

Master's Thesis

September 14, 2022

# CLI-Tutor

Can interactive learning make the command line  
more approachable?

**Qasim Warraich**

of Lahore, Pakistan (18-787-796)

**supervised by**

Prof. Dr. Harald C. Gall

Dr. Carol V. Alexandru-Funakoshi



University of  
Zurich<sup>UZH</sup>





Master's Thesis

---

# CLI-Tutor

Can interactive learning make the command line  
more approachable?

**Qasim Warraich**



University of  
Zurich <sup>UZH</sup>



**Master's Thesis**

**Author:** Qasim Warraich, [qasim.warraich@uzh.ch](mailto:qasim.warraich@uzh.ch)

**URL:** [gitlab.com/qasimwarraich/cli-tutor](https://gitlab.com/qasimwarraich/cli-tutor)

**Project period:** 2022-03-14 - 2022-09-14

Software Evolution & Architecture Lab

Department of Informatics, University of Zurich

---

# Acknowledgements

I would like to acknowledge my thesis advisor Dr Carol V. Alexandru-Funakoshi for his continual support, encouragement and effortless management of competing time zones throughout this thesis work. Additionally, I would like to thank my dear friends Dominique and Jasmin for their design advice, testing support and encouragement. Lastly, I would like to express my gratitude to all the participants of the user study associated with this thesis work. Thank you for your time, valuable feedback and kind words.



---

# Abstract

Despite the arguably dated appearance, difficult learning curve and practical non-existence in the modern personal computing space, Command Line Interfaces (CLIs) have more than stood the test of time in the software development world. There are a multitude of extremely popular tools and applications that primarily focus on the command line as an interaction medium. Some examples include version control software like *git*, compilers and interpreters for programming languages, package managers and various core utilities that are popular in areas such as software development, scripting and system administration. Command line interfaces are also utilised in areas outside of software development. For example, the infamous *Bloomberg Terminal* in the financial sector and in general computing applications such as email e.g. *mutt*, *neomutt* and text editing e.g. *Vim*, *Neovim*, *Wordstar*.

As mentioned before, the use of the command line as an interaction paradigm has effectively disappeared from a mainstream personal computer usage perspective. This reality contributes greatly to the intimidation factor and learning difficulty for those interested in getting into software engineering or system administration. This unfamiliarity, paired with the inevitability of usage of CLIs in the development space, highlights a need to make the command line more accessible to new users for whom text-based interaction with their computer is an alien concept. In recent years, interactive learning tools utilising features such as sandboxed environments have been gaining in popularity and have the potential to be a suitable medium for learning command line basics through actual usage, examples and practise.

In this work, we have created just such an interactive tutoring tool tailored for the command line. *CLI-Tutor* is a forgiving CLI application that aims to teach topics such as shell basics and Unix-like core utility usage through the use of guided lessons with interactive examples and feedback.





---

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem Description . . . . .	1
1.1.1	Solving The Problem . . . . .	1
1.1.2	Research Questions . . . . .	2
1.1.3	Why CLIs? . . . . .	2
1.2	Introducing "CLI-Tutor" . . . . .	3
1.3	Thesis Outline . . . . .	5
<b>2</b>	<b>The CLI-Tutor Tool</b>	<b>7</b>
2.1	Overview . . . . .	8
2.1.1	Curriculum . . . . .	8
2.1.2	Lesson Design . . . . .	9
2.1.3	Usability Considerations . . . . .	10
2.1.4	Safety Considerations . . . . .	13
2.2	Web Application . . . . .	13
<b>3</b>	<b>Design And Implementation</b>	<b>17</b>
3.1	First Attempts . . . . .	17
3.2	Tools and Libraries . . . . .	18
3.2.1	<i>CLI-Tutor</i> . . . . .	18
3.2.2	Web Application . . . . .	18
3.2.3	Server Infrastructure . . . . .	19
3.3	Features and Considerations of <i>CLI-Tutor</i> . . . . .	19
3.3.1	Why <i>Go</i> ? . . . . .	19
3.3.2	Shell Environment . . . . .	19
3.3.3	Lessons . . . . .	20
3.3.4	Lesson Parsing . . . . .	21
3.3.5	Embedded Files . . . . .	23
3.3.6	User Interface . . . . .	23
3.3.7	Validation . . . . .	25
3.3.8	Logging . . . . .	26
3.4	Web Application . . . . .	26
3.4.1	Frontend . . . . .	26
3.4.2	Backend . . . . .	27
3.4.3	<i>Docker Daemon</i> . . . . .	27
3.4.4	CLI Only . . . . .	27

3.4.5	Documentation Website . . . . .	27
3.4.6	Monitoring . . . . .	27
3.5	Extending CLI-Tutor . . . . .	28
<b>4</b>	<b>User Study</b>	<b>31</b>
4.1	Methodology . . . . .	31
4.1.1	Interactive versus Non-interactive . . . . .	31
4.1.2	Structure . . . . .	31
4.2	Participants . . . . .	32
4.2.1	Technical Experience . . . . .	32
4.2.2	Feelings about the CLIs . . . . .	34
4.2.3	CLI Usage . . . . .	36
4.2.4	Learning Preferences . . . . .	38
<b>5</b>	<b>Results</b>	<b>41</b>
5.1	Engineering Results . . . . .	41
5.1.1	Relevant Research Questions . . . . .	42
5.2	User Study Results . . . . .	43
5.2.1	Evaluation section . . . . .	43
5.2.2	Discussion . . . . .	43
5.2.3	Intimidation Factors . . . . .	45
<b>6</b>	<b>Reflections and Related Work</b>	<b>49</b>
6.1	Related Work . . . . .	49
6.2	Improvements . . . . .	49
6.3	Future Work . . . . .	50
<b>7</b>	<b>Conclusion</b>	<b>53</b>
<b>8</b>	<b>Appendix A: User Study Survey Questions</b>	<b>59</b>
8.1	Survey Questions . . . . .	59
8.1.1	User Familiarity Questions . . . . .	59
8.1.2	Evaluation . . . . .	61
8.1.3	Feedback . . . . .	62
8.2	Parser . . . . .	64

## List of Figures

1.1	Screenshot of <i>vimtutor</i> . . . . .	3
1.2	Screenshot of <i>CLI-Tutor</i> . . . . .	4
2.1	Screenshot of <i>CLI-Tutor</i> menu screen. . . . .	11
2.2	Screenshot of the <i>CLI-Tutor</i> lesson view showing the task tracker, feedback mechanism and an in-built helper command. . . . .	12
2.3	Screenshot of <i>CLI-Tutor</i> running in a browser. . . . .	14
2.4	Screenshot of <i>CLI-Tutor</i> showcasing the usage of colours. . . . .	14
2.5	Screenshot of the CLI-only version. . . . .	15
2.6	Screenshot of the welcome screen of the documentation website, linking to the CLI-only version. . . . .	15
2.7	Screenshot of the documentation website showcasing the light mode. . . . .	16
3.1	Screenshot of the progress tracker and interactive task feature. . . . .	20
3.2	Screenshot of a <i>CLI-Tutor</i> lesson showing values interpolated into the lesson. . . .	22
3.3	Screenshot of a hidden file created by the <i>CLI-Tutor</i> as an interactive example. . . .	23
4.1	The distribution of programming experience amongst study participants. . . . .	33
4.2	University level Computer Science experience amongst study participants. . . . .	34
4.3	Chart depicting interest in integrating CLIs more into day-to-day computer use. . .	35
4.4	Chart depicting the self-reported comfort level with CLIs in a Likert-style question. .	36
4.5	Chart depicting the frequency of command line usage amongst participants. . . . .	37
4.6	Chart depicting the frequency of command line usage amongst participants for personal tasks. . . . .	37
4.7	Chart depicting the preferences in learning mediums amongst participants. . . . .	38
4.8	Chart depicting the self-reported effectiveness perception of reading. . . . .	39
4.9	Chart depicting the amount of participants with previous interactive learning experience. . . . .	39
5.1	Chart depicting the evaluation section results of the interactive and non-interactive participant groups. . . . .	45
5.2	Charts comparing the post lesson intimidation levels between the two study groups. .	46
5.3	Charts comparing the post lesson future CLI usage impressions. . . . .	47
8.1	Custom Markdown Parser . . . . .	65

## List of Tables

5.1	Summary of questions answered correctly by method during the evaluation phase. . . .	44
-----	--	----

## List of Listings

2.1	Output of the help flag of <i>CLI-Tutor</i> running in a docker container. . . . .	7
3.1	Data structures for a Lesson and a Task within a Lesson. . . . .	20
3.2	Models used to build the user interface of <i>CLI-Tutor</i> . . . . .	24
3.3	Specification for Markdown lesson files. . . . .	29



# Introduction

## 1.1 Problem Description

Command line interfaces have been with us since the 1950s [1]. The rise of command line interfaces was closely coupled with the rise of time-sharing computer systems. Command line interfaces and time-sharing systems greatly shortened the feedback loop of earlier batch computing systems and allowed programmers to be able to modify their programs in near real time. Time-sharing systems also introduced the concept of a *shell*. A *shell*, a term coined by Louis Pouzin [2], is a program that allows for interaction with the operating system via textual commands [3]. The *Multics* [4] operating system pioneered the concept of operating system shells. The concept proved to be extremely influential and directly influenced the creation of the *Thompson shell* for the *UNIX* [5] operating system. The *UNIX* shell has arguably had the greatest influence on command line interfaces as we know them today, as most modern shells are descendants of the original *UNIX* shell [1].

The command line is still a popular interaction medium for tooling in the areas of software engineering and system administration [6,7]. However, the usage of the command line in the general personal computing space has practically disappeared [8]. This is especially true for younger individuals, whose very first exposure to computers has been working with Graphical User Interfaces (GUIs). This schism in interaction mediums causes issues when the same younger individuals strive to enter technical fields such as software development or system administration. The issue is further propagated by the fact that the command line is its own distinct interaction paradigm based entirely on writing and reading text. This can mean that the usage of traditional learning resources such as documentation and manuals might prove difficult to translate into effective usage without active textual interaction practise on the part of the learner. On the other hand, jumping straight into practising on the command line comes with its caveats. The shell can be an unforgiving tool for a novice user as it is very sensitive to syntax and often provides feedback that is difficult to understand for novice users. The shell also interacts directly with the operating system and does not require confirmation for certain destructive tasks such as file deletion.

### 1.1.1 Solving The Problem

The goal of this Master's thesis is to develop an interactive learning tool that simultaneously aims to address the previously introduced issues of lack of exposure to textual interaction, the novice unfriendly environment of the shell and the difficulties of traditional learning methodologies. The tool should provide a more forgiving experience than using the shell directly whilst still being a

faithful representation of a system shell. Concepts learned during the interactive tutorials should be directly transferrable to a standard Unix-like shell. Furthermore, the tool should be easy to use and encourage experimentation in order to augment the learning experience and in order to better exploit the advantages offered by interactive learning systems.

### 1.1.2 Research Questions

In this work, we set out to address the following research questions:

- RQ1** Are there identifiable patterns of difficulty when it comes to adopting CLIs? Can the "intimidation factor" be pinned down?
- RQ2** How should an interactive learning tool be designed to mitigate the difficulty and intimidation factor of learning CLIs?
- RQ3** How can a "forgiving" shell be implemented on top of an existing shell to enable the transition from learning to real-world usage?
- RQ4** Is the interactive tool more effective than text-based learning methods?
- RQ5** Are novice CLI users more likely to continue using CLI interfaces after using such a tool?

### 1.1.3 Why CLIs?

Before introducing our solution to the stated problem, we would like to discuss the motivations behind using command line interfaces and lowering the barrier of entry to their use.

Command line interfaces are still widely employed and being developed for a variety of uses. Benefitting from their simplicity, Command line interfaces allow for comparatively rapid development and the implementation of features when compared to their graphical counterparts, which would require a much more detailed and complicated implementation and more challenging user interface considerations. Command line programs also further benefit from simplicity of their textual nature when it comes to their ability to work with other programs. The simple notion of textual input and output being the main forms of interaction allow for an extremely flexible interface<sup>1</sup> for chaining tools together. This composition of programs is referred to as *piping* and is a hallmark element of what is considered to be the *UNIX Philosophy* [9]. Since most command line applications share this philosophy of textual input and output there isn't the need for bespoke mediums of communication to be developed and features can be more effectively shared across programs.

Since CLI programs are run within a shell, which has direct access to the underlying operating system, a tight integration between program and system is formed. This tight coupling, paired with the low system resource requirements, the ability to specify inputs and arguments via text and modify program behaviour upon instantiation with flags opens the room for powerful batch processing capabilities, easy automation and the ability to deal with large amounts of input and output efficiently.

Beyond the ease of development and composition, there are user interface elements of command line interfaces that even influence widely used graphical tools. Command line interaction features can commonly be found implemented into the text entry fields of search engines [10] and chat applications. An example of this is the ability to use the up arrow to edit the previously sent message or the ability to use special syntax, like exclamation points, to influence search results.

---

<sup>1</sup>In this case "interface" describes a medium through which programs can communicate, not a user interface.

The learning curve of such command line interfaces is however a strong barrier to widespread adoption and enjoyment of the benefits command line interfaces may bring. For new users, this learning curve is made steeper by the paradigm shift in the personal computing space towards graphical user interfaces. In following section, we will introduce the interactive learning tool, *CLI-Tutor*, developed as a potential solution that addresses the steep learning curve of the command line through interactive lessons and a sandboxed shell environment.

## 1.2 Introducing "CLI-Tutor"

```
=====
=  Welcome to the VIM Tutor  -  Version 1.7  =
=====

Vim is a very powerful editor that has many commands, too many to
explain in a tutor such as this. This tutor is designed to describe
enough of the commands that you will be able to easily use Vim as
an all-purpose editor.

The approximate time required to complete the tutor is 30 minutes,
depending upon how much time is spent with experimentation.

ATTENTION:
The commands in the lessons will modify the text. Make a copy of this
file to practice on (if you started "vimtutor" this is already a copy).

It is important to remember that this tutor is set up to teach by
use. That means that you need to execute the commands to learn them
properly. If you only read the text, you will forget the commands!

Now, make sure that your Caps-Lock key is NOT depressed and press
the  j  key enough times to move the cursor so that lesson 1.1
completely fills the screen.
~~~~~
Lesson 1.1: MOVING THE CURSOR
```

Figure 1.1: Screenshot of *vimtutor*

The *CLI-Tutor* tool is an interactive shell-like tutorial program aimed at making the command line more approachable. We draw inspiration from the *vimtutor* [11] (see: Figure 1.1) utility shipping alongside the popular terminal-based text editor *Vim*. *Vimtutor* is an interactive tutorial for the Vim text editor. It is one long guided lesson presented as a text file. The file itself contains the basic instructions for editing and creating text within Vim and is intended to be modified while proceeding through the lesson. This "learning by doing" approach presented itself as a very suitable approach for a command line tutorial application. Much like with Vim, where one must learn a new paradigm of modal editing. When novices learn the command line they are not only taking on new information about shell usage but also learning an entirely unfamiliar (textual) interaction paradigm. Newer versions of *vimtutor* are even more interactive. In the version included in the popular fork of Vim called *Neovim* [12], lines intended to be edited by the user also provide feedback regarding correctness in the form of green arrows and red crosses.

The *CLI-Tutor* tool introduces users to topics such as shell basics and Unix-like core utility usage through a series of interactive examples. The tool aims to relax the steep learning curve associated with the command line by leveraging interactive examples and a feedback mechanism to instruct and educate the user about the current stage of the lesson and provide some feedback based on the inputs of the user. The core of *CLI-Tutor* is contained within a command line ap-

plication written in *Go*<sup>2</sup> and serves as a standalone application. However, in order to make the learning experience safer for the user and to encourage exploration, the CLI application has been wrapped into a web application to form a sandboxed environment that exposes a terminal over the web. This means the user can use the tutorial application without fear of causing their own system any harm.

The *CLI-Tutor* application is fully open source and also comes with its own custom parser and structure for specifying lessons based on *Markdown*<sup>3</sup> files. This means that the material covered by the lessons can be easily contributed to and distributed, making the tool easily extensible.

