Creating Custom shell using Linux

Hari Prasad Reddy Sheelam
Computer Science
University of North Texas
Texas, US
hariprasadreddysheelam@my.unt.edu

Satishbabu Nalajala
Computer Science
University of North Texas
Texas, US
Satishbabunalajala@my.Unt.edu

Ishitha konda
Computer Engineering
University of North Texas
Texas, US
Ishithakonda@my.unt.edu

Dunnala Naveen
Computer Science
University of North Texas
Texas, US
Naveendunnala@my.unt.edu

***Abstract*—From just a kernel only with a few lines of code to an fully implemented Operating System with more than two million lines of complex code, Linux has grown substantially. Linux has subtly and comparably migrated into a major software, in contrast to previous perceptions that it belonged to enthusiasts, students, and devoted programmers.**

In addition to the paper's discussion of how to create the built-in terminal window which can be seen in Linux distributions using C language, the project includes creating a custom shell. As once shell is created, extra functionality will be introduced and tested.

***Index Terms*—Custom Shell, Linux customized shell, Own shell, New shell, Personalized Linux shell.**

I. Introduction

A. Overview of our project

The shell is a type of software program that interacts with user through a terminal or reads input from a file prior to actually issuing a series of commands to the operating system. In this paper, you will learn how to write your own Linux shell application in C Programming. We will go over some of the internal features and algorithms that a shell employs to function. Once the Linux shell is created, the following steps are taken:
1.If the length of a command is not null when typed, it will be saved to the history.
2. Parsing is the process of separating commands into their individual words and strings.
3.After that, we will look for unique characters, such as pipes.
4.Then, we will determine whether built-in commands are necessary.
5.System commands and libraries will then be executed.
6.The next step is to print the name of the current working directory while simultaneously looking for additional instructions.

B. Problem being addressed

The shell allows the individual to launch programs and direct input and output to return from the files. Shells also include programming constructs such as if, for, while, functions, variables, and so on. Shell programs can help with distinct problems, such as editing lines, retaining command execution history and parsing.

C. Importance

When you create your own shell, users would then gain an in-depth knowledge of the way the operating system and shell interpreter software's interact with each other.

II. Background

ANSI escape codes are currently being used by more applications to interact with the Unix terminal. These are one-of-a-kind codes that your software can use to instruct the terminal. Different terminals accept different parts of the program, and there is no official fixed description of what each program follows.

Several related work has been discussed in the field of creating custom shells to their own liking to manifest the results they want which seems to be very useful in case of command line customization. Nonetheless, it is possible to create programs which use ANSI escape codes and run on standard Unix systems such as Ubuntu or OS-X. This paper will cover the fundamentals of ANSI escape codes and demonstrate how to use them to create a custom shell.

III. Implementation

A. Solutions

We can use various libraries to implement the custom linux terminal. Golang has recently unveiled a new built-in library for building the terminal. In a similar manner, we have developed a brand-new C program to build the terminal.

B. implementation

Through the console, we collect user input for the program. The string is then processed by being cut to length. In a similar manner, we search for unique characters like pipe characters. If it is present, the string is split into two commands, the internal command being executed first and the output being passed to the external command. When there's no pipe symbol, we look for built-in predefined command instructions.

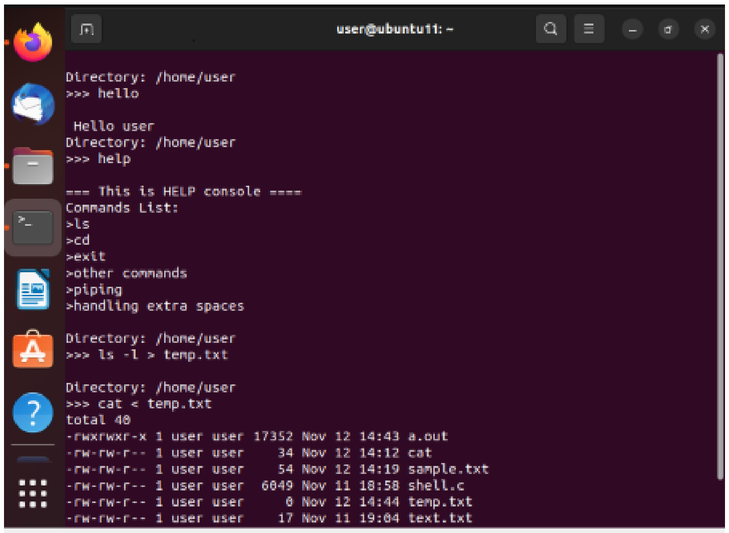
C. Test Cases

We test the program by passing commands like ls, cd, ls -a, an piping, and history commands.

