Batch: A1	**Roll No.: 16010123012**
Experiment / assignment / tutorial No.: 01	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Implementation of Version control System.

AIM: Implementation of Version Control System (Git & GitHub)

Objective: Understand and utilize Git for version control.

- **Tasks:**
  - Initialize a Git repository.
  - Perform basic Git operations: add, commit, push, pull.
  - Create and switch branches.
  - Resolve merge conflicts.
  - Collaborate using pull requests on GitHub

Problem Definition:

The goal of this assignment is to gain practical, hands-on experience with Git and GitHub—essential tools for modern software development and collaboration. Students will explore core Git operations by completing curated interactive exercises from the Git Exercises website (https://gitexercises.fracz.com/). They will demonstrate their understanding of Git commands, branch management, resolving merge conflicts, and contributing to remote repositories hosted on GitHub. The focus is on mastering the Git workflow, from repository initialization to managing collaborative development using pull requests.

Resources used:
https://git-scm.com/docs

Expected OUTCOME of Experiment:
CO1: Build full stack applications in JavaScript using the MERN technologies.

CO5: Deploy MERN applications to cloud platforms like Heroku or AWS and collaborate using GitHub version control.

Books/ Journals/ Websites referred:

1. Shelly Powers Learning Node O' Reilly 2nd Edition, 2016.

Pre Lab/ Prior Concepts:

Implementation Details:

Students have to write stepwise details of implementation.

Exercise Completion Report (PDF or Word)

- For each exercise from gitexercises.fracz.com:
  - Exercise name
  - Objective
  - Steps taken
  - Git commands used
  - Outcome (with sample logs/screenshots)
  - Reflections/Learning

Steps for execution:

1. First type the following into terminal:
2. git clone https://gitexercises.fracz.com/git/exercises.git
3. cd exercises
4. git config user.name "Your name here"
5. git config user.email "Your e-mail here"
6. ./configure.sh
7. git start
8. Then the exercise will start. Steps to complete each command/task has been given above.

Implementation -

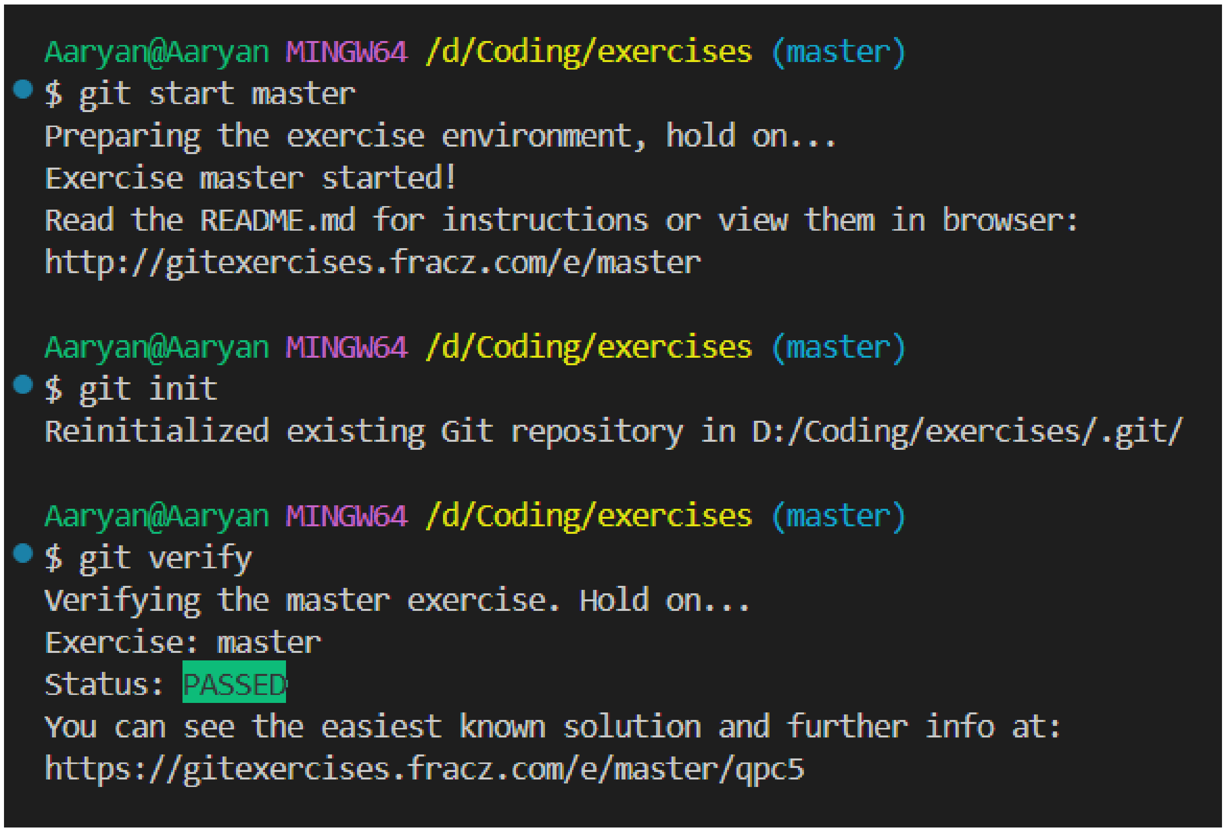
1. **Push a commit you have made**

   Objective: The first exercise is to push a commit that is created when you run the git start command.

   1) Enter "git start master"

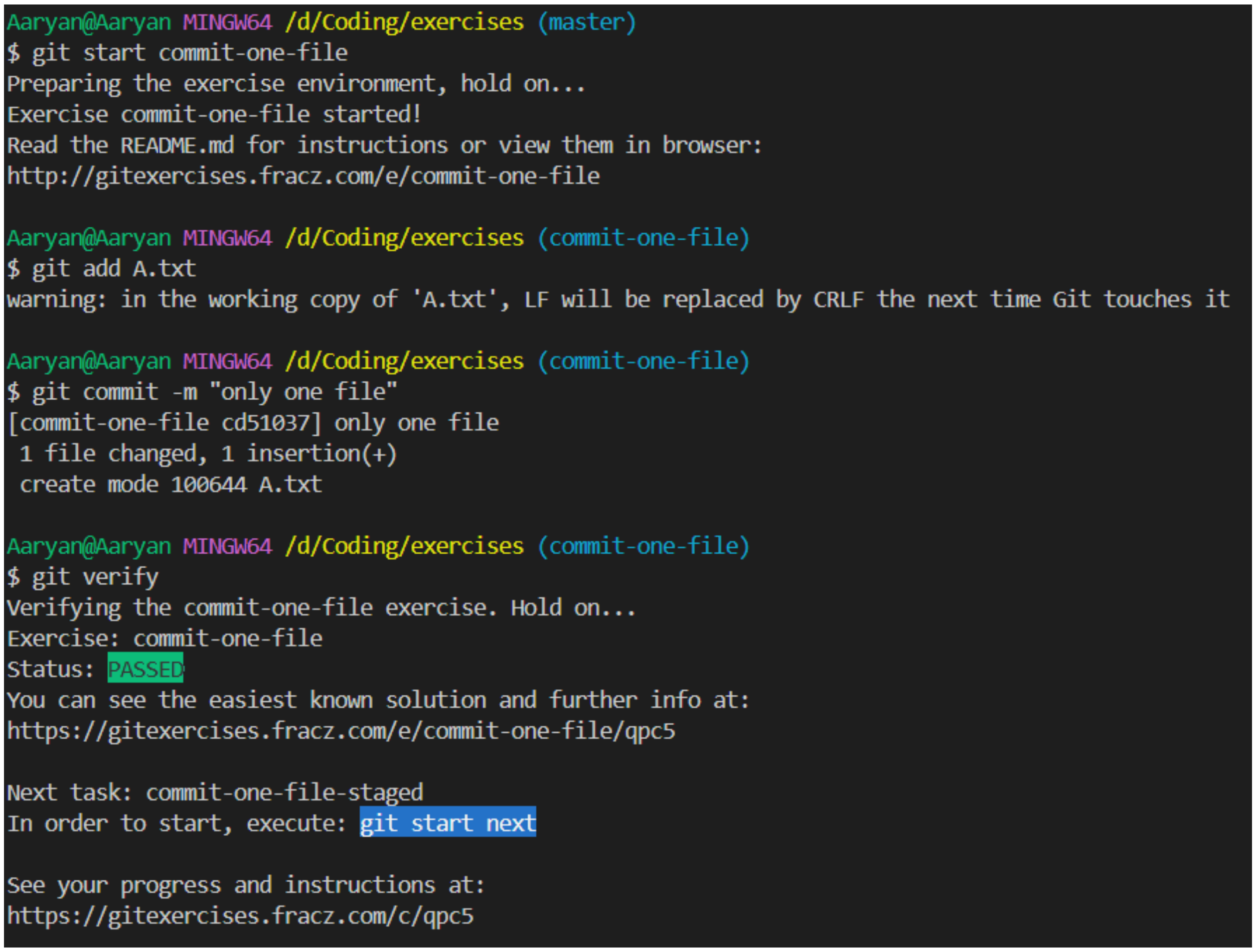
   2) enter "git verify"

   Git init - Initializes a new, empty Git repository in the current directory



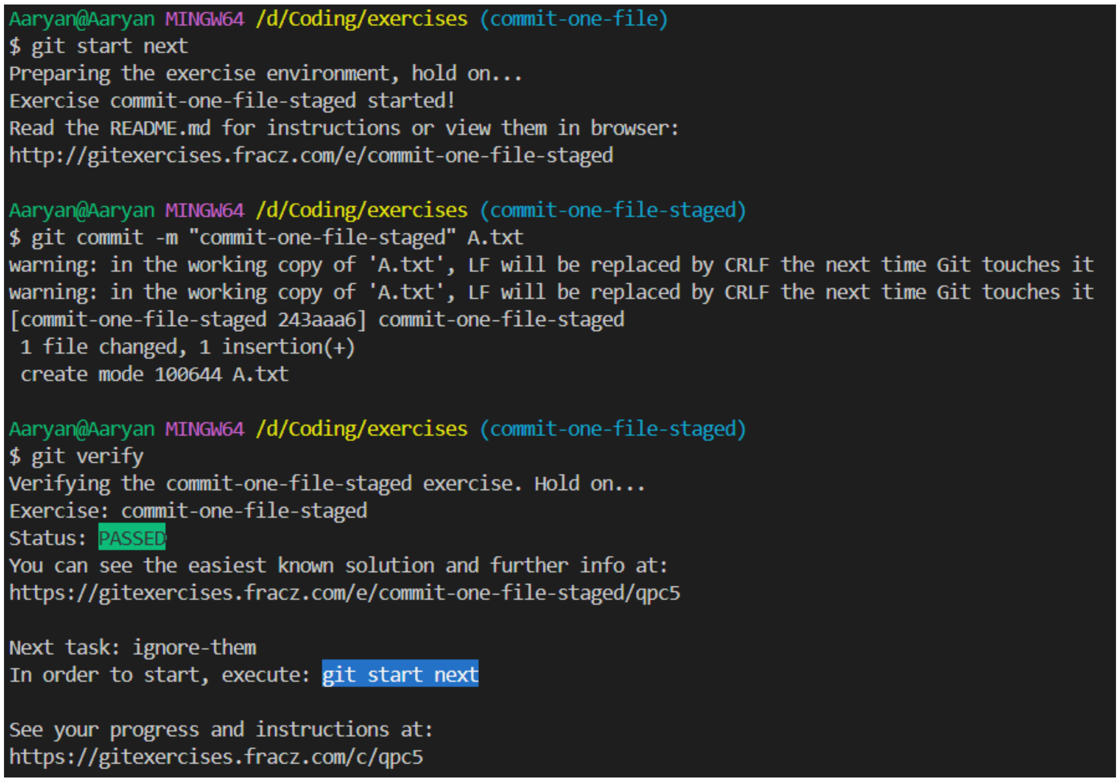
```
Aaryan@Aaryan MINGW64 /d/Coding/exercises (master)
$ git start master
Preparing the exercise environment, hold on...
Exercise master started!
Read the README.md for instructions or view them in browser:
http://gitexercises.fracz.com/e/master

Aaryan@Aaryan MINGW64 /d/Coding/exercises (master)
$ git init
Reinitialized existing Git repository in D:/Coding/exercises/.git/

Aaryan@Aaryan MINGW64 /d/Coding/exercises (master)
$ git verify
Verifying the master exercise. Hold on...
Exercise: master
Status: PASSED
You can see the easiest known solution and further info at:
https://gitexercises.fracz.com/e/master/qpc5
```