```
spam @ spam-linux in /home/spam $
Basics of Textual Interaction : Introduction [0/7]:

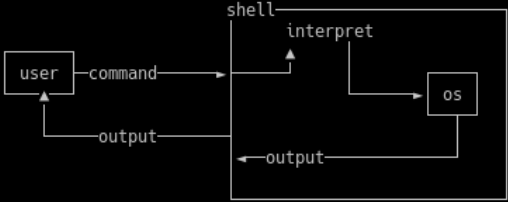
Welcome to the command line interface (or "CLI" for short). It may look
intimidating for now, but this tutorial will familiarise you with the concepts
you need to thrive on the command line.

Below, you can see the command prompt, where you can type commands. When you
are ready, type n or next and hit enter to continue.

spam @ spam-linux in /home/spam $ n
Basics of Textual Interaction : What is the Shell? [1/7]:

Essentially, the shell is a command line interface that lets you "talk
directly to your computer" using text messages. Instead of browsing a hierarchy
of menus and dialogs like in a conventional, graphical user interface, the
shell lets you access any setting or functionality of your computer directly
through dedicated commands. The shell will interpret these commands and
produce appropriate output.

More formally, the shell is a program that wraps your operating system (hence
its name) and acts as an intermediary between you, the user, and the operating
system. It manages the user's interaction by accepting input in the form of
commands and relays responses in the form of produced output, errors or
special shell features like shortcuts.
```



```

graph LR
    user[user] -- command --> shell
    subgraph shell
        interpret[interpret]
    end
    shell --> os[os]
    os -- output --> shell
    shell -- output --> user

```

This graph is an illustration of the interaction cycle between the user and shell.

Type **n** or **next** and hit enter to continue.

```
spam @ spam-linux in /home/spam $
```

Figure 1.2: Screenshot of *CLI-Tutor*

<sup>2</sup>Go programming language: <https://go.dev/>

<sup>3</sup>A popular plain text markup language.



## 1.3 Thesis Outline

Over the next few chapters, we will discuss and show the design, implementation and overall goals of this solution. In Chapter 2, the semantic aspects of the *CLI-Tutor* application and associated web application will be discussed. We will discuss the design and implementation of the solution in Chapter 3. Chapter 4, will discuss the methodology and introduce the participants of the user study conducted during this Master's thesis work. Our findings will be discussed in Chapter 5. In Chapter 6, we will evaluate and reflect upon the solution. We will consider existing approaches and make suggestions regarding building upon and refining *CLI-Tutor*, before, concluding in Chapter 7.



# The CLI-Tutor Tool

```
1 cli-student@3bc86f9090f9:~/tutor$ cli-tutor -h
2
3
4  _ _ _ _ _
5 / _ _ _ _ \
6 / _ _ _ _ \
7 \ _ _ _ _ /
8
9 A simple command line tutor application that aims to introduce users to the
10 basics of command line interaction.
11 Web version is available at https://clitutor.chistole.ch
12
13 Usage:
14 cli-tutor [flags]
15 cli-tutor [command]
16
17 Available Commands:
18 completion  Generate the autocompletion script for the specified shell
19 help        Help about any command
20 info        Prints information about the tool and log collection
21 repo        Prints a url to the git repository for this tool
22 version     Print the version number of cli-tutor
23
24 Flags:
25 -h, --help            help for cli-tutor
26 -n, --no-upload-log   Do not send a copy of the log to the developer
27 -x, --no-welcome     Do not show welcome message when entering tutor
28
29 Use "cli-tutor [command] --help" for more information about a command.
```

**Listing 2.1:** Output of the help flag of *CLI-Tutor* running in a docker container.

*CLI-Tutor* is a command line application written in the *Go* programming language. It is an interactive tutorial application focused on introducing novices to the Linux command line environment. The application is intended to be a forgiving but faithful representation of a shell running *Bash*<sup>1</sup>. In this chapter, we will discuss the semantic design considerations and choices made during the development of *CLI-Tutor*.

## 2.1 Overview

### 2.1.1 Curriculum

*CLI-Tutor* is intended to be used with zero prerequisite knowledge of the command line. To achieve this low barrier to entry, lessons are designed from the perspective of catering to a complete novice user. The curriculum consists of lessons that introduce the very basics of textual interaction and shell usage. As of writing, *CLI-Tutor* has 5 full lessons implemented as a proof of concept and serve as the curriculum in the user study conducted in this thesis work. *CLI-Tutor* is designed to be easily extensible with new lessons being easy to contribute. How this is achieved will be discussed in Chapter 3.

#### Summary of implemented lessons

**Basics of Textual Interaction.** This lesson covers the very foundational concepts of textual interaction. The user is introduced to the term *CLI* and explained what a *shell* is using an ASCII text diagram. The user is then introduced to the concept of issuing a command. To sow interest, the user is then asked to execute some commands to illustrate that the terminal is really interacting with their operating system. The users are asked to use the *curl* command to pull down a weather report from `wttr.in`<sup>2</sup> as an example of what is possible with the command line. Furthermore, the user is introduced to some important in-built commands of *CLI-Tutor* and taught how to clear the screen. Another feature introduced to the user is *zen-mode*, a feature that prevents the screen from being cluttered with output of previous commands to prevent the user from getting overwhelmed. This feature is activated by default in lesson 1 but is then deactivated, unless explicitly set, for the remainder of the curriculum.

**Getting more familiar with the shell.** This lesson jumps deeper into the concept of issuing commands and the "grammar" behind a command. To make this more intuitive the metaphor of a sentence is used, adapted from the documentation of the *cobra* command line tool library [13]. The lesson introduces users to fundamental concepts such as sub-commands and flags. The user is then asked to perform a series of tasks involving the *wc* word counting core utility. These actions are performed on a sample file that *CLI-Tutor* creates when launched and serves as a running example throughout the tutor.

**Basics of the file system and practising commands.** Lesson 3 is all about the file system and file system operations. The user is first introduced to the prompt and explained all the different sections of the prompt line. The prompt in *CLI-Tutor* is modelled after the default prompt of many popular Linux distributions (see: Listing 2.1, line 1), consisting of a username, hostname and current working directory path. The user is introduced to the idea of a hierarchical tree-like file system and is taught how to navigate around it. The *ls* command is also discussed and used

<sup>1</sup>*Bourne Again SHell*: <https://www.gnu.org/software/bash/>

<sup>2</sup>An online weather service that offers a command line friendly output.

to illustrate the concept of hidden files. *CLI-Tutor* also includes a hidden file that it places in the directory the tutor is launched from to help illustrate the concept of hidden files. All the example files included in lessons are deleted upon exiting the tutor. This lesson concludes with a long example for the user to try out that includes all the concepts discussed in the lesson.

**Shell shortcuts and tricks.** This lesson is aimed to be a bit more fun and introduces the user to some tricks and shortcuts available in the shell and the *readline* [14] library used for input in a *bash* shell environment. The user is introduced to the concept of a shell history file and also of the reverse history search feature with *Ctrl+r*. The cancel *Ctrl+c* command is introduced as well as the *!!* operator. As a final step, tab completion is introduced.

**Helping yourself.** This is the last lesson of the current version of *CLI-Tutor* and covers how a user can go about seeking help at the command line and helping themselves. The concept of pagers and some important keybindings of the *less* pager program are first explained before introducing the *man* command to the user. Users are also taught about help flags and encouraged to check out the help command of *CLI-Tutor*, shown at the very beginning of this chapter (see: Listing 2.1).

## 2.1.2 Lesson Design

Lessons are designed to cover one topic per lesson and consist of a short introduction to the concept before jumping into directly applying the subject in a series of interactive tasks. Each lesson, typically, also starts with a very short refresher of the last lesson and ends with a recap of the learnt commands. The lessons are designed to be guided but to leave some room for exploration for the user. *CLI-Tutor* also has some running examples that are included in the form of files that are created upon launching the program and deleted when the program exits. These files are included for the purpose of augmenting the tasks and to allow the user to directly apply commands rather than creating an environment to support their learning themselves.

A lesson consists of a series of tasks. Some of these tasks are purely educational and aim to teach the concept being covered in the lesson and some tasks are interactive. Generally, each task tries to build upon the previous. While writing the lessons, there was a deliberate emphasis on keeping the text short and engaging. The goal was to focus on interactivity and not to provide long text passages for the user to read. In order to achieve this succinctness, the use of examples, metaphors and formatted text to convey ideas was crucial.

Lessons also contain two additional features:

**Vocabulary.** The vocabulary of a lesson is the set of permitted commands in that lesson. This is primarily designed as a safety feature to prevent users from issuing commands incorrectly or bringing damage to their systems. The vocabulary consists of commands that will be covered in the current lesson and also commands previously covered in the tutor.

**Expected values.** Expected values are what drive the interactivity of *CLI-Tutor*. Expected values prompt the tutor to consider the current task as interactive and thus the tutor blocks advancing to the next task unless the interactive task is completed successfully. The expected value pertains to the expected result of performing a certain command or action. This expected value can be specified in three different ways (see: Listing 3.3). Expected values could be a constant values such as a string or number, they could be the result of some function call and even a runtime system call.

### 2.1.3 Usability Considerations

As previously mentioned, lessons are designed with no prior knowledge assumption. In addition to the curriculum, a number of considerations regarding the usability and design of the tool also had to be taken to maintain a low barrier to entry.

#### User Interface

In general, the user interface of *CLI-Tutor* is designed to be very simple and uncluttered. Strictly speaking, despite aiming to be an application that mimics a shell environment, *CLI-Tutor* is a TUI or "Text User Interface" application. A TUI application is a command line application that typically takes control of the entire terminal window and may contain some graphical elements. In the case of *CLI-Tutor*, the menu view contains a graphical list that is "hoverable" and clickable. The lesson view also takes control of the entire terminal window to render text to the screen. Special consideration had to be given to the user interface to make it not only new user-friendly, but also a fair representation of a terminal environment. Unlike tutorial software designed for programming languages, which tend to not have their own distinct environment, the shell is very much its own environment. Helping a user become familiar with the shell environment is crucial if the intimidation factors surrounding command line interfaces are to be overcome.

**Menu view.** The menu view is what the user is greeted with upon launching the program. It lists the lessons in a menu which the user can use to select and start a lesson (see: Figure 2.1). The menu view also has a help bar at the base with the keyboard shortcuts illustrated. The menu view allows for filtering the lessons by using the search functionality. Menu items display the title and description of a lesson, which are parsed directly from the *Markdown* based lesson files.

**Lesson view.** The lesson view is where the shell environment is reproduced. It is designed to look exactly like what interacting with the shell in a terminal looks like but with some added cues from the lesson. These cues include the current task, its instructions and any feedback about the last input of the user (see: Figure 2.2). The lesson view is also running a *Go readline* library to provide a user input mechanism essentially identical to that of a normal shell. The expected shortcuts all work as they should, but certain controls such as *Ctrl+d*, *Ctrl+z* and the *Ctrl+c* interrupts are purposely captured and ignored in order to not close the program unexpectedly.

**Mouse support.** To make the program more usable for individuals used to GUIs, basic mouse support had to be implemented in *CLI-Tutor*. Mouse support is more powerful in the menu view of *CLI-Tutor* than what would be common in most TUI applications. The user can hover to highlight and click to select a lesson. The hitboxes for the menu items are also precisely calculated based on the text rendered to offer a consistent and GUI-like mouse experience. In the lesson view, mouse support is implemented to a degree but still retains the goal of providing an honest shell-like experience. The user can access the scrollbar buffer and use the mouse to select and copy text using the keyboard shortcuts of their terminal emulator. In the web version, the right-click menu that is native to the browser is also available.

**Keyboard shortcuts.** As shown in Figure 2.2, some inbuilt commands available to the user, are not shell commands. For example, the *commands* command prints all the available commands, also known as the lesson's *vocabulary*, to the screen. There are other special commands in the lesson view that allow the user to navigate forward and backwards through the lesson, quit the lesson and toggle *zen-mode*.

```
Which Lesson would you like to start?

5 items

Basics of Textual Interaction
This lesson will introduce you to the very basics of command line interaction.
We will introduce you to the tutor program and how to formulate a command.

Getting more familiar with the shell
In this lesson, we will practice formulating commands and get more comfortable
interacting with the shell.

Basics of the file system and practicing commands
In this lesson, we will apply what you have learnt so far to help you navigate
your computer using the command line.

Shell shortcuts and tricks
In this lesson, we will learn about some shell shortcuts and tricks to make you
more productive at the command line.

Helping yourself!
In this lesson, we will help you help yourself. #RTFM

↑/k up * ↓/j down * / filter * q quit * ? more
```

**Figure 2.1:** Screenshot of *CLI-Tutor* menu screen.

```
cli-student @ 3bc86f9090f9 in /home/cli-student/tutor $  
Getting more familiar with the shell : Let's practice [3/8]:  
Type the wc file.txt command and hit enter.  
  
cli-student @ 3bc86f9090f9 in /home/cli-student/tutor $ wc notfile.txt  
wc: notfile.txt: No such file or directory  
  
That isn't quite right.  
Getting more familiar with the shell : Let's practice [3/8]:  
Type the wc file.txt command and hit enter.  
  
cli-student @ 3bc86f9090f9 in /home/cli-student/tutor $ wc file.txt  
4 26 147 file.txt  
  
Yay you did it!, Let's move to the next task.  
Getting more familiar with the shell : Now let's use some flags [4/8]:  
Type the wc -l file.txt command and hit enter.  
  
cli-student @ 3bc86f9090f9 in /home/cli-student/tutor $ commands  
Available commands: [wc whoami curl clear]  
  
Getting more familiar with the shell : Now let's use some flags [4/8]:  
Type the wc -l file.txt command and hit enter.  
  
cli-student @ 3bc86f9090f9 in /home/cli-student/tutor $
```

**Figure 2.2:** Screenshot of the *CLI-Tutor* lesson view showing the task tracker, feedback mechanism and an in-built helper command.



**Zen mode.** This is a special output printing mode of the lesson view. While not something available by default in a shell, this feature aims to prevent the user from being overwhelmed by long textual output alongside the lesson instructions. When *zen-mode* is activated, the screen clears itself before the output of every command. This has the effect of leaving only the latest output and task on the screen. This feature is activated by default in the very first lesson but then deactivated unless specifically set once the user has been taught about clearing the screen using the *clear* command.

**Feedback colouring.** *CLI-Tutor* utilises colour to differentiate different elements of the lesson. For example, a cyan colour is used to indicate commands, green is used for the task tracker, yellow is used for notes or information about key bindings and finally success and failure feedback is also coded in red and green respectively (see: Figure 2.2 and Figure 2.4). In addition to colour, text formatting elements such as the use of bold and italic text is also utilised to form visual hierarchies and to attract the user's attention.

### 2.1.4 Safety Considerations

**Fake jail warden.** To minimise the disruptions caused by file permissions issues that may be completely unfamiliar to the user, *CLI-Tutor* uses a fake jail to prevent the current working directory from being moved above the home folder of the user.

**Sandbox.** In addition to permissions issues which are merely an inconvenience, accidental file deletion or other potential mistakes during a file system lesson can result in permanent data loss. To mitigate this risk and to create a safe "sandbox" for users to use the application. The simplest way is to run the application in a *docker* [15] container. However, this greatly increases the barrier of entry to using *CLI-Tutor*. To make the tool accessible and easy to distribute a web application to serve instances of *docker* containers running *CLI-Tutor* was created.

## 2.2 Web Application

While the core component of *CLI-Tutor* is a command line application it does have an accompanying web application (see: Figure 2.3). The purpose of creating this web application was briefly discussed in the preceding section. The web application was the medium through which the user study was conducted. Two versions of the web application were created to support the two user groups in the user study (see: Chapter 4). One version contains the *CLI-Tutor* program and the second "CLI-only" version (see: Figure 2.6) serves to provide a sandboxed Linux shell to individuals participating in the user study. Additionally, also to support the User Study a documentation static website (see: Figure 2.7) was also created alongside the web application. The technical details of the web applications will be discussed in Chapter 3.

