Teaching Command Line and Git Skills Using Exercises with Interactive Visualizations

Ryan T. Buxton

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science & Applications

Sara Hooshangi, Chair
Margaret Ellis
Cliff Shaffer

November 30, 2022
Blacksburg, Virginia

Keywords: Interactive Visualizations, Git, Command Line, Computer Science Education

Teaching Command Line and Git Skills Using Exercises with Interactive Visualizations

Ryan T. Buxton

(ABSTRACT)

Command line and version control skills are vital to computer science students during their education and as they enter the software industry. These skills are commonly taught to undergraduate students via traditional lecturing methods and brief hands-on activities. Many students struggle with learning the Git version control system because they are not familiar with the command line, or they do not understand how Git works internally. Recent research highlights the effectiveness of using interactive visualizations to teach computer science concepts. Thus, we developed novel command line and Git exercises with interactive visualizations. These exercises integrate with learning management systems to automate grading. We tested the effectiveness of the exercises in a CS2 course at a large research institution by conducting pre-assessments before and post-assessments after the students completed the exercises. We found that students performed significantly better on both the command line and Git post-assessments than on the pre-assessments. Furthermore, we found that students with less experience with the command line and Git achieved a significantly greater improvement from the pre-assessments to the post-assessments. Additionally, we found that students with different levels of command line and Git experience did not perform differently on the exercises. Therefore, the exercises provide a novel tool for teaching command line and Git concepts to undergraduate computer science students with any level of command line and Git experience.

Teaching Command Line and Git Skills Using Exercises with Interactive Visualizations

Ryan T. Buxton

(GENERAL AUDIENCE ABSTRACT)

Command line is a term used to refer to a text-based user interface that allows users to interact with their computers by inputting commands. Git is a version control system typically used to track the stages of development for a computer program. Command line and Git skills are vital to computer science students during their education and as they enter the software industry. These skills are commonly taught to undergraduate students via traditional lecturing methods and brief hands-on activities. Many students struggle with Git because they are not familiar with the command line, or they do not understand how Git works internally. Recent research highlights the effectiveness of using interactive visualizations to teach computer science concepts. Thus, we developed novel command line and Git exercises with interactive visualizations. These exercises integrate with learning management systems to automate grading. We tested the effectiveness of the exercises in a CS2 course at a large research institution by conducting pre-assessments before and post-assessments after the students completed the exercises. We found that students performed significantly better on the post-assessments than on the pre-assessments. Furthermore, we found that students with less experience with the command line and Git achieved a significantly greater improvement from the pre-assessments to the post-assessments. Therefore, the exercises provide a novel tool for teaching command line and Git concepts to undergraduate computer science students with any level of command line and Git experience.

Acknowledgments

Thank you to my advisor, Dr. Sara Hooshangi, for her support and guidance. I could not have asked for a better advisor.

Thank you to Margaret Ellis for her support and guidance and for serving on my committee. Her hands-on support has helped my work tremendously.

Thank you to Dr. Shaffer for his support and guidance and for serving on my committee.

Special thanks to my parents, Stephen and Julie Buxton; my brother, Tyler Buxton; and my friends who have always supported me. They have shaped who I am today and motivate me to be the best version of myself.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The command line is a text-based user interface that allows users to interact with their computers by inputting commands. Git is a popular distributed version control system used throughout academia and industry that enables users to track changes to their files and collaborate with other users. Computer science students use command line and Git skills in courses, internships, personal projects, undergraduate research, and competitions [26]. Furthermore, command line and Git skills are vital in the software industry. Many students struggle with Git because they do not understand the command line and the internal structure of Git. The goal of this project was to create a novel tool for teaching command line and Git skills that visually showed students how commands affect the file system and the internal Git structure.

1.2 Objective

The main objective of this thesis is to explore the problem of effectively teaching command line and Git concepts to students. We attempted to achieve this objective by developing novel command line and Git exercises with interactive visualizations that integrate with learning management systems using OpenDSA [16]. The exercises were then deployed in the

classroom to assess their effectiveness on student learning.

1.3 Research Questions

The main research questions of this thesis are as follows:

- **RQ1**: Are exercises with interactive visualizations an effective tool for teaching core command line and Git concepts to students?
- **RQ2**: Do students with less command line and Git experience exhibit greater knowledge gain from completing the exercises?
- **RQ3**: Do the exercises reduce the gap in command line and Git knowledge for students with all levels of command line and Git experience?
- **RQ4**: Are the exercises feasible for students with any level of command line and Git experience?

1.4 Major Contributions

The major contributions of this thesis are as follows:

- Developed novel command line exercises with interactive visualizations that integrate with learning management systems using OpenDSA to automate grading. The command line exercises can be accessed at the following URL: http://lti.cs.vt.edu/LTI_ruby/Books/CommandLine/.

- Developed novel Git exercises with interactive visualizations that integrate with learning management systems using OpenDSA to automate grading. The Git exercises can be accessed at the following URL: http://lti.cs.vt.edu/LTI_ruby/Books/Git/.
- Collected and analyzed data to test the effectiveness of the command line exercises.
- Collected and analyzed data to test the effectiveness of the Git exercises.
- Developed novel command line interface within OpenDSA that can be extended to future interactive visualizations that require a command line interface.

1.5 Structure of Thesis

Chapter 2 covers background information, explores relevant research, and discusses some existing command line and Git tutorials. Chapter 3 describes all the development work involved with creating the command line and Git exercises with interactive visualizations. Chapter 4 describes the testing of the exercises in a Computer Science Problem Solving course at Virginia Tech. Chapter 5 describes the results from the testing including all the statistical analysis. Chapter 6 describes potential threats to validity. Chapter 7 concludes the thesis and discusses the direction of future work.

Chapter 2

Background

2.1 Command Line Interface

A command line interface is a text-based user interface that allows users to interact with their computer by inputting commands. A command line interface is commonly used to manipulate the file system and run programs. For example, the commands used in version control systems such as Git, as defined in the next section, are usually run via a command line interface.

2.2 Git

Git [7] is a popular distributed version control system used throughout academia and industry that enables users to track changes to their files. When used with a hosting service such as GitHub [1] or GitLab [6], Git can be used to manage remote files and enable collaboration among users. One key concept within Git is a Git repository. A Git repository is simply a collection of files, which are tracked by Git. Files within a Git repository have three potential states: modified, staged, or committed. A Git repository contains three main areas, which correspond respectively with the states: the working tree, the staging area, and committed files. If a file is changed in any way, it is considered modified and is thus located in the working tree. A user can then transition the file from modified to staged, moving it from the

working tree to the staging area. The file becomes committed once the user commits the file from the staging area. A commit in Git is a snapshot of the state of the files in a repository at the time of the commit. Git stores commits in a commit tree, which is a directed acyclic graph. Every time a new commit is created, the new commit is added to the commit tree. The commit tree provides a history of all the changes made to the repository. Git repositories also contain branches. A branch is simply a pointer to a commit on the commit tree. This allows different histories of the repository to be maintained. Only one branch can be active at any given time. The active branch is known as the HEAD branch. When commits are made, they are added to the HEAD branch. Local commits can be applied to the remote repository via a push in Git. This allows local changes to files to be applied to the remote files. Similarly, remote commits can be applied to a local repository using a pull in Git. This allows changes stored on the remote server to be accessed on the local machine.

2.3 Importance of Command Line and Git Skills

Software engineers use the command line and Git every day. Stack Overflow [4] interviewed 71,379 of its users including professional software developers and people learning to code. They found that of the 53,374 professional software developers surveyed, 96.65% use Git while only 1.38% use no version control system [17]. This result highlights the importance of knowing Git for professional software development. They also found that of 6,157 people learning to code, 81.87% use Git and 17.18% don't use a version control system [17]. This result indicates that most people learning to code also use Git or another version control system, although less than professional software developers. This suggests the need to teach Git and version control to people who are learning to code in order to prepare them for professional software development.

Hooshangi, Buxton, and Ellis [26] explored the usage of command line and Git skills by 341 senior undergraduate computer science students. They found that the following percentages of students used command line skills in each activity: 91% in courses, 65% in internships, 68% in personal projects, 24% in undergraduate research, and 24% in competitions. Similarly, they found that the following percentages of students used Git skills in each activity: 88% in courses, 60% in internships, 62% in personal projects, 23% in undergraduate research, and 21% in competitions. Thus, this research shows that command line and Git skills are important for computer science students academically and for their future careers. Hence, it is important to effectively teach command line and Git skills to computer science students early in their curriculum to prepare them for their future endeavors.

Isomöttönen and Cochez [27] conducted a survey of 21 undergraduate computer science students to explore the challenges students encounter as they learn Git. They found that 100% of the students found Git to be useful, but many students ran into difficulties with Git. One difficulty they found was that students struggled to understand some Git commands because they did not have a strong grasp of the underlying structure of Git. Another difficulty they found was that students with little command line experience struggled with the command line interface aspect of running Git commands. Additionally, students with little command line experience struggled to understand the difference between bash commands used for normal file manipulation and Git commands. The researchers conclude that teaching Git may require more than a simple practice-first approach if students are to understand the underlying structure of Git. This research highlights the importance of teaching command line skills before Git and teaching Git in a way that emphasizes how Git works internally.

Haaren and Lehtinen [23] integrated Git in a computer science course of approximately 200 students. They used Git for the distribution of course materials and for the submission of assignments. At the end of the course, they surveyed the students and found that the

majority of students had a positive experience with using Git in the course. The survey responses highlighted the importance of Git in the software industry, as some students mentioned how they use Git at work. Still, many students expressed difficulty with learning Git and requested more exercises teaching Git before being asked to use Git in practice. Furthermore, they found that some students struggled with Git because they struggled with the command line interface. The results from this study highlight the importance of Git in the professional software industry and show the need for command line skills to use Git effectively. Furthermore, the research suggests a need for better methods of teaching Git before asking students to use it.

Eraslan et al. [21] conducted a study in which students in a software engineering course used Git to work collaboratively. The researchers gathered observations from the course's instructors to create a list of common errors and poor practices for the students with using Git. Many of the errors originated from students not understanding the underlying structure of Git, especially with regards to branches and commits. Thus, this research highlights the importance of teaching Git in a way that exposes the internal structure of Git.

Milliken, Nguyen, and Steeves [28] studied the experiences of researchers using Git through focus groups, a survey, and in-depth one-hour interviews. The focus groups had 12 participants, the survey had 371 respondents, and the interviews had 41 participants. The participants for all three groups included students and faculty who had varying levels of experience with Git. From the focus groups and in-depth interviews, they found that many people struggle to create a mental model of Git commands. Participants claimed that they struggle to mentally visualize or conceptualize what is happening when they run Git commands. This research further suggests the need to teach Git in a way that explains the internal structure of Git and helps with the construction of a visual mental model.

Overall, the research done with regards to teaching Git to students and integrating Git

in classrooms has found that many students struggle to learn Git because they do not have adequate command line interface experience, or they cannot mentally visualize the underlying structure of Git. Thus, in developing a tool to teach Git, it is important to first familiarize the students with the command line. Then, the tool must teach the students how the underlying Git structure is affected as Git commands are executed.

2.4 Interactive Visualizations

Recent literature has shown the effectiveness of using interactive visualizations to teach computer science concepts. Mohammed, Shaffer, and Rodger [29] developed an eTextbook containing exercises with interactive visualizations to teach formal languages and automata concepts. The interactive visualizations provide feedback and grade the students as they complete the exercises. The researchers implemented their eTextbook in a computer science course at Virginia Tech over three different semesters for a total of 271 students. They found that 83% of students found the exercises useful, and the exam scores for students who used the exercises with interactive visualizations were significantly higher than the control group students for questions about topics that were taught in the exercises. Similarly, Farghally et al. [22] developed visualizations that teach algorithm analysis concepts. They implemented the visualizations in a data structures and algorithms course at Virginia Tech with 155 students in the intervention group, and 67 students in the control group. They found that 83% of the students found the visualizations useful, and the exam scores for the students who used the visualizations were significantly higher for questions about topics that were taught in the exercises as compared to the control group. Furthermore, Hamouda et al. [24] developed an interactive tutorial for teaching recursion called RecurTutor. They implemented the tutorial in a CS2 course at Virginia Tech with 168 students in the intervention group and

372 students in the control group. They found that students who used the tutorial spent more time studying recursion and performed significantly better on exam questions about recursion than the students in the control group. Moreover, students who used RecurTutor had greater confidence in their recursion skills. In a similar paper, Hamouda et al. [25] developed an interactive tutorial for teaching recursion in binary trees called BTRecurTutor. They implemented BTRecurTutor in a DSA course at Virginia Tech with 164 students in the control group, 192 students in the first intervention group, and 176 students in the second intervention group. The difference between the two intervention groups is that the second group had a semantical analysis feature with BTRecurTutor that the first group did not have. Both intervention groups performed significantly better than the control group on the final exam question on the topics covered in BTRecurTutor. The second intervention group also performed significantly better than the first intervention group. Additionally, Wang et al. [32] developed an interactive visualization tool that teaches convolutional neural networks. They performed a qualitative study in which they interviewed computer science students. They found that their tool helped users understand the internal structure of convolutional neural networks while being engaging and enjoyable for the user.

Therefore, current research overwhelmingly supports that interactive visualizations are highly effective at teaching computer science concepts to students.

2.5 Existing Tutorials

Many different tutorials currently exist for teaching command line and Git concepts. We list some of the more widely used tutorials below.

2.5.1 Command Line

Many existing command line tutorials are primarily textual. For example, Mozilla [3] provides a textual command line tutorial with some images. These tutorials contain valuable information, but as discussed in Section 2.4, tutorials without interactive aspects are less effective than interactive tutorials.

Additionally, many video command line tutorials exist on sites such as YouTube [5] and LinkedIn Learning [2]. These tutorials also provide valuable information but do not provide the hands-on experience needed to truly learn the command line. Furthermore, LinkedIn Learning videos are hidden behind a paywall that restricts many learners from being able to view them.

Code Academy [19] provides an interactive tutorial for the command line. The tutorial includes multiple lessons. Each lesson contains a description of one or more commands and provides a task for the user to complete. The user must then type commands in the provided command line interface to complete the exercise. Some lessons also provide a visualization of the file system to help users complete the exercises. However, the visualization does not update as users create or reorganize files. The Code Academy tutorial is also hidden behind a paywall that restricts many students from being able to access the tutorial.

A website called Command Challenge [10] provides an interactive tutorial for the command line. Figure 2.1 shows an exercise from Command Challenge. Their tutorial provides valuable hands-on command line experience by providing a task and a command line interface that allows users to run commands to complete the task. One limitation of this tutorial is that it does not contain a visualization of the file system.

Furthermore, a website called OverTheWire has a command line tutorial called Bandit [9]. Bandit requires users to access their servers via Secure Shell (SSH). Users then run commands

to navigate the remote file system and find secret keys to complete various levels. This website has the unique approach of allowing users to use their own local command line interfaces to complete the tutorial. Like Command Challenge, the main limitation of this tutorial is that it does not contain a visualization of the file system. Another limitation is that the Bandit servers cannot handle a large number of users synchronously, so a large computer science course cannot use Bandit in its coursework.

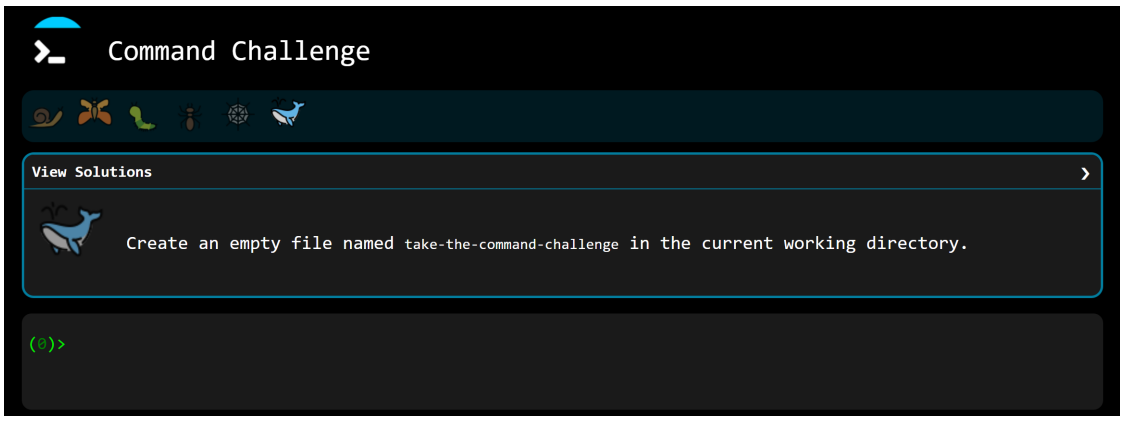


Figure 2.1: Command Challenge Command Line Tutorial

Moreover, game-based command line tutorials exist including Terminus, which was created by students at MIT [15]. As shown in Figure 2.2, Terminus has users explore a game world using commands. Users navigate the world using *cd*, look around using *ls*, interact with items using *less*, and check their current location using *pwd*. Terminus could be effective for learning how to navigate around directories using the command line, but the number of commands learned overall is limited. Furthermore, the only visualization is the image of the current location, which will most likely not help users learn command line skills.

Overall, the existing command line tutorials provide valuable information but could all be improved in some ways. Many of the tutorials do not include visualizations and none of the existing visualizations are interactive. Furthermore, many of the most complete tutorials are hidden behind paywalls. Additionally, none of these tutorials integrate with learning

management systems such as Canvas or Blackboard, and as a result, instructors will have a hard time grading students' completion of the tutorials, making them less attractive for faculty to use in their courses.

Figure 2.2: Terminus Command Line Tutorial

2.5.2 Git

As with the command line tutorials, many existing Git tutorials are primarily textual. For example, Git [8] provides a textual Git tutorial. These tutorials contain valuable information, but as discussed in Section 2.4, tutorials without interactive aspects are less effective than interactive tutorials.

Additionally, many video Git tutorials exist on sites such as YouTube [20] and LinkedIn Learning [14]. These tutorials also provide valuable information but do not provide the hands-on experience needed to truly learn Git. Furthermore, LinkedIn Learning videos are hidden behind a paywall that restricts many people from being able to view them.

Code Academy [11] provides an interactive tutorial for Git. The tutorial includes multiple lessons. Each lesson contains a description of one or more Git commands and provides a

task for the user to complete. The user must then type Git commands in the provided command line interface to complete the exercise. The lessons do not contain visualizations of the internal structure of Git. The Code Academy tutorial is also hidden behind a paywall that restricts many people from being able to access the tutorial.

Learn Git Branching [30] is an open-source project that provides Git exercises with interactive visualizations. Figure 2.3 shows an example of an exercise from Learn Git Branching. The panel on the left describes a challenge for the user to complete by calling git commands using the command line interface at the bottom of the panel. On the right, it visualizes the commit tree and the branches. The website has many exercises and visualizes many commands within Git. However, the major limitation of the tutorial is that it does not include file manipulation. Thus, users cannot get experience with staging files. It is important for new users of Git to understand the staging area and the relationship between changes to files and commits. This tutorial does not establish that commits contain information about changes to files.

Visualizing Git [18] is another open-source project that follows the same structure and visualization style as Learn Git Branching as shown in Figure 2.4. Thus, Visualizing Git has the same limitations as Learn Git Branching in terms of the visualization. Visualizing Git's other limitation is that it does not contain independent exercises; instead, a user can freely run Git commands and see how they affect the Git commit tree. This makes it difficult for new Git users to learn using this website because they may need guided learning.

A Grip on Git [31] is another online Git tutorial that uses visualizations. Figure 2.5 shows one part of the tutorial. On the left side of the screen, the tutorial describes a Git command. As the user scrolls down the page, the box on the left changes to describe different commands. At the same time, the visualization on the right updates to show the change to the Git commit tree as the commands are executed. This tutorial provides valuable information

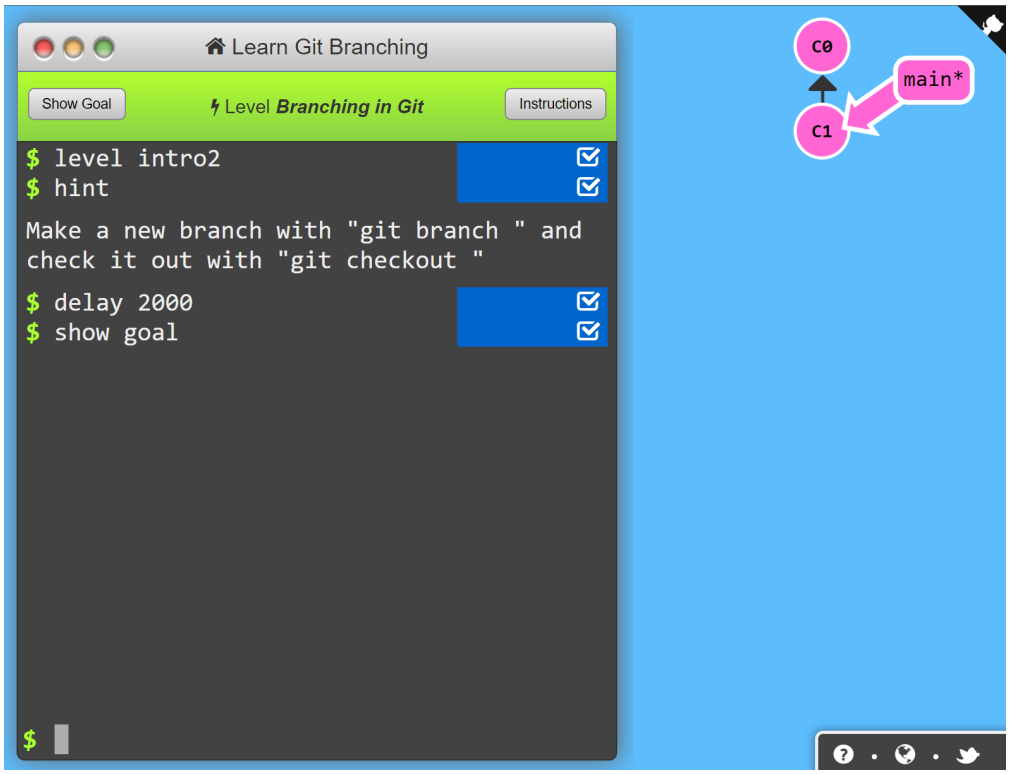


Figure 2.3: Learn Git Branching Tutorial

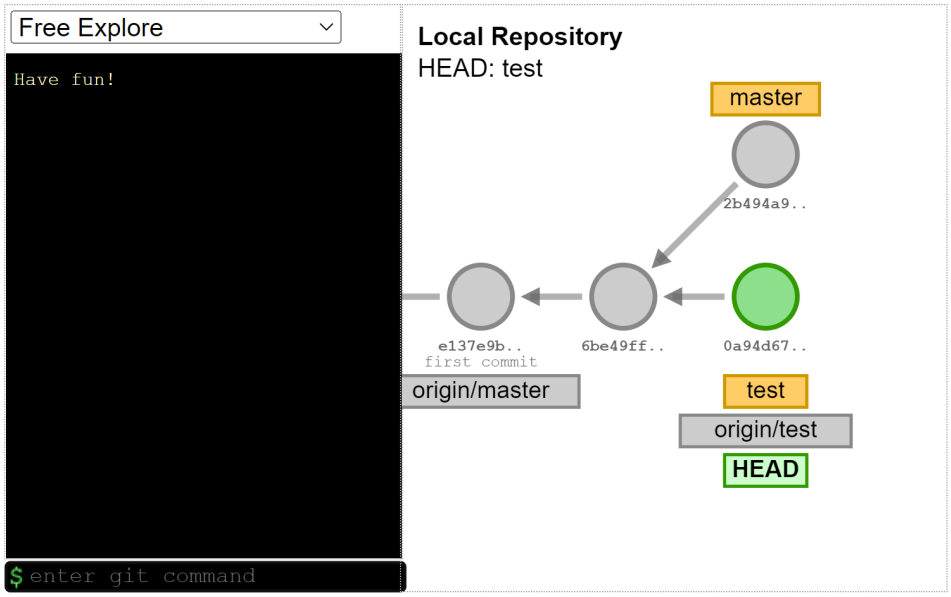


Figure 2.4: Visualizing Git Tutorial

about Git. However, it does not have an interactive component, and the users do not get hands-on experience with trying git commands themselves. Furthermore, the visualization has the same limitations as Learn Git Branching and Visualizing Git because it does not include file manipulation.