2. **Commit one file**
   Objective: There are two files created in the root project directory - A.txt and B.txt. The goal is to commit only one of them.
   Steps taken: first, git start commit-one-file. Then, add any one file, here i added A.txt. Then commit that file.
   Git commands used: git start commit-one-file, git add A.txt, git commit -m "Commit one file"
   git add `file_name` - Stages changes in a file, preparing them to be included in the next commit.
   git commit -m "Commit msg" - Records the staged changes as a new snapshot in the local repository's history with the provided message.

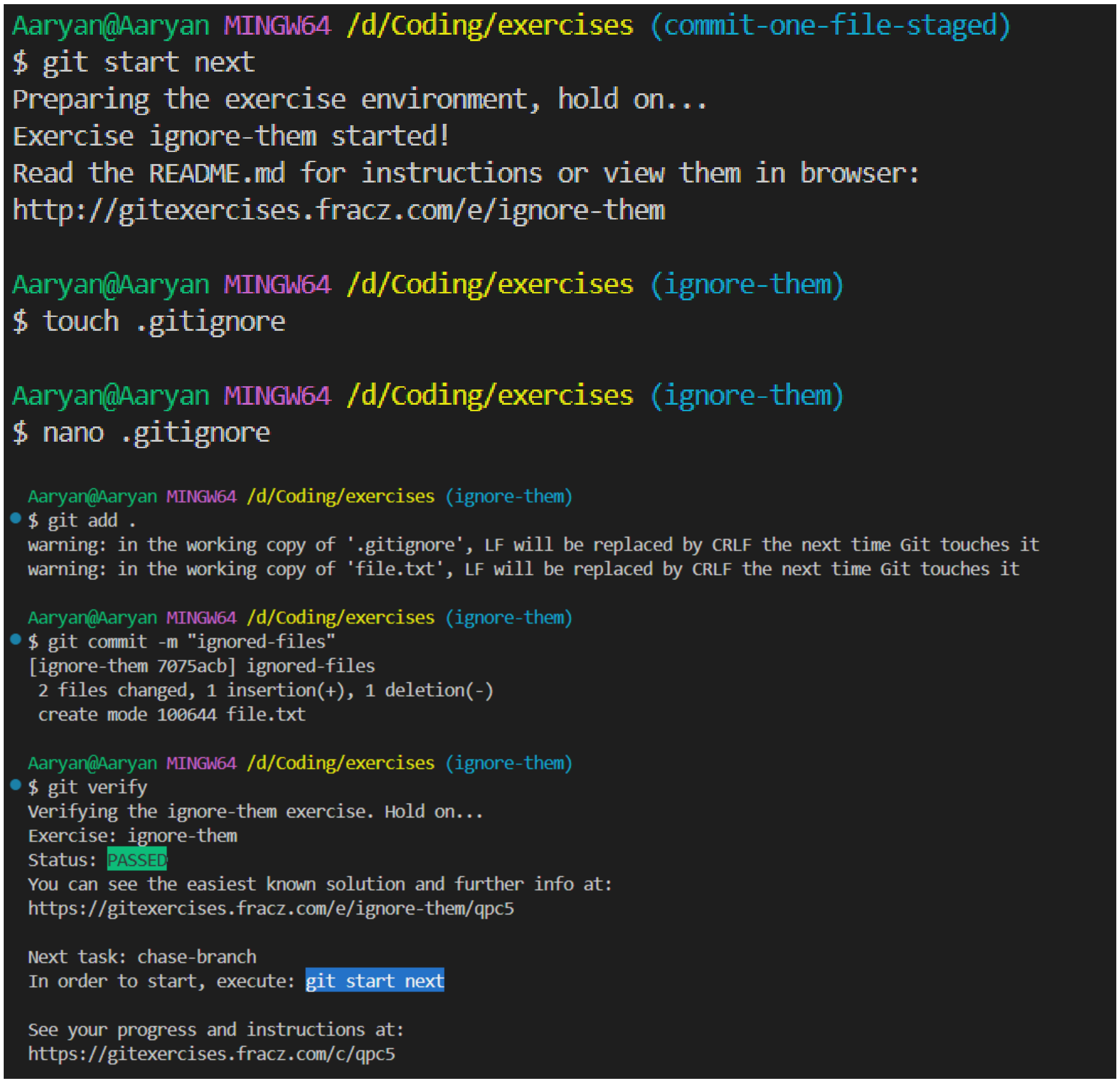


```
Aaryan@Aaryan MINGW64 /d/Coding/exercises (master)
$ git start commit-one-file
Preparing the exercise environment, hold on...
Exercise commit-one-file started!
Read the README.md for instructions or view them in browser:
http://gitexercises.fracz.com/e/commit-one-file

Aaryan@Aaryan MINGW64 /d/Coding/exercises (commit-one-file)
$ git add A.txt
warning: in the working copy of 'A.txt', LF will be replaced by CRLF the next time Git touches it

Aaryan@Aaryan MINGW64 /d/Coding/exercises (commit-one-file)
$ git commit -m "only one file"
[commit-one-file cd51037] only one file
 1 file changed, 1 insertion(+)
 create mode 100644 A.txt

Aaryan@Aaryan MINGW64 /d/Coding/exercises (commit-one-file)
$ git verify
Verifying the commit-one-file exercise. Hold on...
Exercise: commit-one-file
Status: PASSED
You can see the easiest known solution and further info at:
https://gitexercises.fracz.com/e/commit-one-file/qpc5

Next task: commit-one-file-staged
In order to start, execute: git start next

See your progress and instructions at:
https://gitexercises.fracz.com/c/qpc5
```

3. **Commit one file of two currently staged**
   Objective: There are two files created in the root project directory - A.txt and B.txt. They are both added to the staging area. The goal is to commit only one of them.
   Steps taken: git start commit-one-file-staged, then remove one of the added files using reset command. Then commit the other file.
   git reset file - Unstages a file, removing it from the staging area but preserving the changes in your working directory.


```
Aaryan@Aaryan MINGW64 /d/Coding/exercises (commit-one-file)
$ git start next
Preparing the exercise environment, hold on...
Exercise commit-one-file-staged started!
Read the README.md for instructions or view them in browser:
http://gitexercises.fracz.com/e/commit-one-file-staged

Aaryan@Aaryan MINGW64 /d/Coding/exercises (commit-one-file-staged)
$ git commit -m "commit-one-file-staged" A.txt
warning: in the working copy of 'A.txt', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'A.txt', LF will be replaced by CRLF the next time Git touches it
[commit-one-file-staged 243aaa6] commit-one-file-staged
 1 file changed, 1 insertion(+)
 create mode 100644 A.txt

Aaryan@Aaryan MINGW64 /d/Coding/exercises (commit-one-file-staged)
$ git verify
Verifying the commit-one-file-staged exercise. Hold on...
Exercise: commit-one-file-staged
Status: PASSED
You can see the easiest known solution and further info at:
https://gitexercises.fracz.com/e/commit-one-file-staged/qpc5

Next task: ignore-them
In order to start, execute: git start next

See your progress and instructions at:
https://gitexercises.fracz.com/c/qpc5
```

4. **Ignore unwanted files**
   Objective: It is often good idea to tell Git which files it should track and which it should not. Developers almost always do not want to include generated files, compiled code or libraries into their project history. Your task is to create and commit configuration that would ignore all files with exe, o, jar extension. Sample files are generated for you.
   Steps taken: git start then nano. gitignore. inside file add exe,o,jar extensions and library. and then add and commit the file.
   git add .gitignore - Stages the .gitignore file itself, so your project's file-ignore rules become a tracked part of its history.



```
Aaryan@Aaryan MINGW64 /d/Coding/exercises (commit-one-file-staged)
$ git start next
Preparing the exercise environment, hold on...
Exercise ignore-them started!
Read the README.md for instructions or view them in browser:
http://gitexercises.fracz.com/e/ignore-them

Aaryan@Aaryan MINGW64 /d/Coding/exercises (ignore-them)
$ touch .gitignore

Aaryan@Aaryan MINGW64 /d/Coding/exercises (ignore-them)
$ nano .gitignore

Aaryan@Aaryan MINGW64 /d/Coding/exercises (ignore-them)
$ git add .
warning: in the working copy of '.gitignore', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it

Aaryan@Aaryan MINGW64 /d/Coding/exercises (ignore-them)
$ git commit -m "ignored-files"
[ignore-them 7075acb] ignored-files
 2 files changed, 1 insertion(+), 1 deletion(-)
 create mode 100644 file.txt

Aaryan@Aaryan MINGW64 /d/Coding/exercises (ignore-them)
$ git verify
Verifying the ignore-them exercise. Hold on...
Exercise: ignore-them
Status: PASSED
You can see the easiest known solution and further info at:
https://gitexercises.fracz.com/e/ignore-them/qpc5

Next task: chase-branch
In order to start, execute: git start next

See your progress and instructions at:
https://gitexercises.fracz.com/c/qpc5
```