```

CHISTOLE: CLI TUTOR

kill this instance info

command --flags arguments

Now we will practice using some commands to help us navigate around the file system.

Type n or next and hit enter to continue.

clitutor-test @ cli-tutor in /home/cli-student/tutor $

Basics of the file system and practicing commands : A refresher from the last lesson [0/13]:

In the last lesson, we looked at how to formulate commands.
command --flags arguments

Now we will practice using some commands to help us navigate around the file system.

Type n or next and hit enter to continue.

clitutor-test @ cli-tutor in /home/cli-student/tutor $ n

Basics of the file system and practicing commands : Prompt [1/13]:

Before we begin, let's talk a bit more about the command prompt where you've
been entering commands.

It displays information about the:

current user = clitutor-test - You!
host machine = cli-tutor - Your machine
current directory you are in = /home/cli-student/tutor - Where you are on your machine.

The prompt is there to give you some additional context about the state of
your system. Importantly for us, it conveniently tells us where we are on our
computers. This will be very helpful as we proceed.

Type n or next and hit enter to continue.

clitutor-test @ cli-tutor in /home/cli-student/tutor $

```

Figure 2.3: Screenshot of *CLI-Tutor* running in a browser.

```

Welcome to the shell!
Type 'commands' to view a list of available commands or type 'exit' or 'quit' to quit the shell

Basics of Textual Interaction : Introduction [0/7]:

Welcome to the command line interface (or "CLI" for short). It may look
intimidating for now, but this tutorial will familiarise you with the concepts
you need to thrive on the command line.

Below, you can see the command prompt, where you can type commands. When you
are ready, type n or next and hit enter to continue.

cli-student @ 3bc86f9090f9 in /home/cli-student/tutor $

```

Figure 2.4: Screenshot of *CLI-Tutor* showcasing the usage of colours.



CHISTOLE: CLI

kill this instance info docs

```
cli-student@cli-tutor:~/tutor$ ls -la /
total 60
drwxr-xr-x. 1 root root 4096 Sep  5 20:46 .
drwxr-xr-x. 1 root root 4096 Sep  5 20:46 ..
-rwxr-xr-x. 1 root root   0 Sep  5 20:46 .dockerenv
lrwxrwxrwx. 1 root root    7 May 31 15:42 bin -> usr/bin
drwxr-xr-x. 2 root root 4096 Apr 18 10:28 boot
drwxr-xr-x. 5 root root 360 Sep  5 20:46 dev
drwxr-xr-x. 1 root root 4096 Sep  5 20:46 etc
drwxr-xr-x. 1 root root 4096 Aug 17 09:46 home
lrwxrwxrwx. 1 root root    7 May 31 15:42 lib -> usr/lib
lrwxrwxrwx. 1 root root    9 May 31 15:42 lib32 -> usr/lib32
lrwxrwxrwx. 1 root root    9 May 31 15:42 lib64 -> usr/lib64
lrwxrwxrwx. 1 root root   10 May 31 15:42 libx32 -> usr/libx32
drwxr-xr-x. 2 root root 4096 May 31 15:42 media
drwxr-xr-x. 2 root root 4096 May 31 15:42 mnt
drwxr-xr-x. 2 root root 4096 May 31 15:42 opt
dr-xr-xr-x. 198 root root    0 Sep  5 20:46 proc
drwx-----. 2 root root 4096 May 31 15:45 root
drwxr-xr-x. 5 root root 4096 May 31 15:45 run
lrwxrwxrwx. 1 root root    8 May 31 15:42 sbin -> usr/sbin
drwxr-xr-x. 2 root root 4096 May 31 15:42 srv
dr-xr-xr-x. 13 root root    0 Aug 19 11:27 sys
drwxrwxrwt. 2 root root 4096 May 31 15:45 tmp
drwxr-xr-x. 1 root root 4096 May 31 15:42 usr
drwxr-xr-x. 1 root root 4096 May 31 15:45 var
cli-student@cli-tutor:~/tutor$
```

Figure 2.5: Screenshot of the CLI-only version.

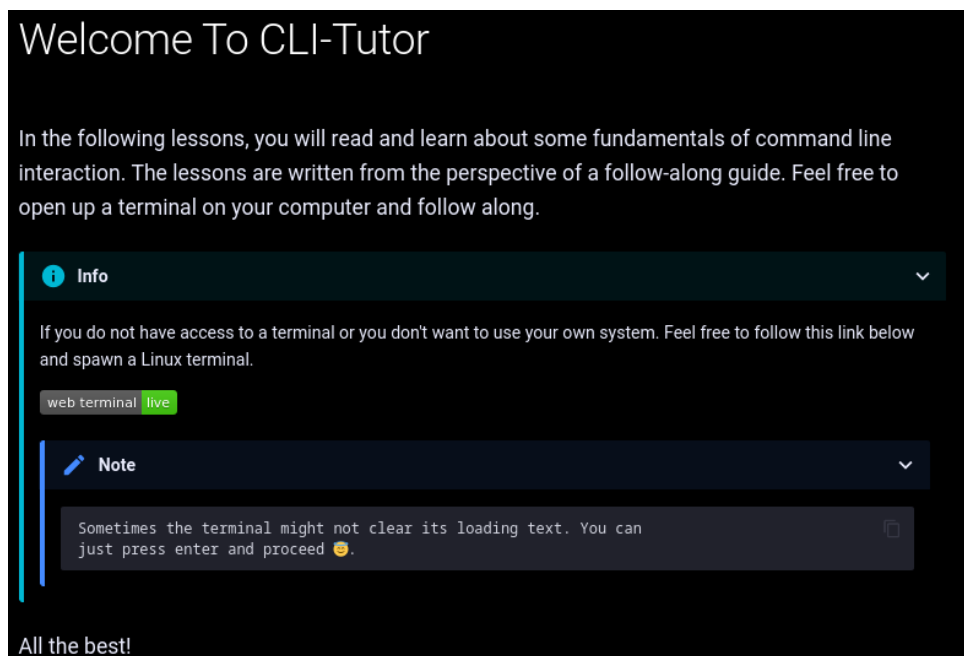





Figure 2.6: Screenshot of the welcome screen of the documentation website, linking to the CLI-only version.


1 - Basics of Textual Interaction


Search

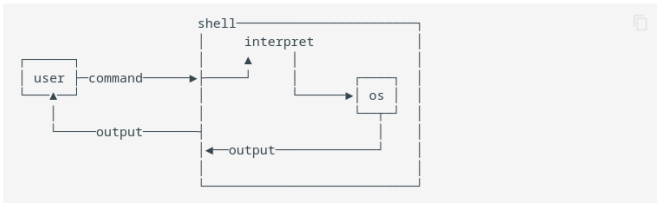

qasimwarraich/cli-tutor  
☆1 📄1

**CLI-Tutor**  
Home  
A Note Before We Begin  
1 - Basics of Textual Interaction  
2 - Getting More Familiar With The Shell  
3 - Basics Of The File System And Practicing Commands  
4 - Shell Shortcuts and Tricks  
5 - Helping Yourself!  
The End, for now!

## What is the Shell?

Essentially, the `shell` is a command line interface that lets you “talk directly to your computer” using text messages. Instead of browsing a hierarchy of menus and dialogs like in a conventional, graphical user interface, the shell lets you access any setting or functionality of your computer directly through dedicated `commands`. The `shell` will interpret these commands and produce appropriate output.

More formally, the `shell` is a program that wraps your operating system (hence its name) and acts as an intermediary between you, the user, and the operating system. It manages the user's interaction by accepting input in the form of `commands` and relays responses in the form of produced output, errors or special shell features like shortcuts.



```

graph LR
    user[user] -- command --> shell[shell]
    subgraph shell
        interpret[interpret]
    end
    shell --> os[os]
    os -- output --> shell
    shell -- output --> user

```

This graph is an illustration of the interaction cycle between the user and shell.

## How to issue commands

Commands supplied to the shell usually adhere to the following pattern:

```
command --flags arguments
```

A `command` is the name of a program the shell can run, while `--flags` and `arguments` are ways to tell the program what to do specifically. Most programs accept multiple flags and/or arguments.

`--flags` are pre-defined cues that alter the program's behaviour. You can imagine these as “switches” that turn certain options on or off.

`arguments` are additional pieces of information supplied as free text which the program will

**Table of contents**  
Introduction  
[What is the Shell?](#)  
How to issue commands  
Try another Command  
Let's try something more fun  
A note about getting overwhelmed  
Congrats! You've made it this far!

**Figure 2.7:** Screenshot of the documentation website showcasing the light mode.

# Design And Implementation

In this chapter, we will dive into the details surrounding the engineering of the *CLI-Tutor* tool, its supporting web application and the online documentation tool. We will first start by sharing the approaches of some of our early prototypes. We will discuss all the considerations and features that shaped the architecture of *CLI-Tutor*. Additionally, the technical stack that *CLI-Tutor* is built on will be introduced. This includes all technical aspects of this work ranging from the core command line application, *CLI-Tutor*, to the servers and tools required to build our web applications and perform our user study. A detailed discussion of the technical aspects around some of the most important features in the current version of *CLI-Tutor* will then follow.

## 3.1 First Attempts

In the early stages of this thesis work, while attempting to model the problem and develop the requirements for a technical solution several prototypes were created. The early prototypes were written in *Python* [16]. It was clear from an early stage that *readline* would be a mandatory element of any approach we would take. Early attempts broadly focused on two different approaches:

- An *Expect* [17] like approach. *Expect* is an add-on to the *Tcl* scripting language and is a way of writing scripted interactions with sub-processes. An *Expect* script can be used to launch a sub-process and communicate with it, for each action expected results can be specified. *Expect* scripts are popular for feigning user input and for testing or combining separate services. It is designed entirely with command line applications in mind and thus seemed a worthwhile approach for *CLI-Tutor*, especially because we wanted to design lessons which had expected values and interactive exercises. Two prototypes were built using *Python* and *Go*. The plan was to use *Expect* to interact with a spawned *Bash* sub-process. While initially promising, the *Expect* approach was abandoned due to limitations in the libraries and concerns around being able to build an appropriate user interface to wrap *Expect*.
- Creating and managing PTYs<sup>1</sup>. Prototypes integrating *readline* implementations were built using this approach with some success. However, this approach necessitated intricate low-level programming and raw communication with the terminal. It was eventually deemed too complicated and operating system specific to be appropriate for *CLI-Tutor*.

Other attempts included writing a GUI application to create a fake terminal environment. However, it was decided that a compiled command line application would be most appropriate

---

<sup>1</sup>PTY refers to a Linux pseudoterminal, the low-level interface through which terminal data flow is implemented in Linux.

as it would allow us to capitalise on the existing interface of the user's terminal and the wealth of libraries and tools available to facilitate command line application development. Creating a mock shell environment and using operating system native system calls was deemed the most suitable approach for *CLI-Tutor*.

## 3.2 Tools and Libraries

### 3.2.1 *CLI-Tutor*

- *readline* is the Go port of the gnu readline [14] line editing software. It forms the backbone of the lesson view in *CLI-Tutor*.
- *bubbletea* is Go framework for building stateful terminal applications using the *Elm* Architecture<sup>2</sup>. *bubbletea* is a part of the *charm*<sup>3</sup> project, which is a host of frameworks and libraries aimed at modernising the command line interface. The project also includes a host of smaller libraries and extensions designed to be used with *bubbletea* applications. *bubbletea* is the TUI framework used to manage switching between the menu and lesson views in *CLI-Tutor*.
  - *bubbles* is a collection of UI components to be used alongside *bubbletea*.
  - *bubblezone* is a community-contributed extension to *bubbles* components that allows for zones and hitboxes to be established in the user interface.
  - *lipgloss* is a styling and layout toolkit for *bubbletea* applications.
- *glamour* is a terminal *Markdown* rendering library also created by the *charm* team. It is used to display lesson tasks during a lesson in *CLI-Tutor*.
- *cobra* is a very popular CLI framework that is used in *CLI-Tutor* for feature flags, the help menu and for managing start-up and clean-up behaviours.
- *goldmark* is a *Markdown* parser written in Go. It is used in *CLI-Tutor* for parsing lesson files into data structures that drive the lesson interface and also for populating the menu with a list of available lessons.
- *termenv* is a Go library used for managing terminal colours and environments. It is used in *CLI-Tutor* for maximising compatibility across terminals, producing colours and for certain terminal controls such as screen clearing.

### 3.2.2 Web Application

- *Svelte* is a JavaScript/TypeScript frontend framework. It was chosen to be the frontend for our web application due to its simplicity, low learning curve and bundled output approach.
- *Vite* is the JavaScript bundler used alongside Svelte to package the project for the web.
- *XtermJs* is a terminal component for frontend web applications. It is the default terminal in the exceptionally popular text editor VSCode<sup>4</sup>. The terminal component is used alongside it's *attach* add-on to establish a connection over WebSockets to a *docker* container. This is how sandboxed environments are provided over the web to users.

---

<sup>2</sup>More on the *Elm* Architecture here: <https://guide.elm-lang.org/architecture/>

<sup>3</sup>More on the *charm* project here: <https://charm.sh>

<sup>4</sup>More about VSCode here: <https://code.visualstudio.com/>

- *Fiber* is the HTTP framework used to create the backend for our web application. It is primarily used as a proxy between the frontend and the *docker daemon* running on our backend server.
- *Docker SDK* is a first party Go Software Development Kit (SDK) for interacting with the *docker daemon*. It was chosen for interoperability with the backend HTTP framework which also uses the Go programming language.
- *MkDocs* is an industry-standard static site generator for documentation websites. It was used to create a documentation website for the user study conducted in this work.
  - *MkDocs-Material* is an extension to *MkDocs* providing an attractive modern theme and some additional visual elements to the static documentation website created for *CLI-Tutor*.

### 3.2.3 Server Infrastructure

- *Linode* is a cloud service provider offering low-cost Virtual Private Servers. Two servers were used to host the web application portion of *CLI-Tutor*.
- *Nginx* is a high performance web server. It was utilised on both our servers to serve as a reverse proxy to allow communication with *docker* and our backend API.
- *Gitlab CI/CD* is a Continuous Integration and Continue Delivery service used for testing, building and deploying our web application.
- *Grafana Cloud* is a monitoring service. It was used in *CLI-Tutor* to monitor uptime and for log agglomeration from our servers to ensure everything was running well.

## 3.3 Features and Considerations of *CLI-Tutor*

### 3.3.1 Why Go?

The Go programming language was selected for *CLI-Tutor* for numerous reasons. Go is a popular language for writing command line applications, with numerous modern CLI applications such as the *GitHub* CLI [18] and *docker* CLI [15]. This popularity in CLI applications also has the side effect of there being a wide selection of libraries available for implementing CLI interfaces (see: section 3.2). In addition to good tooling, Go programs are compiled and thus easier to distribute across operating systems than interpreted alternatives such as *Python* programs. Go also comes with a well-featured standard library with diverse options for interacting with the operating system, a core requirement of *CLI-Tutor*.

### 3.3.2 Shell Environment

Development of *CLI-Tutor* started with the implementation of a shell-like environment. A Go specific *readline*<sup>5</sup> package was used to create a simple *REPL*<sup>6</sup> as a proof of concept. The *readline* library supported a custom prompt and through the use of some system calls a functional prompt with a working *readline* environment was created. The next challenge was how to structure, create and interact with lessons.

<sup>5</sup><https://pkg.go.dev/github.com/chzyer/readline>

<sup>6</sup>Read Evaluate Print Loop

