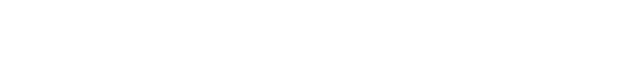


Linux Terminal Simulator



A Guide to Breaking the Program.



Table of Contents

[Introduction 3](#_Toc441087747)

[Summary 4](#_Toc441087748)

[Test Cases 5](#_Toc441087749)

[Test Type: Are there processes? Should they still be alive? 5](#_Toc441087750)

[Test Type: Do the characters accomplish their assigned goal? 6](#_Toc441087751)

[Test Type: Can we break it? We have the technology! 8](#_Toc441087752)

[Conclusion 9](#_Toc441087753)

# Introduction

So you want to break this Linux Terminal Simulator? Well then, there’s a few things you should know. Number one, this program was built and mostly tested on Ubuntu and Fedora systems. First thing you should know is that the “control-\” character is definite signal that I’ve researched and have unable to find a method to catch it. If you wanted to break my program, there you go! You’re free to close this file.

Oh, you’re still reading this… well then, I’ll have some more tests for you!

**All the tests to be done are inputted manually and will first test the basic functionality.**

1. ***Since we are dealing with processes, the program will create three processes that will be assigned to the LinuxTerminal application.******Success conditions****:* 
   1. *will have three processes on execution*
   2. *will have zero processes on regular program termination*
   3. *will have zero processes on regular program termination (no zombie processes)*
2. ***Each character has a specified task and will perform their assigned functionality.******Success conditions***
   1. *‘X’ character erases the formatted text but will appear in the RAW output.*
   2. *‘T’ performs the normal termination of the program. It will also allow each of the child processes to perform their assigned tasks before exiting.*
   3. *‘K’ (That’s an upper-case ‘k’) removes all buffered input.*
   4. *‘ctrl-k’ immediately kills the program. (Hint: the message “killed” is displayed)*
   5. *‘E’ functions similar to the enter key and sends the message buffer to the output to be displayed in raw and formatted forms.*
   6. *‘a’ characters are formatted as ‘z’ characters while other normal characters are untouched.*
   7. *All other normal characters are inserted into the buffer.*
3. ***This is the stress testing portion. There are various tests that have their own defined success conditions in the tests themselves:***
   1. *Inputting characters larger than the buffersize (for this test, the buffersize will be decreased artificially from 4098 to 24).*
   2. *Attempting other control signals into the program.*
   3. *Closing the program while the terminal is open. (Note, don’t do this. It may create zombies that can eat your brians [yes, Brian, as in the person… it’s a joke]).*
   4. *Randomly whacking the keyboard and occasionally hitting shift-E.*

# 

# Summary

&&&&&&&&&&&&&&

&&&&&&&&&&&&&&&7

&&&&&&&&&&&&

# Test Cases

## Test Type: Are there processes? Should they still be alive?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test**  **Number** | **Scenario** | **Expectations** | **Results** | **Conclusion** |
| 1 | Executing the program starts the application. | Three processes will be created that are associated to the Linux Terminal. |  |  |
| 2 | Program started and the ‘T’ character is inputted. No previous input. | There will be no processes associated to the Linux Terminal in the “current processes” list.  -No output! |  |  |
| 3 | Program started and the ‘T’ character is inputted. Previous input. | There will be no processes associated to the Linux Terminal in the “current processes” list.  -Will output whatever was in the input and the new formatted string. |  |  |
| 4 | Program started and ‘ctrl-k’ is inputted. No previous input. | There will be no processes associated to the Linux Terminal in the “current processes” list.  -Display the “Killed” message (spoiler alert) |  |  |
| 5 | Program started and the ‘ctrl-k’ character is inputted. Previous input. | There will be no processes associated to the Linux Terminal in the “current processes” list.  -Display the “Killed” message (spoiler alert) |  |  |
| 6 | Input nothing into the text field and clicking the push button. (Port to Service selected) | Will display usage instructions. |  |  |

