/Shell

```
user@ubuntu:~/Shell$ make
gcc   shell.o   -o shell
user@ubuntu:~/Shell$ make run
./shell
sh> ls
a.out  Makefile  newdir  shell  shell.c  shell.o

sh> pwd
/home/user/Shell

sh> whoami
user

sh> echo "Bye Bye"
"Bye Bye"

sh> exit
user@ubuntu:~/Shell$
```

CHANGE DIRECTORY



```
user@ubuntu:~/Shell$ make run
./shell
sh> mkdir newdir

sh> ls
a.out  Makefile  newdir  shell  shell.c  shell.o

sh> pwd
/home/user/Shell

sh> cd newdir
sh> pwd
/home/user/Shell/newdir

sh> exit
user@ubuntu:~/Shell$
```

Change the current directory to the provided directory by the user so that future functions would be conducted in the changed current directory.