5. Chase branch that escaped

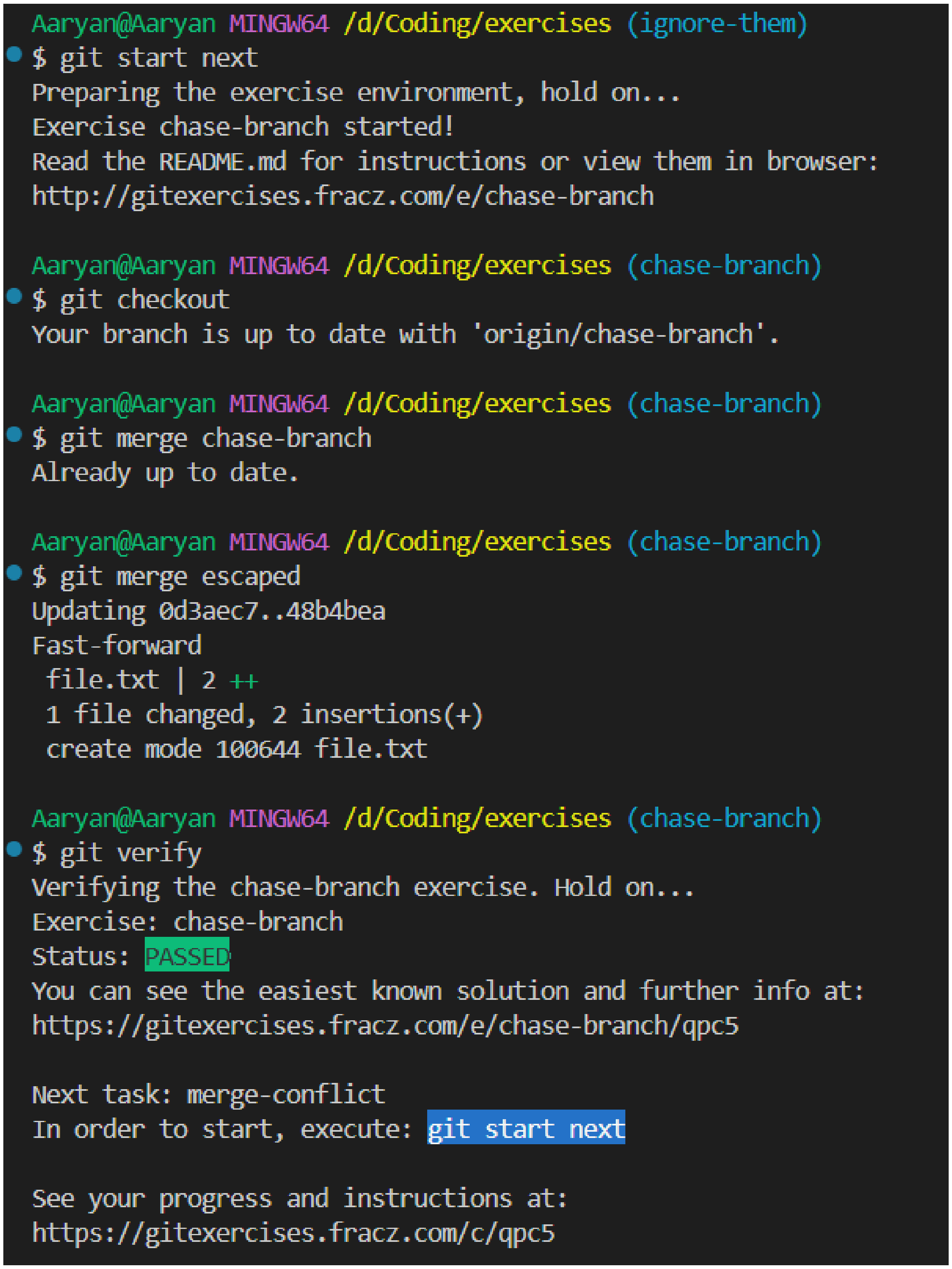
Objective: You are currently on chase-branch branch. There is also escaped branch that has two more commits.

```
  HEAD
   |
chase-branch     escaped
   |               |
   A <----- B <----- C
```

You want to make chase-branch to point to the same commit as the escaped branch.

```
               escaped
                 |
A <----- B <----- C
              |
         chase-branch
              |
            HEAD
```

git merge escaped - Integrates changes from the branch named escaped into your current working branch.



```
Aaryan@Aaryan MINGW64 /d/Coding/exercises (ignore-them)
$ git start next
Preparing the exercise environment, hold on...
Exercise chase-branch started!
Read the README.md for instructions or view them in browser:
http://gitexercises.fracz.com/e/chase-branch

Aaryan@Aaryan MINGW64 /d/Coding/exercises (chase-branch)
$ git checkout
Your branch is up to date with 'origin/chase-branch'.

Aaryan@Aaryan MINGW64 /d/Coding/exercises (chase-branch)
$ git merge chase-branch
Already up to date.

Aaryan@Aaryan MINGW64 /d/Coding/exercises (chase-branch)
$ git merge escaped
Updating 0d3aec7..48b4bea
Fast-forward
 file.txt | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 file.txt

Aaryan@Aaryan MINGW64 /d/Coding/exercises (chase-branch)
$ git verify
Verifying the chase-branch exercise. Hold on...
Exercise: chase-branch
Status: PASSED
You can see the easiest known solution and further info at:
https://gitexercises.fracz.com/e/chase-branch/qpc5

Next task: merge-conflict
In order to start, execute: git start next

See your progress and instructions at:
https://gitexercises.fracz.com/c/qpc5
```

6. Resolve a merge conflict

Objective: Merge conflict appears when you change the same part of the same file differently in the two branches you're merging together. Conflicts require developer to solve them by hand.

Your repository looks like this:

```
      HEAD
       |
   merge-conflict
       |
A <----- B
 \
  \----- C
       |
another-piece-of-work
```

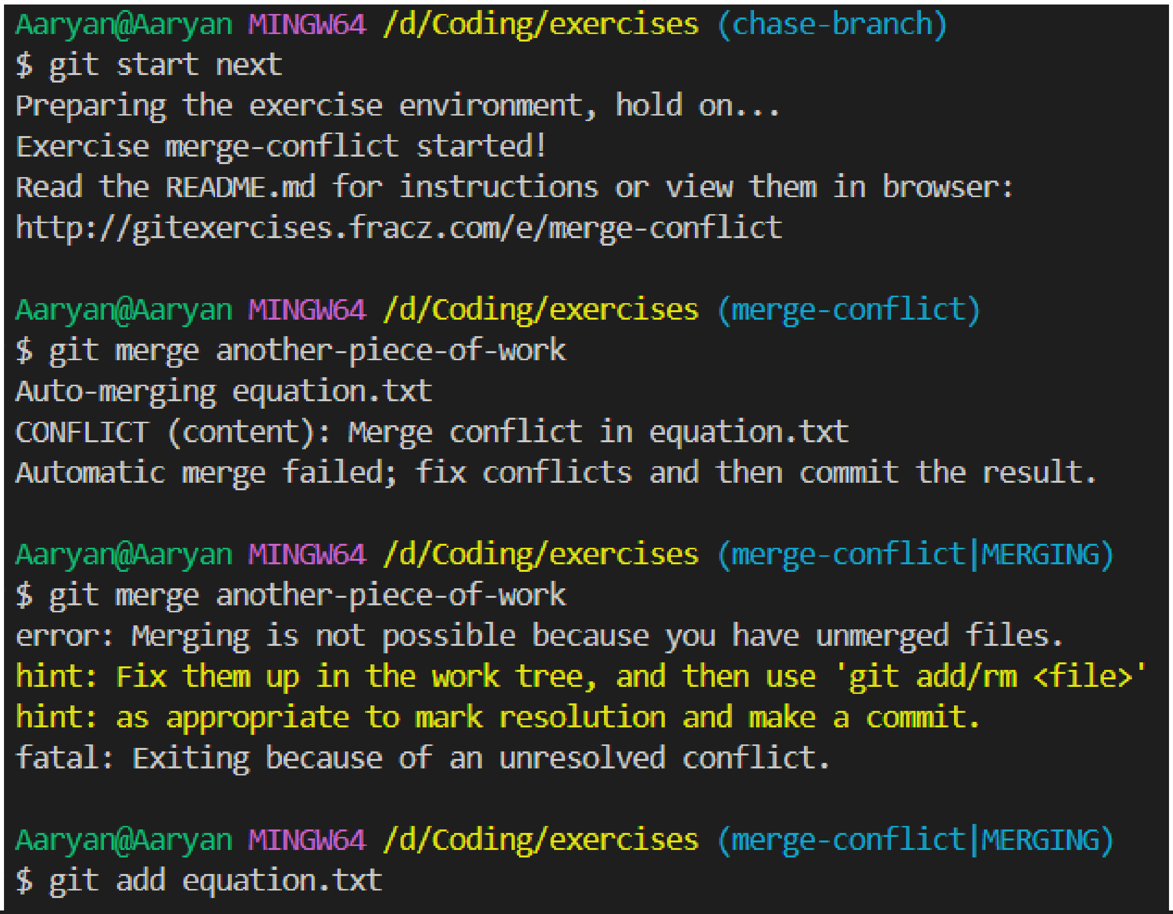
You want to merge the another-piece-of-work into your current branch. This will cause a merge conflict which you have to resolve. Your repository should look like this:

```
             HEAD
              |
          merge-conflict
              |
A <----- B <----- D
 \             /
  \----- C <----/
       |
another-piece-of-work
```

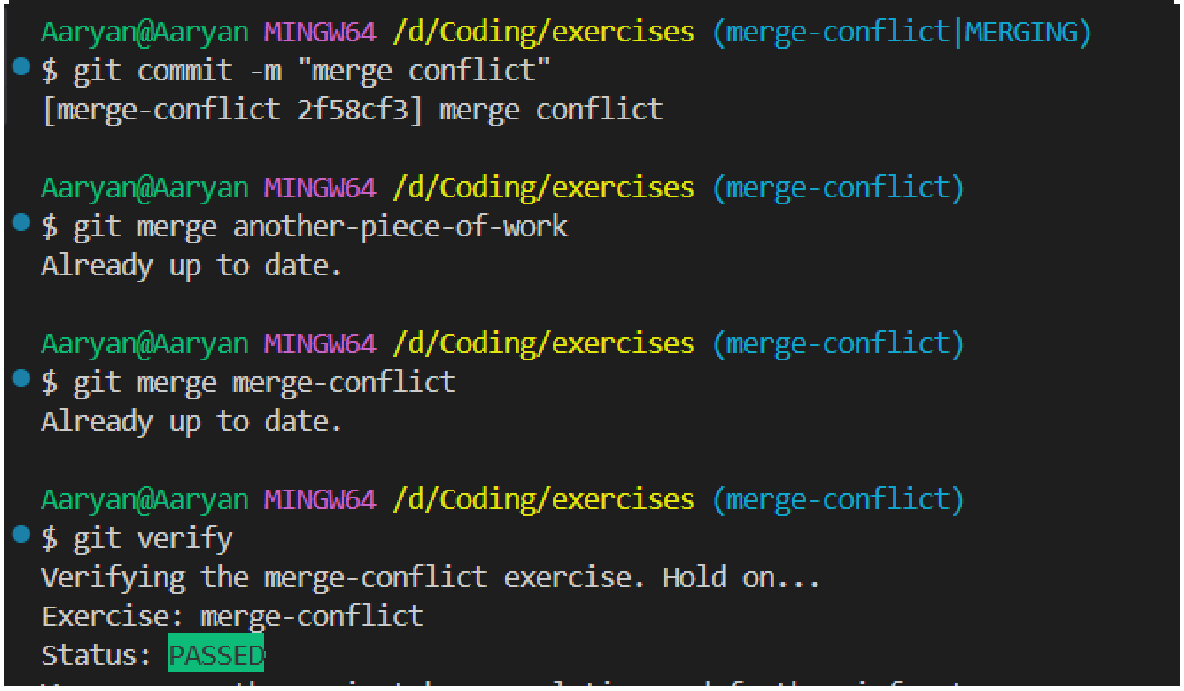
Steps taken: first execute git merge another-piece-of-work command. A merge conflict will occur in equations.txt file, there only 2+3=5, only this line should be present so delete all other lines. then save. and then add and commit the file.

git merge another-piece-of-work - Integrates all commits from the another-piece-of-work branch into your current branch.

git add equation.txt - Stages the current changes in the equation.txt file, marking them for inclusion in the next commit.

