

Assignment 7

- Siddharth Sharma - ssharm24
- Bhushan Thakur - bvthakur
- Karan Jadhav - kjadhav

Q1.

- **Who are the intended users -**
 - People who are new to the CLI, including students, who want to learn and use command line features.
 - Intermediate or expert CLI users, who want to make their usage easier, faster and error-free.
- **What must the intended users do -**
 - Search for any function they want to perform (in the search bar), giving a list of commands which will be executed on selection.
 - Search commands by different tags (categories). Add their own custom command and tag them for easy retrieval.
 - Create pipeline of commands by chaining commands together. When a command needs extra parameters, user will be prompted to enter them with a popup.
- **How to deliver to the intended users (i.e., platform(s)) -**
 - The solution will be a Python or Javascript (NodeJS) based application which runs as local server. Users can access the local web server in any web browser; internet connection is not required. The application is designed to work for Unix CLI and will only work on Unix derivatives - Linux distributions and MacOSX.
- **How to design for the intended usage**
 - Since we are targeting beginners, the UI should be easy to navigate with recognizable actions, and quick feedback. We cannot totally avoid complexity because of the highly configurable nature of CLI, hence tradeoffs are made balanced between ease of use and powerful features.
 - Our app uses Search as the main mode of interaction, with natural language as input. This makes sense because search makes it easier to navigate hundreds if not thousands of commands. Autocomplete will help the user narrow down a result and search flags (to search by categories, and specific command options), provide powerful search capabilities.
 - The use case of chaining commands together(using pipes), is accomplished by providing a builder UI for commands. Buttons such as "+" to add command, and save feature to save the pipeline for later use and modification.

- **How to test your design (methods, participants, anticipated analysis)**
 - Talk aloud and Interview will be our 2 methods to test. First we introduce the participants to the UI elements and the use cases. Then we ask them to perform a set of tasks in the application, and observe them while they describe their actions. We have 3 participants, who are familiar with CLI, but not experts.
 - Analysis will be to see which UI elements aid/hinder the user; recognize any ambiguity in possible interactions. Also to compare against the same tasks in a regular command line as a baseline comparison.
 - Usability test: How accurately the application is able to generate the commands as per user requirement. Or if not a one-time search, then a clearly understood granular search at the command, flag and chaining level
- **How you might iterate based on your test results**
 - Suggested features that add more CLI functionality for example, redirection suggested by one of the participants, will be added
 - Suggested improvements that possibly enhance existing functionality or make it more usable for example, grouping commands into scripts as suggested by another participant, will be incorporated
 - Delays or roadblocks in certain features that take users a long time or are not intuitive enough to use will be analysed and reworked accordingly, by either enhancing existing functionality or adding new ones
- **How you might justify your application**
 - It has been decades since Operating systems have transitioned to GUI interfaces, but CLI usage still exists. This is because it has very powerful features, and great customizability. But the way to use them remains largely unchanged since their inception, and they remain user-hostile in many ways. Our application tries to bring modern features that users have come to expect, like search and autocomplete, history, bookmark and tags, to the CLI.

Q2.

The UI in fig 1 represents the Main UI of the application. The UI has a following set of features.

1. **Search Bar:** User can use this search bar to search for different commands. The search can accept not only description but also a command itself which gives the user further results for customizing with flags. Example: "List files" will give result "ls". If you search "ls" it will display results with different flags - "ls -l", "ls -la" etc. Search will include results from builtin and user saved commands, and from history as well.

2. **Settings view:** This icon navigates the user to the command listing view(fig 2) wherein users can view and modify all the stored commands along with their description and tags.
3. **GUI view:** The GUI provides a visual environment to chain commands, and configure different command parameters. Adding a command in an existing chain is done by clicking add icon beside the last command (similar to adding a browser tab in a browser). Each link in the chain is editable, to configure command parameters. When user is done modifying, they can click 'Run' button to execute the chain. The output would be displayed in the output window. Ex- "ls -l | grep .txt | wc -l" is an example of a chain command which outputs the number of ".txt" files in current directory. It is made of 3 separate commands ls, grep and wc, which in GUI builder will be separately editable.
4. **Terminal view:** This tab view beside the GUI view, displays the regular linux command line interface. The GUI can be seen as a UI wrapper around this terminal interface. This gives the ability to view/modify the command in raw form that's being built in GUI, and to directly interact with the terminal.
5. **History/Starred panel:** This side-panel displays the history of commands executed in most recent first order. User can add these commands into the database which will allow for access via search bar. A Bookmarking feature is also provided for frequently used commands which are displayed in the Starred tab in the panel.
6. **Tags panel:** Tags give quick access to commands relating to a category like Files, Process and Program. Users can create their own tags and add commands to it. Clicking on a tag will result in search query which will display list of commands with that specific tag.