## Test Type: Do the characters accomplish their assigned goal?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test**  **Number** | **Scenario** | **Expectations** | **Results** | **Conclusion** |
| **1** | Input: “workstestXXXXE” | Raw: “workstestXXXXE”  Format: “works” |  |  |
| **2** | Input is: “I Xam Xa Xbanana.E” | Raw: “I Xam Xa Xbanana.E”  Format: “Iamabanana.” | Hilarious. |  |
| **3** | Input is: “XXXXXXXXE” | Raw: XXXXXXXXE  \*Note the lack of a formatted message. |  |  |
| **4** | Input is only: “T” | Raw: T  Format:  \*Note the lack of a formatted message. |  |  |
| **5** | Input is: “this is a Test” | “Raw: this is a T“  “Format: this is a “  \*Note the lack of a formatted message and space AFTER the Raw output. |  |  |
| **6** | Input is: hElElElTJKJKE  \*Note: this also tests ‘T’ and ‘E’ | Raw: hE  Format: h  Raw: lE  Format: l  Raw: lE  Format: l  Raw: lT  Format: l |  |  |
| **7** | Input: [ctrl-k]  \*Note, the brackets (which can be used, indicates the ctrl-k signal) | “Killed” |  |  |
| **8** | Input is: “this is futile[ctrl-k]”  \*Note, the brackets (which can be used, indicates the ctrl-k signal) | “Killed” | . |  |
| **9** | Input is: EEEE | Raw: E  Format:  Raw: E  Format:  Raw: E  Format:  Raw: E  Format: |  |  |
| **10** | Input is: “Since Amy and amanda like apples, I decided to Eat my apple.E” | Raw: Since Amy and amanda like apples, I decided to E  Format: Since Amy znd zmzndz like zpples, I decided to  Raw: at my apple.E  Format: zt my zpple. |  |  |
| **11** | Input is: “these are normal characters that shouldn’t be output onto the screen.” | \*Note, there is NO output because there is no ‘E’ character. |  |  |
| **12** | Input is: “Enter at your own risk, I’m a sign, not a copE” | Raw: E  Format:  Raw: nter at your own risk, I’m a sign, not a copE  Format: nter at your own risk, I’m a sign, not a cop |  |  |

## Test Type: Can we break it? We have the technology!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test**  **Number** | **Scenario** | **Expectations** | **Results** | **Conclusion** |
| **1** | Buffersize changed to MAXIMUM 24 characters and we input more characters than that.  Input:  1234567890  1234567890  1234567890E | Raw: 567890E  Format: 567890  \*Note that the initial characters are completely ignored. |  |  |
| **2** | I will attempt EVERY button with ctrl and only output the ctrl buttons that create an operation! |  |  |  |
| **3** | Closing the terminal in progress. | Zombie processes that will consume your Brians. |  |  |
| **4** | Randomly whacking the keyboard until some sort of error breaks the program. | Unless ctrl-\ is hit, the program will not end. |  |  |

# Conclusion

Our first tests consisted of various smoke tests designed to test if our program even functions such as seeing if empty input affects conversion, having no internet connection and checking to see if the clear button functions as intended. Once those tests have been proven to be a success (or they were intentionally set for failure and those errors were handled), I carried on testing the functionality of each of the four functions.

IP to host proved to be the least intuitive, I had to use the Host To IP in order to acquire the IP’s necessary to test. Nonetheless, all strings that did not follow the format of #.#.#.# only were discarded with the error type being display to the output. All valid input had their aliases as well as the host names printed out in a proper manner. However, when testing with “milliways.bcit.ca”, the host name turned into “ip-142-232-66-1.ptr.bcit.ca”. This is very unusual and probes further investigate at a later time.

Host to IP resolution was very simple and easy to test. Google and localhost was used to test this scenario. All ill-properly formed input was discarded while the input in the format an internet address resolved well. There was a small concern of mine when my localhost became 4711.local however it later made sense because I configured the hosts file on my computer.

Service to Port made me realize that my design could have been improved to allow for extra arguments only if the first two arguments were valid. Nonetheless, all input that did not adhere to the format of “<Service> <Protocol>” were discarded with an error message printing on the screen. Properly formatted input had the output display the port number. Port to Service is a part of the reason where I could have improved my design. Nonetheless, all ill-formed (they did not follow the format “<Port #> <Protocol>”) was discard and the properly formatted input attempted to function. HTTP has a port of 80 with the protocol of 80.

In conclusion, the format of the arguments in the input matter the most when using this application. Domain names, such as google, seem to actually have a lot more than one server than I initially thought. Doing the reverse lookup however on one IP only brings back that server. Also, a small issue I noticed was with supplying additional arguments even though the “required” arguments are valid. Perhaps additional arguments shouldn’t punish the user and still perform the function (assuming that the first two arguments are valid)?