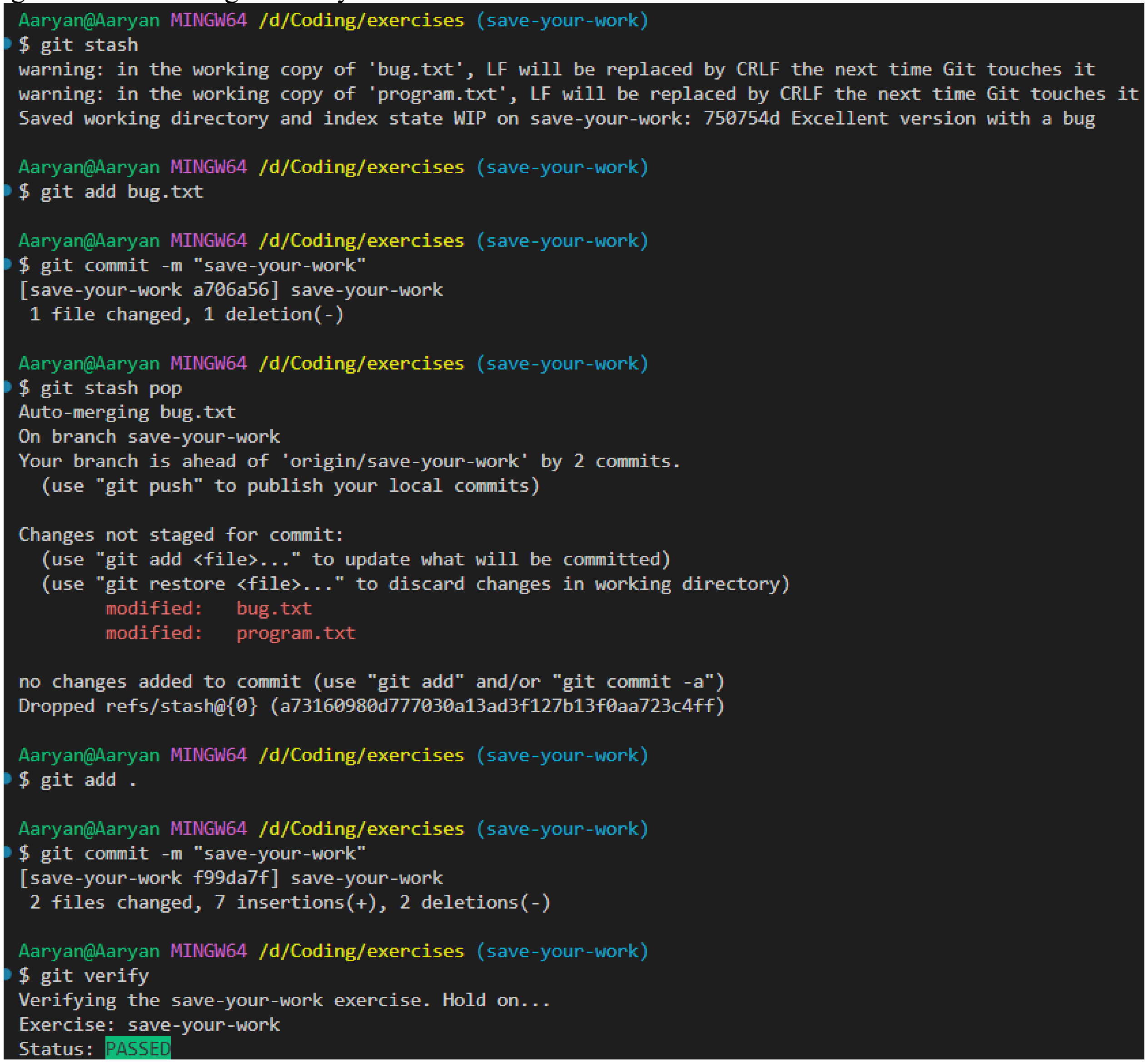


```
Aaryan@Aaryan MINGW64 /d/Coding/exercises (chase-branch)
$ git start next
Preparing the exercise environment, hold on...
Exercise merge-conflict started!
Read the README.md for instructions or view them in browser:
http://gitexercises.fracz.com/e/merge-conflict

Aaryan@Aaryan MINGW64 /d/Coding/exercises (merge-conflict)
$ git merge another-piece-of-work
Auto-merging equation.txt
CONFLICT (content): Merge conflict in equation.txt
Automatic merge failed; fix conflicts and then commit the result.

Aaryan@Aaryan MINGW64 /d/Coding/exercises (merge-conflict|MERGING)
$ git merge another-piece-of-work
error: Merging is not possible because you have unmerged files.
hint: Fix them up in the work tree, and then use 'git add/rm <file>'
hint: as appropriate to mark resolution and make a commit.
fatal: Exiting because of an unresolved conflict.

Aaryan@Aaryan MINGW64 /d/Coding/exercises (merge-conflict|MERGING)
$ git add equation.txt
```



```
Aaryan@Aaryan MINGW64 /d/Coding/exercises (merge-conflict|MERGING)
$ git commit -m "merge conflict"
[merge-conflict 2f58cf3] merge conflict

Aaryan@Aaryan MINGW64 /d/Coding/exercises (merge-conflict)
$ git merge another-piece-of-work
Already up to date.

Aaryan@Aaryan MINGW64 /d/Coding/exercises (merge-conflict)
$ git merge merge-conflict
Already up to date.

Aaryan@Aaryan MINGW64 /d/Coding/exercises (merge-conflict)
$ git verify
Verifying the merge-conflict exercise. Hold on...
Exercise: merge-conflict
Status: PASSED
```

7. **Saving your work**
Objective: You are working hard on a regular issue while your boss comes in and wants you to fix a bug. State of your current working area is a total mess so you don't feel comfortable with making a commit now. However, you need to fix the found bug ASAP. Git lets you to save your work on a side and continue it later. Find appropriate Git tool and use it to handle the situation appropriately.
Look for a bug to remove in bug.txt. After you commit the bugfix, get back to your work. Finish it by adding a new line to bug.txt with Finally, finished it! Then, commit your work after bugfix.
Steps taken: use git stash, then fix the bug in bug.txt file, then commit it. use git stash pop and then add "Finally, finished it!" and then commit it.
git stash - Temporarily shelves your uncommitted local changes (both staged and unstaged) so you can get a clean working directory.



```
Aaryan@Aaryan MINGW64 /d/Coding/exercises (save-your-work)
$ git stash
warning: in the working copy of 'bug.txt', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'program.txt', LF will be replaced by CRLF the next time Git touches it
Saved working directory and index state WIP on save-your-work: 750754d Excellent version with a bug

Aaryan@Aaryan MINGW64 /d/Coding/exercises (save-your-work)
$ git add bug.txt

Aaryan@Aaryan MINGW64 /d/Coding/exercises (save-your-work)
$ git commit -m "save-your-work"
[save-your-work a706a56] save-your-work
 1 file changed, 1 deletion(-)

Aaryan@Aaryan MINGW64 /d/Coding/exercises (save-your-work)
$ git stash pop
Auto-merging bug.txt
On branch save-your-work
Your branch is ahead of 'origin/save-your-work' by 2 commits.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   bug.txt
        modified:   program.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (a73160980d777030a13ad3f127b13f0aa723c4ff)

Aaryan@Aaryan MINGW64 /d/Coding/exercises (save-your-work)
$ git add .

Aaryan@Aaryan MINGW64 /d/Coding/exercises (save-your-work)
$ git commit -m "save-your-work"
[save-your-work f99da7f] save-your-work
 2 files changed, 7 insertions(+), 2 deletions(-)

Aaryan@Aaryan MINGW64 /d/Coding/exercises (save-your-work)
$ git verify
Verifying the save-your-work exercise. Hold on...
Exercise: save-your-work
Status: PASSED
```

8. Change branch history

Objective: You were working on a regular issue while your boss came in and told you to fix recent bug in an application. Because your work on the issue hasn't been done yet, you decided to go back where you started and do a bug fix there.

Your repository look like this:

```
      HEAD
        |
change-branch-history
        |
A <----- B
 \
  \----- C
        |
    hot-bugfix
```

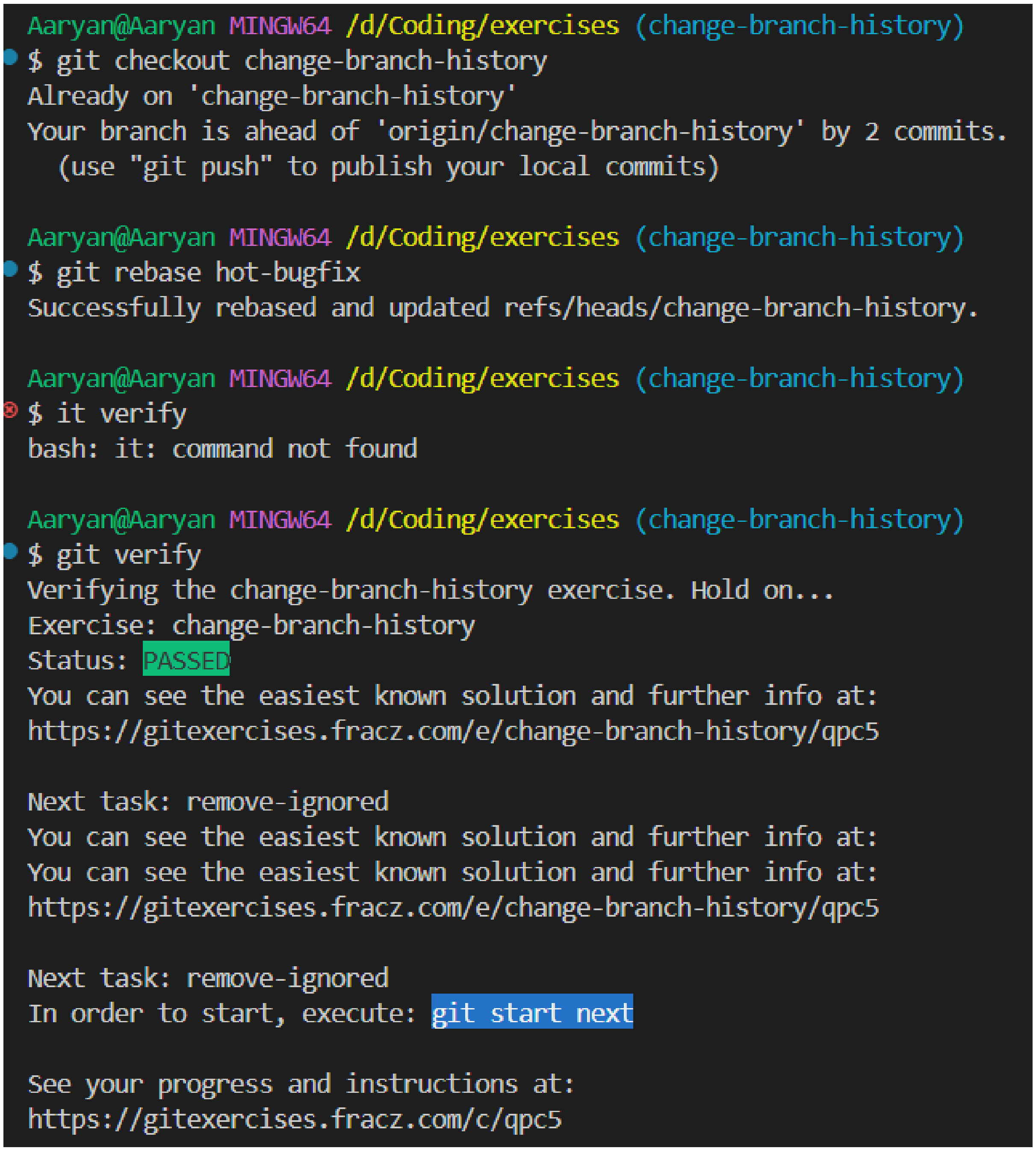
Now you realized that the bug is really annoying and you don't want to continue your work without the fix you have made. You wish your repository looked like you started after fixing a bug.

```
             HEAD
               |
       change-branch-history
               |
A <----- C <----- B
        |
    hot-bugfix
```

Achieve that.