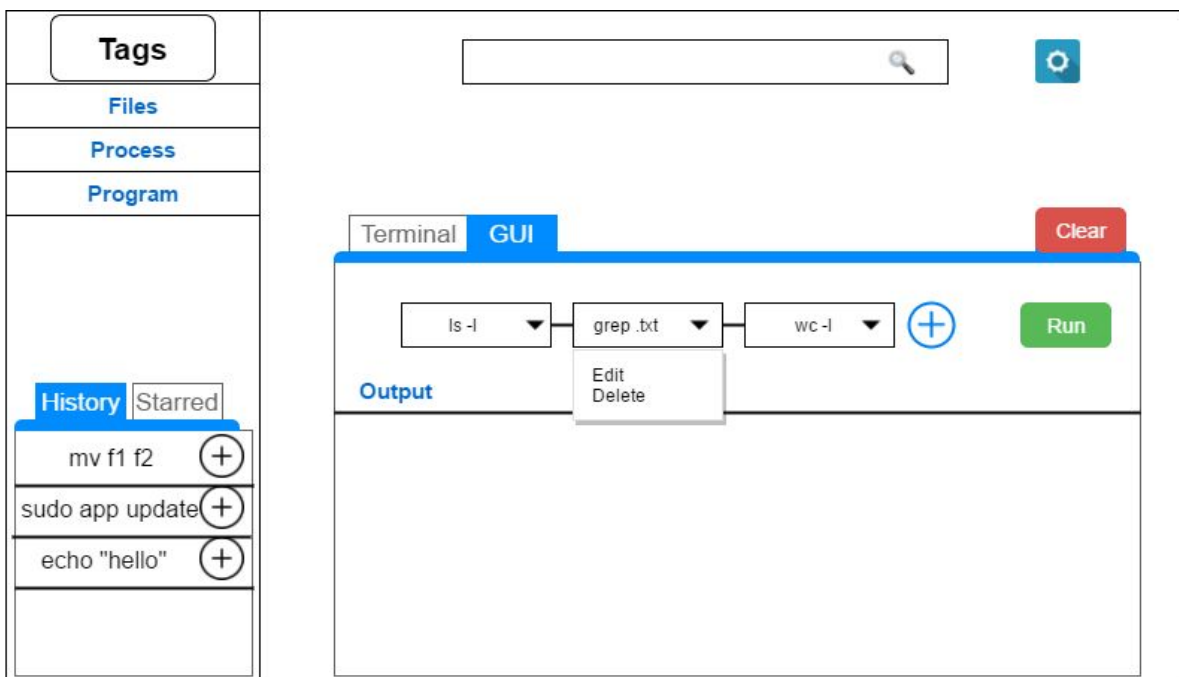



Fig 1: Main UI of application

The UI in fig 2 is of Command listing view. Users can add commands to the command list/database by clicking on add button which will result in a window with all the required fields (fig 3). On entering the information the command will be displayed in the database. Also existing commands can be edited using the rightmost “edit” button.