```
1 type Lesson struct {  
2     Name      string  
3     Vocabulary []string  
4     Description string  
5     Tasks     []Task  
6 }  
7  
8 type Task struct {  
9     Title      string  
10    Description string  
11    Expected    string  
12 }
```

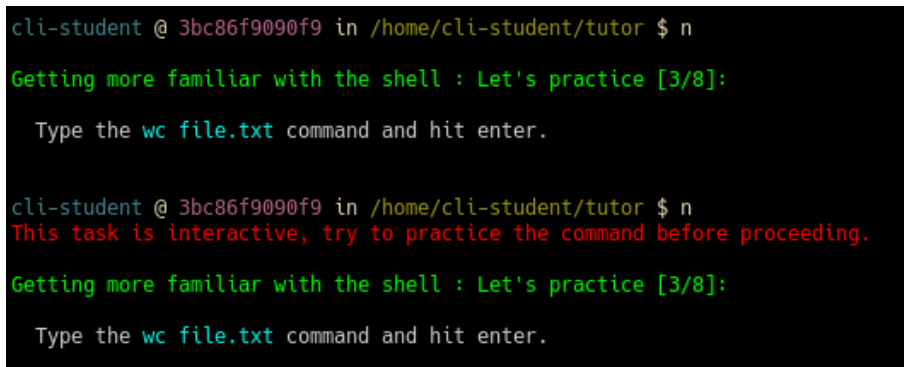
**Listing 3.1:** Data structures for a Lesson and a Task within a Lesson.

### 3.3.3 Lessons

The first task was to model a lesson in the form of a data structure that could be used in the working shell environment we had created at this point. It was decided that a lesson could be modelled as a container that contains a series of tasks, which would be modelled as a separate data structure (see: Listing 3.1).

#### Anatomy of a Lesson

A lesson has a title and description which are used to populate the menu screen of *CLI-Tutor*. Every task contains a title and description as well. The title of a task is displayed in the task tracker and the description is the actual textual content of a task. This content is either explaining a concept to the user, displaying a diagram or prompting the user to input some commands. The amount of tasks dictate the length of the lesson. The user can keep track of their progress inside a lesson via the task tracker, which displays the title of the lesson, the current task as well as a progression counter. A user is free to proceed back and forth through the lesson using the in-built commands *n/next* or *p/prev* unless the task is marked as interactive. In this case, the user must perform an interactive task such as running a specific command relevant to the task at hand. This mechanism as well as the progress tracker are showcased in Figure 3.1.



```
cli-student @ 3bc86f9090f9 in /home/cli-student/tutor $ n  
Getting more familiar with the shell : Let's practice [3/8]:  
Type the wc file.txt command and hit enter.  
  
cli-student @ 3bc86f9090f9 in /home/cli-student/tutor $ n  
This task is interactive, try to practice the command before proceeding.  
Getting more familiar with the shell : Let's practice [3/8]:  
Type the wc file.txt command and hit enter.
```

**Figure 3.1:** Screenshot of the progress tracker and interactive task feature.



## Specifying Lessons

The first versions of a *CLI-Tutor* lesson were written directly in *Go* source code. Not only was this tedious but also extremely specific to this implementation; rendering extensibility and the potential of open-source contributions difficult. It was decided that some sort of data exchange or markup language would be more appropriate for specifying lessons. Early considerations included *JSON*<sup>7</sup> and *YAML*<sup>8</sup>. These solutions proved to be difficult to work with in terms of formatting and did not make the process of contributing lessons any easier for an external contributor.

Ultimately, the decision to implement the lessons in *CommonMark Markdown*<sup>9</sup> was taken. *Markdown* was a pragmatic decision due to its ubiquity in the software development space and abundance of parsers. Furthermore, *Markdown* allowed us to specify our formatting directly in the lesson document and made for a source code free template for specifying lessons.

### 3.3.4 Lesson Parsing

Once *Markdown* was decided to be the implementation language for our lessons the next step was to populate our lesson-related data structures (see: Listing 3.1) in order to drive the tutorial program. To achieve this, a structure for specifying a lesson had to be created. After the specification was complete, a parser had to be written. The *goldmark* markdown parsing library for *Go* was selected. This library allowed us to create a custom parser for our lesson files. The parser first parses a selected lesson file into an *AST* (Abstract Syntax Tree). Once this tree is created, it is traversed using a *Depth First Search* as a means to navigate to all the different syntax nodes in the tree. In our lesson file specification, we make divisions using heading weights in our *Markdown* files.

Level 1 headings and their subsequent nested text are used for *Lesson*<sup>10</sup> level data, specifically the lesson's title and description.

Each subsequent level 2 heading refers to an individual *Task*. The text nested in between the level 2 headings belongs to the *Task* above the text. *Markdown* elements such as code blocks and inline code snippets are used to stylise text during the lesson. The expected value feature of *Tasks* is implemented using the quotation element (`>`) of *Markdown*. As the traversal of the *AST* continues *Tasks* are appended to the *Tasks* array in the *Lesson* data structure. This creates the series of tasks in the lesson, which are eventually navigated through by iterating this *Tasks* array.

Level 3 headings are used to define the lesson's vocabulary or set of permitted commands. There are additional features such as the nature of how interactive task values are to be calculated that can also be specified in the lesson file. For a full *Markdown* specification of a *CLI-Tutor* lesson, see: Listing 3.3 and for the implementation of the custom parser see: Figure 8.1.

## Template Expansion

There is actually one step that takes place just before the parsing of a lesson. This is where values can be interpolated into the *Markdown* files using *Go*'s powerful in-built template<sup>11</sup> library. This allows for special system-specific or environment-specific values to be interpolated into a lesson at the time of parsing, allowing for even more personalised explanations to be baked into the lessons. An example of this is demonstrated in Figure 3.2 where user and system-specific information is presented in the explanation of a shell command prompt. This allows for lessons to be less static

<sup>7</sup>JavaScript Object Notation. See: <https://www.json.org/json-en.html>

<sup>8</sup>Yet Another Markup Language. See <https://yaml.org/>

<sup>9</sup>A popular implementation of *Markdown*. See: <https://commonmark.org/>

<sup>10</sup>*Lesson* and *Task* in this capitalised and italicised form refer to our *Lesson* and *Task* data structures (see: Listing 3.1).

<sup>11</sup>*Go* standard library documentation: <https://pkg.go.dev/text/template>

and further capitalises on the opportunities available in interactive programs compared to written documentation.

```
command --flags arguments

Now we will practice using some commands to help us navigate around the file system.

Type n or next and hit enter to continue.

clitutor-test @ cli-tutor in /home/cli-student/tutor $

Basics of the file system and practicing commands : A refresher from the last lesson [0/13]:

In the last lesson, we looked at how to formulate commands.
command --flags arguments

Now we will practice using some commands to help us navigate around the file system.

Type n or next and hit enter to continue.

clitutor-test @ cli-tutor in /home/cli-student/tutor $ n

Basics of the file system and practicing commands : Prompt [1/13]:

Before we begin, let's talk a bit more about the command prompt where you've
been entering commands.

It displays information about the:

current user = clitutor-test - You!
host machine = cli-tutor - Your machine
current directory you are in = /home/cli-student/tutor - Where you are on your machine.

The prompt is there to give you some additional context about the state of
your system. Importantly for us, it conveniently tells us where we are on our
computers. This will be very helpful as we proceed.

Type n or next and hit enter to continue.

clitutor-test @ cli-tutor in /home/cli-student/tutor $ █
```

**Figure 3.2:** Screenshot of a *CLI-Tutor* lesson showing values interpolated into the lesson.

## Running Examples

Another feature of *CLI-Tutor* is the inclusion of certain files that are created on the user's file system for the duration the tutor program is running. These files serve as running examples and allow for lessons that use real files and data to be crafted. This allows for more creativity and opportunities for engaging the user's when creating interactive tasks.



**Figure 3.3:** Screenshot of a hidden file created by the *CLI-Tutor* as an interactive example.

### 3.3.5 Embedded Files

The two previous features in addition to the goal of being able to distribute *CLI-Tutor* as a single executable binary necessitated the inclusion of both lesson files and files that serve as running examples into the compiled binary. This was achieved using a compiler directive recognised by the Go compiler named *go:embed*<sup>12</sup>. This allowed for the creation of a virtual file system for the program to be able to read files as if it were a real file system. Embedding files directly into the program also greatly contributes towards the ease of distribution and contribution of *CLI-Tutor* as all the parts that make up the tutor reside in one codebase.

### 3.3.6 User Interface

The user interface of *CLI-Tutor* is divided into two main views, the lesson view and the menu view. The default view and what the user is presented with upon launching *CLI-Tutor* is the menu view. The menu is populated using a variation of the parser described in subsection 3.3.4. As previously mentioned, *CLI-Tutor* is a TUI application. To manage the two distinct views and provide interactive user interface elements the *Bubbletea* library was used. *Bubbletea* is a library for making Textual User Interfaces and is inspired by the *Elm* architecture. The *Elm* architecture is a pattern that consists of three-part structure where messages are sent between parts to alter state, update the user interface and act upon messages.

#### The *Elm* Architecture applied to *CLI-Tutor*

The *Elm* architecture's three main parts are:

- *Model*: The model controls the state of the application. The *CLI-Tutor* tool uses three models to manage its state. The *MainModel* is, as the name suggests, the main controller and houses the models for the menu view and the lesson view inside it. The job of the *MainModel* is to present the correct view to the user and to process all the different messages that can be sent back from the lesson and menu models. The selection of views is done by the *state*

---

<sup>12</sup><https://pkg.go.dev/embed>

```

1  const (
2      menuView sessionState = iota
3      lessonView
4  )
5
6  type MainModel struct {
7      state      sessionState
8      menu       tea.Model
9      lesson     tea.Model
10     quitting   bool
11     windowSize tea.WindowSizeMsg
12 }
13
14 type MenuModel struct {
15     list      list.Model
16     choice    string
17     quitting  bool
18     windowSize tea.WindowSizeMsg
19 }
20
21 type LessonModel struct {
22     currentLesson lesson.Lesson
23     rl             *readline.Instance
24     r             *glamour.TermRenderer
25     quitting      bool
26 }

```

**Listing 3.2:** Models used to build the user interface of *CLI-Tutor*.

member of the *MainModel*. The definitions of the three models in *CLI-Tutor* can be found in Listing 3.2.

- *Update*: This is a method that is implemented for a *Model*. The *Update* method consumes the messages sent around between the components of a system built on the *Elm* architecture. The update function can then manipulate the state of its corresponding *Model* to create reactivity. Messages are events generated through interaction with the application and actions such as resizing a window. In *CLI-Tutor*, messages are used to communicate the size of the terminal, manage key presses, enable selections in the menu and handle quitting the application. A message-based architecture allows for a lot of flexibility in terms of defining custom messages that can trigger other behaviours or make state changes. An example is parsing a lesson after it is selected in the menu view and assigning it as the current lesson in the lesson view.
- *View*: The *View* is the last part of the *Elm* architecture and is responsible for rendering the visual environment based on the state of its related *Model*. In *CLI-Tutor*, the view is only used for the menu view of the application. It cannot be used in the lesson view due to the implementation of the *readline* library, which consists of a blocking *for* loop. To get around this we block the *Elm* update loop for the duration of the lesson with the *readline* loop. When a lesson is completed or the user elects to quit the lesson, the *readline* loop breaks and an exit message is sent to the *Update* method *MainModel* signifying the change of state from lesson to menu. This update then triggers a state change on the *MainModel* which results in the user

being presented with the menu view. Because the state is kept throughout the run time of the application the menu will still have the lesson the user selected highlighted, enhancing the user experience.

## Markdown Renderer

We mentioned how we parse the lesson *Markdown* files in subsection 3.3.4. As lines of *Markdown* are parsed they are stored as raw byte arrays in the *Lesson* and *Task* data structures. This has the desired side effect of maintaining the formatting inherently coded into a *Markdown* document. When it comes time to display this information during a lesson, a terminal *Markdown* renderer named *glamour* is used. Rendered text to the terminal is both styled and formatted by *glamour* resulting in a near direct translation between the lesson file and what is displayed to the user in the terminal. Using a *Markdown* renderer also gave us the flexibility of specifying custom styling on all the different *Markdown* style elements and to fine-tune the look and feel of lessons by tweaking the style sheet being used by the renderer.

### 3.3.7 Validation

In the interest of building interactivity into our lessons, we needed to have a way of validating the inputs and actions of our users. Our input validation mechanism is incorporated into the *readline* loop of the lesson view in *CLI-Tutor*. Every time the user issues a command by hitting enter, the string the user entered at the command prompt is sent through a validation cycle. The first step is checking whether the string matches one of the many in-built special commands. These commands do things like proceed back and forth through the lesson, quit the lesson, print out a list of available commands and toggle *zen-mode*. If the string is not one of these special commands then it is assumed that the user is attempting to issue a system or shell command. For convenience, input strings at this point are stripped of white space and split into an array to examine the individual tokens that make up the whole command. The next steps are:

- Checking whether the specified command is in the current lesson's vocabulary and thus permitted to be run. This is done by checking the very first token of the string. Any tokens following the special shell operators `|` and `&` are also validated. If the command passes this stage it means the string contains commands valid and permitted in the current lesson.
- After passing initial validation, further checks are performed to prevent the user from running into permissions issues or working in a sensitive root or system directory. The command also checks for certain special conditions that need to be accounted for. Special conditions such as:
  - Several commands chained as pipes.
  - The presence of shell operators such as `||` and `&&`.
  - Whether the command has arguments or flags specified.
  - Whether the command launches an external command like a pager, which would require a redirection of the spawned sub-process's standard input and output streams to those of *CLI-Tutor*.

Each of these cases requires special handling in the way the system call is placed. The correct method is selected at this point and a system call is issued.

- At this stage the issued command is stored in a variable that tracks the last issued command. This is necessary to support the repeat previous command or `!!` shell operation.

- The next step is to return the combined output of the issued commands standard out and error streams to the user interface. In the case that the command was an interactive sub-process like a pager, the streams of the sub-process are redirected to the user interface at this point.
- A final step is to check where the current *Task* has a defined expected value. If this is the case the returned output of the issued command is compared to this expected value and the appropriate success or failure feedback is provided to the user. If the user successfully completed the task the tutor automatically increments the task tracker, resulting in the user being automatically advanced to the next task in the lesson.

### 3.3.8 Logging

To assist with gathering usage data and user behaviour, *CLI-Tutor* maintains a log file which can be optionally sent back to a server for examination. The log file is a mirror copy of the lesson session with timestamps for every action that occurs during the lesson.

This feature is on by default during the user study phase of this work but can be opted out of with a flag that can be supplied when launching *CLI-Tutor*. The purpose of this feature is entirely in support of the user study conducted in this thesis work and the feature will likely be removed from the tool upon the conclusion of this work.

## 3.4 Web Application

As previously mentioned, the core of *CLI-Tutor* is the command line application described in the previous section. The accompanying web application was created with the goal of simplifying the distribution of the tool and organising the user study. While not originally a design requirement, the idea of sandboxing our application proved very attractive, especially given the nature of our subject matter. While this kind of sandboxing would be easy to achieve with tools such as *docker* or virtual machines. It is too much to expect a novice user to set up and diminishes the potential user base of *CLI-Tutor*. Creating a web application also relaxes the cross-platform compatibility concerns of building an application that interacts directly with the operating system, like a shell.

### 3.4.1 Frontend

The frontend is a simple JavaScript web application written in Svelte. Svelte proved to be a good choice due to its simplicity, small bundle size and developer ergonomics. TypeScript was the main language of development when it came to the frontend. The main role of the frontend application is to provide some instructions and to present the user with a special frontend terminal component. This terminal component is from the popular library *Xterm.js*. This terminal emulation solution was chosen due to its proven robustness as the default terminal in the *VSCode* text editor and due to a first-party add-on that makes it possible to attach the terminal component over the WebSocket<sup>13</sup> protocol to the interactive *docker* container(see: Figure 2.3). The process of provisioning, connecting and cleaning up to *docker* containers is facilitated through the use of API<sup>14</sup> calls to a backend running on the same server.

---

<sup>13</sup>A full-duplex communication protocol native to modern web browsers.

<sup>14</sup>Application Programming Interface

### 3.4.2 Backend

The backend is a REST<sup>15</sup> API, written in *Go* using the *Fiber* library. The API allows the frontend to communicate with the *docker daemon* also running on the same server. Container creation, deletion and resizing of the containers *PTY*<sup>16</sup> are handled by this backend API using the *docker* SDK for *Go*; which interacts directly with the *docker daemon* running on the server. The attaching over *WebSockets* of the frontend to the spawned docker container is achieved through an API call directly to the *docker daemon* using the container's id which is received in a response message from the container creation API endpoint.

### 3.4.3 Docker Daemon

To facilitate the sandboxing via *docker*, the *docker daemon* is exposed over *TCP*<sup>17</sup> with *TLS*<sup>18</sup>. This allows for a direct connection over *WebSockets* to be negotiated between the frontend user interface and the *docker* container running *CLI-Tutor*. For safety and consistency with how the backend API is accessed, the *docker daemon* is accessed via reverse proxy using the *Nginx* web server which manages all incoming *HTTP(S)*<sup>19</sup> connections to the server.

### 3.4.4 CLI Only

As mentioned earlier, a CLI-only version (see: Figure 2.6) was also created using essentially the same architecture as the version that includes *CLI-Tutor*. Slight visual differences were made to differentiate the two however they are based on fundamentally the same *docker* image, with the only difference being that the "CLI-only" version does not contain the *CLI-Tutor* tool and drops the user directly into a Linux shell. The CLI-only version also contains an additional button for navigating to the documentation website.

### 3.4.5 Documentation Website

Created using MkDocs, [19] a widely used tool for documentation in the software development world. The documentation website (see: Figure 2.7) was built to support the User Study. It uses the same lessons as in the *CLI-Tutor* application, albeit with some prompts for user interface actions removed. The fact that the lessons in *CLI-Tutor* are implemented in *Markdown* made creating web pages out of the files straightforward.

### 3.4.6 Monitoring

A monitoring stack using *Grafana* was also set up on the web servers. This was done to monitor performance, availability and to facilitate the debugging of any potential issues. Log agglomeration was also performed with *Loki*<sup>20</sup> and *Promtail*<sup>21</sup>, the latter of which was run in a separate docker container. This information was forwarded to the *Grafana Cloud* web interface.

<sup>15</sup>Representational state transfer: An architectural style for creating interfaces to allow Client-Server communication.

<sup>16</sup>This is necessary to sync the size of the pty created in the frontend terminal with the interactive *docker* container's terminal running on the server.

<sup>17</sup>Transmission Control Protocol

<sup>18</sup>Transport Layer Security

<sup>19</sup>Hypertext Transfer Protocol (Secure)

<sup>20</sup>Log agglomeration tool in the *Grafana* monitoring stack

<sup>21</sup>The server side agent used by *Loki* to scrape system logs.

## 3.5 Extending CLI-Tutor

The extensibility of *CLI-Tutor* has been a priority since the inception of the tool. This is a key factor for the longevity and usefulness of the tool beyond this thesis work. *CLI-Tutor* is open source and hosted in public repositories under a *MIT* [20] license. Contributing lessons to *CLI-Tutor* is uncomplicated owing to the use of *Markdown* files to specify lessons. On the following page, we provide a complete specification and readable example of all the features available during lesson creation and supported by *CLI-Tutor*'s lesson parse, at the time of writing (see: Listing 3.3).