IV. Experimental Results

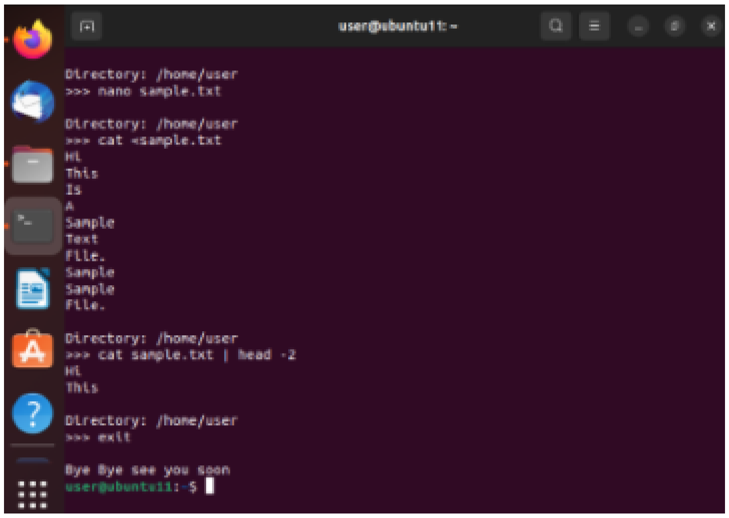
A. Results or Outputs of implementation

We have tested the implemented program with commands like ls, exit, cd, an piping, and help commands.



```
Directory: /home/user
>>> hello

 Hello user
Directory: /home/user
>>> help

=== This is HELP console ====
Commands List:
>ls
>cd
>exit
>other commands
>piping
>handling extra spaces

Directory: /home/user
>>> ls -l > temp.txt

Directory: /home/user
>>> cat < temp.txt
total 40
-rwxrwxr-x 1 user user 17352 Nov 12 14:43 a.out
-rw-rw-r-- 1 user user    34 Nov 12 14:12 cat
-rw-rw-r-- 1 user user    54 Nov 12 14:19 sample.txt
-rw-rw-r-- 1 user user  6049 Nov 11 18:58 shell.c
-rw-rw-r-- 1 user user     0 Nov 12 14:44 temp.txt
-rw-rw-r-- 1 user user    17 Nov 11 19:04 text.txt
```

Fig. 1. Workflow



```
Directory: /home/user
>>> nano sample.txt

Directory: /home/user
>>> cat <sample.txt
Hi
This
Is
A
Sample
Text
File.
Sample
Sample
File.

Directory: /home/user
>>> cat sample.txt | head -2
Hi
This

Directory: /home/user
>>> exit

Bye Bye see you soon
user@ubuntu11:~$ 
```

Fig. 2. Workflow

B. *Interpretation of the results*

The directory can be changed by the cd command. When we run ls, it displays the files in corresponding location; when we run history, it provides a list of all the commands we have recently executed.

V. Conclusion

Using the Linux environment, we were able to implement and run a custom shell Successfully.

References

[1] Schröder, M., Cito, J. An empirical investigation of command-line customization. Empir Software Eng 27, 30 (2022). https://doi.org/10.1007/s10664-021-10036-y

[2] Abdullah Kidwai , Chandrakala Arya b, Prabhishek Singh a, Manoj Diwakar c, Shilpi Singh ,Kanika Sharma e, Neeraj Kumar,"A comparative study on shells in Linux: A review"

[3] Michael Greenberg, Konstantinos Kallas, and Nikos Vasilakis. 2021. Unix shell programming: the next 50 years. In Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS '21). Association for Computing Machinery, New York, NY, USA, 104–111. https://doi.org/10.1145/3458336.3465294