

CS554 Report: Improving Command Line Interfaces

By

Siddharth Sharma: ssharm24 ,

Karan Jadhav: kjadhav ,

Bhushan Thakur: bvthakur ,

North Carolina State University

Abstract:

Graphical User Interface provides simple, and intuitive ways for users to interact with computing devices. In spite of this, Command Line Interface usage still exists. This is on account of CLIs having very powerful features and great customizability. CLIs have remained unchanged for the most part since their inception, because of their usage being targeted at experienced, technical users. This isolates inexperienced and non-technical users from its usage and rich functionality. In addition to this, there is a steep learning curve associated with CLI usage. We propose a application to bridge this gap, by bringing modern features such as autocomplete, history, bookmarks and tags which are based on HCI principles like recognition rather than recall, useful feedback etc. to the CLI and thus make it more user friendly, reduce the associated learning curve and thus allow its functionality to be used by a larger and wider user group.

Background:

The application that most closely resembles ours, is one called CLI Companion[1]. As per it's official description, it allows the storing and running of commands from a GUI, is targeted at users inexperienced with CLI usage, aims to help them unlock the potential of the CLI and also states an added feature for experienced users in terms of the ability to store and search frequently used commands.

CLI Companion provides the following four features - running, adding, deleting and searching commands. At first glance these appear to be the same as the ones in our application, however there are major differences between the two. For starters, commands in CLI Companion exist in a simple, flat repository. In contrast to this, we propose segregating commands based on tags such as text processing, user administration etc. So apart from searching commands, users can also explore commands under one grouping as per their requirement. The second difference is in the commands that can be added. CLI Companion allows the addition of single commands, inclusive of associated flags of course, whereas our application allows piping of several commands, thus allowing the storage and use of complex, chained commands.

The third difference is the history feature. CLI Companion does not have this feature, whereas our application does. It lists commands based on frequency of use as well as how recently they were used. This helps avoid having to search for commonly used commands every time. The next difference is the added feature of output redirection that we provide. Users will have the option to add additional commands after existing ones or redirect their output as required. The last difference, again an added feature, is the capability to group several commands and save them as one script. This is different from chaining commands where the output of one command

is passed as an input to the successive one. In case of a script commands run independent of one another, not necessarily using the output from the previous command.

Lastly, though CLI Companion targets the same goals as our application, namely easier use of CLI via GUI, familiarizing inexperienced users with the CLI, allowing them to utilize terminal features and providing added features for experienced users, we are confident that the added features, in addition to the better designed and more intuitive GUI of our application, will achieve these goals more effectively.

Methods:

Our approach in building and testing our application begins with conceptualizing a feature. Next, we think of this feature in terms of the value it provides to the user, in terms of functionality and ease or intuitiveness of use, and finalize it. After this, we implement the feature or create a prototype and perform user testing. We perform testing as a two pronged approach. We observe the user while he/she uses the feature and after that have a think aloud session with the user. The first approach is targeted more at the inexperienced users, to see how well they're able to use the feature. The second approach, more at experienced users, so we can have inputs from subject matter experts. In this way, we gather feedback on our feature and incorporate it in refining the design or adding additional functionality and subsequently implement the same.

This approach helped us while designing the first prototype of our application. After we had decided on the features and created a design, we proceeded to user testing and feedback. Observing the inexperienced users drove us to refine our design, specifically with respect to the tags section and the editing of chained commands. Feedback from one experienced user made us additional features of output redirection and grouping commands into one script. Based on this feedback we came up with the first prototype.

Our prototype (see fig 1) consisted of features such as search, history, bookmark and tags. While designing we made sure that the UI looks clean and simple enough so that users can use our application without any hassles. The following are the set of features that we included in our initial prototype.

1. **Search Bar:** Users can use this to search for different commands. Search can accept not only descriptions but also commands. Commands from the search result can be further customized. Search will include results from both built-in as well as user saved commands. Results are not necessarily specific to any tag.
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. The user can also add new commands from this section.
3. **GUI view:** The GUI provides a visual environment to chain commands as well as configure different command parameters. Adding a command in an existing chain is done by clicking the add icon beside the last command (similar to adding a browser tab

in a browser). Each command in the chain is modifiable in terms of editing command parameters or removing the command itself. When user is done modifying, they can click 'Run' button to execute the command chain. The output will be displayed in the output window.

4. **Terminal view:** This is the tab beside the GUI view. It displays the regular Linux command line interface. The GUI is a wrapper around this terminal interface. However the CLI is also included for the capability to view/modify the command in its raw form which is being built in the GUI and give the user an experience of the CLI in its pure form.
5. **History/Starred panel:** This side-panel displays the history of commands executed. This is in terms of most recently used as well as the most frequently used commands. User can add commands from this section also into the repository for future use. A Bookmarking feature is also provided in the History section, using which makes a command appear in the Starred section.
6. **Tags panel:** Tags give quick access to commands relating to a category like Files, Process, text processing etc. Users can create their own tags and add commands to it. Clicking on a tag will display the list of commands associated with that specific tag.