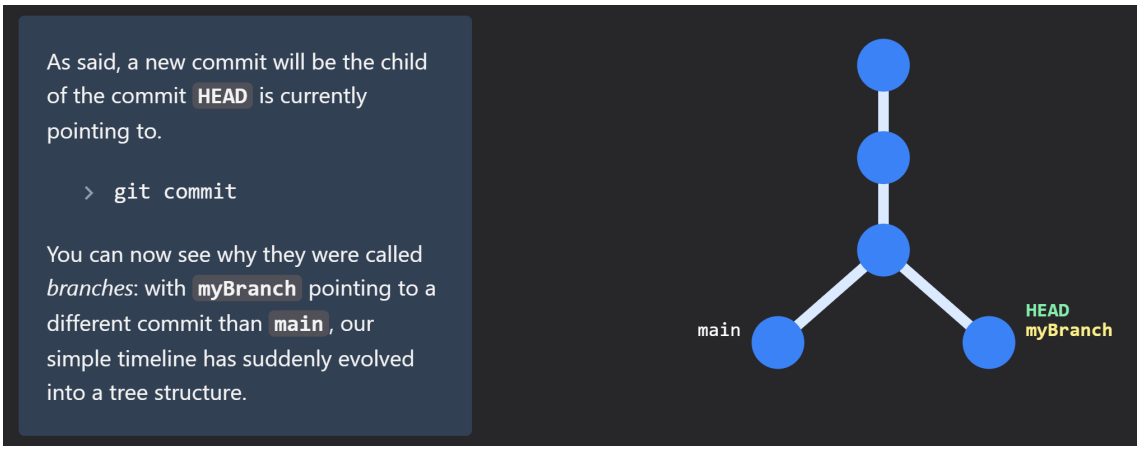


Figure 2.5: A Grip on Git Tutorial

Overall, the existing Git tutorials provide valuable information but could all be improved in some manner. Many of the tutorials either do not include visualizations, do not have interactive visualizations, or do not visualize file manipulation and the relationship between changes to files and commits. Furthermore, many of the most complete tutorials are hidden behind paywalls. Additionally, none of these tutorials integrate with learning management systems such as Canvas or Blackboard, and as a result, instructors will have a hard time grading students' completion of the tutorials, making them less attractive for faculty to use in their courses.

2.6 OpenDSA

OpenDSA [16] is an infrastructure that allows developers to create eTextbooks and deliver the content directly to students via learning management systems such as Canvas, Moodle,

and Blackboard. OpenDSA allows for iFrames to be embedded directly inside the eTextbooks. This allows developers to create content using Javascript, HTML, and CSS. Thus, interactive visualizations and exercises can be created within these Iframes. OpenDSA also provides functionality for grading students as they complete tasks inside the iFrames. Because OpenDSA follows the Learning Tools Interoperability (LTI) protocol, the grading integrates seamlessly with Canvas, Moodle, and Blackboard. Therefore, OpenDSA provides the infrastructure needed to create exercises with interactive visualizations that are automatically graded in Canvas as students complete the exercises.

2.7 Summary

Command line and Git skills are highly important for computer science students during their education and as they enter the software industry. Current methods of teaching Git to undergraduate computer science students commonly involve brief lecture slides, and then students learn through practice. As a result, many students struggle with Git concepts because they are not familiar with the command line, and they do not understand how Git works internally. Recent research has shown that interactive visualizations are effective for teaching computer science skills to students. Existing command line and Git tutorials do not contain sufficient interactive visualizations and do not integrate with learning management systems. OpenDSA provides the infrastructure for creating exercises with interactive visualizations that integrate with learning management systems. Therefore, the development of novel command line and Git exercises with interactive visualizations can be developed using OpenDSA to greatly improve the teaching of command line and Git skills in computer science courses.

Chapter 3

Development

3.1 Technologies Used

We developed the command line and Git exercises within the OpenDSA infrastructure using JavaScript, HTML, and CSS. We used a JavaScript library called jQuery [13] to manipulate the HTML document and handle events such as button clicks and key presses. Additionally, we used a JavaScript library called D3.js [12] to create the interactive visualizations. D3.js provides the functionality for binding data to objects within an SVG element to allow for smooth transitions as the data changes.

3.2 Browser-Based Command Line Interface

We began development by creating a browser-based command line interface that allows users to input commands. The command line interface uses an HTML input field to allow users to input commands. When the user presses the enter key while focused on the input field, the values from the command line are presented above the input as uneditable text. The input field is then cleared, and the user can begin typing their next command. Figure 3.1 shows the browser-based command line interface after the user has inputted two commands: *ls* and *touch test.txt*. Our browser-based command line interface allows custom handler methods to be defined for each command. This allows for easy customization of the command line

interface for different types of exercises and allows custom events to be captured to enable the interactive visualizations.

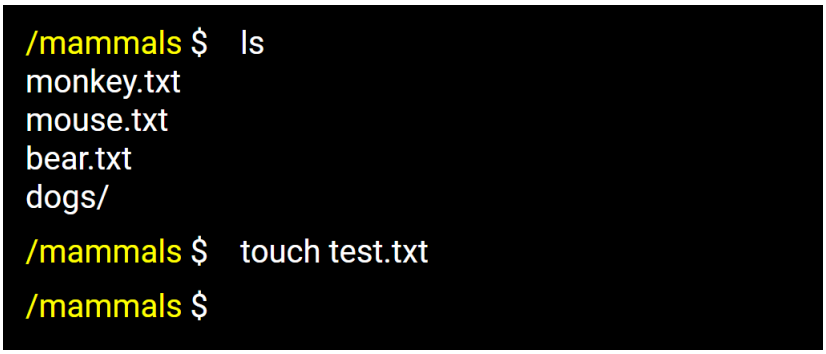


Figure 3.1: Browser-Based Command Line Interface

The prompt for the command line interface is also configurable. For example, Figure 3.2 shows an instance of the browser-based command line interface in which the prompt has been customized for Git to include the current branch.

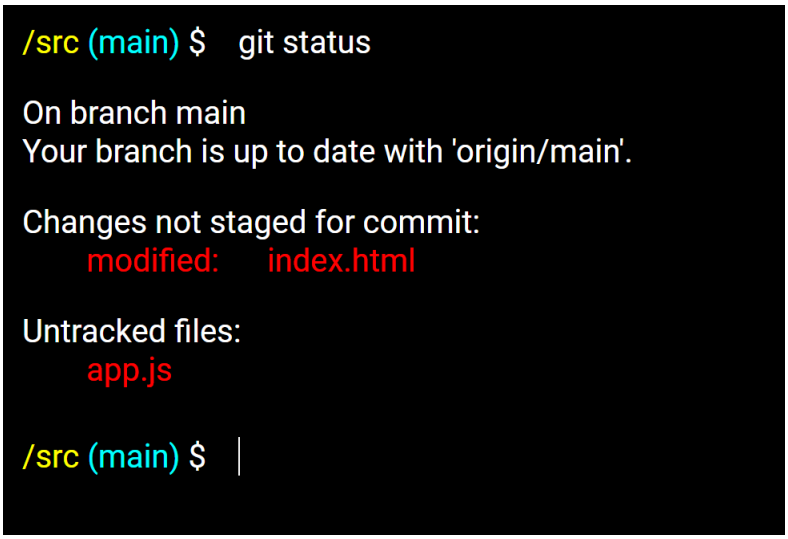


Figure 3.2: Browser-Based Command Line Interface with Git Prompt

3.3 File System Visualization

We continued the development by creating an interactive visualization of a file system. Figure 3.3 shows an instance of the interactive file system visualization. In the visualization, the dark blue rectangles represent directories, and the light blue rectangles represent files. The green rectangle represents the current working directory. The gray lines indicate the hierarchy of the file system. For example, in Figure 3.3, the root directory is represented by the dark blue rectangle containing */*. The root directory then contains four entities: three files named *bird.txt*, *snake.txt*, and *fish.txt* and one directory named *mammals*. The *mammals* directory then contains four entities: three files named *monkey.txt*, *mouse.txt*, and *bear.txt* and one directory named *dogs*. Finally, the *dogs* directory contains three files named *beagle.txt*, *boxer.txt*, and *poodle.txt*.

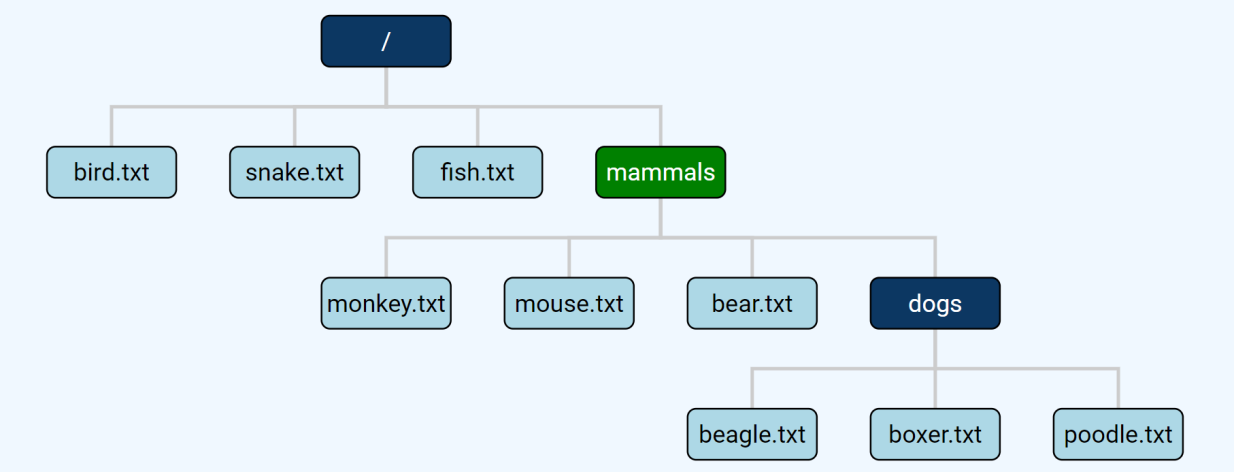


Figure 3.3: File System Visualization

The interactive file system visualization includes transitions to visualize files and directories being created, deleted, or moved. If a file or directory is created, a new rectangle fades into the visualization. If a file or directory is removed, the corresponding rectangle fades out of the visualization. If a file or directory is moved to a new location, the corresponding rectangle smoothly slides to the new location. While these changes occur, the entirety of

the file system visualization reorganizes itself as needed to account for the changes. All of these transitions work together to provide a visualization of the file system as commands are executed.

3.4 Command Line Exercise Component

Using the browser-based command line interface and the interactive file system visualization, we developed the command line exercise component. Figure 3.4 shows an example of a command line exercise.

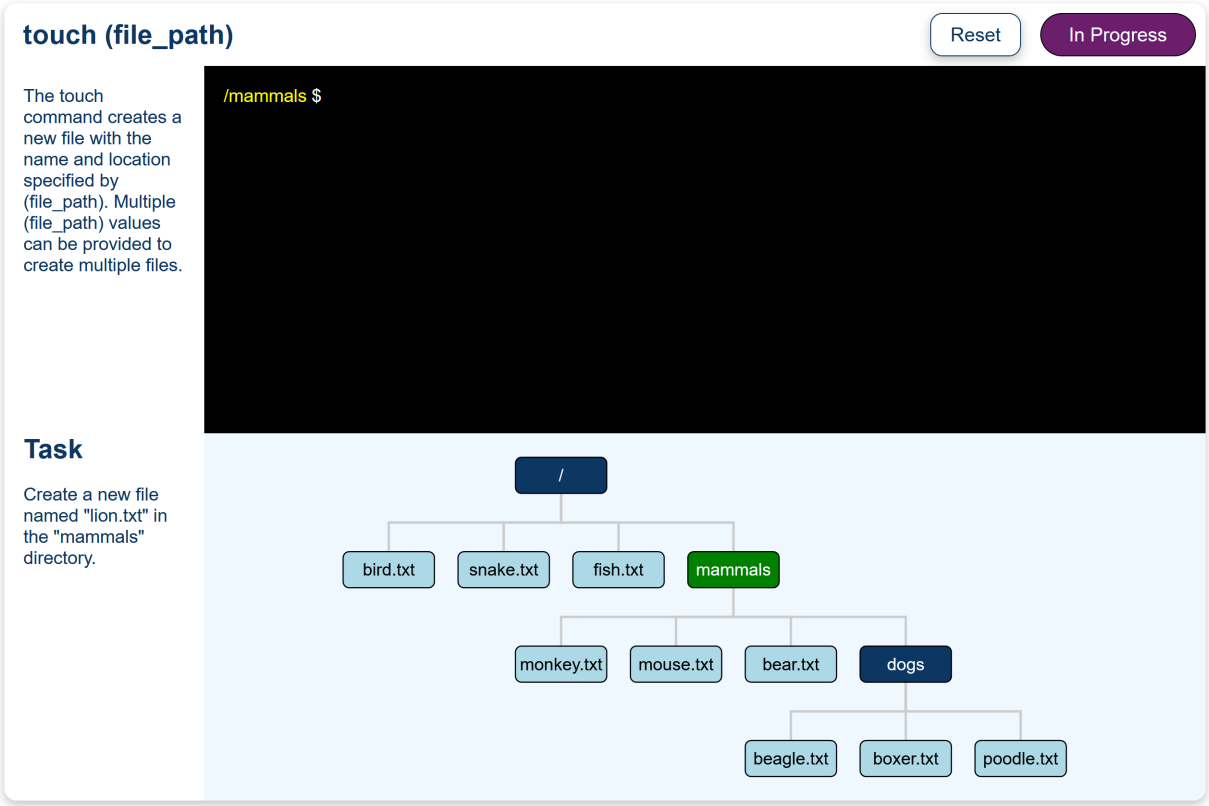


Figure 3.4: Command Line Exercise

Each command line exercise consists of a description, a task, a browser-based command line

interface, and an interactive file system visualization. The description provides some of the information needed to complete the exercise. In most exercises, the description contains a command, its arguments, and a description of the command. The task describes how to complete the exercise. The browser-based command line interface allows the users to input the commands needed to complete the exercise. The browser-based command line interface provides the same functionality for all exercises, meaning the users can execute any of the supported commands as they complete the exercises. Each command line exercise supports the following commands with arguments surrounded by (parentheses) and optional flags surrounded by [brackets]:

- **pwd**: Print the path of the current working directory.
- **ls (directory_path)**: List all files and directories in the current working directory if (directory_path) is not provided. Otherwise, list all files and directories in the directory at the location specified by (directory_path).
- **cd (directory_path)**: Change the current working directory to the directory at the location specified by (directory_path).
- **touch (file_path)**: Create a new file with the name and location specified by (file_path). Provide multiple (file_path) values separated by space characters to create multiple files.
- **mkdir (directory_path)**: Create a new directory with the name and location specified by (directory_path). Provide multiple (directory_path) values separated by space characters to create multiple directories.
- **rm [-r] (path)**: Remove the file or directory at the location specified by (path). Provide multiple (path) values separated by space characters to remove multiple files

or directories. Provide the [-r] flag to remove directories. A directory cannot be removed if the current working directory is a subdirectory of the directory.

- **rmdir (directory_path)**: Remove the directory at the location specified by (directory_path) if the directory is empty. Provide multiple (directory_path) values separated by space characters to remove multiple directories.

- **mv (src_path) (dst_path)**: Move the file or directory from the location specified by (src_path) to the file or directory specified by (dst_path). Provide multiple (src_path) values separated by space characters to move multiple files or directories.

- **cp [-r] (src_path) (dst_path)**: Copy the file or directory from the location specified by (src_path) to the file or directory specified by (dst_path). Provide multiple (src_path) values separated by space characters to copy multiple files or directories. Provide the [-r] flag to copy directories.

As the users execute commands, the interactive file system visualization updates accordingly. This helps the user understand the impact of the commands on the file system. Every time the user executes a command, the exercise checks the state of the file system to determine if the user has completed the exercise. This is done using a custom method defined with each exercise. This method is run after each command execution and has access to the state of the file system and the most recent command. If this method returns true, then the exercise has been completed. Once the exercise has been completed, the purple *In Progress* indicator at the top of the exercise changes to a green *Completed* indicator. The description, task, initial file system state, and desired file system state are configurable so that new command line exercises can be created easily.

3.5 Command Line Exercises

Using the configurable command line exercise component, we developed 17 command line exercises. The command line exercises were designed to teach many basic commands that are essential for using any command line interface effectively. Most exercises focus on teaching one command. However, the last three exercises are challenge exercises that require multiple commands to be completed. Table 3.1 shows the topics focused on by each command line exercise. With the command line exercises, we created an About page that explains what the command line is, how paths work, and how the exercises work. The About page also lists key command line terms and all the commands supported by the command line exercises.

Table 3.1: Command Line Exercises Topics

Exercise	Topic
Exercise 1	pwd
Exercise 2	pwd in a different current working directory
Exercise 3	pwd in a different current working directory
Exercise 4	ls
Exercise 5	cd
Exercise 6	cd to parent directory
Exercise 7	cd with complex path to target directory
Exercise 8	touch
Exercise 9	mkdir
Exercise 10	rm
Exercise 11	rm with -r flag
Exercise 12	rmdir
Exercise 13	mv
Exercise 14	cp
Exercise 15	mkdir and touch for multiple files
Exercise 16	mkdir and mv for multiple files
Exercise 17	cp with the -r flag and rm for multiple files

3.6 Git Visualization

Next, we developed an interactive Git visualization. Figure 3.5 shows an example of an interactive Git visualization. On the left, the visualization shows the local Git repository. On the right, the visualization depicts the remote Git repository. Each repository contains an interactive file system visualization and an interactive Git tree visualization. As in the interactive file system visualization used for command line, the dark blue rectangles on the top half of the visualization represent directories, and the light blue rectangles represent files. The purple rectangle in the top half of the visualization represents the current working directory.

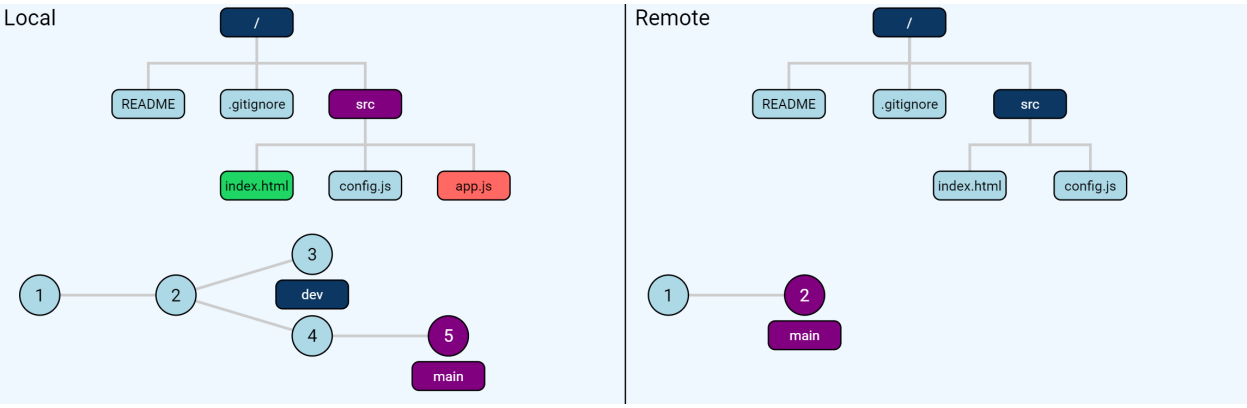


Figure 3.5: Git Visualization

For example, in Figure 3.5, *src* is the current working directory. The green rectangles represent files that have been added to the Git staging area. *index.html* has been added to the Git staging area. The red rectangles represent files that have been changed but have not been added yet to the Git staging area. *app.js* has been changed. The light blue circles represent commits. Each commit is labeled with a number. The number matches local commits with the corresponding remote commit. The gray lines connecting the circles represent a parent-child relationship between commits where the parent commit is connected to the left end of the line, and the child commit is connected to the right end of the line. The

dark blue rectangles in the bottom half of the visualization represent branches, which point to whichever commit appears directly above them. *dev* is a branch, which is a pointer to commit number 3. The purple rectangle in the bottom half of the visualization represents the current branch. Similarly, the purple circle represents the head commit, which is the commit the current branch points to. The current branch is *main*, which is a pointer to the head commit labeled number 5.