Steps taken: Here we just had to rearrange the branch, to point to the hot-bugfix. So just used the command "git rebase hot-bugfix.

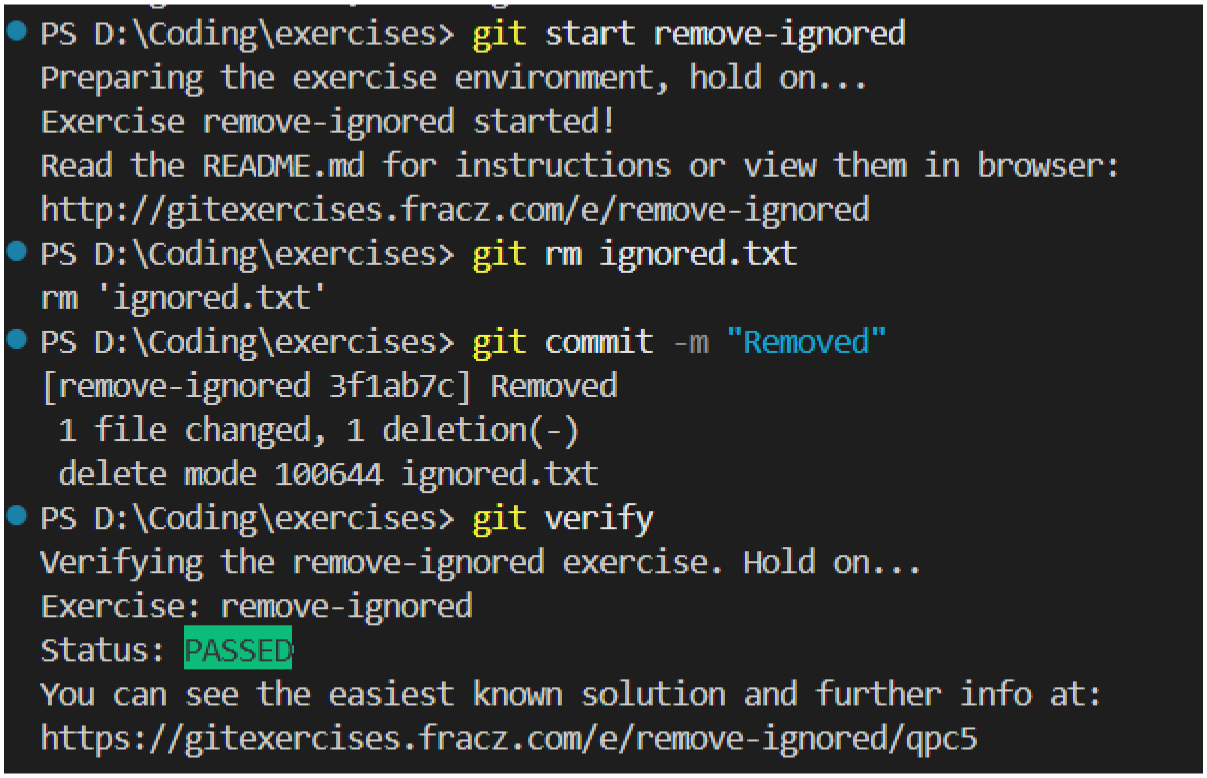
git rebase hot-bugfix - Moves your current branch's commits to the tip of the hot-bugfix branch, creating a cleaner, linear project history.



```
Aaryan@Aaryan MINGW64 /d/Coding/exercises (change-branch-history)
$ git checkout change-branch-history
Already on 'change-branch-history'
Your branch is ahead of 'origin/change-branch-history' by 2 commits.
  (use "git push" to publish your local commits)

Aaryan@Aaryan MINGW64 /d/Coding/exercises (change-branch-history)
$ git rebase hot-bugfix
Successfully rebased and updated refs/heads/change-branch-history.

Aaryan@Aaryan MINGW64 /d/Coding/exercises (change-branch-history)
$ it verify
bash: it: command not found

Aaryan@Aaryan MINGW64 /d/Coding/exercises (change-branch-history)
$ git verify
Verifying the change-branch-history exercise. Hold on...
Exercise: change-branch-history
Status: PASSED
You can see the easiest known solution and further info at:
https://gitexercises.fracz.com/e/change-branch-history/qpc5

Next task: remove-ignored
You can see the easiest known solution and further info at:
You can see the easiest known solution and further info at:
https://gitexercises.fracz.com/e/change-branch-history/qpc5

Next task: remove-ignored
In order to start, execute: git start next

See your progress and instructions at:
https://gitexercises.fracz.com/c/qpc5
```

9. **Remove ignored file**
   Objective: File ignored.txt is ignored by rule in. gitignore but is tracked because it had been added before the ignoring rule was introduced. Remove it so changes in ignored.txt file is not tracked anymore.
   Steps taken: here we just have to remove the ignored.txt file and commit the rest, so used: git rm ignored.txt and then commit.

git rm ignored.txt - Removes the file ignored.txt from both your working directory and the Git staging area, queuing it for deletion on the next commit.



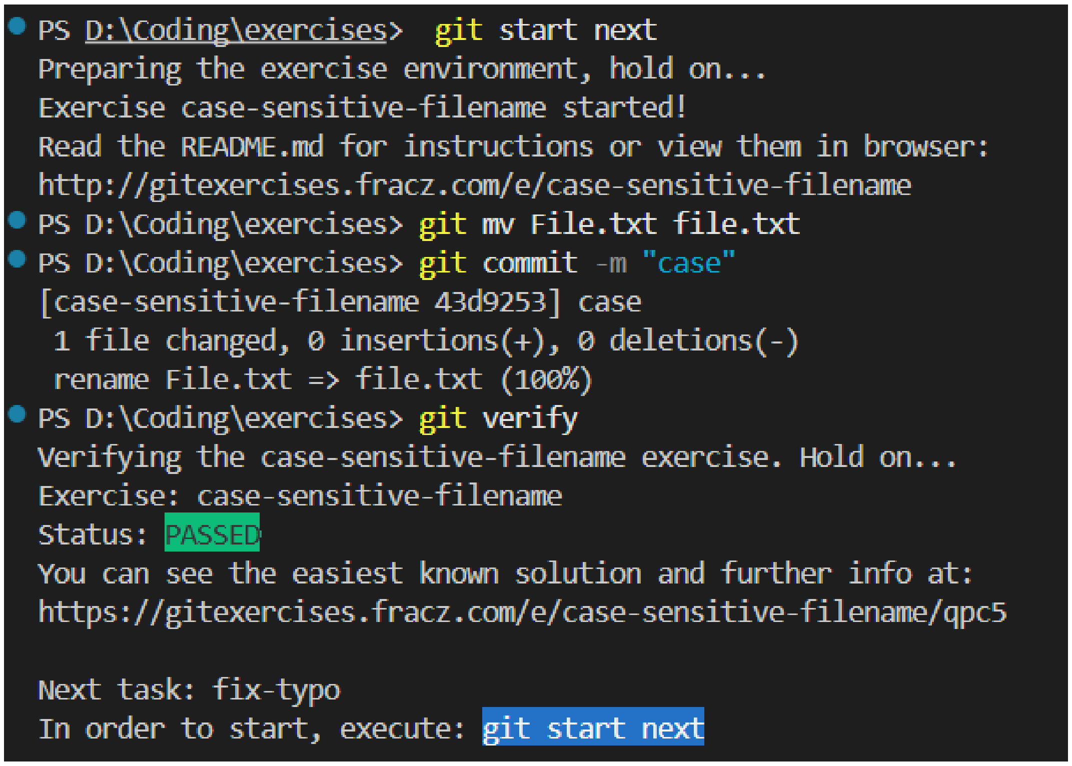
```
PS D:\Coding\exercises> git start remove-ignored
Preparing the exercise environment, hold on...
Exercise remove-ignored started!
Read the README.md for instructions or view them in browser:
http://gitexercises.fracz.com/e/remove-ignored
PS D:\Coding\exercises> git rm ignored.txt
rm 'ignored.txt'
PS D:\Coding\exercises> git commit -m "Removed"
[remove-ignored 3f1ab7c] Removed
 1 file changed, 1 deletion(-)
 delete mode 100644 ignored.txt
PS D:\Coding\exercises> git verify
Verifying the remove-ignored exercise. Hold on...
Exercise: remove-ignored
Status: PASSED
You can see the easiest known solution and further info at:
https://gitexercises.fracz.com/e/remove-ignored/qpc5
```

10. Change a letter case in the filename of an already tracked file

Objective: You have committed a File.txt but then you realized the filename should be all lowercase: file.txt. Change the filename.

Steps taken: we first use the git mv command to interchange the locations of the two files. Then we commit the lowercase file.

git mv File.txt file.txt - Renames File.txt to file.txt, reliably handling case changes and automatically staging the move for the next commit.