```
1 # Lesson Title
2
3 A line under a level 1 heading is the lesson description. This is also
4 displayed in the menu.
5
6 ## First Task Title
7
8 Task instructions are parsed as normal lines under a level 2 heading.
9
10 Even line breaks and nested elements will reflect in the lesson. This
11 greatly enhances readability.
12
13 `Commands` are highlighted with backticks.
14
15 Text can be injected into a lesson at the time of parsing with template
16 functions like this --> {{SomeFunc}}
17
18 ## Second Task Title
19
20 This is the first line following a second level 2 heading and thus the text
21 of task #2.
22
23 ```
24 Code blocks can also be used to represent larger blocks of instructions or
25 for ASCII diagrams.
26 ```
27 ## Interactive tasks with expected values
28
29 If the current task is expecting a certain output to a command the user
30 types, we can specify that using the `>` syntax.
31
32 > {{TestFunc}}
33
34 ## Runtime expected value calculation
35
36 Sometimes the correct value for a given task can only be computed at run
37 time, to achieve this we can specify the expected value of a task to
38 be the output of a system call by prepending a `!` and then the expected
39 command.
40
41 > !ls -la
42
43 ### Lesson Vocabulary (Provided as a comma-separated list of values under a
44 ↪ level 3 heading )
45
46 vim, ls, cp, cat, echo <--- only these commands will be permitted in this
47 ↪ lesson.
```

**Listing 3.3:** Specification for Markdown lesson files.



# User Study

In order to test and validate the effectiveness of our solution in reference to our research questions (see: subsection 1.1.2), a user study was conducted. The goal of this user study was two-part. Firstly, we were interested in assessing the usability and response to *CLI-Tutor*. Secondly, we wanted to ascertain if interactive learning would be a more effective medium to teach command line interaction than traditional alternatives such as online documentation or books. In this chapter, we will describe our user study in detail.

## 4.1 Methodology

The user study for this thesis work was conducted remotely and asynchronously. We designed an online survey using the *LimeSurvey* [21] tool made available to us by the University of Zurich.

The focus of the user study was primarily on the comparison between interactive learning approaches such as that of *CLI-Tutor* and traditional ones, which are mostly reading-based. In the modern software development space, online documentation is the status quo and the medium we choose to compare our solution against. As discussed in Chapter 3, *CLI-Tutor* uses *Markdown* to specify lessons. Many static documentation generation websites use *Markdown* files to generate documentation. This is also true for our chosen generator. This enables us to objectively compare the interaction medium rather than the lesson content, since the exact same lessons can be used in both tools. Furthermore, due to the popularity of *MkDocs*, it is a very realistic representation of documentation that individuals may encounter in the wild.

### 4.1.1 Interactive versus Non-interactive

We designed our user study intending to perform A/B style testing comparing learning mediums. To support this we used the pseudorandom number generation feature of *LimeSurvey* to assign our participants to one of two groups, interactive and non-interactive.

### 4.1.2 Structure

In this section, we provide a structural overview of our online survey. The entire survey, complete with possible answers, is available in the appendix.

Our online survey was divided into the following steps:

1. **User Familiarity:** In this section, participants answered questions relating to their experience, interest and preferences to provide us with some insights on each individual participant.
2. **Assignment:** All participants were divided into one of two groups, interactive or non-interactive. The assignment value is unknown to the participant at the time of starting the survey. Our survey tool then conditionally rendered a URL for the participants to follow to the next section of the survey.
3. **Tutorial:** At this stage, once the participants have been assigned to one of the two groups, they will either be sent to our web application running *CLI-Tutor* or to our documentation website. If assigned to the non-interactive group participants were also given the option of navigating to the *CLI-only* version of our tool, in case they did not have access to a Unix-like terminal (see: Figure 2.6).
4. **Evaluation:** The evaluation stage is where participants were asked a series of basic questions relating to the lessons they took in the previous step. Questions were a mixture of multiple choice and free text questions. This section was identical for both user groups as the lessons were also identical.
5. **Feedback:** In this section, participants were able to provide feedback regarding their experience. All but one of the questions in this section were identical for both user groups. The non-interactive group were asked one additional question regarding interactive learning: *Do you think an interactive command line tutorial application would improve the learning process?*
6. **Feedback Opposite (optional):** This section was optional and included only one quick feedback question. At the end of the survey, participants were then given a chance to try out the opposite tool to which they were assigned for the user study. No evaluation or tutorial was mandated here and participants were given one free text field to report on their feelings using the alternative tool.

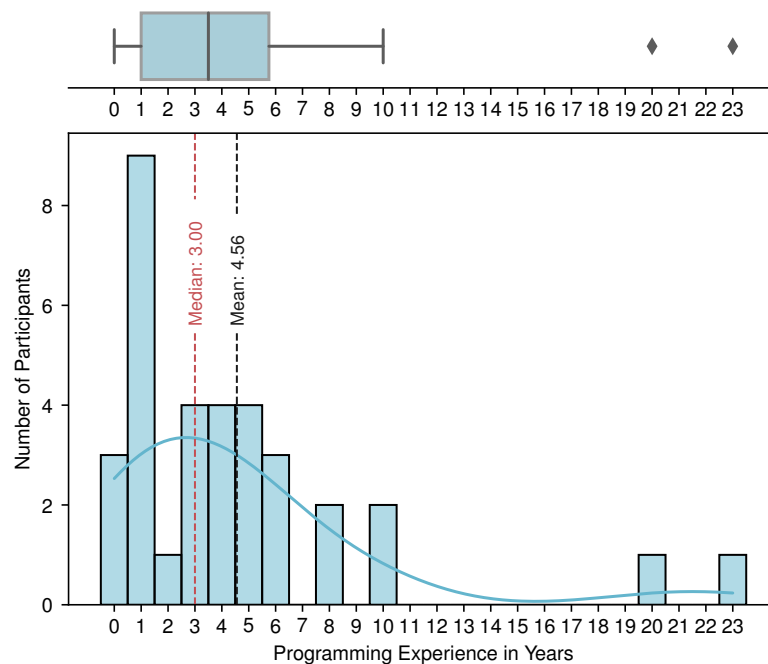
## 4.2 Participants

In this section we will share insights we gathered from the first section of the survey, where we asked experience, familiarity and more general questions.

In total, 34 participants took part in our user study. 19 of whom were assigned to the interactive group and 15 to the non-interactive group. Recruitment was primarily done through University channels though some participants were also sourced through work emails and word of mouth. While not limited to individuals in software related fields, over 60% (see: Figure 4.2) of the participants were from technical backgrounds or currently computer science students.

### 4.2.1 Technical Experience

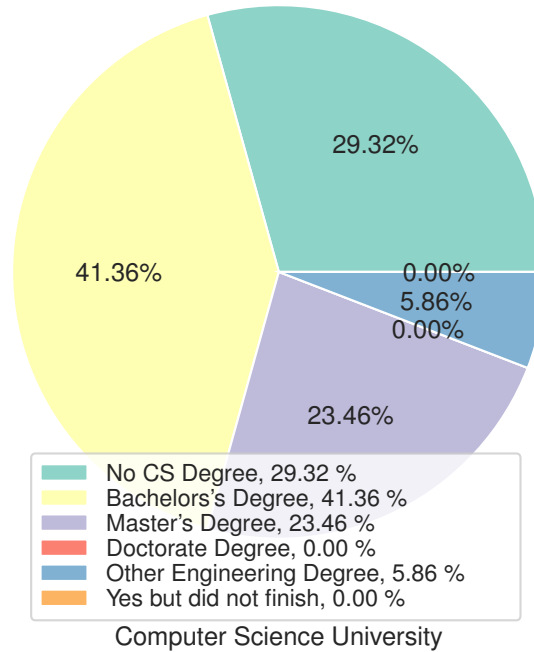
Given the goals of *CLI-Tutor*, it comes as no surprise that most of the individuals who participated in our user study were not very highly experienced, though there were two outliers with over twenty years of programming experience. Most of the participants were far less experienced with a median experience of 3 years (see: Figure 4.1). While not very highly experienced in programming, most of our participants did come from some sort of Computer Science or engineering background (see: Figure 4.2).



**Figure 4.1:** The distribution of programming experience amongst study participants.

In the above figure, the programming experience of all the participants is presented as a histogram. Programming experience is not a metric that can reliably predict command line proficiency but can indicate the levels of exposure to command line interfaces a participant may have had. In general, most participants have been programming for 5 years or less and the most common group was individuals who have been programming for 1 year. This makes sense as a significant portion of participants were sourced through a recruiting email distributed in the University of Zurich.

Participants were also asked about whether they had any university experience in computer science or related fields. Most participants had some degree of experience at the Bachelor's and Master's level but despite this 29.32% reported not having any university experience in Computer Science. Two participants, came from a mechanical and electrical engineering backgrounds. The complete distribution across all levels of university experience can be found on the following page in Figure 4.2.



**Figure 4.2:** University level Computer Science experience amongst study participants.

### 4.2.2 Feelings about the CLIs

To gauge interest and feelings, questions regarding interest and comfort level with CLIs were also asked. In general, the interest in integrating the command line more into day-to-day computer use was high. With 79.41% (see: Figure 4.3) of respondents reporting that they were interested in doing so. Participants were also asked to explain their motivations. Motivations were varied, but there were some commonalities in the motivations to integrate CLIs more in daily computer usage. For some, the motivation was a curiosity about getting to know their computers better:

*"It is an additional tool, where one can learn and do a myriad of things with while also improving one's understanding of computer systems as a whole."*

*"to build up a better understanding of how things work behind the scenes."*

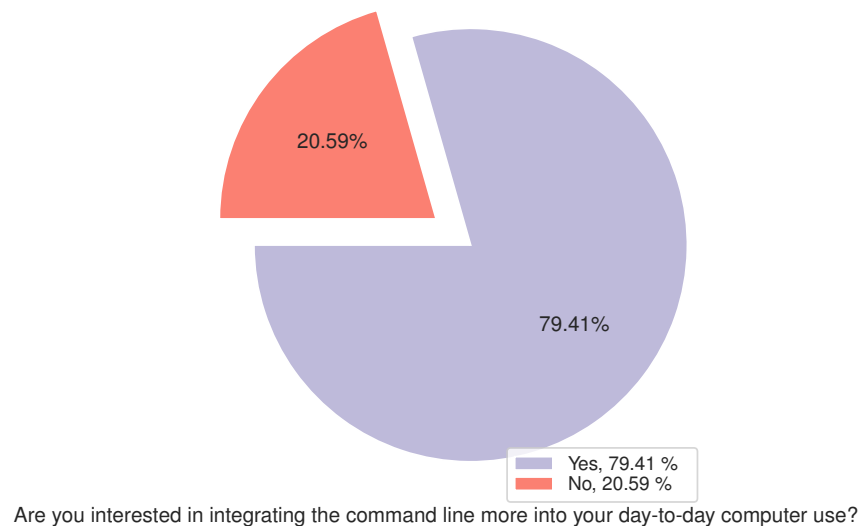
For one respondent, this sentiment was even partially altruistic.

*"For using to fix ppl's and my computer if it breaks/bugs out"*

Other sentiments expressed interest in integrating CLIs more in the work environment as a

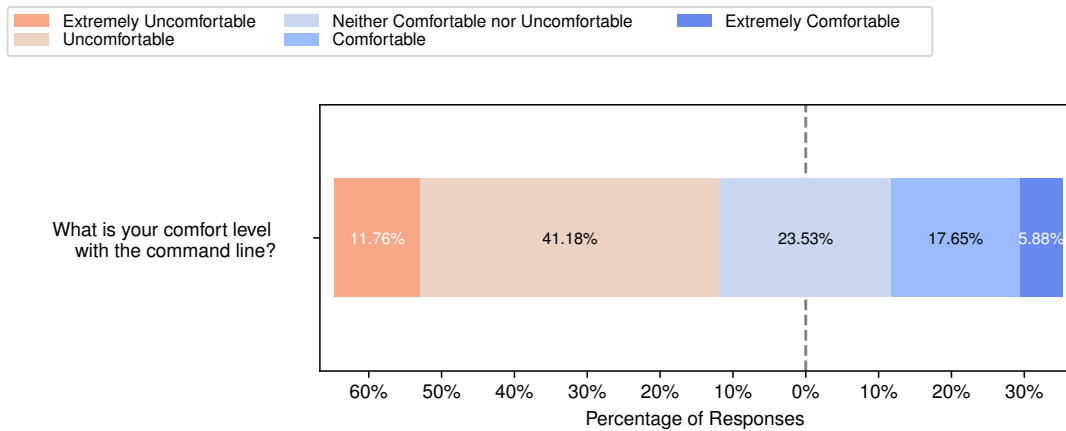
productivity booster or as an important skill.

- *"Due to my field of work, more familiarity and proficiency with any CLI would be helpful"*
- *"to optimize the workflow"*
- *"It's much easier to replicate/reproduce the results than using GUI"*



**Figure 4.3:** Chart depicting interest in integrating CLIs more into day-to-day computer use.

Another factor the participants were questioned about was their existing comfort level with command line interfaces. This question was presented in a Likert-style fashion with answers ranging from "Extremely Uncomfortable" to "Extremely Comfortable" (see: Figure 4.4).



**Figure 4.4:** Chart depicting the self-reported comfort level with CLIs in a Likert-style question.

More than half of the participants responded that they were extremely or at least uncomfortable with using the command line. The percentage of individuals who identified as at least comfortable was 23.53%. This large amount of discomfort aligns with the higher amount of low-experience participants. Another potential contributing factor is the fact that most people were university students who generally have only ever worked with GUIs in their personal computer usage.

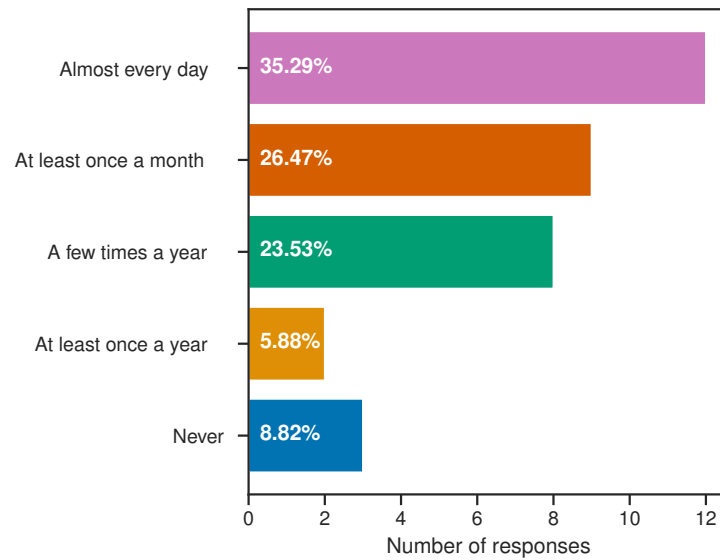
### 4.2.3 CLI Usage

Beyond comfort and interest, the amount of existing experience specifically with command line interfaces is another interesting factor. Participants were questioned about their existing usage of command line interfaces. These questions were presented in a multiple choice and single-answer form. Two questions were asked of the participants, and they were intentionally divided in order to highlight the differences between general usage (see: Figure 4.5) and individuals who also use CLIs in their personal time (see: Figure 4.6). Furthermore, this intentional division was intended to balance the fact that a lot of individuals participating in the user study were computer science students and might have had some experience that does not reflect their personal affinity towards CLIs.

The usage of command line interfaces amongst our participant groups with higher than anticipated. Over half of respondents stated they use command line applications or tools at least once a month, with 35.29% stating that they almost use them every day. This finding, strongly highlights the continuing relevance of command line interfaces, especially in the software development space. Fascinatingly, when asked about the usage of command line applications or tools in day-to-day personal computing the statistics are almost mirrored between the previous question. Almost 38.24% stated that they never use command line applications for personal tasks. This indicates that CLIs are perceived and used mostly as work-related tools and their qualities in personal computing are not considered or are not accessible. This point could be related to the fact that since CLIs are no longer the default paradigm of interaction with the computer, and they are usually presented to individuals in environments where the tool is considered the *De Facto tool for the job*, such as with *git* [6].

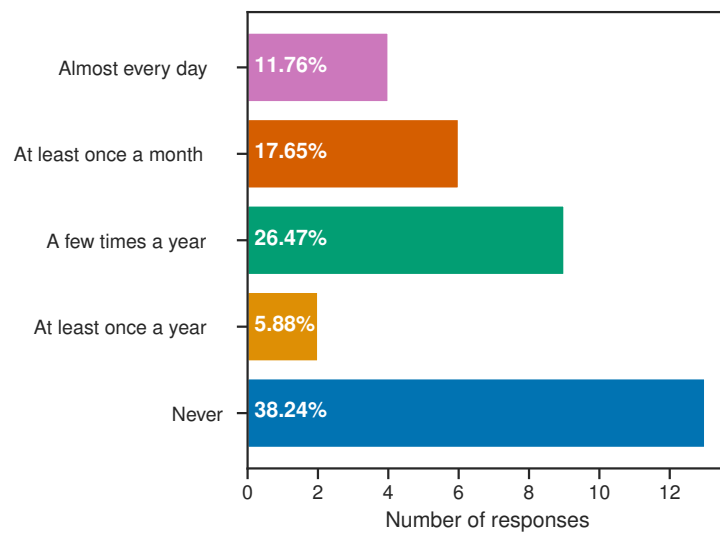


On average how often do you use command line applications or terminal based tools?



**Figure 4.5:** Chart depicting the frequency of command line usage amongst participants.

On average, How often do you perform regular computing tasks such as file management from the command line?

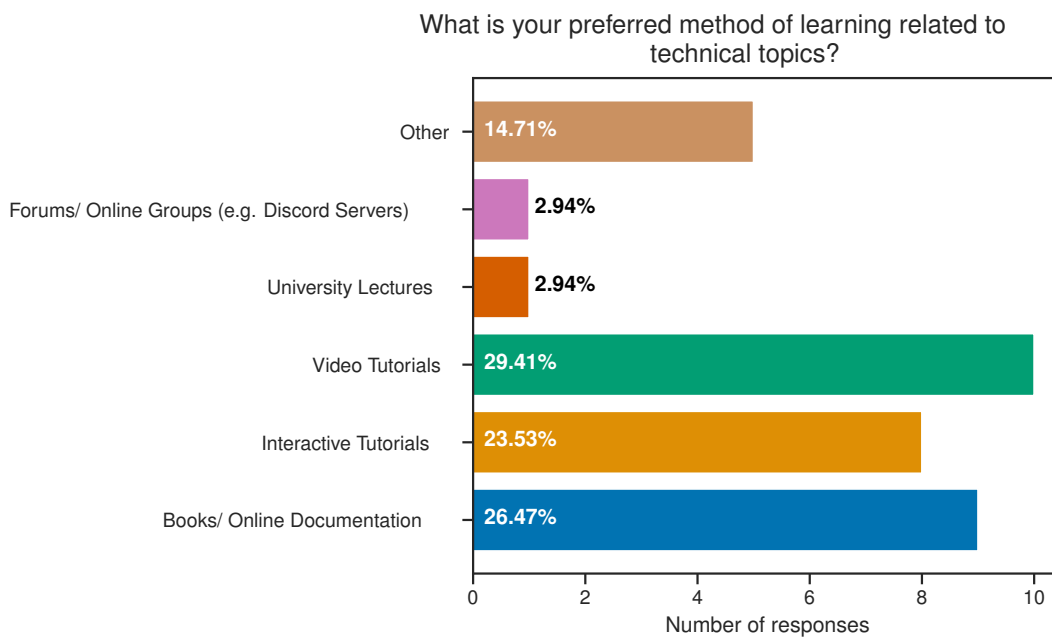


**Figure 4.6:** Chart depicting the frequency of command line usage amongst participants for personal tasks.

### 4.2.4 Learning Preferences

The preferences regarding learning mediums are another important factor relevant to this study. As also mentioned in our research questions (see: subsection 1.1.2), analysing the effectiveness of interactive learning tools is part of the goals of this work.

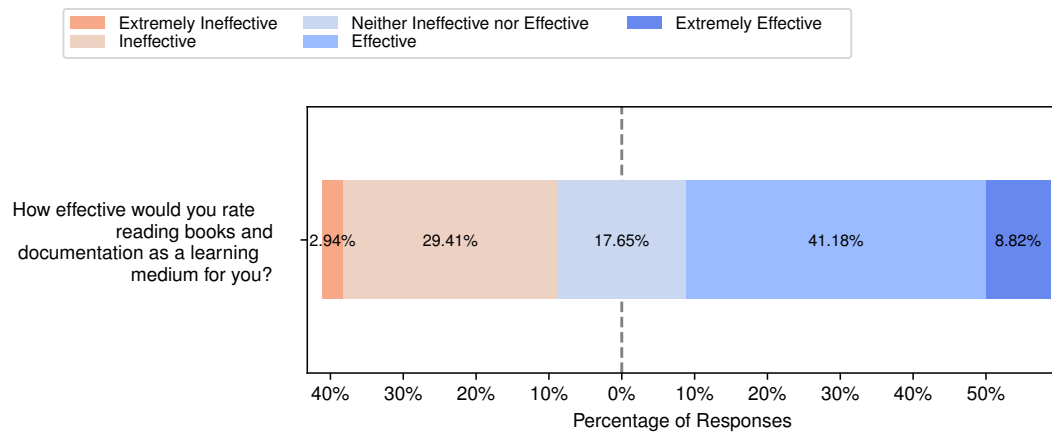
To get an idea of the existing preferences of our study participants, they were asked to select what mediums they preferred when learning technical topics. Encouragingly, over 50% of participants indicated their preferred method of learning was at least partially interactive, with most indicating that their preferred method was video tutorials. This is most likely due to the vast amount of video content, especially technical content, available on the web. This large proportion of individuals preferring video and interactive tutorials is a good indicator of the potential utility of interactive learning tools such as *CLI-Tutor*. 26.37% of participants stated their preferred method of learning was via books or online documentation.



**Figure 4.7:** Chart depicting the preferences in learning mediums amongst participants.

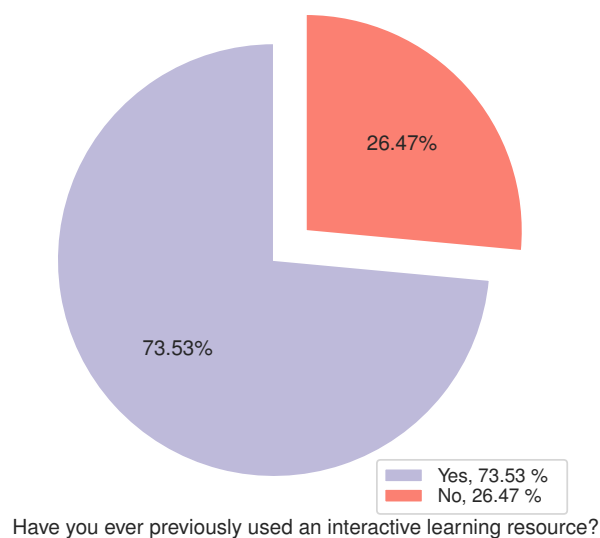
From the above question, the verbatim responses for *Other* were:

- "All of the above"
- "Work (good mix between pressure to get it right and being paid for it)"
- "Starting projects and learning from tutorials/documentation as I go"
- "testing curiosities"
- "DIY"



**Figure 4.8:** Chart depicting the self-reported effectiveness perception of reading.

Similar to the question regarding comfort with CLIs, the perceived effectiveness of written documentation was also inquired about in a Likert-style, with options ranging from extremely effective to extremely ineffective. Though a majority of individuals considered reading to be an effective means of learning, a large proportion (29.4%) reported reading to be ineffective and a significant 17.65% were ambivalent about the question.



**Figure 4.9:** Chart depicting the amount of participants with previous interactive learning experience.

Participants were also questioned about their existing experience with interactive learning resources. The vast majority, (73.53%) indicated that they had previous experience with interactive learning resources (see: Figure 4.9).



# Results

In this chapter, we will discuss the outcomes of this thesis work. We will look into our findings in light of the research questions we defined in Chapter 1 in subsection 1.1.2. Furthermore, we will divide our results into two distinct sections, focusing on the results of the engineering effort in building *CLI-Tutor* and secondly, the findings from the user study discussed in the preceding chapter.

## 5.1 Engineering Results

From an engineering perspective, we managed to achieve almost all the technical goals we set out for *CLI-Tutor*. Design goals relating to ease of distribution and access were fulfilled as *CLI-Tutor* is a single binary that is easy to distribute and containerise. *CLI-Tutor* also works natively on *GNU/Linux* and *Macintosh* operating systems and thanks to the associated web application and docker images for the tool, it is possible to use *CLI-Tutor* on almost any mainstream operating system. The web application and ease of containerisation also helped achieve another goal of creating a safe and forgiving sandboxed environment for users to experiment within.