The interactive Git visualization includes transitions to visualize the effects of commands on the file system and the internal Git state. If a file is changed, its color changes to red. If a file is added to the Git staging area, its color changes to green. If a commit is created, the files changed in that commit are shown fading into the new circle representing the commit. If a user pushes changes from the local repository (left side) to the remote repository (right side), the new commits and the changed files are shown sliding from the local repository to the remote repository. Similarly, if a user pulls changes from the remote repository to the local repository, the new commits and changed files are shown sliding from the remote repository to the local repository. If a user clones the remote repository, the file system and Git tree are shown sliding from the remote repository to the local repository. If a user switches branches, the colors of the branches update accordingly. If a user creates a new branch, a new branch rectangle fades in. All of these transitions work together to provide a visualization of the file systems and internal Git states as commands are executed.

3.7 Git Exercise Component

Using the browser-based command line interface and the interactive Git visualization, we developed the Git exercise component. Figure 3.6 shows an example of a Git exercise.

The Git exercise component follows the same structure as the command line exercise compo-

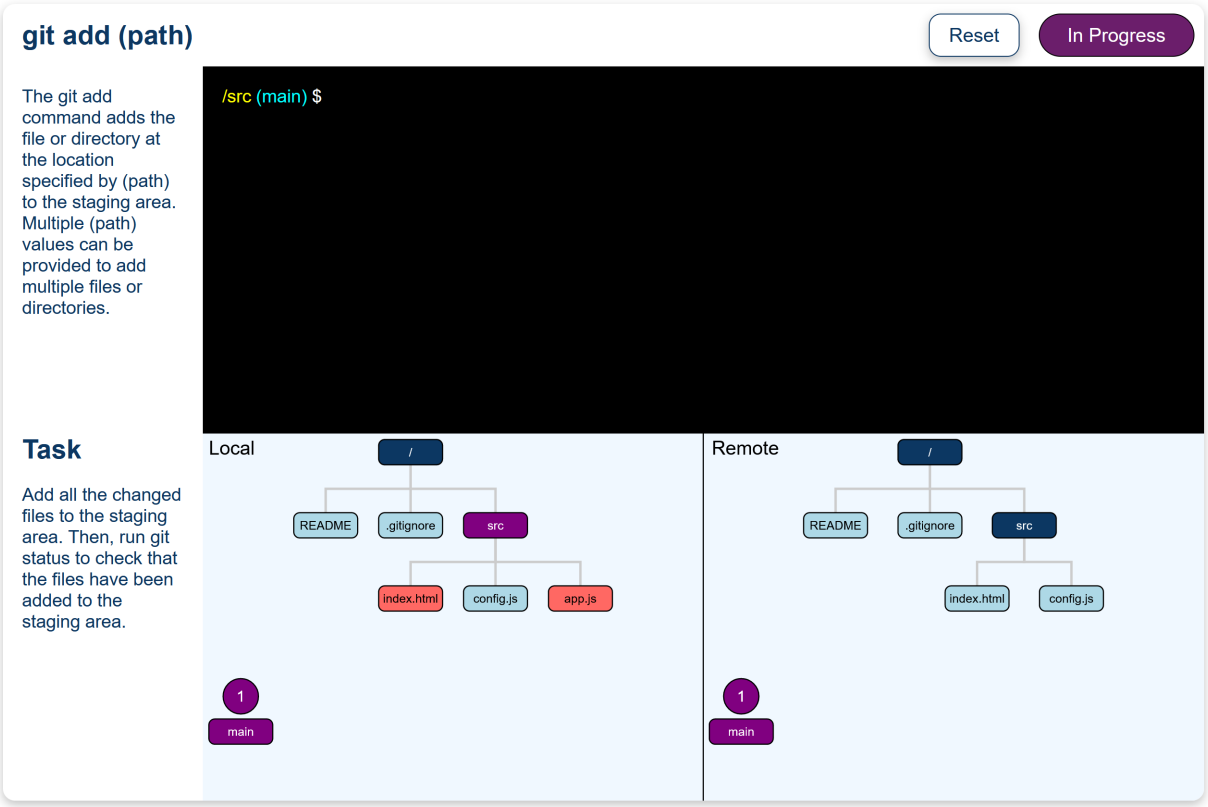


Figure 3.6: Git Exercise

nent with a description, a task, a browser-based command line interface, and an interactive Git visualization. The description provides some of the information needed to complete the exercise. In most exercises, the description contains a command, its arguments, and a description of the command. The task describes how to complete the exercise. The browser-based command line interface allows users to input the commands needed to complete the exercise. The browser-based command line interface provides the same functionality for all exercises, meaning the users can execute any of the supported commands as they complete the exercises. The Git exercise component supports all the commands from the command line exercise component as well as the following commands with arguments surrounded by (parentheses) and optional flags surrounded by [brackets]:

- **git clone (url)**: Clone the remote repository at the location specified by (url) and copy the contents of the remote repository to a new directory on the local machine.
- **git status**: Print the status of the local repository including information about the working tree, the staging area, commits, and the active branch.
- **git add (path)**: Add the file at the location specified by (path) to the staging area. If (path) specifies a directory, all the changed files in the directory are added to the staging area. Provide multiple (path) values separated by space characters to add multiple files.
- **git rm [-r] (path)**: Remove the file at the location specified by (path), and add the file to the staging area. Provide the [-r] flag to remove directories. If (path) specifies a directory, all the files in the directory are removed and added to the staging area. Provide multiple (path) values separated by space characters to remove multiple files.
- **git restore [--staged] (path)**: Revert the changes made to the file at the location specified by (path). Provide the [--staged] flag to move a file from the staging area to the working tree. If (path) specifies a directory, all the changed files in the directory are restored. Provide multiple (path) values separated by space characters to restore multiple files.
- **git commit -m (message) [-a] (path)**: Create a commit containing the changes in the staging area. The -m flag is required and must be followed by a nonempty (message). Provide the [-a] flag to add all files to the staging area before creating the commit. Provide one or more (path) values to create a commit containing only the changes to the files at the location or locations specified by (path). If (path) specifies a directory, all the changed files in the directory are committed. Untracked files are not included in the commit even if the [-a] flag or the (path) arguments are used.

- **git push**: Push new commits from the current branch of the local repository to the corresponding branch of the remote repository. The commit or commits contain the changes to the files that are applied to the remote repository.

- **git pull**: Pull new commits from the current branch of the remote repository to the corresponding branch of the local repository. The commit or commits contain the changes to the files that are applied to the local repository.

- **git branch (branch_name)**: Create a branch with the name specified by (branch_name).

- **git switch [-c] (branch_name)**: Change the current branch to the branch with the name specified by (branch_name). Provide the [-c] flag to create a new branch with the name specified by (branch_name) if the branch does not exist.

- **git checkout [-b] (branch_name)**: Change the current branch to the branch with the name specified by (branch_name). Provide the [-b] flag to create a new branch with the name specified by (branch_name) if the branch does not exist. Provides the same functionality as git switch. In practice, it is better to use git switch for changing branches because git is migrating from checkout to switch for changing branches.

As the users execute commands, the interactive Git visualization updates accordingly. This helps the user understand the impact of the commands on the file system and the internal Git state. Every time the user executes a command, the exercise checks the state of the file system and the internal Git data to determine if the user has completed the exercise. This is done using a custom method defined with each exercise. This method is run after each command execution and has access to the state of the file system, the internal Git data, and the most recent command. If this method returns true, then the exercise has been completed. Once the exercise has been completed, the purple *In Progress* indicator at the

top of the exercise changes to a green *Completed* indicator. The description, task, initial state, and desired state are configurable so that new Git exercises can be created easily.

3.8 Git Exercises

Using the configurable Git exercise component, we developed 18 Git exercises. The Git exercises were designed to teach many basic Git commands that are essential for using Git effectively. Most exercises focus on teaching one Git command. However, the last three exercises are challenge exercises that require multiple commands to be completed. Table 3.2 shows the topics focused on by each Git exercise. With the Git exercises, we created an About page that explains what Git is and how the exercises work. The About page also lists key Git terms and all the commands supported by the Git exercises.

Table 3.2: Git Exercises Topics

Exercise	Topic
Exercise 1	git clone
Exercise 2	git status
Exercise 3	git add
Exercise 4	git rm
Exercise 5	git commit
Exercise 6	git push
Exercise 7	git restore
Exercise 8	git restore with --staged flag
Exercise 9	git pull
Exercise 10	git commit with -a flag
Exercise 11	git commit for specific files
Exercise 12	git branch
Exercise 13	git switch
Exercise 14	git switch with -c flag
Exercise 15	git switch with diverged branches
Exercise 16	Creating new file and pushing to remote repository
Exercise 17	Pushing new branch to remote repository
Exercise 18	Cloning remote repository and pushing new change to remote repository

3.9 Pilot Testing

After developing the exercises, we carried out pilot testing by asking several CS faculty and advanced-level CS students in our department to try out the exercises and provide feedback. Table 3.3 summarizes the most impactful feedback received.

Table 3.3: Pilot Testing Feedback

The tab key does not auto-complete the directory or file name in the command line interface.
The up and down arrow keys do not allow the user to scroll through the history of the commands run in the command line interface.
The command line interface prompt does not include the current working directory or the current branch for the Git exercises.
The command line interface allows users to input the incorrect number of arguments for commands.
The command line interface allows users to remove the current working directory using the rm command.

We were able to address all of this feedback. We added a tab key auto-complete functionality so that users can auto-complete a directory or file name using the tab key. We added command line history functionality so that users can scroll through the history of commands using the up and down arrow keys. We added the absolute path of the current working directory to the command line interface prompt so that users can easily determine the current working directory. Similarly, we added the name of the current branch to the command line interface prompt for the Git exercises so that users can easily determine the current branch. We added error handling so that if a user inputs too few or too many arguments for a command, the command line interface outputs an error message and does not run the command. We added error handling for the rm command that checks whether the current working directory would be deleted. If so, the command line interface outputs an error message and does not execute the rm command.

3.10 Canvas Integration

Using the functionality built into OpenDSA, we integrated the command line and Git exercises with Canvas. Because the exercises are integrated with Canvas, instructors can easily assign the exercises to students. Students can then complete the exercises directly in Canvas without needing to access an external site. Because OpenDSA follows the LTI protocol, Canvas also automatically records the students' grades as they complete the exercises. Thus, instructors do not have to manually record students' grades on the exercises. Figure 3.7 shows an example of a command line exercise in Canvas.

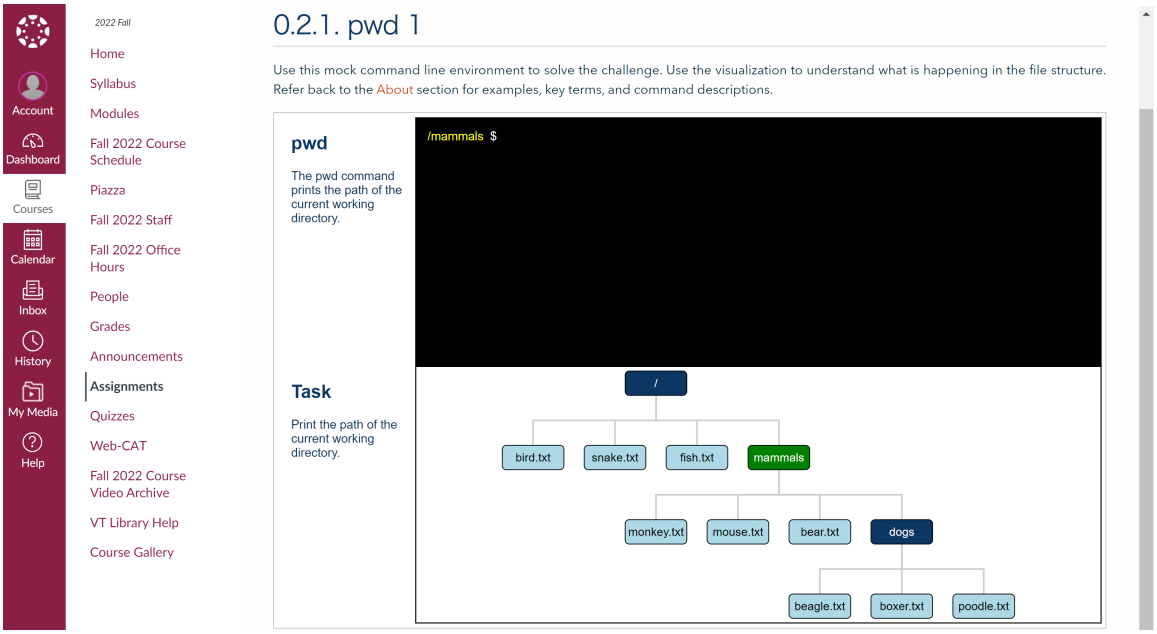


Figure 3.7: Command Line Exercise Integrated in Canvas

Chapter 4

Course Deployment

4.1 Assessment Design

To test the effectiveness of the exercises, we developed an assessment for both command line and Git to be used as pre- and post-assessment of the exercises. Each assessment consisted of 10 multiple choice questions. The pre-assessment for both command line and Git contained an additional question about the student's level of experience.

4.1.1 Command Line

Table 4.1 shows the topics covered in the command line assessment. The questions aim to test the core command line concepts taught in the command line exercises.

Table 4.1: Command Line Assessment Questions Topics

Question	Topic
Question 1	ls
Question 2	touch and ls
Question 3	cd to parent directory
Question 4	pwd and ls
Question 5	cd to parent directory and ls
Question 6	touch and ls
Question 7	mkdir
Question 8	rm with -r flag
Question 9	mv
Question 10	cp and rm with -r flag

Appendix A.1 contains the question about the student's level of experience with the command line, and Appendix A.2 contains the command line assessment questions.

4.1.2 Git

Table 4.2 shows the topics covered in the Git assessment. The questions aim to test the core Git concepts taught in the Git exercises.

Table 4.2: Git Assessment Questions Topics

Question	Topic
Question 1	git commit
Question 2	git push
Question 3	git clone
Question 4	Pulling from remote repository and pushing new change to remote repository
Question 5	git switch
Question 6	git push and branches
Question 7	git add
Question 8	git commit
Question 9	git pull
Question 10	git clone

Appendix A.3 contains the question about the student's level of experience with Git, and Appendix A.4 contains the Git assessment questions.

4.2 Classroom Deployment

After developing the command line and Git exercises, we tested the effectiveness of the exercises in the CS2104 Introduction to Problem Solving course across the 4 different class sections. The command line and Git exercises were tested separately with one day of class used to test the command line exercises and a second day of class used to test the Git exercises. The structure of the testing was the same for both the command line and Git exercises.

Students first took the assessment before completing the exercises. They were given 10 minutes to complete the assessment and were not allowed to return to a question after moving past it. Students were told they would receive full credit for the pre-assessment as long as they completed it so that students could answer the questions to the best of their knowledge without feeling pressured by their grade. After completing the pre-assessment, students moved on to the exercises. Students could not access the exercises until the pre-assessment was completed so that the pre-assessment accurately recorded students' knowledge before completing the exercises. The command line exercises were given as an assignment worth 20 points with the first fourteen exercises being worth one point each, and the last three exercises being worth two points each because they were challenge exercises. The Git exercises were given as an assignment worth 21 points with the first fifteen exercises being worth one point each, and the last three exercises being worth two points each because they were challenge exercises. The students were given as much time as they needed to complete the exercises so that they could work at their own pace and reference the documentation as they worked through the exercises. After completing the exercises, students were required to take the same assessment again as a post-assessment. Students were allowed to take the post-assessment without completing all the exercises in case they were not able to complete some of the exercises. Like with the pre-assessment, students were given 10 minutes to complete the post-assessment and were not allowed to return to a question after moving past it. Contrary to the pre-assessment, students were told that the post-assessment would be graded based on correctness. We did this to incentivize students to think carefully as they answer the questions. Students were given approximately 30 minutes of in-class time to complete the pre-assessment, the exercises, and the post-assessment. If they did not complete everything in that time, they were able to finish after class.

4.3 Feedback

From the deployment, we received feedback by talking to students as they completed the exercises. Table 4.3 lists the most impactful feedback.

Table 4.3: Classroom Deployment Feedback

Users have difficulty clicking into the command line interface input field to input commands.
If the user reaches a state that does not allow them to complete an exercise, they have to refresh the page to reset the exercise state.

We were able to address this feedback. We added an on-click handler to the command line interface so that when users click anywhere in the command line interface, the input field is focused. This makes it easier for users to focus the input field and input commands. Additionally, we added a reset button to the top right of the exercises. The users can use this button to reset the exercise to its initial state.

Chapter 5

Results

5.1 Command Line

5.1.1 Descriptive Statistics

Figure 5.1 shows the distribution of students' levels of experience with the command line based on four different levels of working experience with the command line. Most students have had multiple experiences using the command line, have used the command line once or twice, or have never used the command line before. Students are distributed somewhat evenly among those three categories with most students having used the command line once or twice. Significantly fewer students use the command line regularly compared to any of the other categories with only 9.9% of students using the command line regularly. This indicates that although some sophomore-level computer science students have multiple experiences with the command line, most sophomore-level computer science students do not use the command line regularly.

Table 5.1 and Figure 5.2 show the mean scores out of 10 on the command line pre-assessment and post-assessment grouped by the various levels of command line experience. On average, the mean scores on the command line pre-assessment increased with the level of command line experience. The mean scores on the post-assessment were greater than the mean scores on the pre-assessment for all levels of experience. The mean scores on the post-assessment

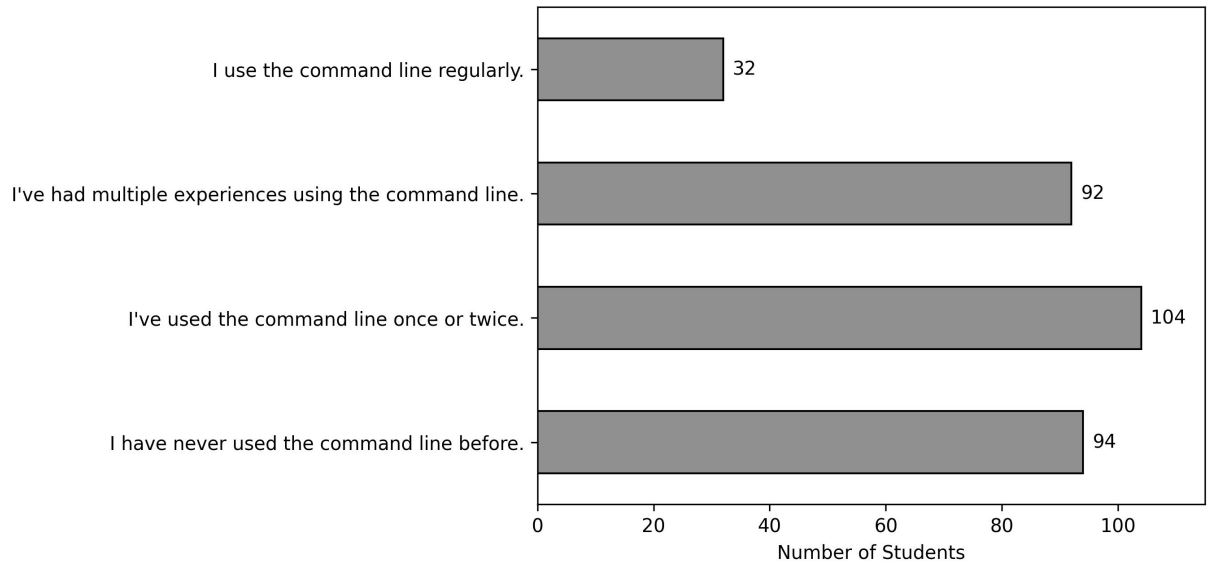


Figure 5.1: Distribution of levels of command line experience for 322 students

mostly increased with the level of experience. However, students who have had multiple experiences using the command line had a greater mean score on the post-assessment than students who use the command line regularly. The post-assessment scores deviated less than the pre-assessment scores between each level of experience. These results indicate that on average, the command line exercises increased students' scores from pre-assessment to post-assessment regardless of levels of experience.

Table 5.1: Mean scores on command line assessments and exercises

Experience Level	Count	Pre/10	Post/10	Exercises/20
I use the command line regularly.	32	8.094	8.750	19.625
I've had multiple experiences using the command line.	92	7.304	9.033	18.826
I've used the command line once or twice.	104	5.510	8.375	19.135
I have never used the command line before.	94	4.564	8.351	18.532
All Levels	322	6.003	8.593	18.919

Table 5.2 summarizes the scores on the command line pre- and post-assessments. The mean, Q1, median, Q3, and mode of the scores were greater for the command line post-assessment than the command line pre-assessment. However, the standard deviation of the scores was

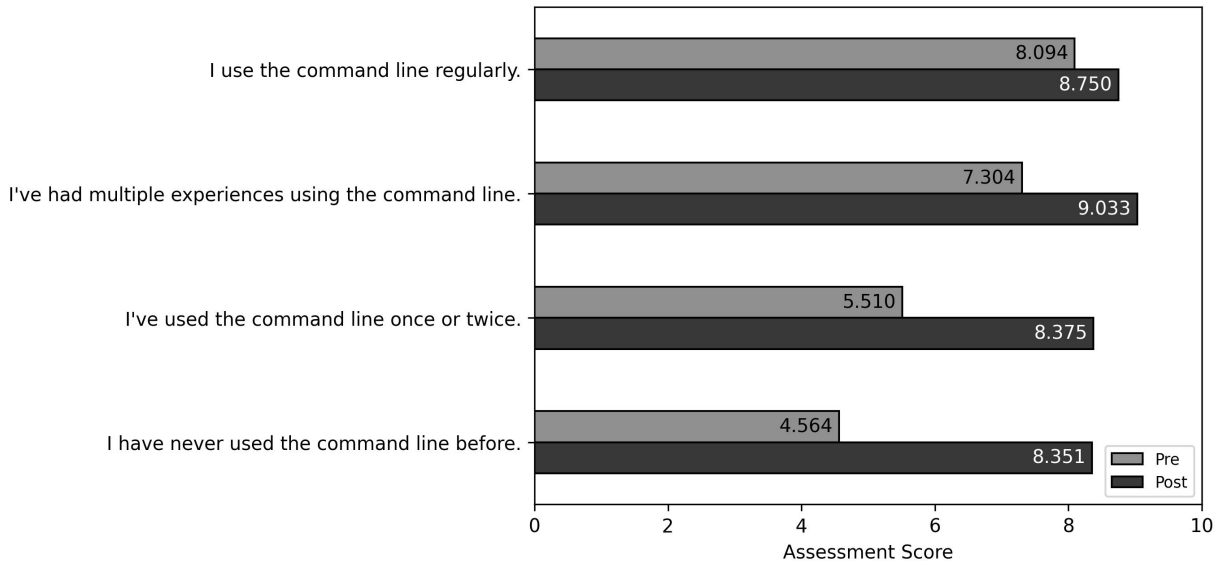


Figure 5.2: Comparison of mean command line pre- and post-assessment scores by experience

greater for the command line pre-assessment than the command line post-assessment.