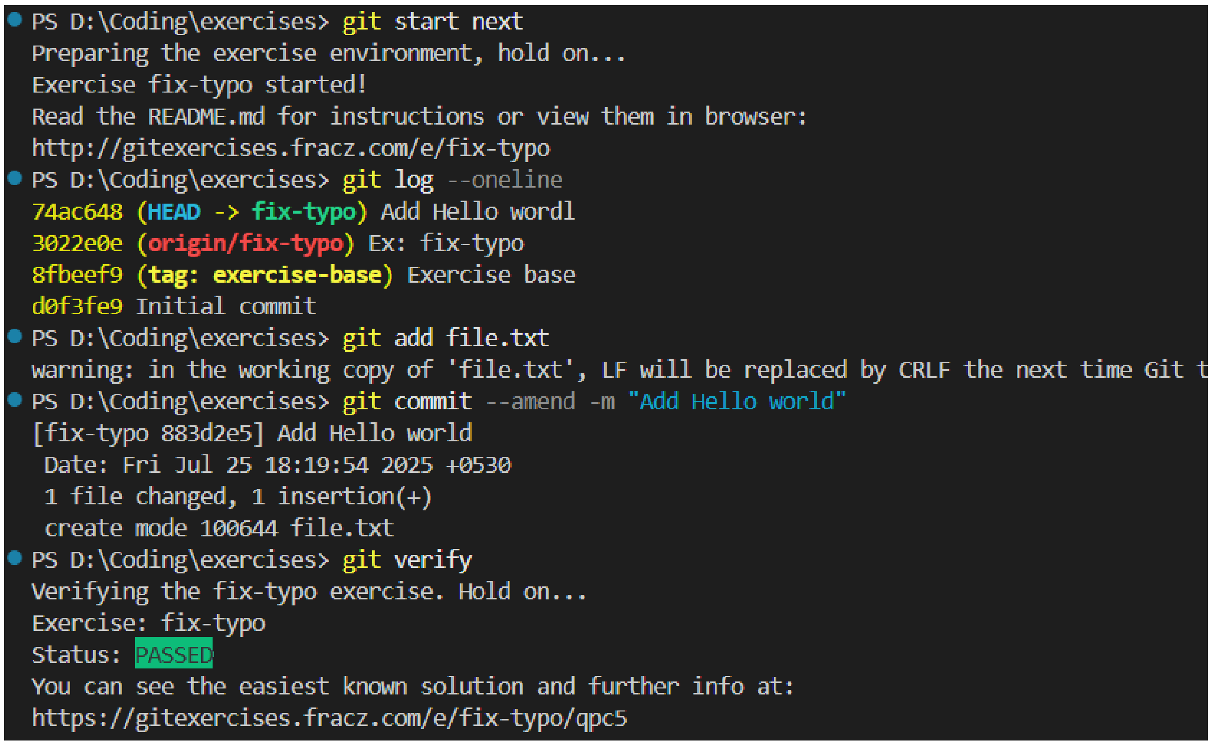
```
PS D:\Coding\exercises>  git start next
Preparing the exercise environment, hold on...
Exercise case-sensitive-filename started!
Read the README.md for instructions or view them in browser:
http://gitexercises.fracz.com/e/case-sensitive-filename
PS D:\Coding\exercises> git mv File.txt file.txt
PS D:\Coding\exercises> git commit -m "case"
[case-sensitive-filename 43d9253] case
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename File.txt => file.txt (100%)
PS D:\Coding\exercises> git verify
Verifying the case-sensitive-filename exercise. Hold on...
Exercise: case-sensitive-filename
Status: PASSED
You can see the easiest known solution and further info at:
https://gitexercises.fracz.com/e/case-sensitive-filename/qpc5

Next task: fix-typo
In order to start, execute: git start next
```

11. Fix typographic mistake in the last commit

Objective: You have committed file.txt but you realized you made a typo - you wrote wordl instead of world. Edit previous commit so no one would realize you haven't checked the file before committing it.

Steps taken: First do nano file.txt and fix the typo in the file. Then amend it and then fixx the typo in the commit message.

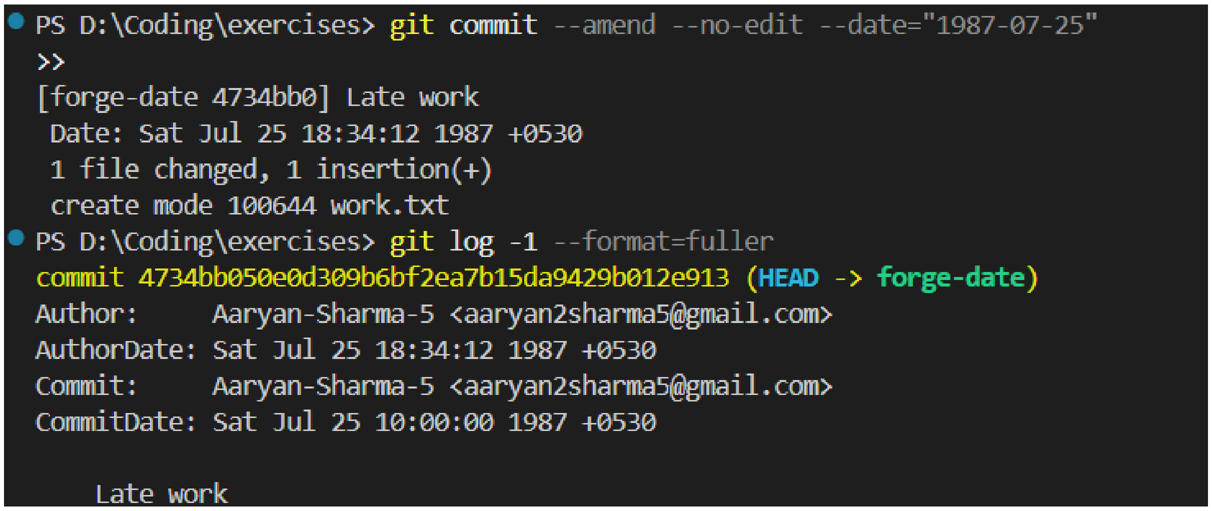
git commit –amend - Redoes the most recent commit, letting you add more staged changes or edit its commit message.



```
PS D:\Coding\exercises> git start next
Preparing the exercise environment, hold on...
Exercise fix-typo started!
Read the README.md for instructions or view them in browser:
http://gitexercises.fracz.com/e/fix-typo
PS D:\Coding\exercises> git log --oneline
74ac648 (HEAD -> fix-typo) Add Hello wordl
3022e0e (origin/fix-typo) Ex: fix-typo
8fbeef9 (tag: exercise-base) Exercise base
d0f3fe9 Initial commit
PS D:\Coding\exercises> git add file.txt
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git t
PS D:\Coding\exercises> git commit --amend -m "Add Hello world"
[fix-typo 883d2e5] Add Hello world
 Date: Fri Jul 25 18:19:54 2025 +0530
 1 file changed, 1 insertion(+)
 create mode 100644 file.txt
PS D:\Coding\exercises> git verify
Verifying the fix-typo exercise. Hold on...
Exercise: fix-typo
Status: PASSED
You can see the easiest known solution and further info at:
https://gitexercises.fracz.com/e/fix-typo/qpc5
```

12. Forge the commit's date

Objective: You should have finished your work a week ago. However, you had some more important things to do so you have committed the work just now. As a git expert, change the date of the last commit. Don't be modest - make it look like it was committed in 1987!
Steps taken: just use the commit and amend command and edit the date feature.
git commit --amend --noedit --date - Updates the author date of the most recent commit to a specified time, without changing its message or files.



```
PS D:\Coding\exercises> git commit --amend --no-edit --date="1987-07-25"
>>
[forge-date 4734bb0] Late work
 Date: Sat Jul 25 18:34:12 1987 +0530
 1 file changed, 1 insertion(+)
 create mode 100644 work.txt
PS D:\Coding\exercises> git log -1 --format=fuller
commit 4734bb050e0d309b6bf2ea7b15da9429b012e913 (HEAD -> forge-date)
Author:     Aaryan-Sharma-5 <aaryan2sharma5@gmail.com>
AuthorDate: Sat Jul 25 18:34:12 1987 +0530
Commit:     Aaryan-Sharma-5 <aaryan2sharma5@gmail.com>
CommitDate: Sat Jul 25 10:00:00 1987 +0530

    Late work
```

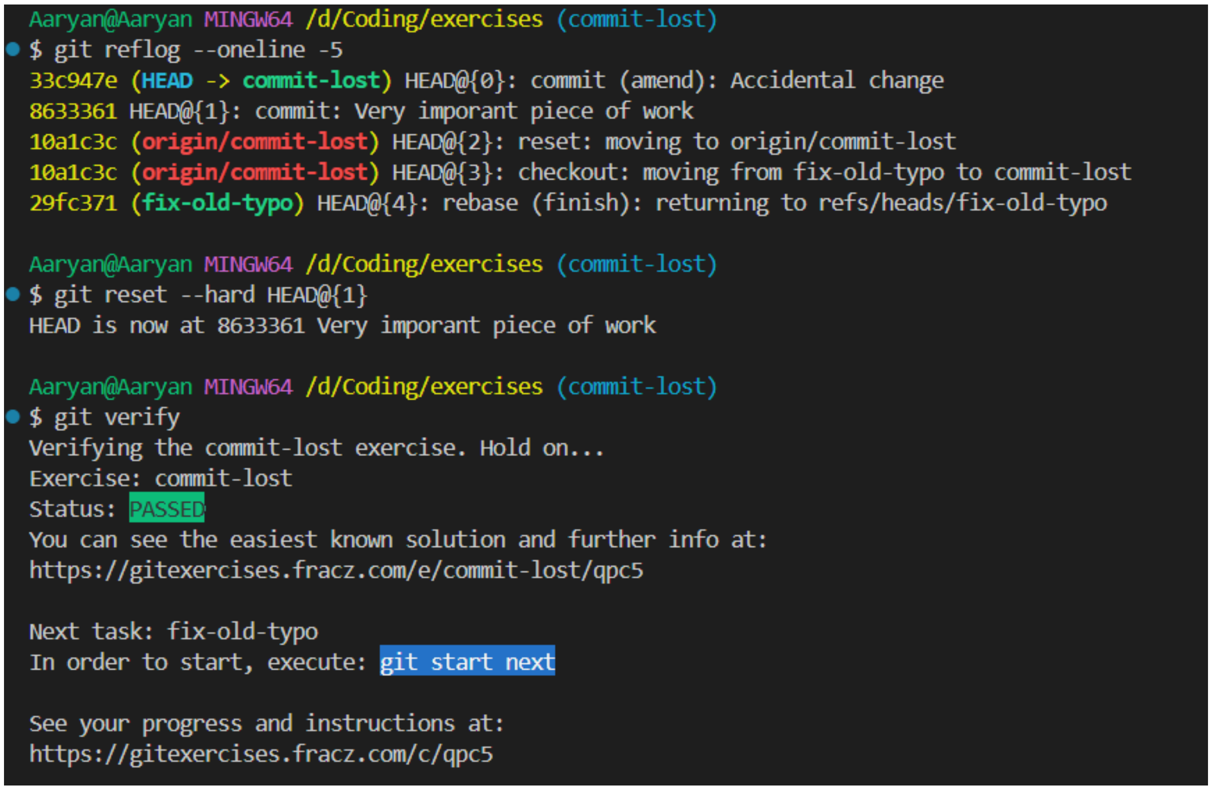
13. Find a commit that has been lost

Objective: You have created a commit with very important piece of work. You then wanted to fix something in the last commit so you have amended it. However, you have just realized you have accidentally committed the wrong changes and you desperately need the first version of the commit you have just amended. However, there is no previous version in the history - you have edited the last commit with git commit --amend. Your goal is to find the first version of the commit in the repository. It must be somewhere.... Once found, force the commit-lost branch to point at it again and verify the solution.