This ease of experimentation and application of learnt skills is an important factor when it comes to developing confidence with CLIs as they are a foreign interaction paradigm to most novices. Ease of experimentation and ability to apply learnt skills is not purely achievable through technical design alone. The curriculum of *CLI-Tutor* also reflects an emphasis on giving the user space to try things out without worry or the tutorial software actually getting in the way. One participant in the user study particularly admired this aspect of *CLI-Tutor*:

*"I think the cli-tutor really provides the right amount of guidance, while leaving enough freedom to experiment and test things. Thus it was trivial to examine small questions one asks themselves like: what would happen if one calls "rmdir" on a directory that contains another empty directory. All in all a great experience, and a drastic improvement compared to non-interactive 2 hour lectures."*

Based, on feedback received by participants in the user study we can conclude that the curriculum was well received and appropriately designed as many respondents found this to be a particularly strong aspect of *CLI-Tutor*.

*"Very good tutorial, great work! It starts at the beginning and fills the gaps that are necessary to make working with the command line more quicker and more useful."*

*"It was overall very well structured. I felt like the tutorial was a very smooth process, taking you from one topic to the most closely related next topic."*

*"Great work, short but precise! I would love to do more of your tutorials :)"*

The level of difficulty was also appropriate as many participants found the course content appropriate for newcomers. This is encouraging as tutorial tools are generally targeted at beginners and *CLI-Tutor* is no exception.

*"Was a pleasant and well thought out lesson. Would recommend this for newcomers to the CLI, as it does have a friendly and fun tone to the entire teaching environment."*

There were also multiple participants who particularly enjoyed the interactive aspect *CLI-Tutor*. This was of particular interest to use given our a major intention of our research was to assess the effectiveness of interactive learning tools for the command line.

*"The cli-tutor made it really easy for me to learn how to use the command line in an uncomplicated way. The interactive part was the best part about the tool. It required me to directly put my understanding into practice, making the learning process very convenient."*

*its very nice. i think there is a lot of learning when u actually have to type in stuff than just passively reading. i know from myself than i need to write stuff down to learn, so beeing forced to type in stuff to get in another level helps me.*

Even when interactive styled learning was not the preference of the participant, as is the case with the participant below. They still were able to find some merits in the interactive approach.

*"i personally prefer to acquire knowledge first and then put it into practice on my own. but i think interactive learning is a very effective method for many people to learn new things. especially when it comes to something that triggers fear of contact (for example the cli, or git). the cli-tutor is a really cool tool, visually and in terms of content! the most important things are repeated more than once, and repetitions are very important for the learning effect."*

### 5.1.1 Relevant Research Questions

**RQ2** and **RQ3** are technically orientated research questions. Based on what we learnt in the feedback provided by participants in the user study we will now provide some insights into potential answers to these research questions.

**RQ2.** *How should an interactive learning tool be designed to mitigate the difficulty and intimidation factor of learning CLIs?*

Based on the feedback received during the user study, it can be said that in order to build effective interactive learning tools the structure and content of lessons are of high importance. The correct sequencing, building up and repetition of examples are key to creating an effective curriculum. Allowing room for exploration and experimentation are other important aspects of an effective interactive learning tool. To some participants the ability to directly try out the lesson content made for a stronger learning experience.

**RQ3.** *How can a 'forgiving' shell be implemented on top of an existing shell to enable the transition from learning to real-world usage?*

The approach taken by *CLI-Tutor* based on its reception seems to be promising. Creating an intermediary between the user and the shell allowed for tailoring the environment specifically to beginners while still being able to be faithful to the shell interaction experience. Wrapping the shell also enabled safety measures like not allowing the user into the sensitive root level directories, which contributes to the safety aspect and reduces the overall intimidation factor. The sandboxing approach takes this point and pushes it even further as then the users have no fears of any potential negative consequences to their systems.

## 5.2 User Study Results

### 5.2.1 Evaluation section

The following subsection, addresses research question **RQ4**: *Is the interactive tool more effective than text based learning methods?*

As mentioned in Chapter 4, the evaluation section of our user study is what we used to evaluate the effectiveness of the lessons our study participants took. Since the content of all the lessons was exactly the same across both interaction mediums this offers a reasonable way to focus our comparison on only the interaction mediums.

A question-by-question breakdown of the percentage of correct responses per question for both user groups is shown in Table 5.1. When averaged across all questions in the evaluation section, the interactive group performed 9.34% better than their non-interactive counterparts. When looking deeper at the differences in performance there was an observable trend. Individuals who took the interactive version of the tutor seemed to score significantly higher on questions that related to topics that contained lots of interactive examples in *CLI-Tutor*. This is especially true for questions relating to flags and the file system which consisted of many interactive examples.

For a visual representation of the differences between both participant groups in the evaluation section (see: Figure 5.1).

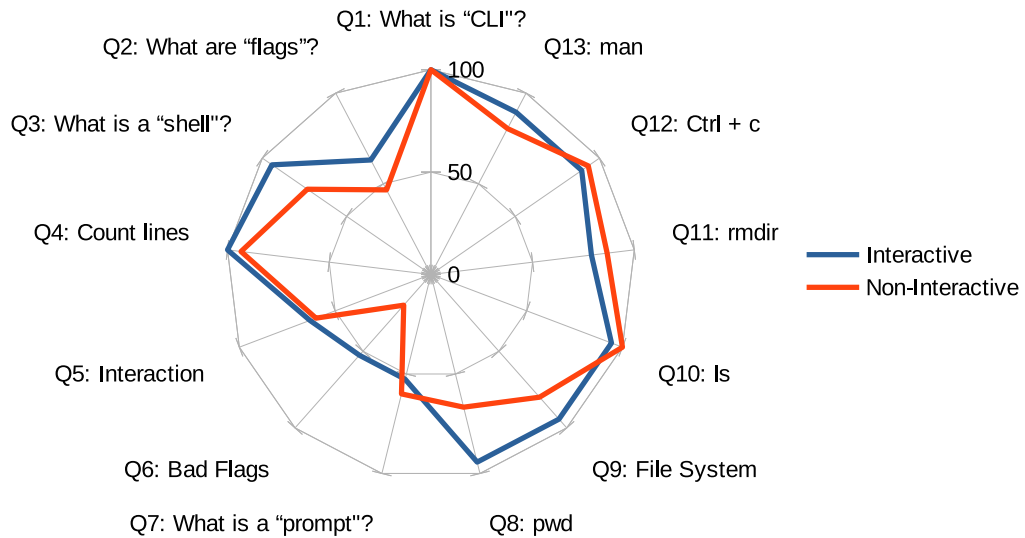
### 5.2.2 Discussion

Despite the positive results, the higher level of performance cannot be conclusively attributed to the interaction medium alone. There are factors such as existing experience, outliers and a small study size that must be considered when analysing these results. However, given that the performance of the interactive group was comparatively higher on more interactively practised topics in the tutor, there appears to be a correlation between practise and success when it comes to the command line.

Question	Interactive Correct %	Non Interactive Correct %
What does CLI stand for?	100.00%	100.00%
Which one of the following best describes the role of flags when issuing a command?	63.16%	46.67%
In your own words can you describe what the shell is?	94.28%	73.34%
How would you count the number of lines in a given file with the word count utility?	100.00%	93.34%
Which one of the following flow diagrams best describes textual interaction with the operating system	63.16%	60.00%
Which of the following is an incorrect usage of flags?	52.63%	20.00%
What is the role of the prompt?	52.63%	60.00%
What is the command to see where you are on your file system and what does the abbreviation stand for?	94.28%	66.67%
Which of the following structures describes the file system best?	94.28%	80.00%
What command do you use to list the contents of your current directory?	94.28%	100.00%
Can you explain in what situations the rmdir command will not delete a directory?	78.95%	86.67%
What shell keyboard shortcut cancels a command or input?	89.47%	93.34%
What is the name of the command that brings up documentation about a command? What does this command stand for?	89.47%	80.47%
<b>Overall Score</b>	<b>82.19%</b>	<b>72.85%</b>

**Table 5.1:** Summary of questions answered correctly by method during the evaluation phase.





**Figure 5.1:** Chart depicting the evaluation section results of the interactive and non-interactive participant groups.

### 5.2.3 Intimidation Factors

In the user study, participants were asked about intimidation factors regarding command line usage to better understand the difficulties faced by beginners in order to further improve *CLI-Tutor* and any other future research in the area of making command line interfaces more approachable.

**RQ1.** *Are there identifiable patterns of difficulty when it comes to adopting CLIs? Can the ‘intimidation factor’ be pinned down?*

Based on the results from the feedback sections of the user study, we can say based on the data that individuals who used the interactive *CLI-Tutor*, on average, left less intimidated than their non-interactive tutor counterparts. As shown in Figure 5.2, a mere 5.26% of respondents assigned to the interactive group reported feeling more intimidated after their experience with *CLI-Tutor*. On the non-interactive side this figure was considerably higher with 26.67% of respondents reporting being more intimidated by the command line after the user study.

In terms of identifiable patterns of difficulty and intimidation factors, participants were asked to identify their personal intimidation factors. Responses included fears about lack of safety mechanisms or unintentional errors.

■ "irreversible actions"

■ "something fatal happens that I didn't want to to (something like rm without wanting it/typos/...)"

One of the most often cited intimidation factors was related to the mental overhead of remem-

bering all the commands.

■ *"If you don't use it that often, the overwhelming number of commands makes it difficult to use."*

■ *remembering all the things to write in order to give the command correctly*

■ *difficult to memorise the command and flags, especially when I'm not using them frequently*

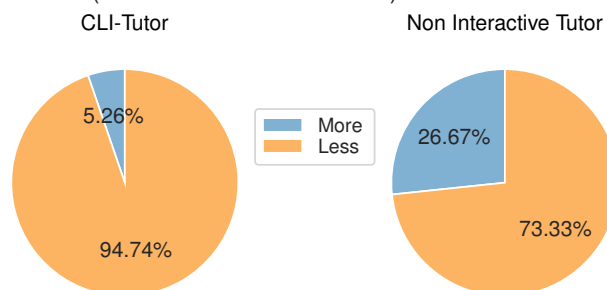
One participant even feared that the memorisation requirements might even nullify any potential productivity gains of using the command line.

■ *not knowing the commands by heart and needing to look it up (taking more time in the process)*

Additionally, there was also mention of the unfamiliarity with command line interfaces as an additional intimidation factor.

■ *unfamiliar setting, no overview*

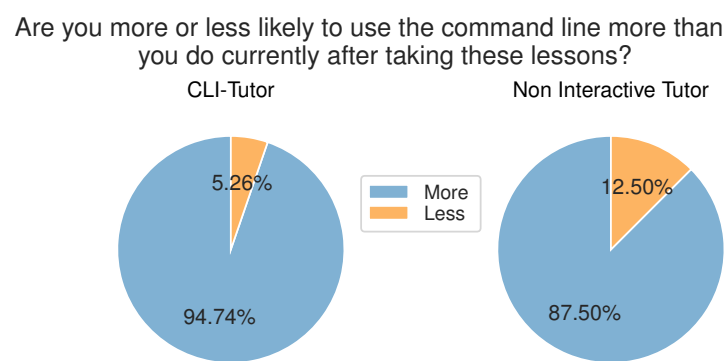
Do you feel more or less intimidated by the command line after this (interactive/non-interactive) tutor?



**Figure 5.2:** Charts comparing the post lesson intimidation levels between the two study groups.

**RQ5.** *Are novice CLI users more likely to continue using CLI interfaces after using such a tool?*

Participants were questioned during the feedback section of the user study whether their experience in this user study had any effect on their anticipated usage of CLIs going forward. The differences across interaction groups are more subtle compared to the previous comparison. In the interactive group, 94.74% (see: Figure 5.3) of participants indicated that they would use CLIs more in the future following their experience with *CLI-Tutor*. On the non-interactive side, this group was slightly smaller, but at 87.50% it still represented the majority of participants. While the differences across interaction paradigms are less pronounced in this result, it does indicate that the curriculum designed in this work targeted intimidation factors well and generally inspired confidence in users.



**Figure 5.3:** Charts comparing the post lesson future CLI usage impressions.



# Reflections and Related Work

## 6.1 Related Work

Interactive learning tools are not a new area of research in the software development field. Tutorial software for programming languages has existed and has been researched for decades [22–29].

Interactive tutorial tools in the software development space are primarily built for the purpose of introducing specific programming languages [26,28,30,31]. Pillay et al. laid out a framework for how to develop programming language tutors in 2003 [32]. However, Due to the recent growth of remote learning and remote working, asynchronous learning and computer-based learning is seeing a fresh surge in research. Not only are intelligent tutoring systems being developed and tested against traditional learning media but also against other asynchronous computer-based systems such as video lectures [33,34].

There are also a handful of interactive tutorial tools that are built primarily for the web. [26,30,35,35,36]. This category of tools often goes beyond the realm of programming languages and often serves as interactive elements in the official documentation of certain web technologies, such as with [35,37]. Another interesting aspect of these web-based tools is that they try to implement a sandboxed environment to make the tools more accessible and lower the barrier to use. This sandboxing has long been identified as a pragmatic design choice with research dating as far back as the 1985 work of Anderson et al. [22], *LISP TUTOR*, one of the earliest interactive programming tutors. In fact, the work of Anderson et al. highlights how access to tutoring tools may have the potential to greatly improve performance particularly for economically disadvantaged students, who may not have access to the same resources as their counterparts. Another more recent work looking into this sandboxing idea as a means of providing flexibility and safety to users is, [25].

## 6.2 Improvements

The user study and the associated feedback provided useful insights into how subsequent versions of *CLI-Tutor* could be improved. Some users suggested that adopting a more hybrid approach between the interactive and non-interactive tools could improve *CLI-Tutor*.

*It's a great tool, feels great to use. Personally i prefer guide books / video guides simply because going back and forth to look for desired info is easier than a more linear interactive approach but i can greatly appreciate such "learning games", and would love to see more of it*

*Still i prefer to be able to quickly look up something in a textbook / scroll back in a video, because it allows to connect information more intricately and more adjusted to my way of storing / recalling information.*

These suggestions could improve the utility of *CLI-Tutor* as a reference tool and allow for continued use of the tool beyond taking the first lesson. This hybrid documentation and interactive sandbox approach is a design style that is seeing an increase in popularity recently, especially in the web development documentation world. A very popular *JavaScript* framework, *React*<sup>1</sup> has recently released a beta version of their documentation that employs exactly this hybrid approach [37]. Another popular *JavaScript* framework, *Svelte*<sup>2</sup>, has also employed a similar hybrid approach for their introductory tutorials [35].

Other areas for potential improvement to *CLI-Tutor* include:

- *Dynamic feedback*: Integrating some form of feedback that dynamically adjusts according to the user's input or offers some intelligent suggestions. Such an approach would have the potential to offer more personalised and specific feedback than the approach in the current version of *CLI-Tutor*. [38–40] are all works targeting this domain of intelligent feedback generation.
- *Mobile version*: A mobile-friendly version of *CLI-Tutor*, particularly the web application, would have the potential of widening the potential user group and simultaneously reducing the barrier to entry for *CLI-Tutor*. To implement this, some attention to the sizing of elements in the web application would be necessary. Furthermore, a mobile-friendly version would necessitate the integration of a modified on-screen keyboard to ease interaction with the tool, as it is primarily keyboard driven.
- *Progress tracking*: The integration of some sort of progress tracking would improve the user experience of *CLI-Tutor*. This could open the doors to elements such as gamification or performance analytics and even more refined feedback to be integrated. Implementing statefulness in *CLI-Tutor* could potentially be achieved using lightweight client-side databases such as *SQLite*<sup>3</sup>.