Table 5.2: Summary statistics for 322 scores on command line pre- and post-assessments

Assessment	Mean	Standard Deviation	Min	Q1	Median	Q3	Max	Mode
Pre	6.003	2.467	0	4	6	8	10	8
Post	8.593	1.502	0	8	9	10	10	9

Table 5.1 also shows the mean scores out of 20 on the command line exercises. The mean scores on the exercises mostly increased with the level of experience. However, students who have used the command line once or twice performed better on the exercises than students who have had multiple experiences with the command line. Despite the slight score improvement based on level of experience, all students were able to complete almost all of the exercises on average. This indicates that the exercises are feasible for students with any level of experience. Figure 5.3 shows the distribution of students' scores on the command line exercises and further supports that most students were able to complete all of the exercises.

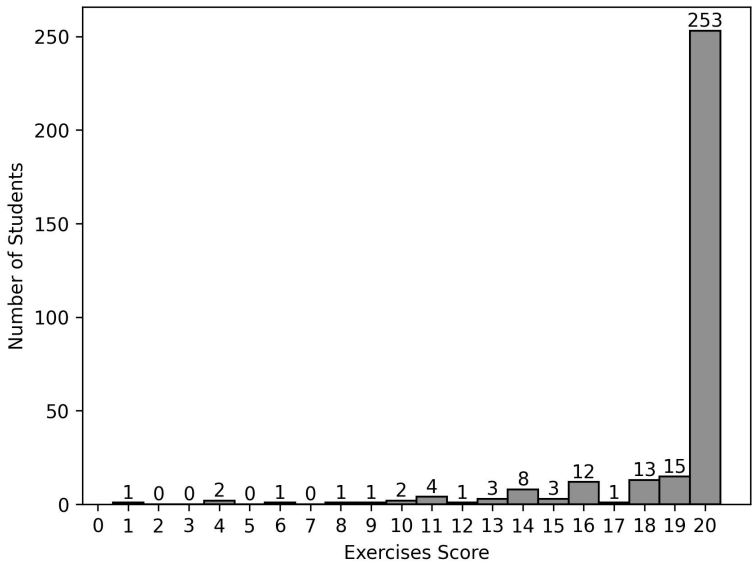


Figure 5.3: Command line exercises scores

5.1.2 Test Statistics

Table 5.3 shows the results of a two-sample Welch's t-test comparing the scores on the command line pre-assessment and post-assessment grouped by the various levels of command line experience. Excluding those who use the command line regularly, students with all levels of experience performed statistically significantly better on the post-assessment than on the pre-assessment. This indicates that the command line exercises were effective for students who have had multiple experiences using the command line, have used the command line once or twice, or have never used the command line before. Although students who use the command line regularly performed better on the post-assessment than the pre-assessment, this result was not statistically significant. This result could be due to the fact that students who use the command line regularly performed well on the pre-assessment, and thus had less room for improvement on the post-assessment.

An ANOVA test was conducted for the scores on the command line pre-assessment grouped by the various levels of command line experience. The ANOVA test determined a *p-value* of

Table 5.3: Comparison of mean command line pre- and post-assessment scores by experience

Experience Level	Pre/10	Post/10	*p-value*
I use the command line regularly.	8.094	8.750	.136
I've had multiple experiences using the command line.	7.304	9.033	**<.001*** **
I've used the command line once or twice.	5.510	8.375	**<.001*** **
I have never used the command line before.	4.564	8.351	**<.001*** **
All Levels	6.003	8.593	**<.001*** **

<.001. Thus, there is a statistically significant difference between the mean scores on the command line pre-assessment based on the level of command line experience. This indicates that a student's level of experience affected their score on the pre-assessment. Table 5.4 shows the results of performing a Tukey HSD test on the command line pre-assessment scores grouped by the various levels of command line experience. The difference between the pre-assessment scores for most pairs of command line experience was statistically significantly different. The only difference that was not statistically significant was the difference between the pre-assessment scores for students who use the command line regularly and who have had multiple experiences using the command. These results further indicate that a student's level of experience affected their score on the pre-assessment. However, the results also indicate that students who use the command line regularly and who have had multiple experiences using the command line did not perform differently on the pre-assessment. Overall, the results from the ANOVA test and the Tukey HSD test are expected because students with less command line experience should have less knowledge about the command line, and thus perform worse on the pre-assessment. Hence, this result suggests that the pre-assessment was designed reasonably because people with more command line knowledge performed better than people with less command line knowledge.

An ANOVA test was conducted for the scores on the command line post-assessment grouped by the various levels of command line experience. The ANOVA test determined a *p-value*

Table 5.4: Comparison of mean command line pre-assessment scores by experience

Categories Pair	Difference	*p-value*
I use the command line regularly. I've had multiple experiences using the command line.	0.789	.271
I use the command line regularly. I've used the command line once or twice.	2.584	**<.001*** **
I use the command line regularly. I have never used the command line before.	3.530	**<.001*** **
I've had multiple experiences using the command line. I've used the command line once or twice.	1.795	**<.001*** **
I've had multiple experiences using the command line. I have never used the command line before.	2.741	**<.001*** **
I've used the command line once or twice. I have never used the command line before.	0.946	**.010***

of **.005**. Thus, there is a statistically significant difference between the mean scores on the command line post-assessment based on the level of command line experience. This indicates that a student's level of command line experience affected their score on the command line post-assessment. Table 5.5 shows the results of performing a Tukey HSD test on the command line post-assessment scores grouped by the various levels of command line experience. The difference between the post-assessment scores for most pairs of command line experience was not statistically significantly different. The only two differences that were statistically significant were the differences between the post-assessment scores for students who have had multiple experiences and who have either used the command line once or twice or never. Although these results indicate that a student's level of experience affected their score on the post-assessment in some cases, the results also indicate that the post-assessment scores were affected less by the level of experience than the pre-assessment scores. This can be seen in that the p-values from the Tukey HSD test for comparing the post-assessment scores shown in Table 5.5 are much greater than the p-values from the Tukey HSD test for comparing the pre-assessment scores shown in Table 5.4. Overall, the results from the ANOVA test and the Tukey HSD test indicate that the level of experience affected the scores on the

post-assessment for some levels of experience but not all levels of experience. These results suggest that the command line exercises helped reduce the gap in command line knowledge of students with different levels of command line experience.

Table 5.5: Comparison of mean command line post-assessment scores by experience

Categories Pair	Difference	p-value
I use the command line regularly. I've had multiple experiences using the command line.	-0.283	.788
I use the command line regularly. I've used the command line once or twice.	0.375	.592
I use the command line regularly. I have never used the command line before.	0.399	.552
I've had multiple experiences using the command line. I've used the command line once or twice.	0.658	**.011***
I've had multiple experiences using the command line. I have never used the command line before.	0.682	**.010***
I've used the command line once or twice. I have never used the command line before.	0.024	.999

Table 5.6 shows the score improvements from the command line pre-assessment to the command line post-assessment grouped by the various level of experience. The scores improved from the pre-assessment to the post-assessment for all levels of experience. The scores improved by a greater magnitude as the level of command line experience decreased. This indicates that students with any level of command line experience benefited from the exercises with students who have less experience benefiting more.

Table 5.6: Mean score improvements between command line pre- and post-assessments

Experience Level	Pre/10	Post/10	Improvement
I use the command line regularly.	8.094	8.750	0.656
I've had multiple experiences using the command line.	7.304	9.033	1.728
I've used the command line once or twice.	5.510	8.375	2.865
I have never used the command line before.	4.564	8.351	3.787
All Levels	6.003	8.593	2.590

An ANOVA test was conducted for the score improvements from the command line pre-

assessment to the command line post-assessment grouped by the various levels of command line experience. The ANOVA test determined a *p-value* of **<.001**. Thus, there is a statistically significant difference between the mean score improvements from the command line pre-assessment to the command line post-assessment based on the level of command line experience. This indicates that a student's level of command line experience affected their score improvement from the pre-assessment to the post-assessment. Table 5.7 shows the results of performing a Tukey HSD test on the command line score improvements from the pre-assessment to the post-assessment grouped by the various levels of command line experience.

Table 5.7: Comparison of mean command line assessments score improvements by experience

Categories Pair	Difference	p-value
I use the command line regularly. I've had multiple experiences using the command line.	-1.072	.101
I use the command line regularly. I've used the command line once or twice.	-2.209	**<.001*** **
I use the command line regularly. I have never used the command line before.	-3.131	**<.001*** **
I've had multiple experiences using the command line. I've used the command line once or twice.	-1.137	**.003** **
I've had multiple experiences using the command line. I have never used the command line before.	-2.059	**<.001*** **
I've used the command line once or twice. I have never used the command line before.	-0.922	**.024***

The difference between the score improvements for most pairs of command line experience was statistically significant. The only difference that was not statistically significant was the difference between the score improvements for students who use the command line regularly and who have had multiple experiences using the command line. These results indicate that a student's level of experience affected how much their post-assessment score improved from their pre-assessment score. Overall, the results from the ANOVA test and the Tukey HSD test are expected because students with less experience generally performed worse on the

pre-assessment and thus had more room to improve on the post-assessment. Regardless, these results suggest that the command line exercises were more effective for students with less command line experience.

We explored how the levels of command line experience affected the scores on the command line exercises by conducting an ANOVA test for the scores on the command line exercises grouped by the various levels of command line experience. The ANOVA test determined a *p-value* of .241. Thus, there is not a statistically significant difference between the mean scores on the command line exercises based on the level of command line experience. This indicates that a student's level of command line experience does not affect their score on the command line exercises. Table 5.8 shows the results of performing a Tukey HSD test on the command line exercises scores grouped by the various levels of command line experience. The difference between the exercises scores for all pairs of command line experience was not statistically significantly different. These results further indicate that a student's level of experience did not affect their score on the command line exercises. Overall, the results from the ANOVA test and the Tukey HSD test suggest that the command line exercises are equally feasible for students with any level of command line experience.

Table 5.8: Comparison of mean command line exercises scores by experience

Categories Pair	Difference	p-value
I use the command line regularly. I've had multiple experiences using the command line.	0.799	.539
I use the command line regularly. I've used the command line once or twice.	0.490	.838
I use the command line regularly. I have never used the command line before.	1.093	.258
I've had multiple experiences using the command line. I've used the command line once or twice.	-0.309	.880
I've had multiple experiences using the command line. I have never used the command line before.	0.294	.901
I've used the command line once or twice. I have never used the command line before.	0.603	.465

5.1.3 Experienced vs Inexperienced

We also analyzed the data by splitting the students into two groups, inexperienced and experienced, based on their experience level with the command line. A student is considered inexperienced if they have either only used the command line once or twice or never used the command line before. A student is considered experienced if they either use the command line regularly or have had multiple experiences using the command line. Table 5.9 shows the results of a two-sample Welch's t-test comparing students' command line pre-assessment scores, post-assessment scores, assessment score improvements, and exercises scores grouped by inexperienced and experienced students.

Table 5.9: Comparison of mean command line assessments and exercises scores by experience

	Inexperienced	Experienced	*p-value*
Pre/10	5.061	7.508	**<.001*** **
Post/10	8.364	8.960	**<.001*** **
Improvement	3.303	1.452	**<.001*** **
Exercises/20	18.848	19.032	.573

Experienced students performed statistically significantly better than inexperienced students on the command line pre-assessment and post-assessment. This result is expected because students with more command line experience are expected to perform better on the assessments. Furthermore, contrary to the results from the ANOVA and Tukey HSD tests for the post-assessment, the result suggests that the command line exercises did not reduce the gap in command line knowledge of experienced and inexperienced students because the experienced students still performed statistically significantly better on the command line post-assessment than the inexperienced students. Inexperienced students had a statistically significantly greater score improvement than experienced students from the pre-assessment to the post-assessment. As with the result from the ANOVA and Tukey HSD tests for the score improvements, this result is expected because inexperienced students generally per-

formed worse on the pre-assessment and thus had more room for improvement on the post-assessment. Still, this result suggests that the command line exercises were more effective for inexperienced students. Experienced students did not perform statistically significantly better than inexperienced students on the command line exercises. This result indicates that the command line exercises are not significantly easier for experienced students.

5.1.4 Question Analysis

Table 5.10 shows the number and percent of students who correctly answered each question on the command line pre- and post-assessments. Q9, Q8, and Q3 were the most missed questions on the command line pre-assessment. Q9, Q10, and Q8 were the most missed questions on the command line post-assessment.

Table 5.10: Number of students out of 322 who correctly answered each question on the command line pre- and post-assessment

Question	Pre	Post
Q9	77 (23.9%)	134 (41.6%)
Q8	157 (48.8%)	280 (87.0%)
Q3	159 (49.4%)	300 (93.2%)
Q10	192 (59.6%)	258 (80.1%)
Q6	203 (63.0%)	315 (97.8%)
Q5	208 (64.6%)	292 (90.7%)
Q2	215 (66.8%)	311 (96.6%)
Q7	222 (68.9%)	282 (87.6%)
Q1	236 (73.3%)	319 (99.1%)
Q4	264 (82.0%)	317 (98.4%)

Many fewer students correctly answered Q9 compared to any other question on both the command line pre-assessment and post-assessment, which indicates that Q9 is more difficult than the other questions. Figure 5.4 shows Q9 from the command line assessments. Figure 5.5 shows the correct answer, and Figure 5.6 shows the most common incorrect answer. Students may have selected this answer choice because they thought the double periods indi-

cated that the file should move up one directory from its current location instead of moving up one directory from the current working directory. The poor performance on Q9 suggests that the exercises may need to better enforce the meaning of the double periods.

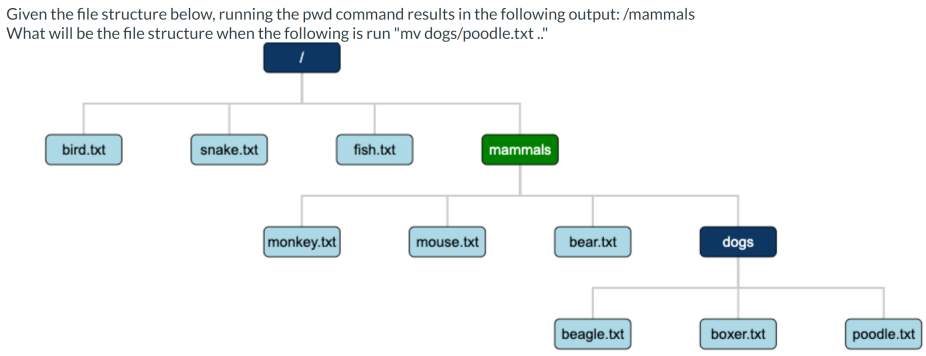


Figure 5.4: Command Line Assessment Question 9

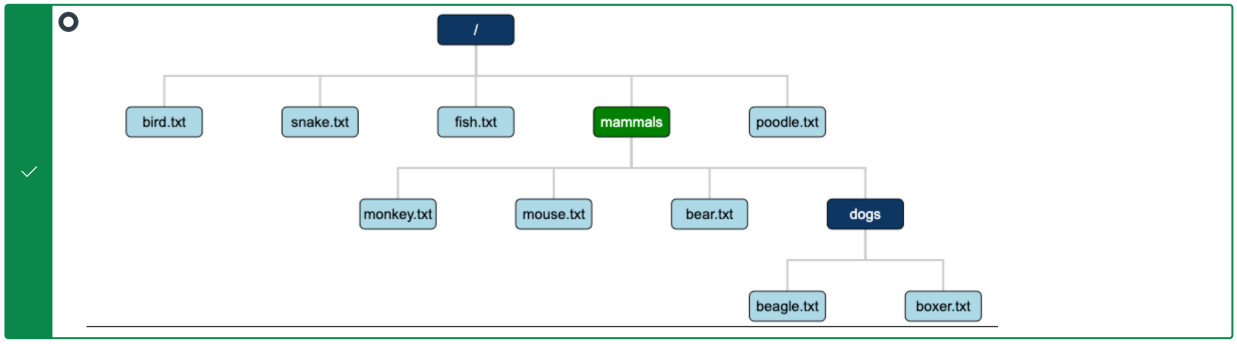


Figure 5.5: Command Line Assessment Question 9 Correct Answer

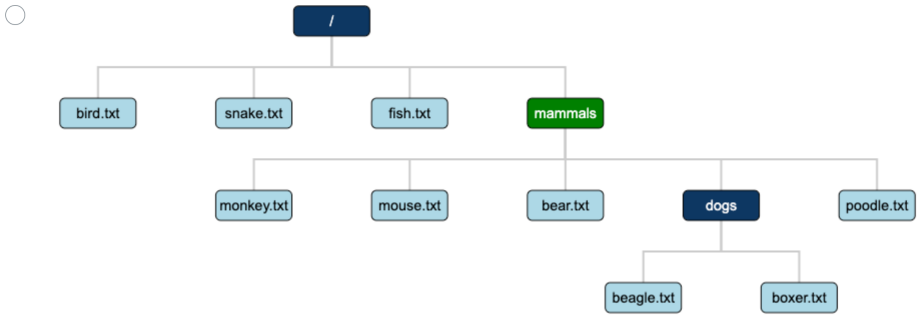


Figure 5.6: Command Line Assessment Question 9 Most Common Incorrect Answer

Figure 5.7 shows Q8 from the command line assessments. The poor performance on Q8 sug-

gests that the exercises may need to better enforce how the -r flag works with rm command. However, students may have had difficulty because the question may have been confusing due to the _____ notation.

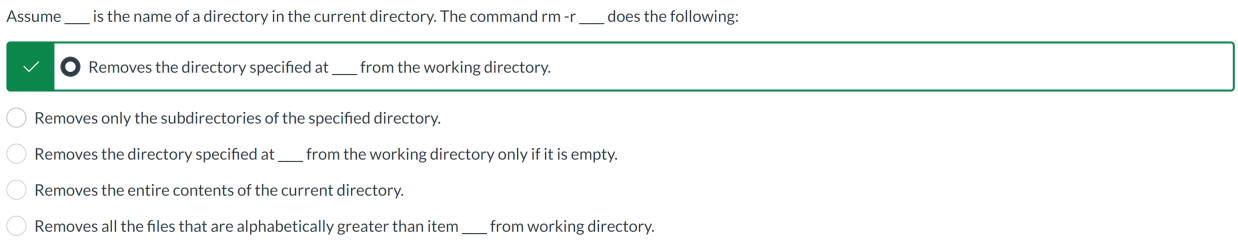


Figure 5.7: Command Line Assessment Question 8

Figure 5.8 shows Q3 from the command line assessments. Similar to Q9, Q3 required students to understand the usage of double periods. The poor performance on Q3 in the command line pre-assessment shows the need for teaching the usage of the double periods. Contrary to Q9, Q3 was not one of the most missed questions on the command line post-assessment although both questions required the students to understand the usage of double periods. Students may have struggled more on Q9 because they were expected to understand the usage of double periods in the context of moving files. The students may have focused on the mv command portion of Q9 and may not have thought carefully about the double periods. Thus, after completing the exercises, students seemed to understand the usage of double periods in the context of changing directories but not as well in the context of moving files.

Although Q10 was the fourth most missed question on the command line pre-assessment, Q10 was the second most missed question on the command line post-assessment. Figure 5.9 shows Q10 from the command line assessments. The difficulty on Q10 may have come from Q10 requiring students to mentally keep track of the file state as multiple commands are run, which no other question requires.