Steps taken: Just reset the branch head to the very first commit.

Git commands: git reset --hard HEAD@ {1}

git reflog - Shows a log of all recent changes to your local repository's branch tips, acting as a safety net to recover lost work



```
Aaryan@Aaryan MINGW64 /d/Coding/exercises (commit-lost)
$ git reflog --oneline -5
33c947e (HEAD -> commit-lost) HEAD@{0}: commit (amend): Accidental change
8633361 HEAD@{1}: commit: Very imporant piece of work
10a1c3c (origin/commit-lost) HEAD@{2}: reset: moving to origin/commit-lost
10a1c3c (origin/commit-lost) HEAD@{3}: checkout: moving from fix-old-typo to commit-lost
29fc371 (fix-old-typo) HEAD@{4}: rebase (finish): returning to refs/heads/fix-old-typo

Aaryan@Aaryan MINGW64 /d/Coding/exercises (commit-lost)
$ git reset --hard HEAD@{1}
HEAD is now at 8633361 Very imporant piece of work

Aaryan@Aaryan MINGW64 /d/Coding/exercises (commit-lost)
$ git verify
Verifying the commit-lost exercise. Hold on...
Exercise: commit-lost
Status: PASSED
You can see the easiest known solution and further info at:
https://gitexercises.fracz.com/e/commit-lost/qpc5

Next task: fix-old-typo
In order to start, execute: git start next

See your progress and instructions at:
https://gitexercises.fracz.com/c/qpc5
```

14. Split the last commit

Objective: You have committed both first.txt and second.txt as one commit. However, you made a mistake. You intended to commit first.txt in the first commit and the second.txt in the second one.

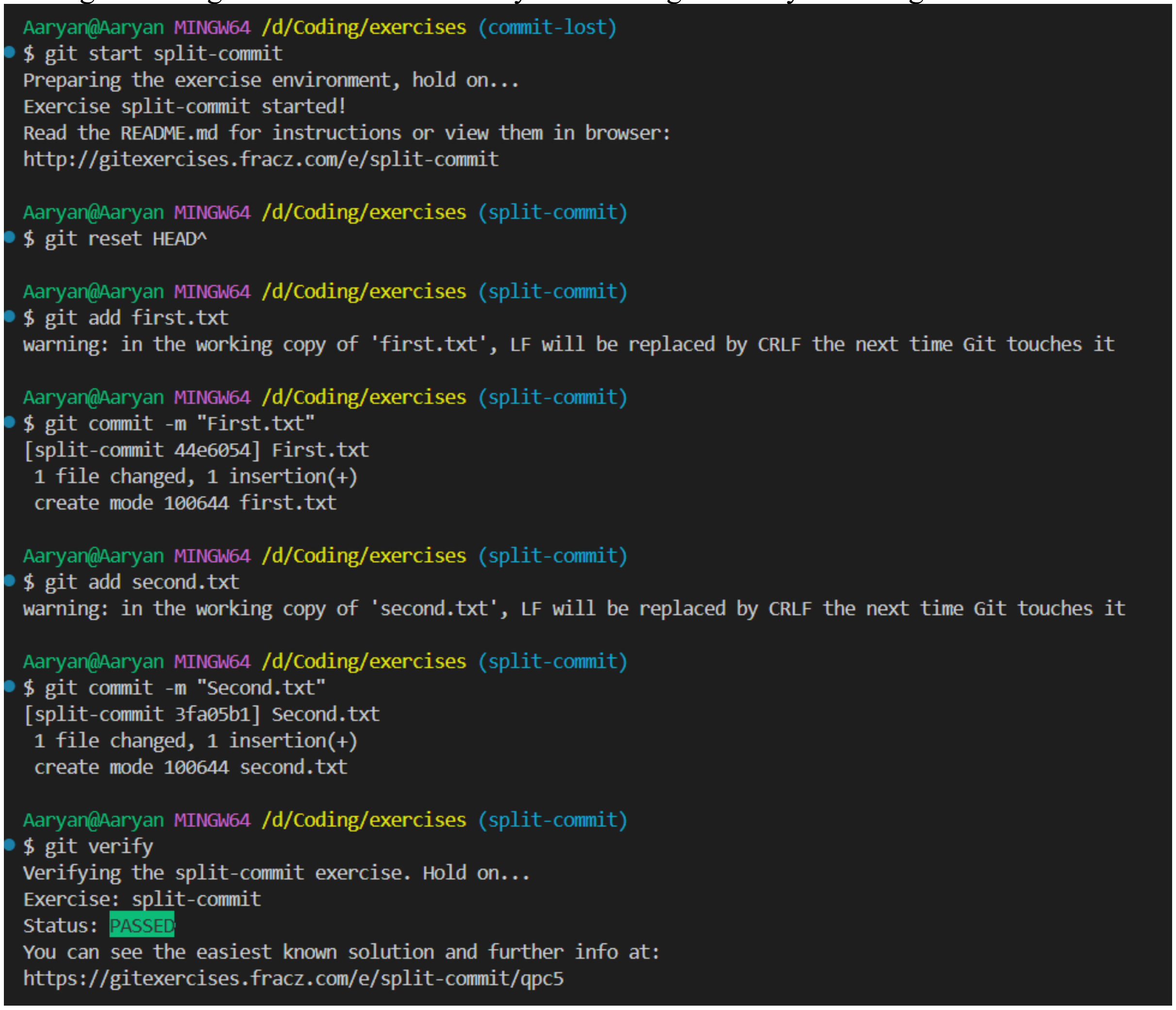
Steps taken: First reset the HEAD back 1 commit, then add and commit the first file, then add and commit the second file.

Git commands: git reset HEAD^

git add first.txt

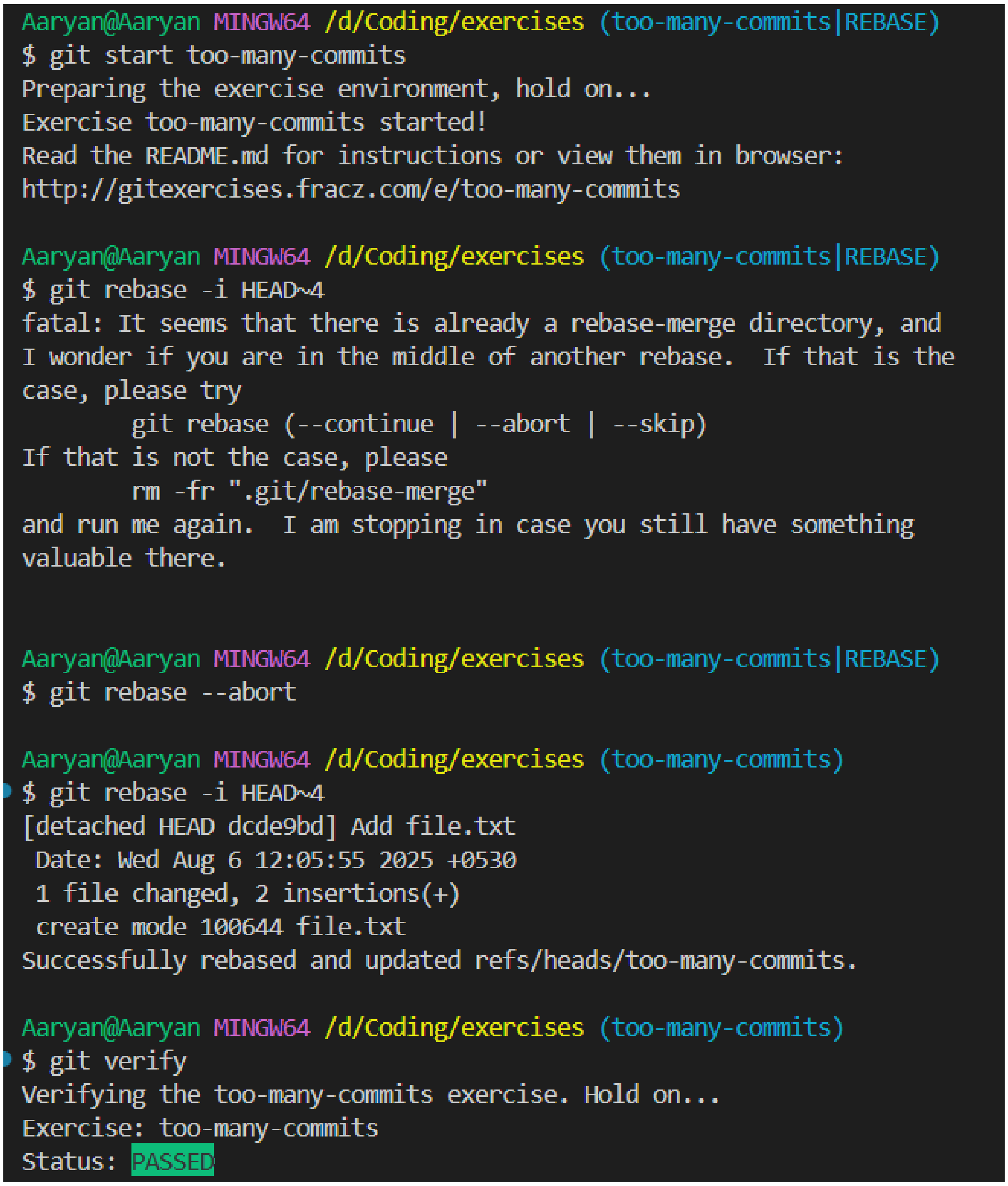
git commit -m "First.txt"

git add second.txt
git commit -m "Second.txt"
git reset HEAD^ - Undoes the most recent commit by moving the branch pointer back one step, leaving the changes from that commit in your working directory as unstaged modifications



```
Aaryan@Aaryan MINGW64 /d/Coding/exercises (commit-lost)
$ git start split-commit
Preparing the exercise environment, hold on...
Exercise split-commit started!
Read the README.md for instructions or view them in browser:
http://gitexercises.fracz.com/e/split-commit

Aaryan@Aaryan MINGW64 /d/Coding/exercises (split-commit)
$ git reset HEAD^

Aaryan@Aaryan MINGW64 /d/Coding/exercises (split-commit)
$ git add first.txt
warning: in the working copy of 'first.txt', LF will be replaced by CRLF the next time Git touches it

Aaryan@Aaryan MINGW64 /d/Coding/exercises (split-commit)
$ git commit -m "First.txt"
[split-commit 44e6054] First.txt
 1 file changed, 1 insertion(+)
 create mode 100644 first.txt

Aaryan@Aaryan MINGW64 /d/Coding/exercises (split-commit)
$ git add second.txt
warning: in the working copy of 'second.txt', LF will be replaced by CRLF the next time Git touches it

Aaryan@Aaryan MINGW64 /d/Coding/exercises (split-commit)
$ git commit -m "Second.txt"
[split-commit 3fa05b1] Second.txt
 1 file changed, 1 insertion(+)
 create mode 100644 second.txt

Aaryan@Aaryan MINGW64 /d/Coding/exercises (split-commit)
$ git verify
Verifying the split-commit exercise. Hold on...
Exercise: split-commit
Status: PASSED
You can see the easiest known solution and further info at:
https://gitexercises.fracz.com/e/split-commit/qpc5
```