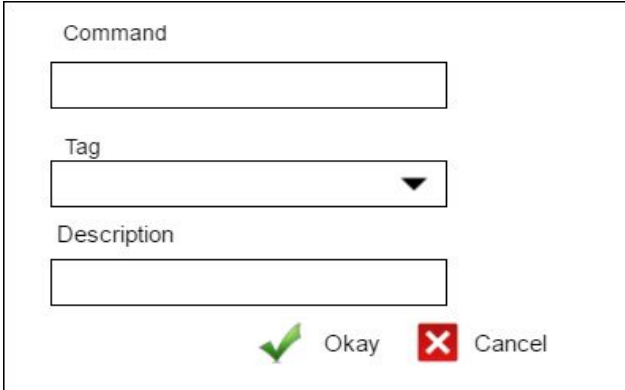
| # | Command | Description | Tags | |
|---|-----------------|-----------------|----------|---|
| 1 | ls -l | List all files | files |  |
| 2 | shutdown | shutdown | utility |  |
| 3 | sudo apt update | software update | software |  |

Fig 2: list view



Command

Tag

Description



 Okay  Cancel

Fig 3: Popup for adding new command to Database

Some commands require user input such as path (in case of rename or delete command) or search string(in case of grep or find command). The Input is taken via a popup which is prompted when user selects the command.The UI for this displayed in fig 4.

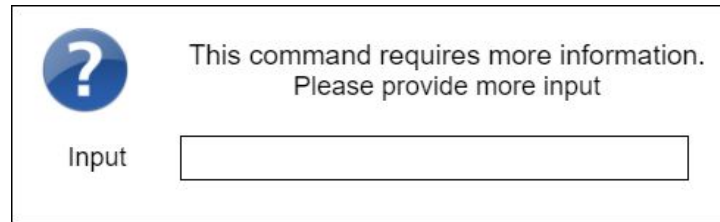


Fig 4: Popup for commands which require user input

Q3.

- **Process of evaluation:** Think-aloud while doing tasks and personal interview with questions about usability of UI, comments regarding overall design and the way it looks etc, list of features that users would like etc.
- **Evaluation Tasks:** The 3 tasks described in our testing methods were used:
 - Search and execute: Users were asked about the approach they would use to find all running processes on our UI.
 - Bookmark and tag it:
 - Chain commands

Evaluation:

- **User 1** - This particular user is an experienced CLI user. He recognizes the fact that piping commands together is a complex task with no immediate feedback and is receptive to proposed improvements.
 - **UI feedback** - Settings button is ambiguous, doesn't clearly represent the command listing view.
 - **Suggested Improvements**

- **UI for redirecting input and output**, i.e. taking input from a file, and saving terminal output to a file. This can be accomplished by adding a "Save to file" button in the bottom of the output window.
- **Templates for Commonly used commands and workflows**. Example - search for a process in a all running processes "ps aux | grep process". This is a common command and should be included.
- **User 2** - This user is an intermediate user, and provides multiple suggestions for improving the immediate feedback of the application in conditions like error or permission requirement.
 - **Suggested Improvements**
 - **Don't have root permission** - When a command cannot execute because the user doesn't have proper permission, then ask automatically for user password
 - **Display useful error messages**. Provide a useful error message and a possible solution over it. For ex: If a user mistypes a command we could suggest user closest matching command to user.
 - **Aggregate useful commands from internet** and present them in a UI - There are websites like bashoneliners.com and commandlinefu.com which maintain a list of useful commands, which can be exported into the database automatically.
- **User 3** - This user is intermediate user, and suggests more flexible saving features, like scripts, and features which are present in CLI such as tab completion for input.
 - **Suggested Improvements**
 - **Saving Multiple lines of command** - Bash scripts are used to automate tasks or accomplish a complex task. This can be a feature similar to commands in database, making it searchable, and immediately executable from the application.
 - **AutoCompletion for paths, filenames** - when giving input the user shouldn't be expected to type correct paths or filenames, and they should be detected like autocomplete.
 - **Shortcuts for run, bookmark** etc. - When using the application, after a while, users might want quick ways to run or bookmark a command, which can be accomplished by providing keyboard shortcuts.
- **Next design iteration:**
 - **Features from User feedback** - These would include automatically asking for password permission, listing most commonly used commands, and providing helpful error message and resolving them.
 - **meaningful interaction with output** - Our application mostly deals with solving the problem of easily providing input to the CLI, but meaningful interaction with the output of CLI is also an area that can be worked upon. Example - Output of

list files (ls) command is a list of files, these can have checkboxes, which can be used to delete or rename from the UI itself. Another example - Output of list processes command is list of running processes, and checkboxes here can be used to kill desired processes.