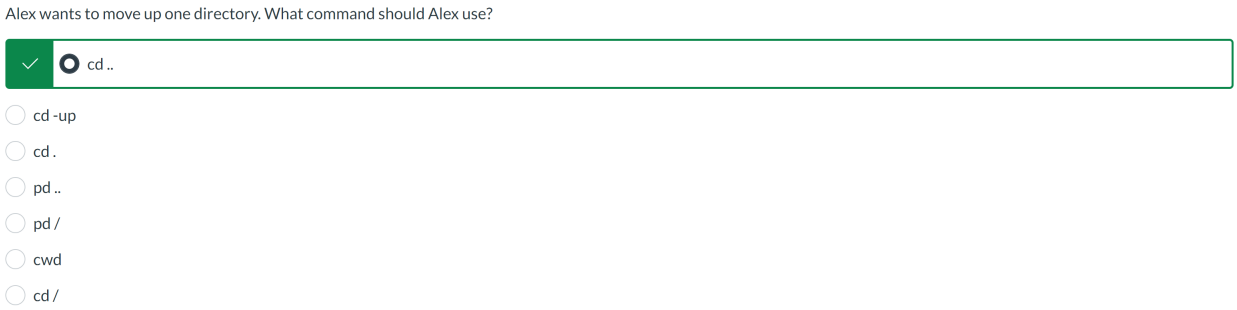


Figure 5.8: Command Line Assessment Question 3

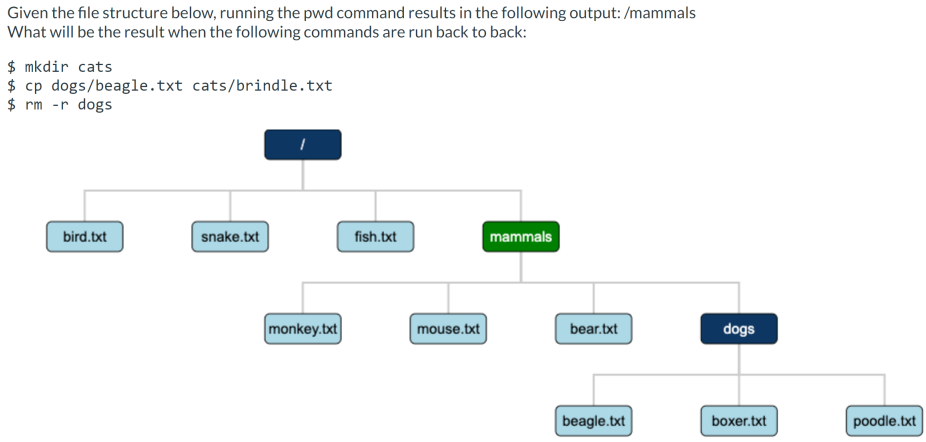


Figure 5.9: Command Line Assessment Question 10

Despite the scores being lower on certain questions, the number of students who got each question correct improved from the command line pre-assessment to the command line post-assessment for every question, which indicates that the command line exercises helped teach the students all of the concepts tested by the assessments.

We analyzed the number of students who answered each question correctly on the command line pre-assessment for each level of command line experience in order to gain insight into which command line concepts students struggle with before completing our tutorial. Table 5.11 shows the number and percent of students who use the command line regularly who correctly answered each question on the command line pre-assessment. For students who use the command line regularly, Q9, Q8, Q10, and Q3 were the most missed questions on

the pre-assessment. The top four most missed questions for this category are the same as for all categories combined, although the order of Q10 and Q3 is switched.

Table 5.11: Number of students out of 32 who use the command line regularly who correctly answered each question on the command line pre-assessment

Question	Pre
Q9	10 (31.2%)
Q8	19 (59.4%)
Q10	25 (78.1%)
Q3	26 (81.2%)
Q7	28 (87.5%)
Q5	29 (90.6%)
Q1	30 (93.8%)
Q2	30 (93.8%)
Q4	31 (96.9%)
Q6	31 (96.9%)

Table 5.12 shows the number and percent of students who have had multiple experiences using the command line who correctly answered each question on the command line pre-assessment. For students who have had multiple experiences using the command line, Q9, Q8, Q10, and Q3 were the most missed questions on the pre-assessment. The top four most missed questions for this category are the same as for students who use the command line regularly.

Table 5.13 shows the number and percent of students who have used the command line once or twice who correctly answered each question on the command line pre-assessment. For students who have used the command line once or twice, Q9, Q3, Q8, and Q10 were the most missed questions on the pre-assessment. The top four most missed questions for this category are the same as for students who use the command line regularly and for students who have had multiple experiences with the command line. However, the ordering of Q3, Q8, and Q10 are different.

Table 5.14 shows the number and percent of students who have never used the command

Table 5.12: Number of students out of 92 who have had multiple experiences using the command line who correctly answered each question on the command line pre-assessment

Question	Pre
Q9	33 (35.9%)
Q8	45 (48.9%)
Q10	60 (65.2%)
Q3	65 (70.7%)
Q5	70 (76.1%)
Q6	75 (81.5%)
Q2	77 (83.7%)
Q7	81 (88.0%)
Q4	82 (89.1%)
Q1	84 (91.3%)

Table 5.13: Number of students out of 104 who have used the command line once or twice who correctly answered each question on the command line pre-assessment

Question	Pre
Q9	19 (18.3%)
Q3	42 (40.4%)
Q8	51 (49.0%)
Q10	56 (53.8%)
Q5	59 (56.7%)
Q6	61 (58.7%)
Q7	64 (61.5%)
Q2	66 (63.5%)
Q1	71 (68.3%)
Q4	84 (80.8%)

line before who correctly answered each question on the command line pre-assessment. For students who have never used the command line before, Q9, Q3, Q6, Q2, and Q8 were the most missed questions on the pre-assessment. Q9, Q3, and Q8 appear in the top four most missed questions for the other levels of experiences. However, Q6 and Q2 do not. Both Q6 and Q2 require students to understand the usage of the touch command. Thus, students who have never used the command line before may have struggled with this question because the functionality of touch is less intuitive based on its name compared to many other commands.

Table 5.14: Number of students out of 94 who have never used the command line before who correctly answered each question on the command line pre-assessment

Question	Pre
Q9	15 (16.0%)
Q3	26 (27.7%)
Q6	36 (38.3%)
Q2	42 (44.7%)
Q8	42 (44.7%)
Q7	49 (52.1%)
Q5	50 (53.2%)
Q1	51 (54.3%)
Q10	51 (54.3%)
Q4	67 (71.3%)

Overall, students with all levels of command line experience performed the worst on Q9. Although the students performed worse on certain questions, the performance on each question improved from the command line pre-assessment to the command line post-assessment, indicating that the command line exercises helped improve the scores on all the questions.

5.2 Git

5.2.1 Descriptive Statistics

Figure 5.10 shows the distribution of students' levels of experience with Git based on four different levels of working experience with Git. The number of students with each level of Git experience approximately doubles as the level of Git experience decreases. Thus, the most common category is students who have never used Git or version control before with more than half of the students falling into this category. Furthermore, only 6.6% of students use Git regularly. This indicates that many sophomore-level computer science students have little to no experience with Git or version control.

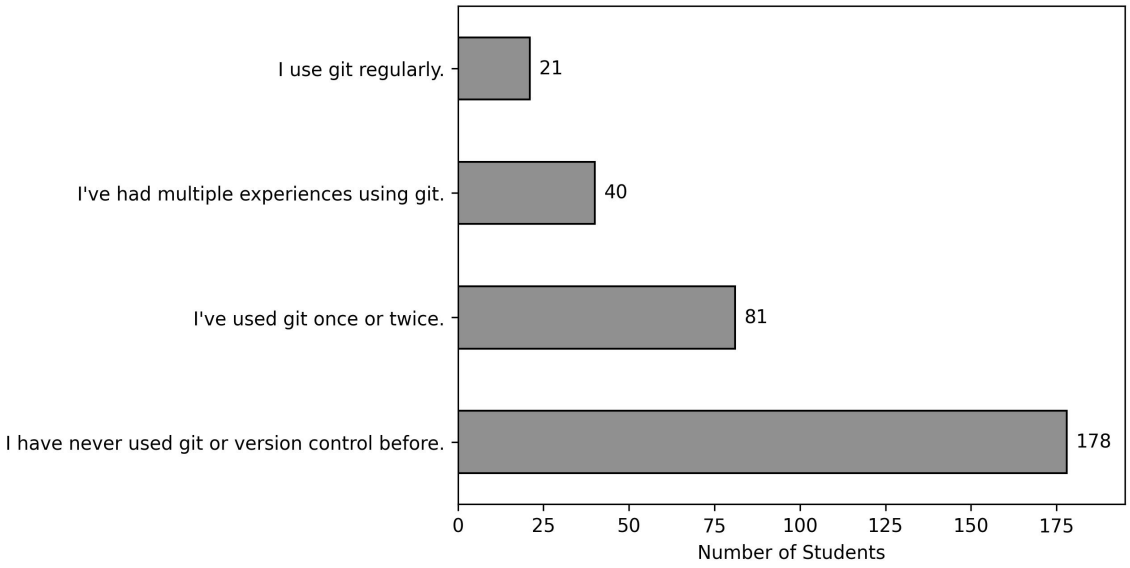


Figure 5.10: Distribution of levels of Git experience for 320 students

Table 5.15 and Figure 5.11 show the mean scores out of 10 on the Git pre-assessment and post-assessment grouped by the various levels of Git experience. On average, the mean scores on the Git pre-assessment increased with the level of Git experience. The mean scores on the post-assessment were greater than the mean scores on the pre-assessment for all levels

of experience. The mean scores on the Git post-assessment increased with the level of Git experience. Also, the Git post-assessment scores deviated less than the Git pre-assessment scores between each level of Git experience. These results indicate that on average, the Git exercises increased students' scores from pre-assessment to post-assessment regardless of levels of experience.

Table 5.15: Mean scores on Git assessments and exercises

Experience Level	Count	Pre/10	Post/10	Exercises/21
I use Git regularly.	21	7.238	9.190	18.143
I've had multiple experiences using Git.	40	5.725	8.725	18.425
I've used Git once or twice.	81	3.741	8.432	17.531
I have never used Git or version control before.	178	2.882	8.101	16.567
All Levels	320	3.741	8.334	17.147

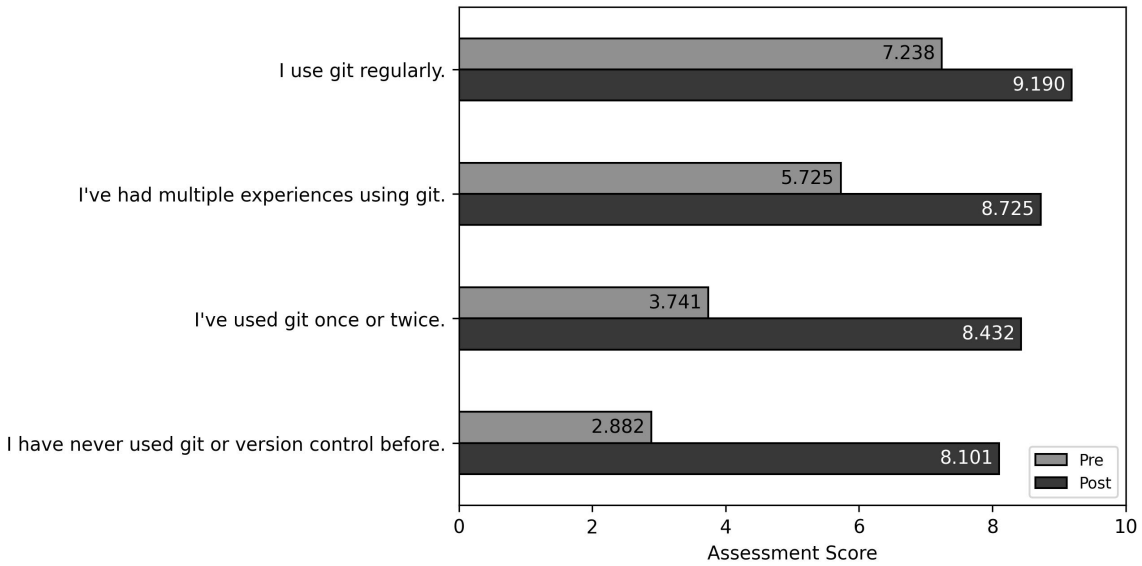


Figure 5.11: Comparison of mean Git pre- and post-assessment scores by experience

Table 5.16 summarizes the scores on the Git pre- and post-assessments. The mean, Q1, median, Q3, and mode of the scores were greater for the Git post-assessment than the Git pre-assessment. However, the standard deviation of the scores was greater for the Git pre-assessment than the Git post-assessment.

Table 5.16: Summary statistics for 320 scores on Git pre- and post-assessments

Assessment	Mean	Standard Deviation	Min	Q1	Median	Q3	Max	Mode
Pre	3.741	2.271	0	2	3	5	10	4
Post	8.334	2.105	0	8	9	10	10	10

Table 5.15 also shows the mean scores out of 21 on the Git exercises. The mean scores on the Git exercises mostly increased with the level of Git. However, students who have had multiple experiences using Git performed better on the Git exercises than students who use Git regularly. Despite the slight score improvement based on level of experience, all students were able to complete most of the Git exercises on average. This indicates that the Git exercises are feasible for students with any level of Git experience. However, the Git exercises did appear to be slightly more challenging for students who have never used Git or version control before. Figure 5.12 shows the distribution of students' scores on the Git exercises and further indicates that most students were able to complete most of the exercises.

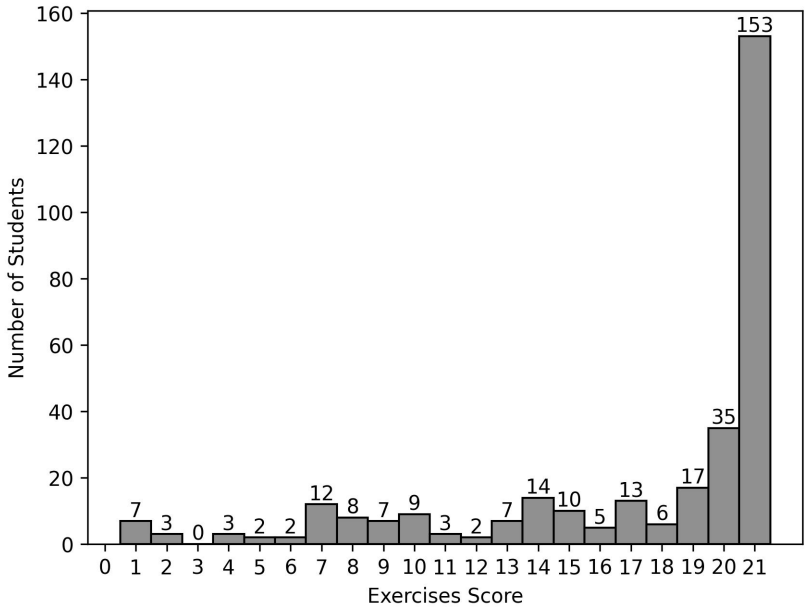


Figure 5.12: Git exercises scores

5.2.2 Test Statistics

Table 5.17 shows the results of a two-sample Welch's t-test comparing the scores on the Git pre-assessment and post-assessment grouped by the various levels of Git experience. Students with all levels of Git experience performed statistically significantly better on the Git post-assessment than on the Git pre-assessment. This indicates that the Git exercises were effective for students with all levels of Git experience and suggests that students of all levels of Git experience learned something new from the Git exercises.

Table 5.17: Comparison of mean Git pre- and post-assessment scores by experience

Experience Level	Pre/10	Post/10	*p-value*
I use Git regularly.	7.238	9.190	**<.001*** **
I've had multiple experiences using Git.	5.725	8.725	**<.001*** **
I've used Git once or twice.	3.741	8.432	**<.001*** **
I have never used Git or version control before.	2.882	8.101	**<.001*** **
All Levels	3.741	8.334	**<.001*** **

An ANOVA test was conducted for the scores on the Git pre-assessment grouped by the various levels of Git experience. The ANOVA test determined a *p-value* of **<.001**. Thus, there is a statistically significant difference between the mean scores on the Git pre-assessment based on the level of Git experience. This indicates that a student's level of experience affected their score on the pre-assessment. Table 5.18 shows the results of performing a Tukey HSD test on the Git pre-assessment scores grouped by the various levels of Git experience.

The difference between the pre-assessment scores for all pairs of Git experience was statistically significantly different. These results further indicate that a student's level of experience affected their score on the pre-assessment. Overall, the results from the ANOVA test and the Tukey HSD test are expected because students with less Git experience should have less knowledge about Git, and thus perform worse on the pre-assessment. Hence, this result suggests that the pre-assessment was designed reasonably because people with more Git

Table 5.18: Comparison of mean Git pre-assessment scores by experience

Categories Pair	Difference	p-value
I use Git regularly. I've had multiple experiences using Git.	1.513	**.015***
I use Git regularly. I've used Git once or twice.	3.497	**<.001****
I use Git regularly. I have never used Git or version control before.	4.356	**<.001****
I've had multiple experiences using Git. I've used Git once or twice.	1.984	**<.001****
I've had multiple experiences using Git. I have never used Git or version control before.	2.843	**<.001****
I've used Git once or twice. I have never used Git or version control before.	0.859	**.004****

knowledge performed better than people with less Git knowledge.

An ANOVA test was conducted for the scores on the Git post-assessment grouped by the various levels of Git experience. The ANOVA test determined a *p-value* of .065. Thus, there is not a statistically significant difference between the mean scores on the Git post-assessment based on the level of Git experience. This indicates that a student's level of Git experience did not affect their score on the Git post-assessment. Table 5.19 shows the results of performing a Tukey HSD test on the Git post-assessment scores grouped by the various levels of Git experience.

The difference between the post-assessment scores for all pairs of Git experience was not statistically significantly different. These results further indicate that a student's level of experience did not affect their score on the post-assessment. Overall, the results from the ANOVA test and the Tukey HSD test both indicate that the level of experience did not affect the scores on the post-assessment. These results suggest that the Git exercises helped reduce the gap in Git knowledge of students with different levels of Git experience.

Table 5.20 shows the score improvements from the Git pre-assessment to the Git post-

Table 5.19: Comparison of mean Git post-assessment scores by experience

Categories Pair	Difference	p-value
I use Git regularly. I've had multiple experiences using Git.	0.465	.842
I use Git regularly. I've used Git once or twice.	0.758	.450
I use Git regularly. I have never used Git or version control before.	1.089	.110
I've had multiple experiences using Git. I've used Git once or twice.	0.293	.887
I've had multiple experiences using Git. I have never used Git or version control before.	0.624	.323
I've used Git once or twice. I have never used Git or version control before.	0.331	.639

assessment grouped by the various level of experience. The scores improved from the pre-assessment to the post-assessment for all levels of experience. The scores improved by a greater magnitude as the level of Git experience decreased. This indicates that students with any level of Git experience benefited from the exercises with students who have less experience benefiting more.

Table 5.20: Mean score improvements between Git pre- and post-assessments

Experience Level	Pre/10	Post/10	Improvement
I use Git regularly.	7.238	9.190	1.952
I've had multiple experiences using Git.	5.725	8.725	3.000
I've used Git once or twice.	3.741	8.432	4.691
I have never used Git or version control before.	2.882	8.101	5.219
All Levels	3.741	8.334	4.594

An ANOVA test was conducted for the score improvements from the Git pre-assessment to the Git post-assessment grouped by the various levels of Git experience. The ANOVA test determined a *p-value* of $\mathbf{<.001}$. Thus, there is a statistically significant difference between the mean score improvements from the Git pre-assessment to the Git post-assessment based on the level of Git experience. This indicates that a student's level of Git experience affected

their score improvement from the pre-assessment to the post-assessment. Table 5.21 shows the results of performing a Tukey HSD test on the Git score improvements from the pre-assessment to the post-assessment grouped by the various levels of Git experience.

Table 5.21: Comparison of mean Git assessments score improvements by experience

Categories Pair	Difference	p-value
I use Git regularly. I've had multiple experiences using Git.	-1.048	.461
I use Git regularly. I've used Git once or twice.	-2.739	**<.001*** **
I use Git regularly. I have never used Git or version control before.	-3.267	**<.001*** **
I've had multiple experiences using Git. I've used Git once or twice.	-1.691	**.006** **
I've had multiple experiences using Git. I have never used Git or version control before.	-2.219	**<.001*** **
I've used Git once or twice. I have never used Git or version control before.	-0.528	.450

The difference between the score improvements for most pairs of Git experience was statistically significant. The only two differences that were not statistically significant were the differences between the score improvements for students who use Git regularly and who have had multiple experiences using Git and between the score improvements for students who have used Git once or twice and who have never used Git or version control before. These results indicate that a student's level of experience affected how much their post-assessment score improved from their pre-assessment score. However, students with similar but slightly different levels of experience did not have different score improvements in some cases. Overall, the results from the ANOVA test and the Tukey HSD test are expected because students with less experience generally performed worse on the pre-assessment and thus had more room to improve on the post-assessment. Regardless, these results suggest that the Git exercises were more effective for students with less Git experience.

We explored how the levels of Git experience affected the scores on the Git exercises by conducting an ANOVA test for the scores on the Git exercises grouped by the various levels of Git experience. The ANOVA test determined a *p-value* of .172. Thus, there is not a statistically significant difference between the mean scores on the Git exercises based on the level of Git experience. This indicates that a student's level of Git experience does not affect their score on the Git exercises. Table 5.22 shows the results of performing a Tukey HSD test on the Git exercises scores grouped by the various levels of Git experience. The difference between the exercises scores for all pairs of Git experience was not statistically significantly different. These results further indicate that a student's level of experience did not affect their score on the Git exercises. Overall, the results from the ANOVA test and the Tukey HSD test suggest that the Git exercises are equally feasible for students with any level of Git experience.

Table 5.22: Comparison of mean Git exercises scores by experience

Categories Pair	Difference	p-value
I use Git regularly. I've had multiple experiences using Git.	-0.282	.998
I use Git regularly. I've used Git once or twice.	0.612	.970
I use Git regularly. I have never used Git or version control before.	1.575	.616
I've had multiple experiences using Git. I've used Git once or twice.	0.894	.842
I've had multiple experiences using Git. I have never used Git or version control before.	1.858	.233
I've used Git once or twice. I have never used Git or version control before.	0.963	.575

5.2.3 Experienced vs Inexperienced

We also analyzed the data by splitting the students into two groups, inexperienced and experienced, based on their experience level with Git. A student is considered inexperienced if they have either only used Git once or twice or never used Git or version control before. A student is considered experienced if they either use Git regularly or have had multiple experiences using Git. Table 5.23 shows the results of a two-sample Welch's t-test comparing students' Git pre-assessment scores, post-assessment scores, assessment score improvements, and exercises scores grouped by inexperienced and experienced students.

