POSIX SHELL

PROJECT REPORT

(Group No 25)

PRAKHAR GUPTA

SHARANYA BHATTACHARYA

ADESH INGLE

NEERAJ ASDEV

MENTOR: SARTHAK VERMA

Table of Contents

OVERVIEW

Developed a working POSIX compatible shell with a subset of feature support of the default shell. The following features have been implemented (basic and extended both inclusive).

- ✓ Configuration file
- ✓ Parser
- ✓ History
- ✓ Command autocompletion
- ✓ Output redirection
- ✓ Alarm

FEATURES IN BRIEF

CONFIGURATION FILE

- Sets the environment for shell on startup.
- All the initialization environment variables like PATH, HOME, USER, HOSTNAME, PS1 are defined and implemented here in this file. Other variables like HISTSIZE are also included.
- This file is preprocessed during startup of this shell and all the variables can be persistently accessed whilst the shell is still active. This file is included as a header in the MAIN file.
- Support for PS1 as a combination of HOME and HOSTNAME variables to be displayed on the CLI is handled. HOME variable is linked and associated with "~" variable as mentioned.
- In the program, an "init()" function is defined where all the variables are initialized using configuration object in the raw mode.

GENERAL INPUT OUTPUT & PARSING

- Firstly, input is tokenized where tokens are stored as strings. NOTE that they are space separated values in the input.
- Then each token will be interpreted as either built-in(command) or user defined parameter. Each built-in will have a different behavior. For example, “gotoDir()” is an implementation of the built in “cd” command.
- Every input prompt will be recognized as a combination of pre-defined keys. For example, UP arrow is a combination of sequence of keys and so on.

HISTORY IMPLEMENTATION

- Every entered input will be logged in the history data structure implemented as deque.
- Users can use UP ARROW key and DOWN ARROW key to browse commands in backward (least recent) and forward (most recent) respectively.
- Deque size is bounded by HISTSIZE environment variable. Therefore, in overflow, front element of deque or least recent command will be popped.
- Consecutive repetitive elements will be stored only ONCE.
- On typing “history” command, the CLI will display all the n entries from least recent to most recent (including the history command typed) where n is the size of the history deque.
- Command can be executed from anywhere.

COMMAND AUTOCOMPLETION

- Trie is an efficient data structure to store strings and perform autocompletion.
- Once reaching the node till the last character of the typed string, autocompletion can be formed to fetch all the possible strings.
- All the commands defined in shell are physically stored inside “bin” folder in the “root” directory. These commands are stored in the trie.
- Now whenever user presses TAB key during string input, a function will autocomplete and print all the possible suggestions with max 5 suggestions per line.
- String input with only 1 command matched will replace the input in the same line as it is.

OUTPUT REDIRECTION

- The work of any command is either taking input or giving an output or both. So, Linux has some command or special character to redirect these input and output functionalities such as “>”, “>>”, etc.
- The motive behind this is that we don’t want the output to be displayed on the terminal. Instead, say, we want it to be copied in a text file.
- “>” denotes overwrite operation. For e.g. “cat > sample.txt” will overwrite the contents of the sample text file with the output of the cat command.
- Similarly, “>>” will just append the destination file and keep the original contents as it is.
- Note that the content will be taken as output when typed after the “cat >> destination” command.

ALARM

- This feature is implemented to set an alarm or reminder for the delivery of a signal.
- Here, the signal should be executed whenever the time set during the creation is up.

- To store the time::data pair, map data structure is used in order to map the data with time where time is key and data is value.
- The alarm can be triggered anytime during the execution of the program in the shell.

PIPELINE

- A pipe is a type of redirection (move of standard result to another objective) that is utilized in Linux and other Unix-like working frameworks to send the result of one order/program/cycle to another order/program/process for additional handling. The Unix/Linux frameworks permit stdout of an order to be associated with stdin of another order. You can cause it to do as such by utilizing the line character '|'.
- Pipe is utilized to join at least two orders, and in this, the result of one order goes about as contribution to another order, and this order's result might go about as contribution to the following order, etc. It can likewise be imagined as a transitory association between at least two orders/programs/processes. The order line programs that do the further handling are alluded to as channels.
- This immediate association between orders/programs/processes permits them to work all the while and licenses information to be moved between them consistently instead of going it through impermanent text records or through the presentation screen.
- Pipes are unidirectional i.e information streams from left to directly through the pipeline.

Prakhar Gupta	2022202021	Configuration setup, Handling environment variables alarm function, alias execution, basic inbuilt commands
Neeraj Asdev	2022201056	Autocomplete, History completion, execution of certain commands
Sharanya Bhattacharya	2022201040	Parsing input, integration between modules, basic inbuilt commands
Aadesh Ingle	2022202017	IO redirection, pipelining record statement