15. **Too many commits**
Objectives: You were working on an issue and created two commits introducing very small change. You don't want to mess up your project history so you want to make only one commit that contains changes made in the last two.
git rebase -i HEAD^^
# "squash" the second commit
squash - combining a series of consecutive commits into a single, new commit

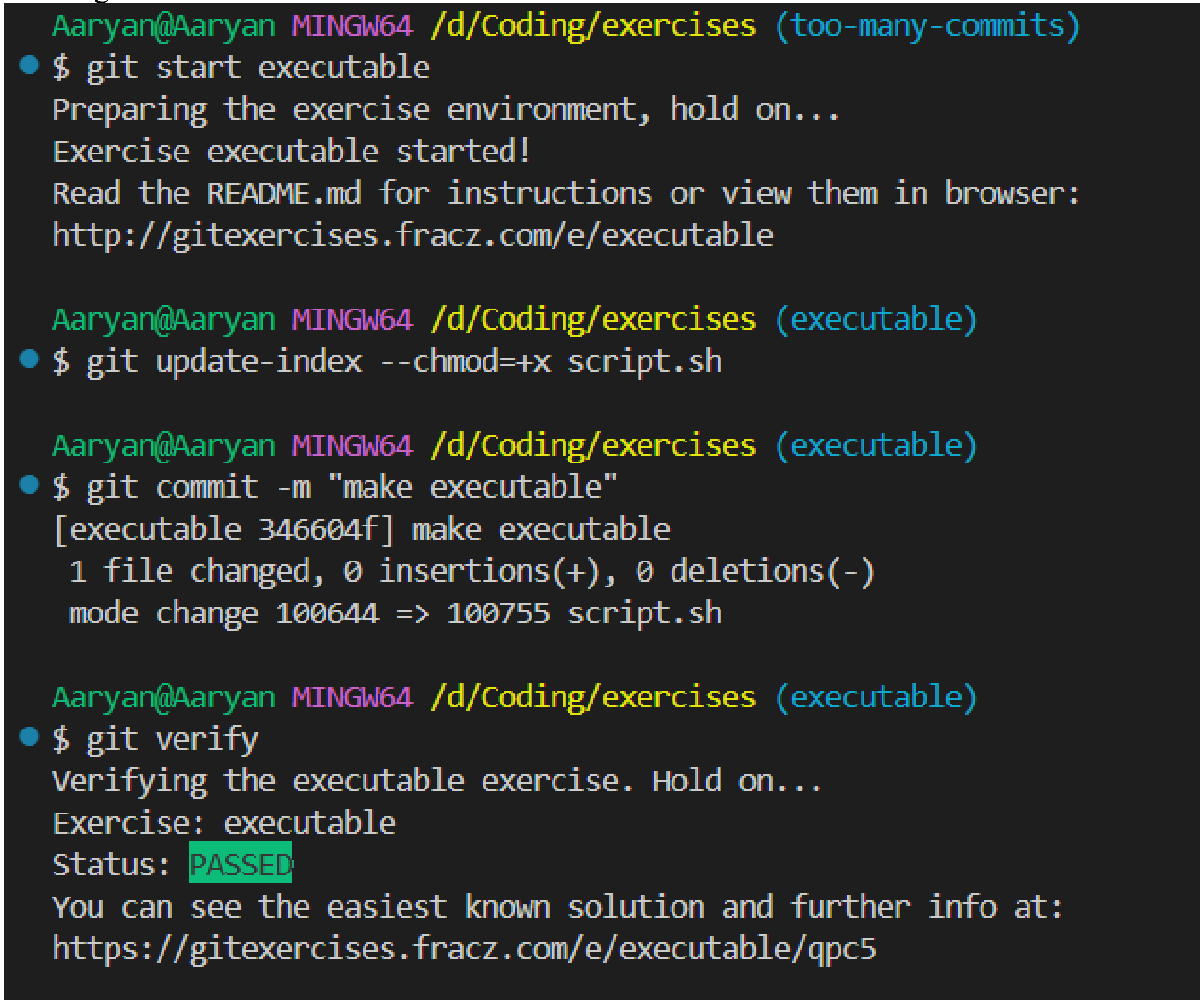


```
Aaryan@Aaryan MINGW64 /d/Coding/exercises (too-many-commits|REBASE)
$ git start too-many-commits
Preparing the exercise environment, hold on...
Exercise too-many-commits started!
Read the README.md for instructions or view them in browser:
http://gitexercises.fracz.com/e/too-many-commits

Aaryan@Aaryan MINGW64 /d/Coding/exercises (too-many-commits|REBASE)
$ git rebase -i HEAD~4
fatal: It seems that there is already a rebase-merge directory, and
I wonder if you are in the middle of another rebase.  If that is the
case, please try
        git rebase (--continue | --abort | --skip)
If that is not the case, please
        rm -fr ".git/rebase-merge"
and run me again.  I am stopping in case you still have something
valuable there.


Aaryan@Aaryan MINGW64 /d/Coding/exercises (too-many-commits|REBASE)
$ git rebase --abort

Aaryan@Aaryan MINGW64 /d/Coding/exercises (too-many-commits)
$ git rebase -i HEAD~4
[detached HEAD dcde9bd] Add file.txt
 Date: Wed Aug 6 12:05:55 2025 +0530
 1 file changed, 2 insertions(+)
 create mode 100644 file.txt
Successfully rebased and updated refs/heads/too-many-commits.

Aaryan@Aaryan MINGW64 /d/Coding/exercises (too-many-commits)
$ git verify
Verifying the too-many-commits exercise. Hold on...
Exercise: too-many-commits
Status: PASSED
```

16. Make the file executable by default

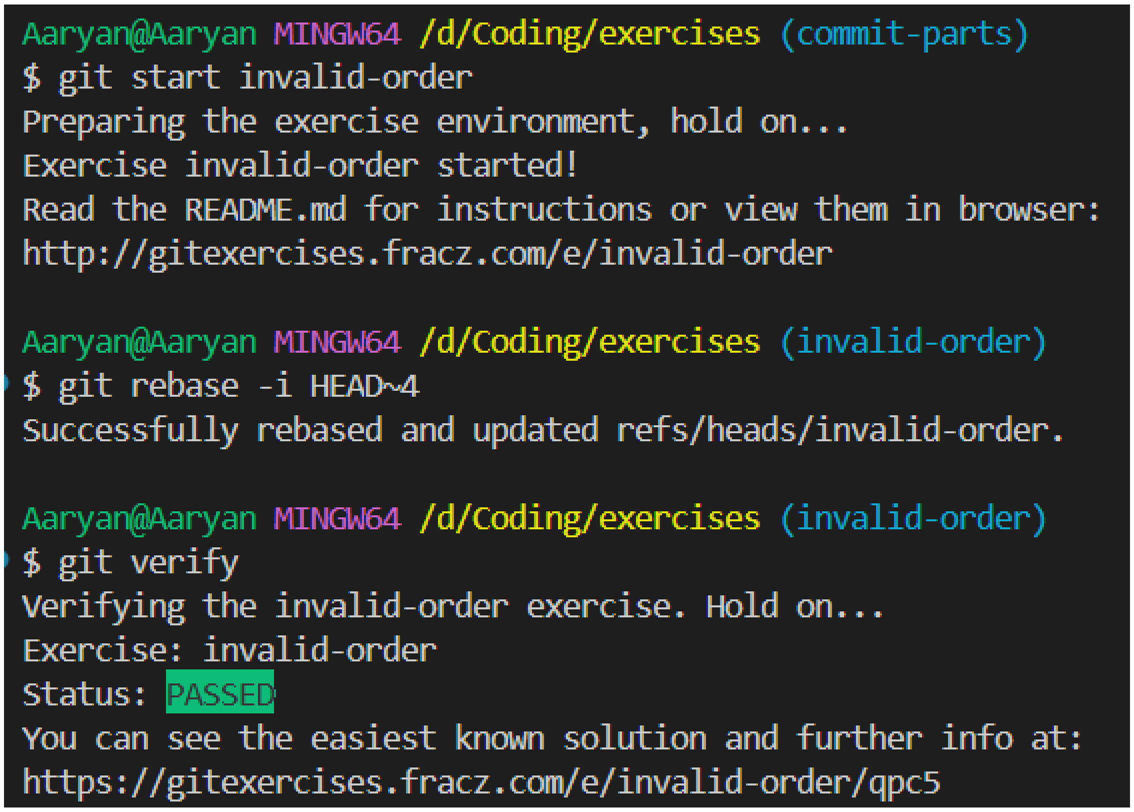
Objective: You have created a simple bash script in script.sh. However, when you check it out on Unix, it does not have required execute permissions so you can't launch it with ./script.sh without

performing chmod +x script.sh beforehand. Fix it by adding an executable bit for script.sh in Git history.
Steps taken: we need to use the git update-index command and edit the chmod by adding an executable.
git update-index --chmod=+x script.sh - Directly adds the executable permission to script.sh in the staging area, ensuring it's treated as executable within the repository, without changing your local working file.



```
Aaryan@Aaryan MINGW64 /d/Coding/exercises (too-many-commits)
$ git start executable
Preparing the exercise environment, hold on...
Exercise executable started!
Read the README.md for instructions or view them in browser:
http://gitexercises.fracz.com/e/executable

Aaryan@Aaryan MINGW64 /d/Coding/exercises (executable)
$ git update-index --chmod=+x script.sh

Aaryan@Aaryan MINGW64 /d/Coding/exercises (executable)
$ git commit -m "make executable"
[executable 346604f] make executable
 1 file changed, 0 insertions(+), 0 deletions(-)
 mode change 100644 => 100755 script.sh

Aaryan@Aaryan MINGW64 /d/Coding/exercises (executable)
$ git verify
Verifying the executable exercise. Hold on...
Exercise: executable
Status: PASSED
You can see the easiest known solution and further info at:
https://gitexercises.fracz.com/e/executable/qpc5
```

17. **Invalid Order**
Objectives: You have committed two changes but you don't like the order in which they appear in the history. Switch them.
git rebase -i HEAD~2 - Opens an interactive editor to let you modify the last two commits, allowing you to reorder, squash, or reword them



```
Aaryan@Aaryan MINGW64 /d/Coding/exercises (commit-parts)
$ git start invalid-order
Preparing the exercise environment, hold on...
Exercise invalid-order started!
Read the README.md for instructions or view them in browser:
http://gitexercises.fracz.com/e/invalid-order

Aaryan@Aaryan MINGW64 /d/Coding/exercises (invalid-order)
$ git rebase -i HEAD~4
Successfully rebased and updated refs/heads/invalid-order.

Aaryan@Aaryan MINGW64 /d/Coding/exercises (invalid-order)
$ git verify
Verifying the invalid-order exercise. Hold on...
Exercise: invalid-order
Status: PASSED
You can see the easiest known solution and further info at:
https://gitexercises.fracz.com/e/invalid-order/qpc5
```

Conclusion:
I have successfully completed the experiment on the Implementation of Version Control Systems using Git and GitHub. By working through the series of interactive exercises on gitexercises.fracz.com , I have gained practical, hands-on experience with the essential commands and workflows that are fundamental to modern software development.