Table 5.23: Comparison of mean Git assessments and exercises scores by experience

	Inexperienced	Experienced	*p-value*
Pre/10	3.151	6.246	**<.001*** **
Post/10	8.205	8.885	**.007** **
Improvement	5.054	2.639	**<.001*** **
Exercises/21	16.869	18.328	.050

Experienced students performed statistically significantly better than inexperienced students on the Git pre-assessment and post-assessment. This result is expected because students with more Git experience are expected to perform better on the assessments. Furthermore, contrary to the results from the ANOVA and Tukey HSD tests for the post-assessment, the result suggests that the Git exercises did not reduce the gap in Git knowledge of experienced and inexperienced students because the experienced students still performed statistically significantly better on the Git post-assessment than the inexperienced students. Inexperienced students had a statistically significantly greater score improvement than experienced students from the pre-assessment to the post-assessment. As with the result from the ANOVA and Tukey HSD tests for the score improvements, this result is expected because inexperienced students generally performed worse on the pre-assessment and thus had more room for improvement on the post-assessment. Still, this result suggests that the Git exercises were

more effective for inexperienced students. Experienced students did not perform statistically significantly better than inexperienced students on the Git exercises. This result indicates that the Git exercises are not significantly easier for experienced students.

5.2.4 Question Analysis

Table 5.24 shows the number and percent of students who correctly answered each question on the Git pre- and post-assessments. Q1, Q8, and Q9 were the most missed questions on the Git pre-assessment. Q1, Q8, and Q9 were also the most missed questions on the Git post-assessment.

Table 5.24: Number of students out of 320 who correctly answered each question on the Git pre- and post-assessment

Question	Pre	Post
Q1	55 (17.2%)	211 (65.9%)
Q8	102 (31.9%)	228 (71.2%)
Q9	108 (33.8%)	261 (81.6%)
Q2	117 (36.6%)	292 (91.2%)
Q6	117 (36.6%)	271 (84.7%)
Q5	119 (37.2%)	303 (94.7%)
Q3	120 (37.5%)	263 (82.2%)
Q10	124 (38.8%)	268 (83.8%)
Q7	152 (47.5%)	292 (91.2%)
Q4	183 (57.2%)	278 (86.9%)

Many fewer students correctly answered Q1 compared to any other question on both the Git pre-assessment and post-assessment, which indicates that Q1 is more difficult than the other questions. Figure 5.13 shows Q1 from the Git assessments. The most commonly selected incorrect answer for Q1 on both the Git pre-assessment and post-assessment was: *The commit command commits changes to the remote server.* Students may selected this answer because they confused the git commit command with the git push command. The relatively poor performance on Q1 on the Git post-assessment indicates that the Git exercises

need to better enforce what a commit is and how that is different from pushing to the remote repository.

Figure 5.13: Git Assessment Question 1

Figure 5.14 shows Q8 from the Git assessments. Like Q1, Q8 requires the student to understand the git commit command. This further indicates that the Git exercises need to better enforce how the commit command works.

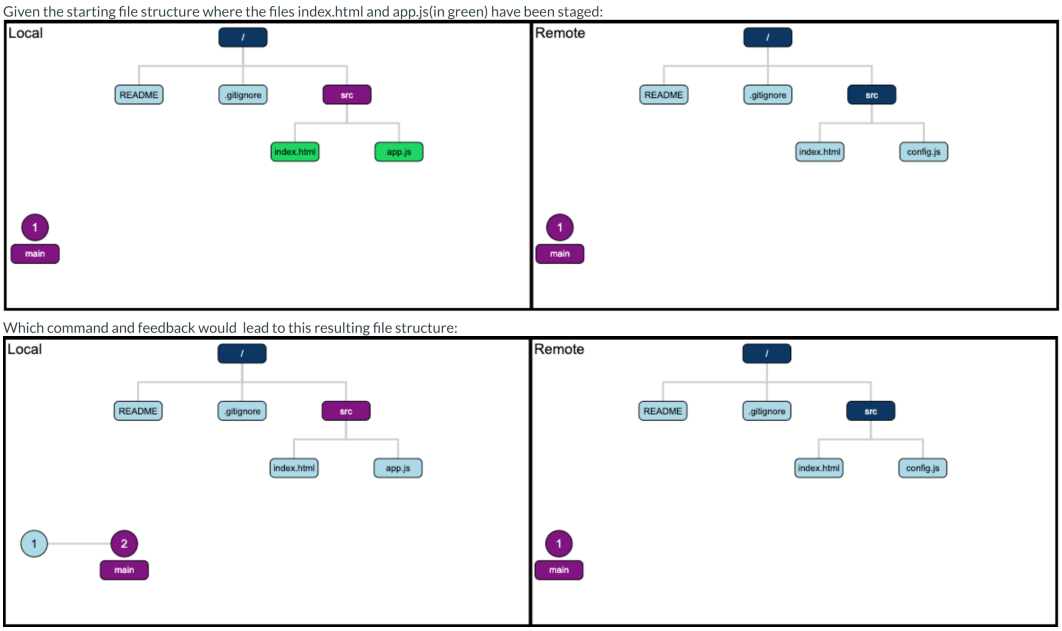


Figure 5.14: Git Assessment Question 8

Figure 5.15 shows Q9 from the Git assessments. Q9 tests the student's understanding of the git pull command, so the relatively poor performance on Q9 on the Git post-assessment indicates that the Git exercises need to better enforce the usage of the git pull command.

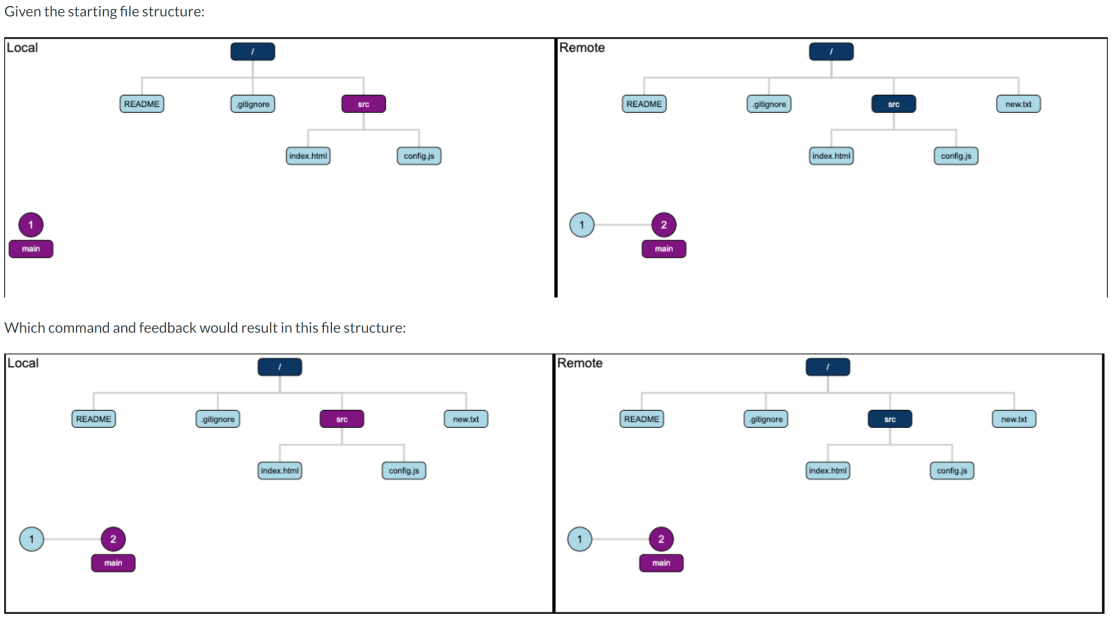


Figure 5.15: Git Assessment Question 9

Despite the scores being lower on certain questions, the number of students who got each question correct improved from the Git pre-assessment to the Git post-assessment for every question, which indicates that the Git exercises helped teach the students all of the concepts tested by the assessments.

We analyzed the number of students who answered each question correctly on the Git pre-assessment for each level of Git experience in order to gain insight into which Git concepts students struggle with before completing our tutorial. Table 5.25 shows the number and percent of students who use Git regularly who correctly answered each question on the Git pre-assessment. For students who use Git regularly, Q1, Q7, and Q8 were the most missed questions on the pre-assessment. The top three most missed questions for this category are similar as for all categories combined except for Q7. Figure 5.16 shows Q7 from the Git assessments. Students who use Git regularly may have struggled with this question on the Git pre-assessment because they are familiar with using git add but may not know or remember the name of the staging area.

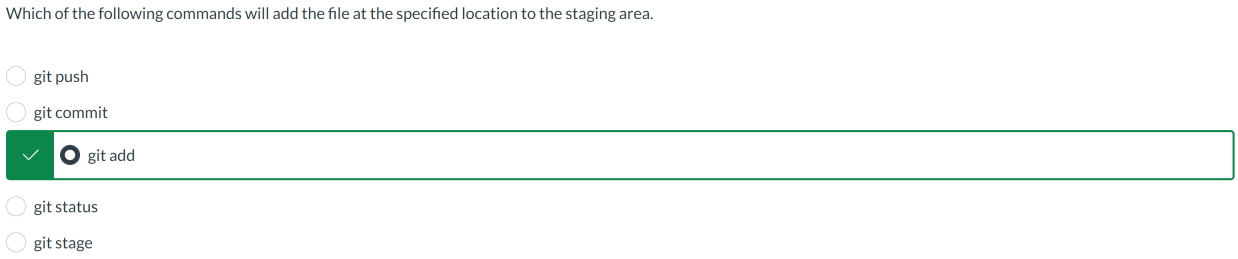


Figure 5.16: Git Assessment Question 7

Table 5.25: Number of students out of 21 who use Git regularly who correctly answered each question on the Git pre-assessment

Question	Pre
Q1	7 (33.3%)
Q7	11 (52.4%)
Q8	11 (52.4%)
Q9	15 (71.4%)
Q2	16 (76.2%)
Q5	17 (81.0%)
Q10	17 (81.0%)
Q6	18 (85.7%)
Q3	19 (90.5%)
Q4	21 (100.0%)

Table 5.26 shows the number and percent of students who have had multiple experiences using Git who correctly answered each question on the Git pre-assessment. For students who have had multiple experiences using Git, Q1, Q5, Q6, and Q8 were the most missed questions on the pre-assessment. The top four most missed questions for this category are similar as for all categories combined and for students who use Git regularly except for Q5 and Q6. Q5 and Q6 required the students to understand git switch and git push, respectively. Students with multiple experiences using Git may have never needed to switch branches before or may have used git checkout instead of git switch and thus did not know the usage of git switch. Another possibility is that students have used git switch and git push before but did not have enough experience with the technical vocabulary used in Git that was needed to answer the questions correctly.

Table 5.26: Number of students out of 40 who have had multiple experiences using Git who correctly answered each question on the Git pre-assessment

Question	Pre
Q1	12 (30.0%)
Q5	17 (42.5%)
Q6	19 (47.5%)
Q8	19 (47.5%)
Q2	24 (60.0%)
Q7	24 (60.0%)
Q9	24 (60.0%)
Q10	24 (60.0%)
Q3	31 (77.5%)
Q4	35 (87.5%)

Table 5.27 shows the number and percent of students who have used Git once or twice who correctly answered each question on the Git pre-assessment. For students who have used Git once or twice, Q1, Q8, and Q5 were the most missed questions on the pre-assessment. The top three most missed questions for this category are similar for students who have had multiple experiences with Git, although the ordering is different.

Table 5.27: Number of students out of 81 who have used Git once or twice who correctly answered each question on the Git pre-assessment

Question	Pre
Q1	13 (16.0%)
Q8	23 (28.4%)
Q5	24 (29.6%)
Q6	29 (35.8%)
Q10	29 (35.8%)
Q9	30 (37.0%)
Q2	33 (40.7%)
Q3	34 (42.0%)
Q7	42 (51.9%)
Q4	46 (56.8%)

Table 5.28 shows the number and percent of students who have never used Git before who correctly answered each question on the Git pre-assessment. For students who have never

used Git, Q1, Q3, and Q9 were the most missed questions on the pre-assessment. The top three most missed questions for this category are similar to all categories combined except for Q3. The top three most missed question for this category are different than all the other categories except for Q1. This difference could be due to students with no experience guessing on questions.

Table 5.28: Number of students out of 178 who have never used Git before who correctly answered each question on the Git pre-assessment

Question	Pre
Q1	23 (12.9%)
Q3	36 (20.2%)
Q9	39 (21.9%)
Q2	44 (24.7%)
Q8	49 (27.5%)
Q6	51 (28.7%)
Q10	54 (30.3%)
Q5	61 (34.3%)
Q7	75 (42.1%)
Q4	81 (45.5%)

Overall, students with all levels of Git experience performed the worst on Q1 and the best on Q4. Although the students performed worse on certain questions, the performance on each question improved from the Git pre-assessment to the Git post-assessment, indicating that the Git exercises helped improve the scores on all the questions.

5.3 Discussion

The following summarizes our main findings from the course deployment:

- The scores on the command line post-assessment were significantly greater than the scores on the command line pre-assessment for all levels of command line experience except for students who use the command line regularly.

- The scores on the Git post-assessment were significantly greater than the scores on the Git pre-assessment for all levels of Git experience.
- As the level of command line experience decreased, the improvement from the command line pre-assessment to the command line post-assessment increased.
- As the level of Git experience decreased, the improvement from the Git pre-assessment to the Git post-assessment increased.
- The scores on the command line post-assessment were not significantly different based on the level of command line experience for most levels of experience.
- The scores on the Git post-assessment were not significantly different based on the level of Git experience.
- The scores on the command line exercises were not significantly different based on the level of command line experience.
- The scores on the Git exercises were not significantly different based on the level of Git experience.

These findings provide insight into RQ1 by supporting that the command line and Git exercises were successful in teaching core command line and Git concepts. Thus, the findings are consistent with recent literature [22, 24, 25, 29, 32] that interactive visualizations are effective for teaching computer science concepts to students. Furthermore, the findings provide insight into RQ2 by suggesting that the exercises were more effective for students with less command line or Git experience. This result may be caused by more experienced students performing better than less experienced students on the pre-assessment, so the more experienced students had less room for improvement. Additionally, the findings provide insight into RQ3 by suggesting that the command line and Git exercises successfully reduced the

gap in students' knowledge of the command line and Git across different levels of experience because the scores on the post-assessments were not significantly different across most levels of experience. Moreover, the findings provide insight into RQ4 by showing that the exercises are feasible for people with any level of command line or Git experience. Overall, the findings support that the exercises are an effective tool for teaching command line and Git concepts to computer science students with any level of command line and Git experience.

Chapter 6

Threats to Validity

6.1 Internal Validity

Given there is only one data point collected for assessing our tool, there is no potential for history or maturation threats. A direct causality between the completion of the exercises and our outcomes cannot be inferred because no distinct control and experimental groups have been identified in this study. The sample size used to assess the tool is large enough to avoid selection bias in the analysis. Because students have diverse academic backgrounds, differential experience may be a potential threat to this experiment.

Another potential threat to the validity of this experiment is that some of the assessment questions contain images from the interactive visualizations. Students have to understand the structure of the visualizations to answer the question correctly. When taking the pre-assessment, students have never seen the visualizations before so the meaning of the visualizations may be confusing. Thus, the students may have performed better on the post-assessment than the pre-assessment for these types of questions because the exercises exposed them to how the visualizations work. Hence, the improvement on these questions may not indicate that the students knew more about the command line or Git after completing the exercises. Instead, they may simply know more about the structure of the visualizations. However, the visualizations were designed to be intuitive and knowledge about the command line and Git should correlate with understanding of the structure of the visualizations. Also,

most questions did not include an image of the interactive visualizations. Therefore, the effect of this potential threat on the validity of this experiment should be minor.

An additional threat to the validity of this experiment is that we do not have data to conclude that the command line and Git skills students learned from completing the exercises will help students use the command line and Git effectively in the future. The improvements from the pre-assessments to the post-assessments show that the students learned the concepts tested in the assessments, but we cannot conclude that students are proficient with the command line and Git after completing the exercises. However, the concepts taught in the exercises were chosen based on the command line and Git concepts taught in previous semesters of CS2104 Introduction to Problem Solving and based on our experience using the command line and Git, so the exercises should provide students with a suitable background for using the command line and Git.

6.2 External Validity

Our findings may not be generalizable to all undergraduate students because our results are based on data for students at only one large research institution. However, students who enroll in introductory CS courses at large research institutions throughout the United States exhibit similar characteristics as our students.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this work, we explored the problem of effectively teaching practical computer science skills to students through exercises with interactive visualizations. We specifically focused on teaching command line and Git skills. Hence, we developed novel command line and Git exercises with interactive visualizations. The command line exercises visualize the file system, and the Git exercises visualize the file system and Git tree for both a local repository and a remote repository. We hypothesized that students who used interactive visualizations to learn command line and Git concepts would show significant knowledge gain on the topics of command line and Git. We implemented the command line and Git exercises in a problem solving CS course. To test knowledge gain, we conducted a pre-assessment before the students completed the exercises and a post-assessment after the students completed the exercises. The results showed that students exhibited significant knowledge gain of both command line and Git concepts on average (Table 5.3 and Table 5.17). Furthermore, the results showed that students with less experience showed greater knowledge gain (Table 5.6 and Table 5.20). Moreover, the results showed that experience level did not affect a student's ability to complete the exercises (Table 5.8 and Table 5.22). Thus, the results suggest that the command line and Git exercises are applicable to students of all levels of experience, and the exercises successfully increased students' command line and Git knowledge. Therefore, the

results of this work support that interactive visualizations can be an effective tool for teaching practical computer science skills. Overall, the results support that we were successful in creating novel command line and Git exercises with interactive visualizations to teach core command line and Git concepts.

7.2 Future Work

We have created 17 command line exercises and 18 Git exercises. These suites of exercises are geared towards teaching students with little to no experience with the command line or Git. Hence, many of the exercises have low levels of complexity. In future work, more complex command line and Git exercises could be created. A suite of exercises could be created for students with more experience that aims to teach them more complicated command line and Git concepts so that they can further master the skills. The command line and Git exercise components are configurable, so future work creating new exercises would be feasible and take limited additional development effort.

Currently, the exercises support many fundamental commands needed to perform common tasks using the command line and Git. In future work, the command line and Git exercises could be expanded to support more commands. For example, the command line exercises could be extended to include piping and input/output redirection functionality. Additionally, the Git exercises could be extended to include commands such as git reset, git merge, and git rebase.

The browser-based command line interface is entirely configurable with regards to the command line prompt and the supported commands. Thus, future work could include the development of other novel interactive visualizations that teach practical computer science skills. For example, an interactive visualization could be developed for Docker and Kubernetes.

The exercises were designed to teach command line and Git concepts to learners who have no experience with command line or Git. All of the testing for the exercises was performed in a computer science course at Virginia Tech, so although some of the students did not have experience with the command line or Git, all students have some computer science background. Thus, future work could be conducted in which the exercises are tested on students from different majors. The exercises could be tested on students from other STEM majors and also from non-STEM majors. The results from such a study would be interesting to explore whether the exercises are effective for students with less technical experience in addition to no command line or Git experience.

We currently do not have data to show that students learned more about command line and Git from completing our exercises than they learned in CS2104 Introduction to Problem Solving in previous semesters. Thus, in future work, we could analyze students' performance on assessment questions related to command line and Git from previous semesters and compare the performance to the semester in which the students used our tutorials. This would allow us to determine whether the tutorials are more effective than methods used to teach command line and Git in previous semesters.

Furthermore, we have not analyzed how long the students spent completing our tutorials. In future work, we could analyze the time spent on the tutorials to gain insight into how long the tutorials take to complete and determine whether the length should be adjusted.

Bibliography

[1] GitHub. URL https://github.com/.

[2] The Command Line. URL https://www.linkedin.com/learning/autocad-2023-essential-training/the-command-line?autoplay=true&u=57888345.

[3] Command Line Crash Course. URL https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Understanding_client-side_tools/Command_line.

[4] Stack Overflow. URL https://stackoverflow.com/.

[5] Command Line Crash Course. URL https://www.youtube.com/watch?v=yz7nYlnXLfE.

[6] GitLab. URL https://about.gitlab.com/.

[7] About - Git. URL https://git-scm.com/about.

[8] Git - gittutorial. URL https://git-scm.com/docs/gittutorial.

[9] Bandit. URL https://overthewire.org/wargames/bandit/bandit0.html.

[10] Command Challenge. URL https://cmdchallenge.com/.

[11] Learn Git and Github. URL https://www.codecademy.com/courses/learn-git/lessons/git-workflow/exercises/git-commit.

[12] D3.js. URL https://d3js.org/.

[13] jQuery. URL https://jquery.com/.

[14] Git from Scratch. URL https://www.linkedin.com/learning/git-from-scratch/git-from-scratch?autoplay=true&u=57888345.

[15] Terminus. URL https://web.mit.edu/mprat/Public/web/Terminus/Web/main.html.

[16] OpenDSA. URL https://opendsa-server.cs.vt.edu/.

[17] Stack Overflow Developer Survery 2022. URL https://survey.stackoverflow.co/2022/.

[18] Visualizing Git. URL https://git-school.github.io/visualizing-git/.

[19] Learn the Command Line. URL https://www.codecademy.com/courses/learn-the-command-line/lessons/navigation.

[20] Git Crash Course. URL https://www.youtube.com/watch?v=RGOj5yH7evk.

[21] Sukru Eraslan, Julio César Cortés Rios, Kamilla Kopec-Harding, Suzanne M Embury, Caroline Jay, Christopher Page, and Robert Haines. Errors and poor practices of software engineering students in using git. In *Proceedings of the 4th Conference on Computing Education Practice 2020*, pages 1–4, 2020.

[22] Mohammed F. Farghally, Kyu Han Koh, Hossameldin Shahin, and Clifford A. Shaffer. Evaluating the effectiveness of algorithm analysis visualizations. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, page 201–206, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346986. doi: 10.1145/3017680.3017698. URL https://doi.org/10.1145/3017680.3017698.

[23] Lassi Haaranen and Teemu Lehtinen. Teaching git on the side: Version control system as a course platform. In *Proceedings of the 2015 ACM conference on innovation and technology in computer science education*, pages 87–92, 2015.

[24] Sally Hamouda, Stephen H. Edwards, Hicham G. Elmongui, Jeremy V. Ernst, and Clifford A. Shaffer. Recurtutor: An interactive tutorial for learning recursion. *ACM Trans. Comput. Educ.*, 19(1), nov 2018. doi: 10.1145/3218328. URL https://doi.org/10.1145/3218328.