Tasks and Timeline:

To develop our application we plan to use a waterfall model wherein we would develop the infrastructure for the application followed by agile method of software development; to add new features iteratively in the existing software infrastructure. Following is the timeline for the different tasks that we plan to do in order to successfully develop and test our application.

1. Initial requirement gathering (Duration: 2 Weeks)
 - Goals and Objectives (Duration: 1 Week) : Identify the HCI problems that we plan to tackle and list them down.
 - Literature Review (Duration: 1 Week) : Study existing applications and identify problems in the architecture. Also identify what are the features that those applications lack.
2. Design (Duration: 4 Weeks)
 - Architecture (Duration: 2 Weeks): Design the architecture such that it solves the problems which other applications had.
 - Prototyping (Duration: 1 Week): Transform the architecture into a user interface.
 - Get user survey and feedback(Duration: 1 Week) : Walk the user through the interface design and get their feedback. Conduct personalized interview with questions about features, usability and overall aesthetics of the application. Ask user for any improvement that they could think for a particular feature.
3. Agile development cycle (Duration:)
 - Development (Duration: 2 Weeks) : Implement the features that user want in a system.
 - Testing (Duration: 3 Weeks) : Perform usability tests.

- Feedback (Duration: 1 Week) : Ask user for a feedback
- Improvement (Duration: 1 Week) : Make improvements if any.

References:

1. CLI Companion
2. Inky: a sloppy command line for the web with rich visual feedback
<https://dl.acm.org/citation.cfm?id=1449737&CFID=741268924&CFTOKEN=55053021>
3. Integrating a command line shell into a browser:
<https://dl.acm.org/citation.cfm?id=1267724.1267739>
4. Gracoli: a graphical command line user interface
<https://dl.acm.org/citation.cfm?id=2442019&CFID=741268924&CFTOKEN=55053021>

Figures:

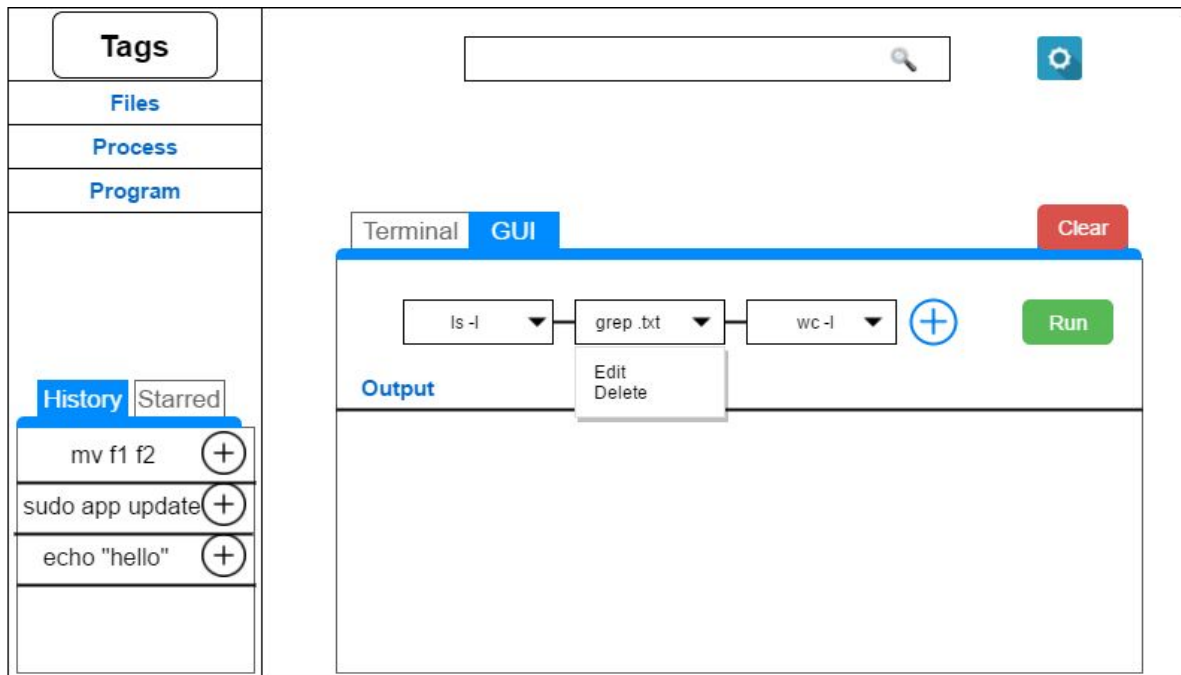


Fig 1:Main UI of Application




Add				
#	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



This command requires more information.
 Please provide more input

Input

Fig 4: Popup for commands which require user input`