Project Overview

The goal of this project was to explore and execute various git reset commands (--hard, --mixed, and --soft) and analyze their effects on the Git repository. The process also involved tracking changes, managing the staging index, committing changes, and pushing updates to a remote repository.

Steps and Challenges Encountered

1. Create Directory and Initialize Git Repository

- **Action**: I created a new directory called git_reset_test and initialized a new Git repository within it.
- **Analysis**: The repository was successfully initialized as confirmed by the presence of the. git directory. No changes were made at this point, and the repository was clean.

2. Create Empty File and Add to Staging Index

- **Action**: I created an empty file called reset_lifecycle_file and added it to the staging index using the git add command.
- **Analysis**: When I ran git status, it indicated that the file was in the staging area, ready to be committed. The file was successfully tracked by Git, and no changes were pending.

3. Commit File

- **Action**: I committed the file with a message: "Added reset_lifecycle_file."
- **Analysis**: After committing, I ran git status, and the result showed "nothing to commit" since all changes were already committed. This meant the repository was in sync with the commit history.

4. Modify File and Analyze Status

- **Action**: I added the text "hello git reset" to the reset_lifecycle_file and ran git status to analyze the results.

- **Analysis**: git status showed that the file was modified but not yet staged for commit. The file was marked as "modified" in the working directory. Running git ls-files -s confirmed the file was tracked by Git with the new content but not yet committed.

5. Run Git Reset --Hard

- **Action**: I created a new file called new_file, added it to the staging index, and added new content to reset_lifecycle_file. I then performed a git reset --hard to reset the state of the repository.
- **Analysis**: After the hard reset, both the staging area and the working directory were reverted to the state of the last commit, discarding the changes. Running git status showed that there were no changes to commit, indicating that all modifications had been undone. The hard reset also removed the new file, as confirmed by git ls-files -s, which showed only the committed file, reset_lifecycle_file, in the index.

6. Run Git Status and Git ls-files -s After Hard Reset

- **Action**: After the reset, I ran git status and git ls-files -s to analyze the effects of the hard reset.
- **Analysis**:
  - **git status**: The output confirmed that there were no uncommitted changes, and the working directory was clean. The repository was in sync with the most recent commit, and no staged or untracked files remained.
  - **git ls-files -s**: This command displayed the reset_lifecycle_file as staged for commit, confirming that only the committed file was tracked in the index. There were no untracked or staged files left after the hard reset. This confirmed that the hard reset had successfully reverted all changes.

7. Run Git Reset --Mixed

- **Action**: After performing the hard reset, I created a new file new_file, added it to the staging index, and added new content to reset_lifecycle_file. I then ran git reset --mixed.
- **Analysis**:

- **git status**: The output indicated that the new_file was no longer staged, but the changes in the working directory were preserved. This showed that the mixed reset only affected the staging area and did not remove the file or modifications from the working directory.
- **git ls-files -s**: The output showed reset_lifecycle_file as staged, but new_file was no longer in the staging area. The reset preserved the file in the working directory but un staged it. This confirmed that the mixed reset moved changes out of the staging area while keeping them in the working directory.

8. Run Git Reset --Soft

- **Action**: I added reset_lifecycle_file to the staging index and ran git reset --soft.
- **Analysis**:
  - **git status**: After the soft reset, the changes remained staged in the index, as indicated by the "Changes to be committed" section in the output. The working directory was unchanged, and no modifications were made. This showed that the soft reset moved the HEAD to the previous commit but did not affect the staging area or working directory.
  - **git ls-files -s**: The output confirmed that reset_lifecycle_file was staged, and no other changes were present. The reset had moved the HEAD back but left all files in the staging area, ready for a new commit.

9. Run Git Reset --Soft with COMMIT_HASH

- **Action**: I ran git log to find the commit hash of the initial commit and then performed a git reset --soft COMMIT_HASH.
- **Analysis**:
  - **git status**: The output showed that the working directory had no changes, but the file reset_lifecycle_file was still staged for commit. This confirmed that the git reset --soft command moved the HEAD back to the initial commit without modifying the working directory or staging area.

    - **git ls-files -s**: The output showed that reset_lifecycle_file remained staged, confirming that the soft reset had preserved the staged changes. The file was still in the staging area, ready to be committed.

10. Run Git Status and Git ls-files -s After Soft Reset

- **Action**: After the soft reset, I ran git status and git ls-files -s again to analyze the results.
- **Analysis**:
    - **git status**: The output confirmed that there were no new changes to commit and reset_lifecycle_file remained in the staging area. The reset had successfully moved the HEAD to the initial commit while preserving the changes in the staging area, ready for a commit.
    - **git ls-files -s**: The reset_lifecycle_file was shown as staged for commit, and no other changes were present. This confirmed that the reset had moved the HEAD but left all modifications staged, as expected.

11. Push Changes to Remote Repository

- **Action**: Once the repository was in the desired state, I pushed the changes to the remote repository on GitHub.
- **Analysis**: Before pushing, I ran git status to ensure there were no uncommitted changes. After setting up the remote repository and pushing, git status showed that the local branch was in sync with the remote. The push was successful, and the changes were uploaded to GitHub.

By analyzing the results of each command carefully, you can observe the impact of different git reset options on the staging area, working directory, and commit history. This exercise helped deepen my understanding of how Git handles changes at various stages and how it maintains the integrity of the repository across different types of resets.