[25] Sally Hamouda, Stephen H Edwards, Hicham G Elmongui, Jeremy V Ernst, and Clifford A Shaffer. Btrecurtutor: a tutorial for practicing recursion in binary trees. *Computer Science Education*, 30(2):216–248, 2020.

[26] Sara Hooshangi, Ryan Buxton, and Margaret Ellis. Integration of practical computing skills and co-curricular activities in the curriculum. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, pages 61–67, 2022.

[27] Ville Isomöttönen and Michael Cochez. Challenges and confusions in learning version control with git. In *International Conference on Information and Communication Technologies in Education, Research, and Industrial Applications*, pages 178–193. Springer, 2014.

[28] Genevieve Milliken, Sarah Nguyen, and Vicky Steeves. A behavioral approach to understanding the git experience. In *Proceedings of the 54th Hawaii International Conference on System Sciences*.

[29] Mostafa Mohammed, Clifford A. Shaffer, and Susan H. Rodger. Teaching formal languages with visualizations and auto-graded exercises. In *Proceedings of the 52nd ACM*

Technical Symposium on Computer Science Education, SIGCSE '21, page 569–575, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380621. doi: 10.1145/3408877.3432398. URL https://doi.org/10.1145/3408877.3432398.

[30] Peter Cottle. Learn Git Branching. URL https://learngitbranching.js.org.

[31] Vincent Tunru. A Grip On Git. URL https://agripongit.vincenttunru.com/.

[32] Zijie J. Wang, Robert Turko, Omar Shaikh, Haekyu Park, Nilaksh Das, Fred Hohman, Minsuk Kahng, and Duen Horng Polo Chau. Cnn explainer: Learning convolutional neural networks with interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1396–1406, 2021. doi: 10.1109/TVCG.2020.3030418.

Appendices

Appendix A

Pre-Assessment and Post-Assessment Questions

A.1 Command Line Experience Question

Figure A.1 shows the question that asked the students their level of experience with the command line.

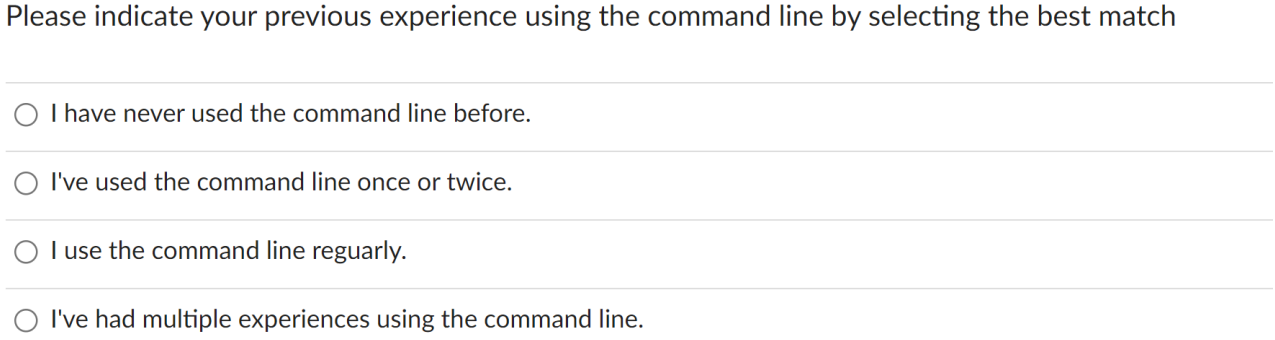


Figure A.1: Command Line Experience Question

A.2 Command Line Assessment Questions

The following figures show the questions from the command line assessments. Some questions are split into multiple figures due to the size of the questions. The correct answer is indicated by the checkmark and the green border.

Figure A.2: Command Line Assessment Question 1

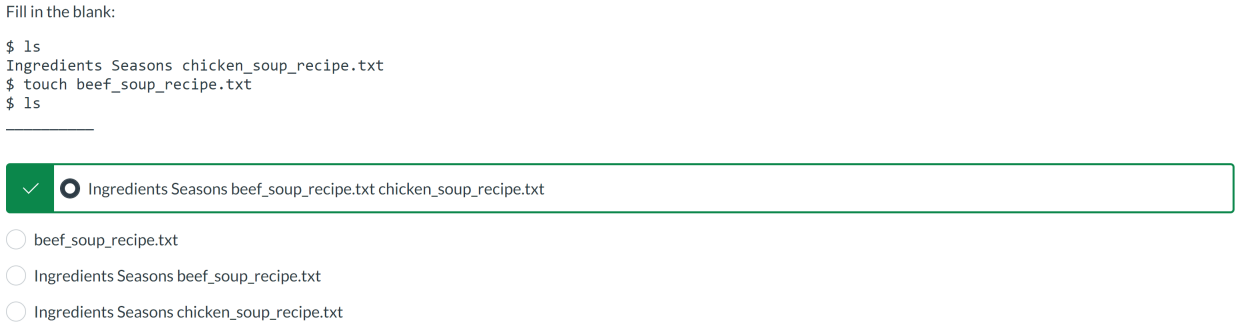


Figure A.3: Command Line Assessment Question 2

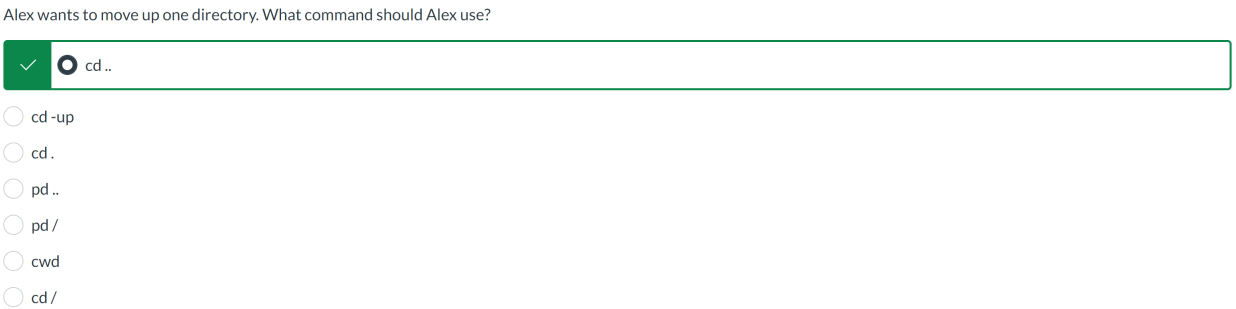


Figure A.4: Command Line Assessment Question 3

Given the file structure below, running the pwd command results in the following output: /mammals/dogs
What will be the output when the command "ls" is run right after pwd command?

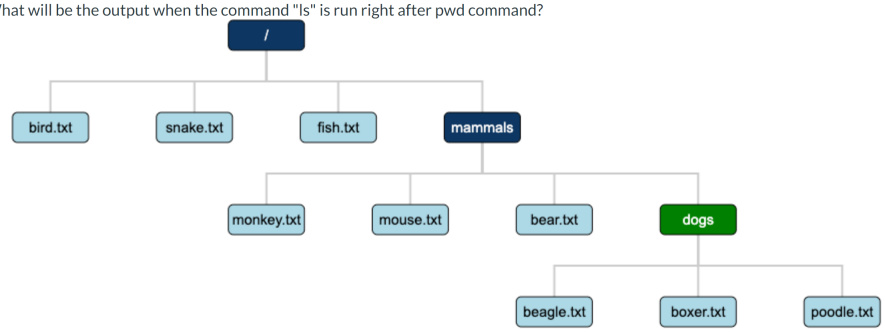


Figure A.5: Command Line Assessment Question 4

bear.txt
monkey.txt
mouse.txt
dogs/

bird.txt
fish.txt
snake.txt
mammals/

beagle.txt
boxer.txt
poodle.txt

bear.txt
monkey.txt
mouse.txt

Figure A.6: Command Line Assessment Question 4 Answer Choices

Given the file structure below, running the pwd command results in the following output: /mammals/dogs
What will be the output when the following two commands are run back to back:

```
$cd ..
$ls
```

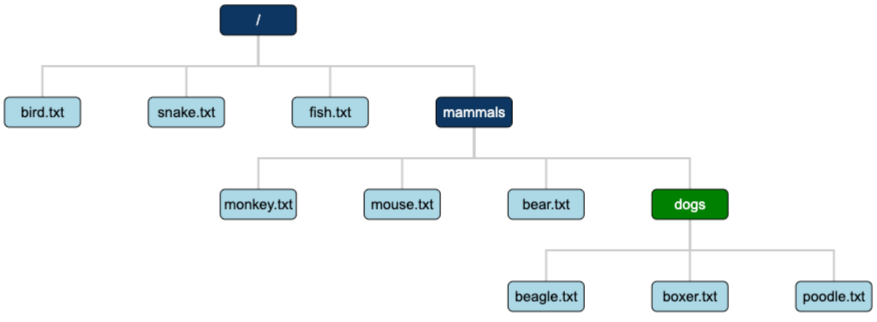


Figure A.7: Command Line Assessment Question 5

beagle.txt
boxer.txt
poodle.txt

beagle.txt
boxer.txt
poodle.txt
dogs/

bird.txt
fish.txt
snake.txt
mammals/

bear.txt
monkey.txt
mouse.txt
dogs/

Figure A.8: Command Line Assessment Question 5 Answer Choices

Given the file structure below, running the pwd command results in the following output: /mammals
What will be the output when the following two commands are run back to back:

```
$touch iris.txt
$ls
```

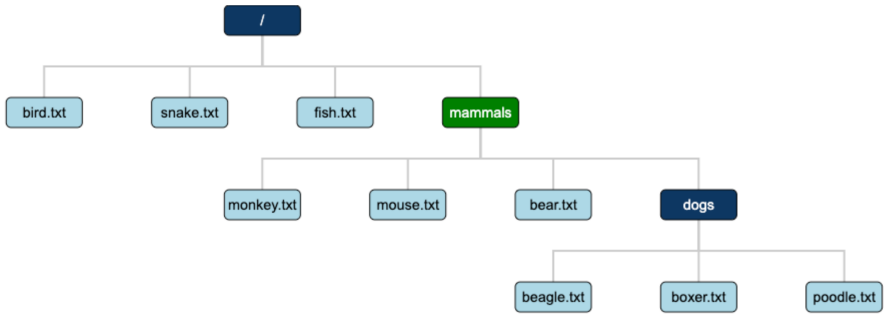


Figure A.9: Command Line Assessment Question 6

iris.txt

bear.txt
iris.txt
monkey.txt
mouse.txt
dogs/

bird.txt
fish.txt
iris.txt
snake.txt
mammals/

beagle.txt
boxer.txt
iris.txt
poodle.txt

Figure A.10: Command Line Assessment Question 6 Answer Choices

Given the file structure used in previous questions, the pwd command results in the following output:
/mammals/dogs
What would the file structure look like if the command "mkdir folder1" is run and then the user navigates to the root directory?

Figure A.11: Command Line Assessment Question 7

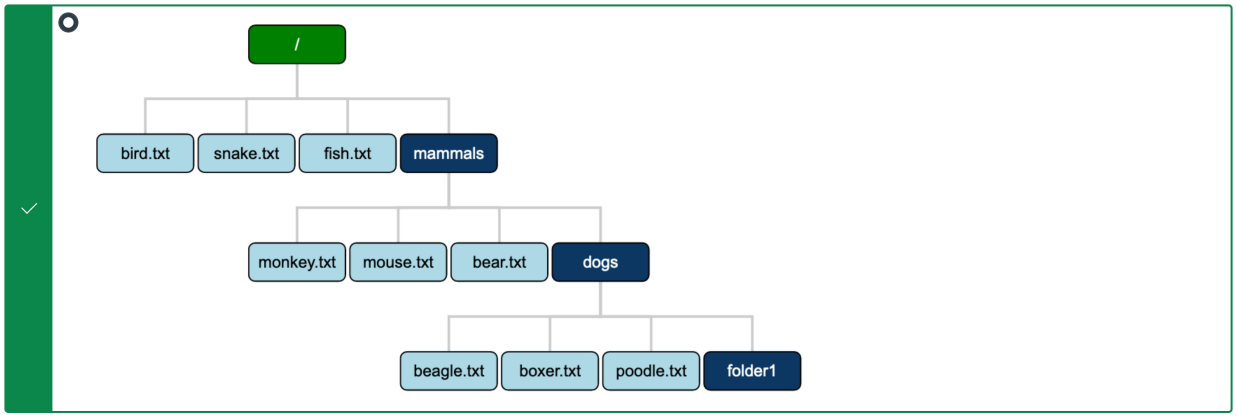


Figure A.12: Command Line Assessment Question 7 Answer Choice 1

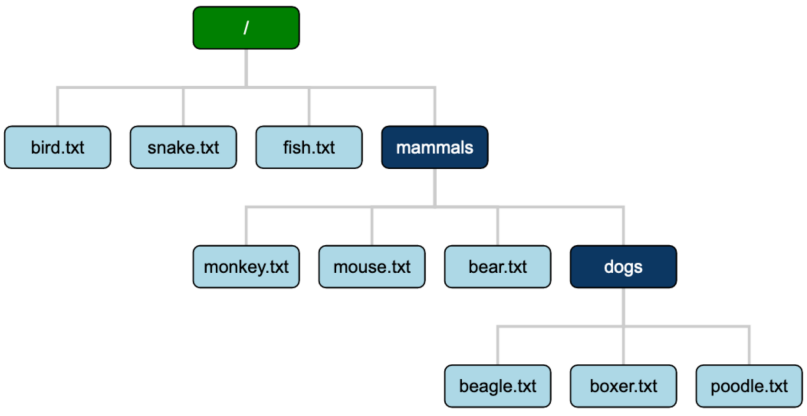


Figure A.13: Command Line Assessment Question 7 Answer Choice 2

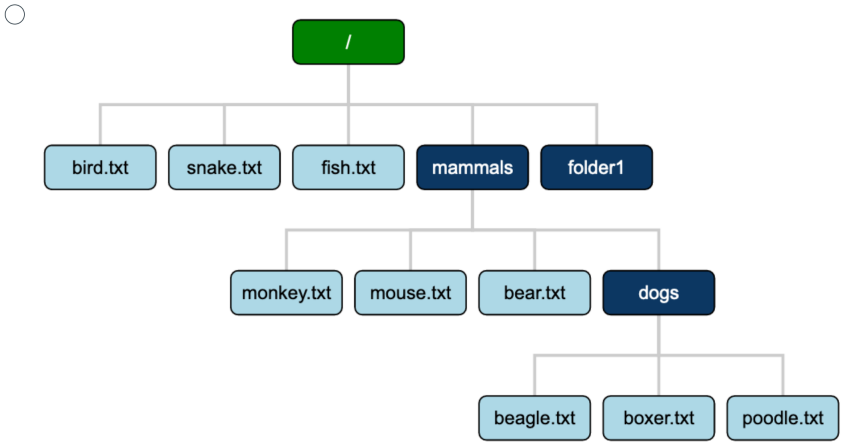


Figure A.14: Command Line Assessment Question 7 Answer Choice 3

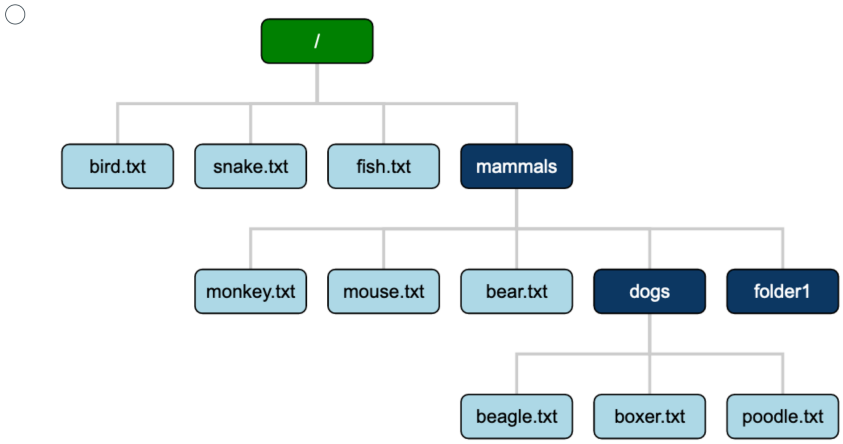


Figure A.15: Command Line Assessment Question 7 Answer Choice 4

Assume ___ is the name of a directory in the current directory. The command rm -r ___ does the following:

- ✓ Removes the directory specified at ___ from the working directory.
- Removes only the subdirectories of the specified directory.
- Removes the directory specified at ___ from the working directory only if it is empty.
- Removes the entire contents of the current directory.
- Removes all the files that are alphabetically greater than item ___ from working directory.

Figure A.16: Command Line Assessment Question 8

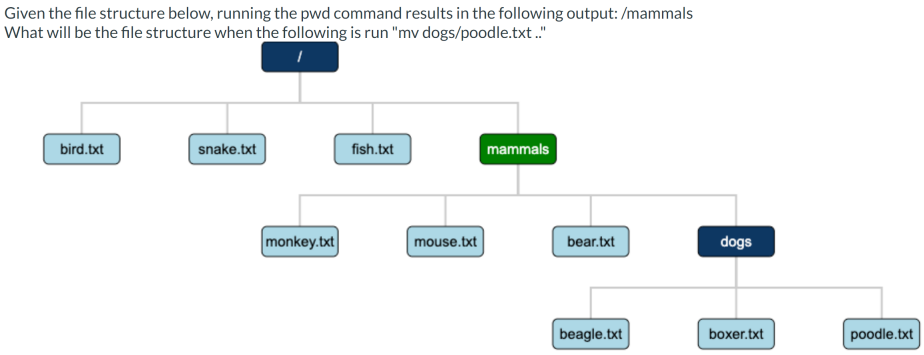


Figure A.17: Command Line Assessment Question 9

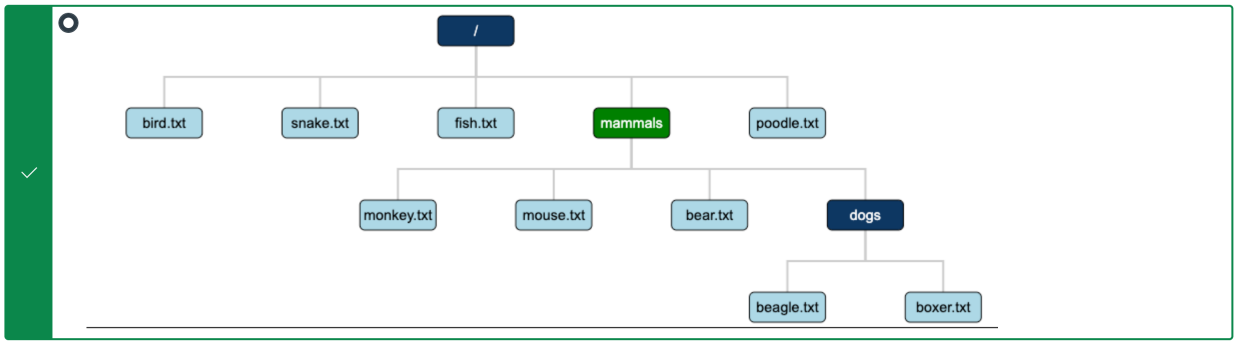


Figure A.18: Command Line Assessment Question 9 Answer Choice 1

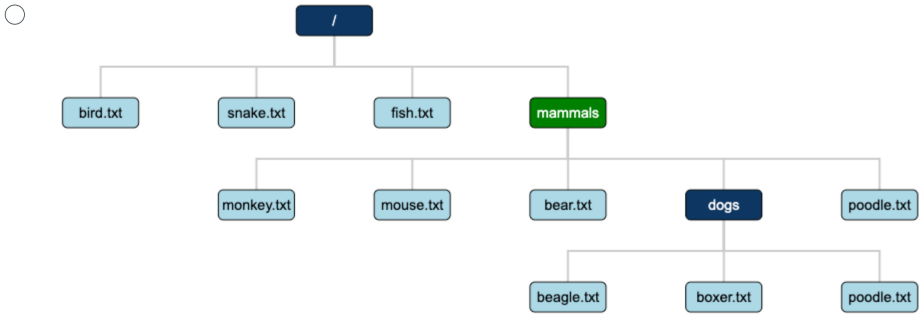


Figure A.19: Command Line Assessment Question 9 Answer Choice 2

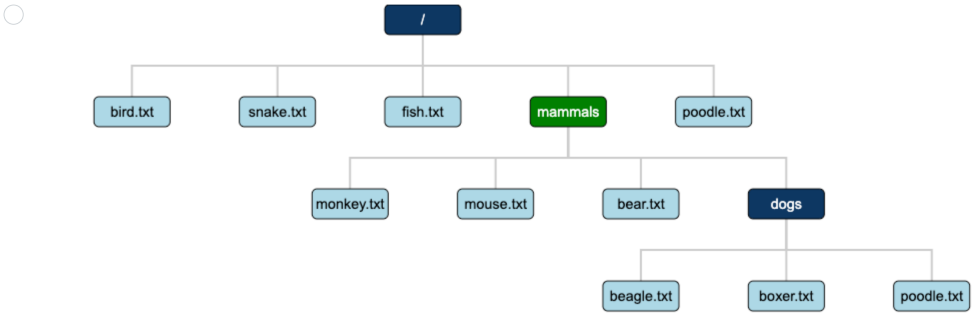


Figure A.20: Command Line Assessment Question 9 Answer Choice 3

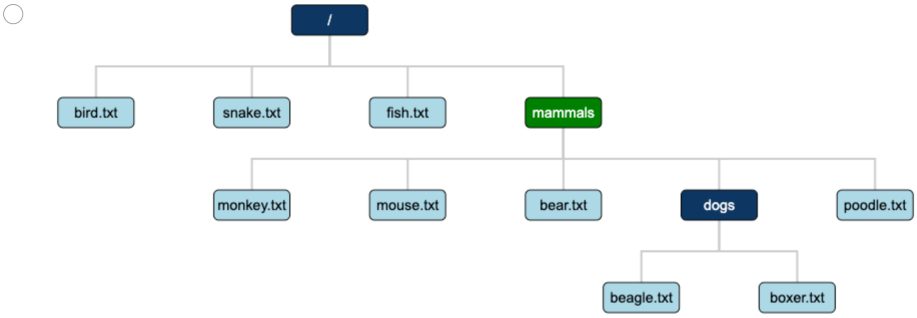


Figure A.21: Command Line Assessment Question 9 Answer Choice 4

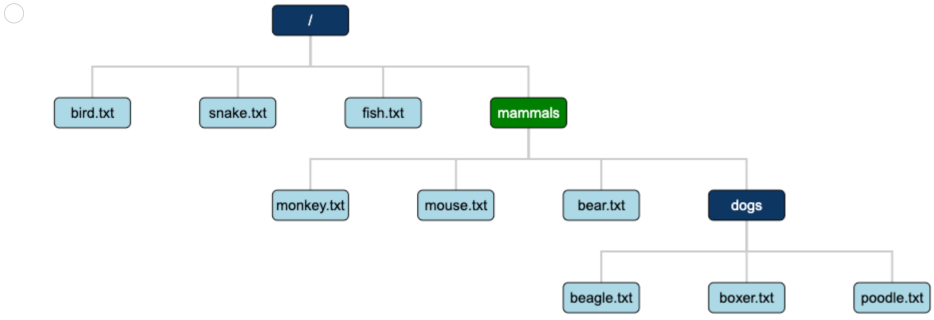


Figure A.22: Command Line Assessment Question 9 Answer Choice 5

Given the file structure below, running the pwd command results in the following output: /mammals
What will be the result when the following commands are run back to back:

```
$ mkdir cats
$ cp dogs/beagle.txt cats/brindle.txt
$ rm -r dogs
```