- *Styling*: A more robust and user-modifiable styling configuration has the potential to make the interface of *CLI-Tutor* more inviting and cater better to the preferences of users. Finer control over the appearance also has the potential to make *CLI-Tutor* more accessible as colourblind-friendly modes, and other accessibility features such as font control could be integrated into the application.

## 6.3 Future Work

Building upon this work can provide valuable insights into how to make the command line more approachable and how to build better interactive learning software in general. The addition of some suggested improvements in the preceding chapter can open the doors to a wide array of future research in this domain. Furthermore, the user study in this work could benefit from reproduction with a larger number of participants. Another potential modification for future research is could be to perform a user study with absolute beginners, or individuals with no programming or command line experience.

---

<sup>1</sup><https://reactjs.org/>

<sup>2</sup><https://svelte.dev/>

<sup>3</sup><https://sqlite.org/>

---

Another avenue of research could be to look into more advanced topics and see if the benefits of an interactive learning environment still reflect when the subject matter is more complicated or targeted at more advanced users.





# Conclusion

In this work, we presented *CLI-Tutor*, an interactive tutorial tool for introducing the command line to beginners. We demonstrated that an interactive approach to teaching the concepts of command line interaction has the potential to be more effective than the traditional documentation-based approaches that exist. The *CLI-Tutor* succeeds in its intended goals of offering an interactive and low-barrier to access tutorial tool for the command line.

As a result of a user study comparing *CLI-Tutor* to a state-of-the-art documentation tool, we were able to produce encouraging results. Our findings indicate that tutorials that incorporate interactive elements have the potential to not only produce better learning results but also provide an engaging and safe environment for experimentation and exploration. The research performed as part of this work also brings to light some difficulties related specifically to the command line that could prove beneficial to future research in this area. Mitigation of these identified intimidation factors offers a framework for understanding the difficulties and issues regarding command line interfaces. This has the potential not only to produce better tutorials and tutorial software but can also be used to improve the design of command line interfaces in general. Finally, this work makes suggestions and outlines potential improvements for building future interactive learning tools for the command line and beyond.



---

# Bibliography

- [1] E. S. Raymond and R. W. Landley, *The art of unix usability*. Pearson Education, Inc, 2004, [Retrieved 02-Aug-2022]. [Online]. Available: <http://www.catb.org/esr/writings/taouu/taouu.html>
- [2] L. Pouzin, "The origin of the shell," [Retrieved 02-Aug-2022]. [Online]. Available: <https://multicians.org/shell.html>
- [3] J. R. Mashey, "Using a command language as a high-level programming language," in *Proceedings of the 2nd international conference on Software engineering*. Citeseer, 1976, pp. 169–176.
- [4] F. J. Corbató and V. A. Vyssotsky, "Introduction and overview of the multics system," in *Proceedings of the November 30–December 1, 1965, fall joint computer conference, part I*, 1965, pp. 185–196.
- [5] D. M. Ritchie and K. Thompson, "The unix time-sharing system," *Communications of the ACM*, vol. 17, no. 7, pp. 365–375, 1974.
- [6] S. Hultstrand and R. Olofsson, "Git-cli or gui: Which is most widely used and why?" 2015.
- [7] L. Takayama and E. Kandogan, "Trust as an underlying factor of system administrator interface choice," in *CHI'06 extended abstracts on Human factors in computing systems*, 2006, pp. 1391–1396.
- [8] J. Reimer, "A history of the gui," *Ars Technica*, vol. 5, pp. 1–17, 2005.
- [9] M. McIlroy, E. Pinson, and B. Tague, "Unix time-sharing system," *The Bell system technical journal*, vol. 57, no. 6, pp. 1899–1904, 1978.
- [10] D. Norman, "The next ui breakthrough: Command lines," *Interactions*, vol. 14, no. 3, p. 44–45, may 2007. [Online]. Available: <https://doi.org/10.1145/1242421.1242449>
- [11] M. C. Pierce, R. K. Ware, C. Smith, and B. Moolenaar, "vimtutor - the vim tutor," Nov 2019, [Retrieved 02-Aug-2022]. [Online]. Available: <https://github.com/vim/vim/blob/master/runtime/tutor/tutor>
- [12] Neovim Contributors, "Home - Neovim — neovim.io," [Retrieved 02-Aug-2022]. [Online]. Available: <https://neovim.io>
- [13] S. Francia, "Cobra.dev," [Retrieved 02-Apr-2022]. [Online]. Available: <https://cobra.dev/#concepts>

- [14] C. Ramey, B. Fox, and Open Source Contributors, "The gnu readline library," [Retrieved 02-Apr-2022]. [Online]. Available: <https://tiswww.case.edu/php/chet/readline/rltop.html>
- [15] Docker Inc, "Docker homepage," Aug 2022, [Retrieved 02-Aug-2022]. [Online]. Available: <https://www.docker.com/>
- [16] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.
- [17] D. Libes, *Exploring Expect: a Tcl-based toolkit for automating interactive programs*. " O'Reilly Media, Inc.", 1995.
- [18] GitHub Team and Open Source Contributors, "Github cli," [Retrieved 02-Aug-2022]. [Online]. Available: <https://cli.github.com/>
- [19] MkDocs Team and Open Source Contributors, "Mkdocs," [Retrieved 02-Aug-2022]. [Online]. Available: <https://www.mkdocs.org/>
- [20] Software Package Data Exchange (SPDX), "Mit license : Software package data exchange (spdx)," [Retrieved 02-Aug-2022]. [Online]. Available: <https://spdx.org/licenses/MIT.html>
- [21] C. Schmitz and LimeSurvey Team, "Limesurvey." [Online]. Available: <https://www.limesurvey.org/>
- [22] J. R. Anderson and B. J. Reiser, "The lisp tutor," *Byte*, vol. 10, no. 4, pp. 159–175, 1985.
- [23] J. R. Anderson and E. Skwarecki, "The automated tutoring of introductory computer programming," *Commun. ACM*, vol. 29, no. 9, p. 842–849, sep 1986. [Online]. Available: <https://doi.org/10.1145/6592.6593>
- [24] A. Gerdes, J. Jeuring, and B. Heeren, "An interactive functional programming tutor," in *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, 2012, pp. 250–255.
- [25] T. Permpool, S. Nalintippayawong, and K. Atcharyachanvanich, "Interactive sql learning tool with automated grading using mysql sandbox," in *2019 IEEE 6th International Conference on Industrial Engineering and Applications (ICIEA)*. IEEE, 2019, pp. 928–932.
- [26] C. Lee and M. S. Baba, "The intelligent web-based tutoring system using the c++ standard template library," *Malaysian Online Journal of Instructional Technology*, vol. 2, no. 3, pp. 34–42, 2005.
- [27] J. Jeuring, A. Gerdes, and B. Heeren, "A programming tutor for haskell," in *Central European Functional Programming School*. Springer, 2011, pp. 1–45.
- [28] J. Holland, A. Mitrovic, and B. Martin, "J-latte: a constraint-based tutor for java," in *Hong Kong: 17th International on Conference Computers in Education (ICCE 2009)*. University of Canterbury. Computer Science and Software Engineering, 2009, pp. 142–146.
- [29] S. Schez-Sobrinho, C. Gmez-Portes, D. Vallejo, C. Glez-Morcillo, and M. A. Redondo, "An intelligent tutoring system to facilitate the learning of programming through the usage of dynamic graphic visualizations," *Applied sciences*, vol. 10, no. 4, p. 1518, 2020.
- [30] C. Done, "Try haskell! an interactive tutorial in your browser," [Retrieved 04-May-2022]. [Online]. Available: <https://www.tryhaskell.org/>

- [31] A. Ajayi, E. Olajubu, D. Ninan, S. Akinboro, and H. Soriyan, "Development and testing of a graphical fortran learning tool for novice programmers," *Interdisciplinary Journal of Information, Knowledge, and Management*, vol. 5, 2010.
- [32] N. Pillay, "Developing intelligent programming tutors for novice programmers," *ACM SIGCSE Bulletin*, vol. 35, no. 2, pp. 78–82, 2003.
- [33] B. A. Becker and K. Quille, "50 years of cs1 at sigcse: A review of the evolution of introductory programming education research," in *Proceedings of the 50th acm technical symposium on computer science education*, 2019, pp. 338–344.
- [34] E. Ossovski and M. Brinkmeier, "Comparing video and interactive learning material styles for programming," in *EDULEARN22 Proceedings*. IATED, 2022, pp. 5381–5390.
- [35] R. Harris and Svelte Contributors, "Svelte tutorial," [Retrieved 04-May-2022]. [Online]. Available: <https://svelte.dev/tutorial/basics>
- [36] I. Herweijer, "Got 30 minutes? give ruby a shot right now!" [Retrieved 04-May-2022]. [Online]. Available: <https://try.ruby-lang.org/>
- [37] React Team Meta/Facebook, "Quick start," [Retrieved 04-May-2022]. [Online]. Available: <https://beta.reactjs.org/learn>
- [38] H. Keuning, B. Heeren, and J. Jeuring, "Strategy-based feedback in a programming tutor," in *Proceedings of the computer science education research conference*, 2014, pp. 43–54.
- [39] A. Gerdes, B. Heeren, J. Jeuring, and L. T. Van Binsbergen, "Ask-elle: an adaptable programming tutor for haskell giving automated feedback," *International Journal of Artificial Intelligence in Education*, vol. 27, no. 1, pp. 65–100, 2017.
- [40] K. Rivers and K. R. Koedinger, "Data-driven hint generation in vast solution spaces: a self-improving python programming tutor," *International Journal of Artificial Intelligence in Education*, vol. 27, no. 1, pp. 37–64, 2017.



# Appendix A: User Study Survey Questions

## 8.1 Survey Questions

### 8.1.1 User Familiarity Questions

- Please provide a name or identify yourself
  - *user textual input*
- How many years of programming experience do you have (if any)?
  - *user numerical input*
- Are you currently enrolled in or did you ever participate in a Computer Science degree or a similar computation/information theory degree at a University level?
  - No.
  - Yes, at a Bachelor's level.
  - Yes, at a Master's level.
  - Yes, at a Doctorate level.
  - Yes, but I did not complete my studies.
  - Other
- What is your preferred method of learning related to technical topics?
  - Books/ Online Documentation
  - Interactive Tutorials
  - Video Tutorials
  - University Lectures
  - Forums/ Online Groups  
(e.g. Discord Servers)
  - Other

- **How effective would you rate reading books and documentation as a learning medium for you?**

(1 is extremely ineffective and 5 is extremely effective. This includes online resources such as documentation, blogs and tutorials.)

- 1
- 2
- 3
- 4
- 5

- **Have you ever previously used an interactive learning resource?**

(An interactive learning resource must allow some sort of input and verification from the user during the learning process. Youtube or non interactive online video based courses do not satisfy this requirement.)

- Yes
- No

- **How comfortable would you describe yourself with command line applications?**

(1 = Very uncomfortable 5 = Extremely comfortable)

- 1
- 2
- 3
- 4
- 5

- **On average how often do you use command line applications or terminal based tools?**

- Never
- A few times a year
- At least once a year
- At least once a month
- Almost every day

- **On average, How often do you perform regular computing tasks such as file management from the command line?**

- Never
- A few times a year
- At least once a year
- At least once a month
- Almost every day

- **Are you interested in integrating the command line more into your day-to-day computer use? If so, why?**

- Yes
- No



## 8.1.2 Evaluation

- **What does CLI stand for?**
  - Command Line Interface
  - Command Line Interaction
  - Command Line Instrument
  - Cool Linux Interaction
- **Which one of the following best describes the role of flags when issuing a command?**
  - They modify a program's behaviour
  - They modify the input given to a program
  - They are the input to a program
  - They add functionality to a program
  - I don't know
- **In your own words can you describe what the shell is?**
  - *user textual input*
- **How would you count the number of lines in a given file with the word count utility?**  
(Please choose all that apply)
  - `wc -l file.txt`
  - `wc lines file.txt`
  - `wc file.txt --lines`
  - `wc file.txt`
  - `lines file.txt`
- **Which one of the following flow diagrams best describes textual interaction with the operating system using a shell?**
  - user <=> shell <=> os
  - user <=> operating system
  - user -> shell -> os -> user
  - user <=> shell
  - I don't know
- **Which of the following is an incorrect usage of flags?**
  - `wc --lineswords file.txt`
  - `wc -lw file.txt`
  - `wc --lines --words file.txt`
  - `wc file.txt --words -l`
  - I don't know
- **What is the role of the prompt?**

- *user textual input*
- **What is the command to see where you are on your file system, and what does the abbreviation stand for?**
  - *user textual input*
- **Which of the following structures describes the file system best?**
  - Tree
  - Folder
  - Database
  - Chain
  - I don't know
- **What command do you use to list the contents of your current directory?**
  - ls
  - wc
  - pwd
  - list
  - I don't know
- **Can you explain in what situations the rmdir command will not delete a directory?**
  - *user textual input*
- **What shell keyboard shortcut cancels a command or input?**
  - Control + c
  - Control + Alt + Delete
  - Escape
  - Alt + F4
  - I don't know
- **What is the name of the command that brings up documentation about a command? What does this command stand for?**
  - *user textual input*

### 8.1.3 Feedback

Feedback was provided by a combination of two choice (e.g. *More* or *Less*) questions with a comment field where participants could expand on their answers.

### Interactive Group

- Do you feel more or less intimidated by the command line after this interactive tutor?
  - More
  - Less
  - *user textual input*
- What intimidation factors or difficulties did you personally experience when it comes to using the command line?
  - *user textual input*
- Are you more or less likely to use the command line more than you do currently after using this interactive tutor application?
  - More
  - Less
  - *user textual input*
- What are your feelings regarding interactive learning tools after this experience?
  - *user textual input*
- Did you learn anything new? If so, what was the most interesting or useful thing you learnt
  - *user textual input*
- We would love to hear some of your general feedback or reflections about the cli-tutor and the interactive learning process?
  - *user textual input*

### Non-Interactive Group

- Do you feel more or less intimidated by the command line after taking these lessons?
  - More
  - Less
  - *user textual input*
- What intimidation factors or difficulties did you personally experience when it comes to using the command line?
  - *user textual input*
- Are you more or less likely to use the command line more than you do currently after taking these lessons?
  - More
  - Less
  - *user textual input*

- **What are your feelings regarding learning technical topics by reading documentation after this experience?**
  - *user textual input*
- **Do you think an interactive command line tutorial application would improve the learning process?**
  - Yes
  - No
  - It would make no difference
  - *user textual input*
- **Did you learn anything new? If so, what was the most interesting or useful thing you learnt**
  - *user textual input*
- **We would love to hear some of your general feedback or reflections about the cli-tutor and the interactive learning process?**
  - *user textual input*

## Non-Interactive Group

Users were also encouraged to optionally try the alternative tool to which they were assigned and asked to provide feedback.

- **Please provide some thoughts or feedback about your experience trying out the alternative version of the tutorial tool.**(e.g. Is there something you prefer or disfavour about one approach or the other? Is there a difference in levels of intimidation or difficulty with one approach or the other?)
  - *user textual input*

## 8.2 Parser

Here is the custom parser used for parsing lesson files into the *Lesson* and *Task* datastructures:

```

1 package main
2
3 func ParseLesson(content []byte) Lesson {
4     Lesson := new(Lesson)
5     parsed := goldmark.DefaultParser().Parse(text.NewReader(content))
6
7     ast.Walk(parsed, func(n ast.Node, entering bool) (ast.WalkStatus, error) {
8         s := ast.WalkStatus(ast.WalkContinue)
9         if !entering || n.Type() == ast.TypeDocument {
10             return ast.WalkContinue, nil
11         }
12         var err error
13
14         if n.Kind() == ast.KindHeading {
15             currentHeading := n.(*ast.Heading)
16             if currentHeading.Level == 1 {

```

```

17         Lesson.Name = string(currentHeading.Text(content))
18     }
19 }
20
21     if n.Kind() == ast.KindParagraph {
22         if n.PreviousSibling() != nil && n.PreviousSibling().Kind() ==
↪ ast.KindHeading {
23             parentHeading := n.PreviousSibling().(*ast.Heading)
24             if parentHeading.Level == 1 {
25                 lessonString := AssembleLines(n, content)
26                 Lesson.Description = lessonString
27             }
28             if parentHeading.Level == 2 {
29                 currentTask := new(Task)
30                 currentTask.Title = string(parentHeading.Text(content))
31                 taskString := AssembleLines(n, content)
32
33                 // Handles nested paragraphs
34                 if n.NextSibling() != nil && n.NextSibling().Kind() !=
↪ ast.KindHeading {
35                     for p := n; p.NextSibling() != nil && p.NextSibling().Kind()
↪ != ast.KindHeading; p = p.NextSibling() {
36                         if p.NextSibling().Kind() == ast.KindCodeBlock {
37                             currentTask.Expected = AssembleLines(p.NextSibling(),
↪ content)
38                             continue
39                         }
40                         if p.NextSibling().Kind() == ast.KindBlockquote {
41                             currentTask.Expected =
↪ AssembleLines(p.NextSibling().FirstChild(), content)
42                             continue
43                         }
44                         if p.NextSibling().Kind() == ast.KindFencedCodeBlock {
45                             taskString = taskString + "\n```\n" +
↪ AssembleLines(p.NextSibling(), content) + "\n```"
46                         } else {
47                             taskString = taskString + "\n\n" +
↪ AssembleLines(p.NextSibling(), content)
48                         }
49                     }
50                 }
51                 currentTask.Description = taskString
52                 Lesson.Tasks = append(Lesson.Tasks, *currentTask)
53             }
54             if parentHeading.Level == 3 {
55                 vocabularyString := n.Text(content)
56                 vocabulary := strings.Split(string(vocabularyString), ", ")
57                 Lesson.Vocabulary = vocabulary
58             }
59         }
60     }
61     return s, err
62 })
63 return *Lesson
64 }

```

Figure 8.1: Custom Markdown Parser