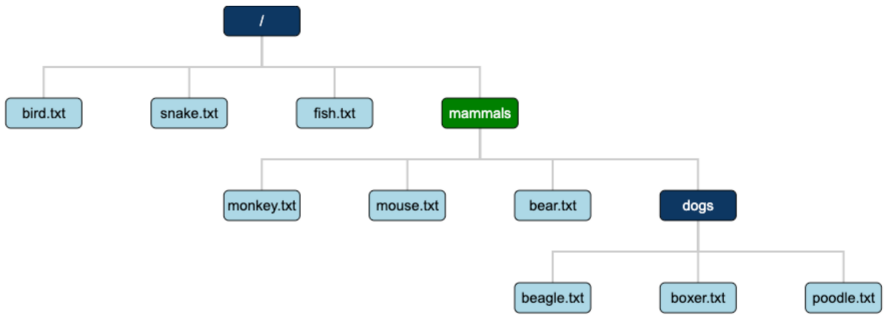


Figure A.23: Command Line Assessment Question 10

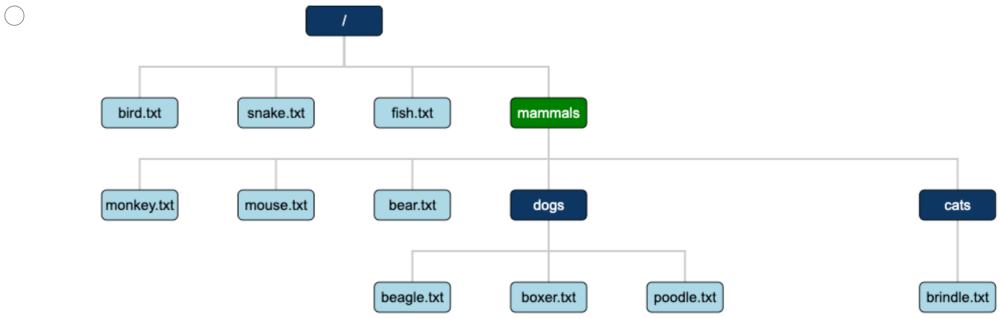


Figure A.24: Command Line Assessment Question 10 Answer Choice 1

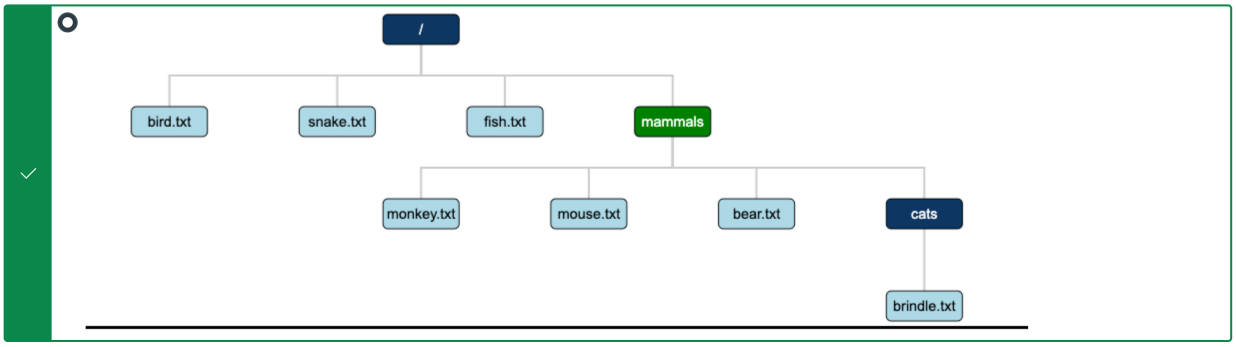


Figure A.25: Command Line Assessment Question 10 Answer Choice 2

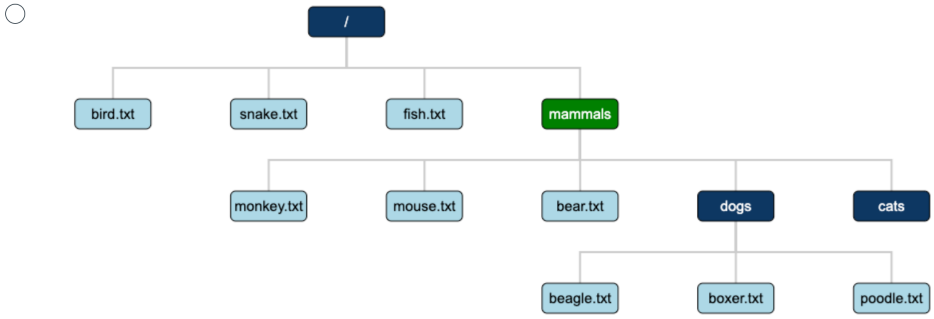


Figure A.26: Command Line Assessment Question 10 Answer Choice 3

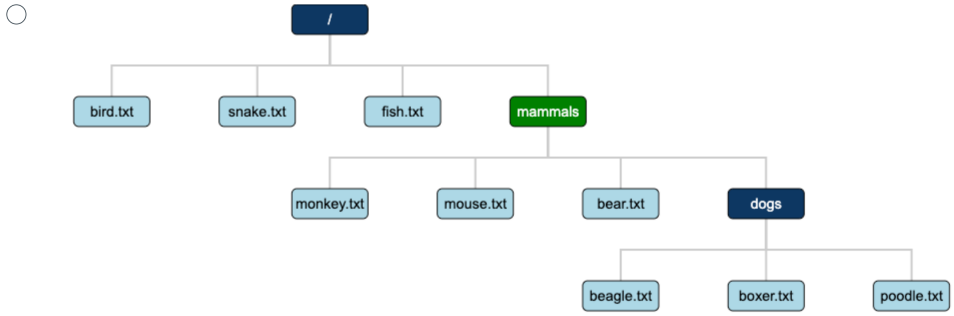


Figure A.27: Command Line Assessment Question 10 Answer Choice 4

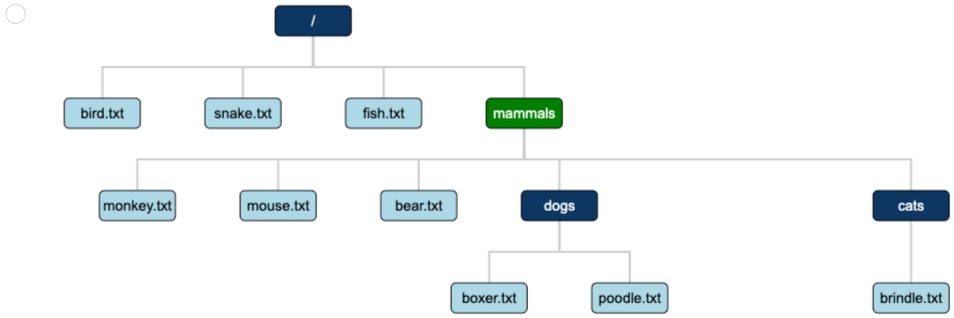


Figure A.28: Command Line Assessment Question 10 Answer Choice 5

A.3 Git Experience Question

Figure A.29 shows the question that asked the students their level of experience with Git.

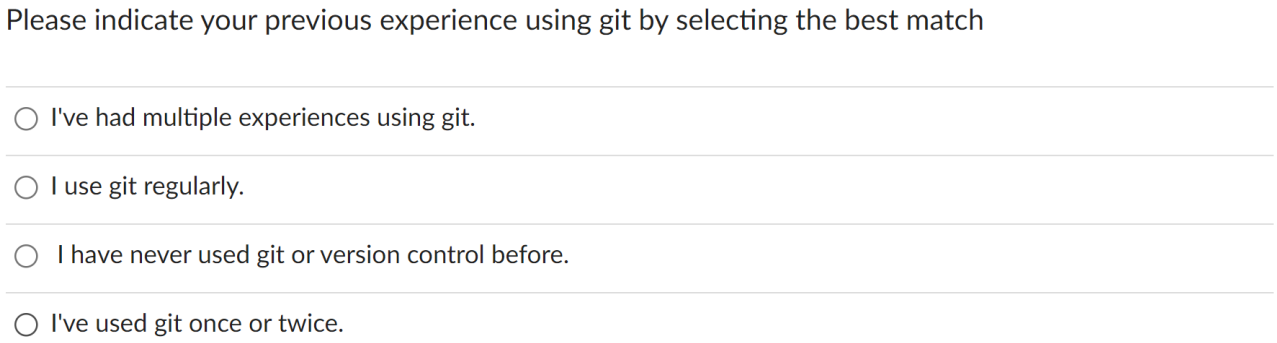


Figure A.29: Git Experience Question

A.4 Git Assessment Questions

The following figures show the questions from the Git assessments. Some questions are split into multiple figures due to the size of the questions. The correct answer is indicated by the checkmark and the green border.

Figure A.30: Git Assessment Question 1

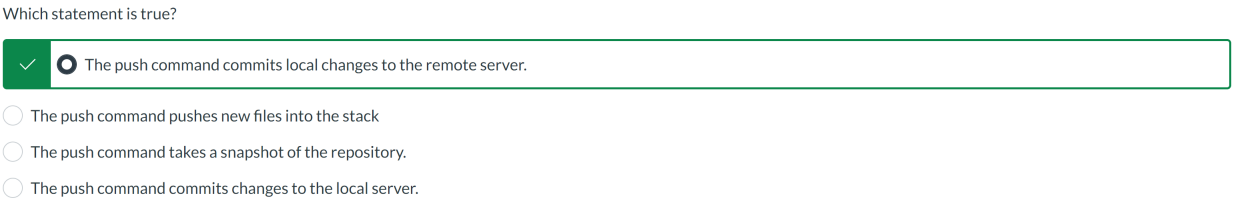


Figure A.31: Git Assessment Question 2

Figure A.32: Git Assessment Question 3

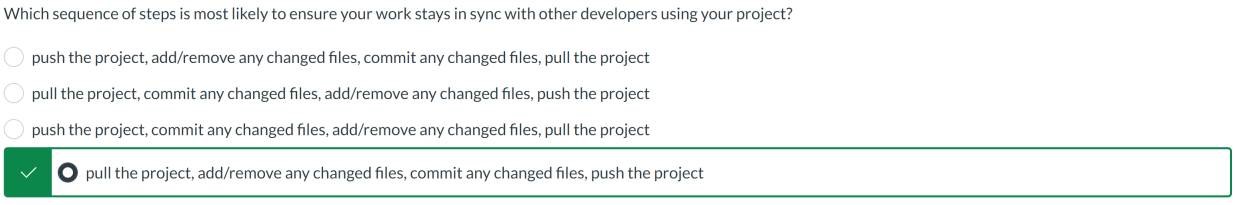


Figure A.33: Git Assessment Question 4

Which of the following commands will change the current branch to the branch with the specified name.

- git cb
- git status
- git cd
- ✓ git switch

Figure A.34: Git Assessment Question 5

Which of the following commands will copy the current branch of the local repository to the corresponding branch of the remote repository.

- git status
- git add
- git switch
- ✓ git push
- git commit

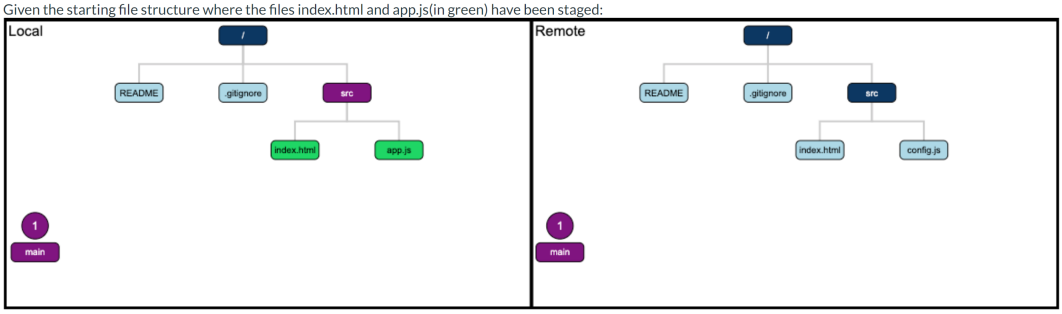
Figure A.35: Git Assessment Question 6

Which of the following commands will add the file at the specified location to the staging area.

- git push
- git commit
- ✓ git add
- git status
- git stage

Figure A.36: Git Assessment Question 7

Given the starting file structure where the files index.html and app.js(in green) have been staged:



Which command and feedback would lead to this resulting file structure:

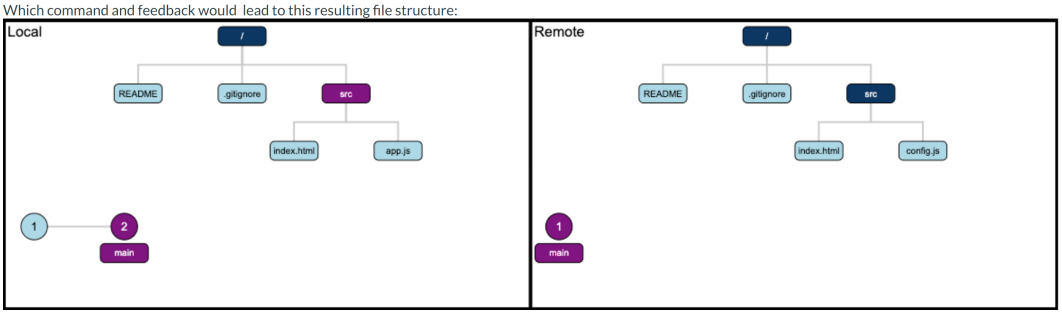


Figure A.37: Git Assessment Question 8

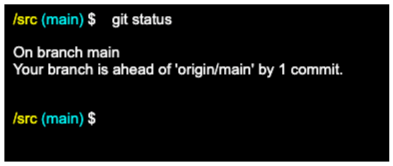


Figure A.38: Git Assessment Question 8 Answer Choices

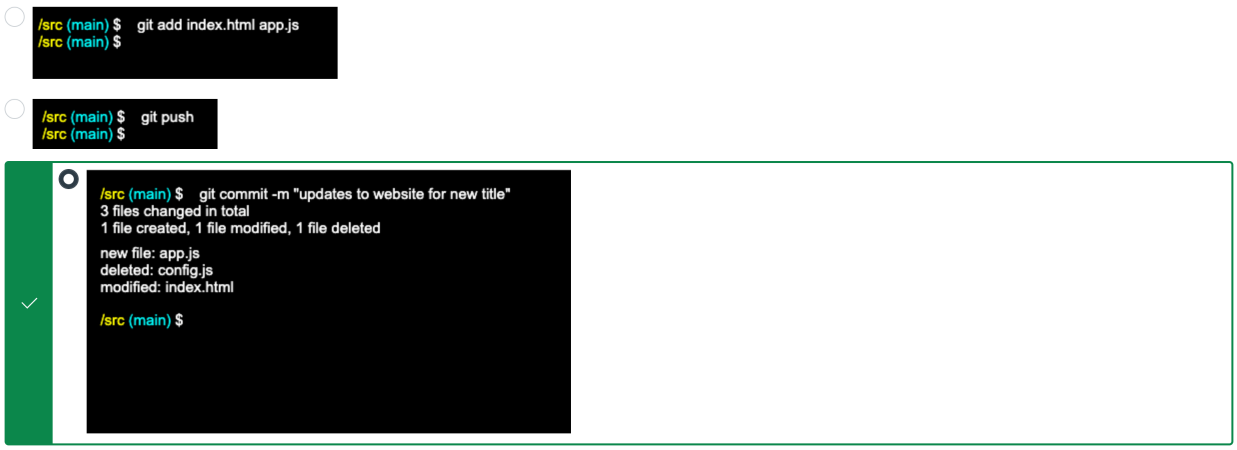
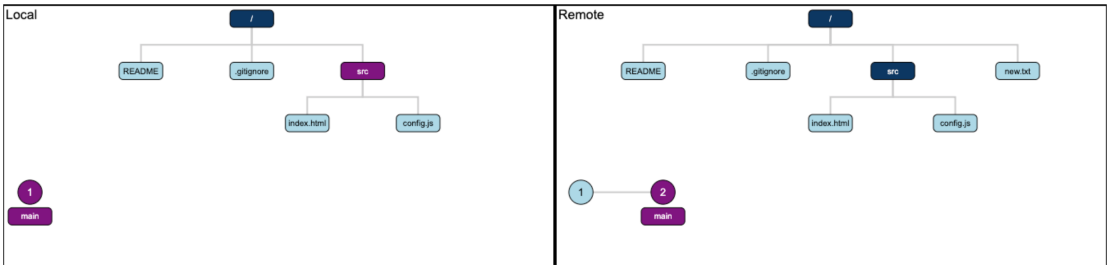


Figure A.39: Git Assessment Question 8 Answer Choices Continued

Given the starting file structure:



Which command and feedback would result in this file structure:

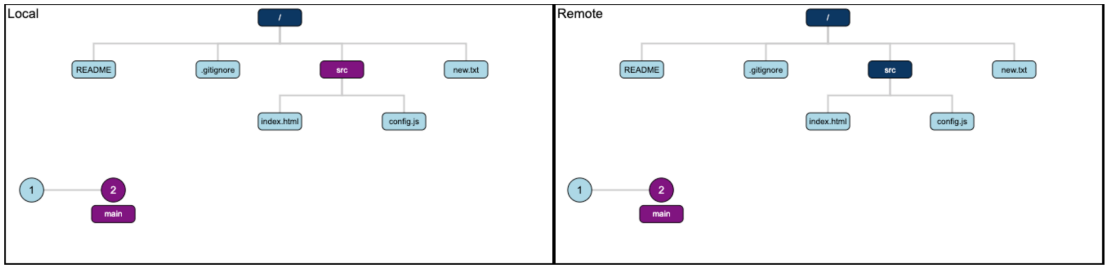


Figure A.40: Git Assessment Question 9

```
/src (main) $   git push
/src (main) $
```

```
/src (main) $   git status

On branch main
Your branch is behind 'origin/main' by 1 commit.

/src (main) $
```

```
/src (main) $   git status

On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
      new file:     app.js
      deleted:      config.js
      modified:     index.html

/src (main) $
```

Figure A.41: Git Assessment Question 9 Answer Choices

```
/src (main) $   git pull
/src (main) $
```

```
/src (main) $   git push
Nothing to push. Everything up to date

/src (main) $
```

Figure A.42: Git Assessment Question 9 Answer Choices Continued

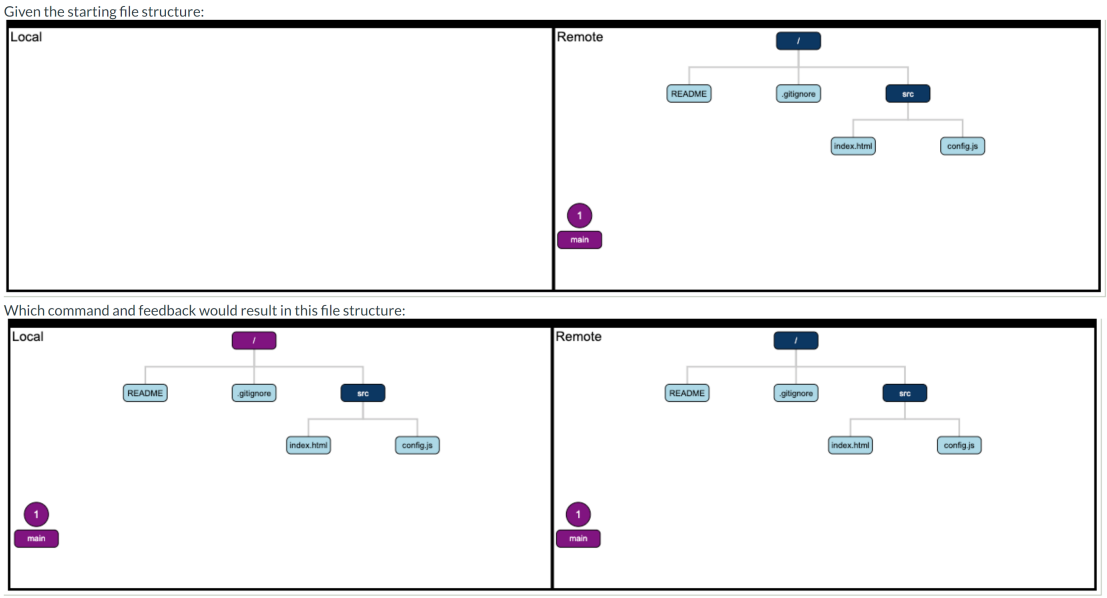


Figure A.43: Git Assessment Question 10

Figure A.44: Git Assessment Question 10 Answer Choices