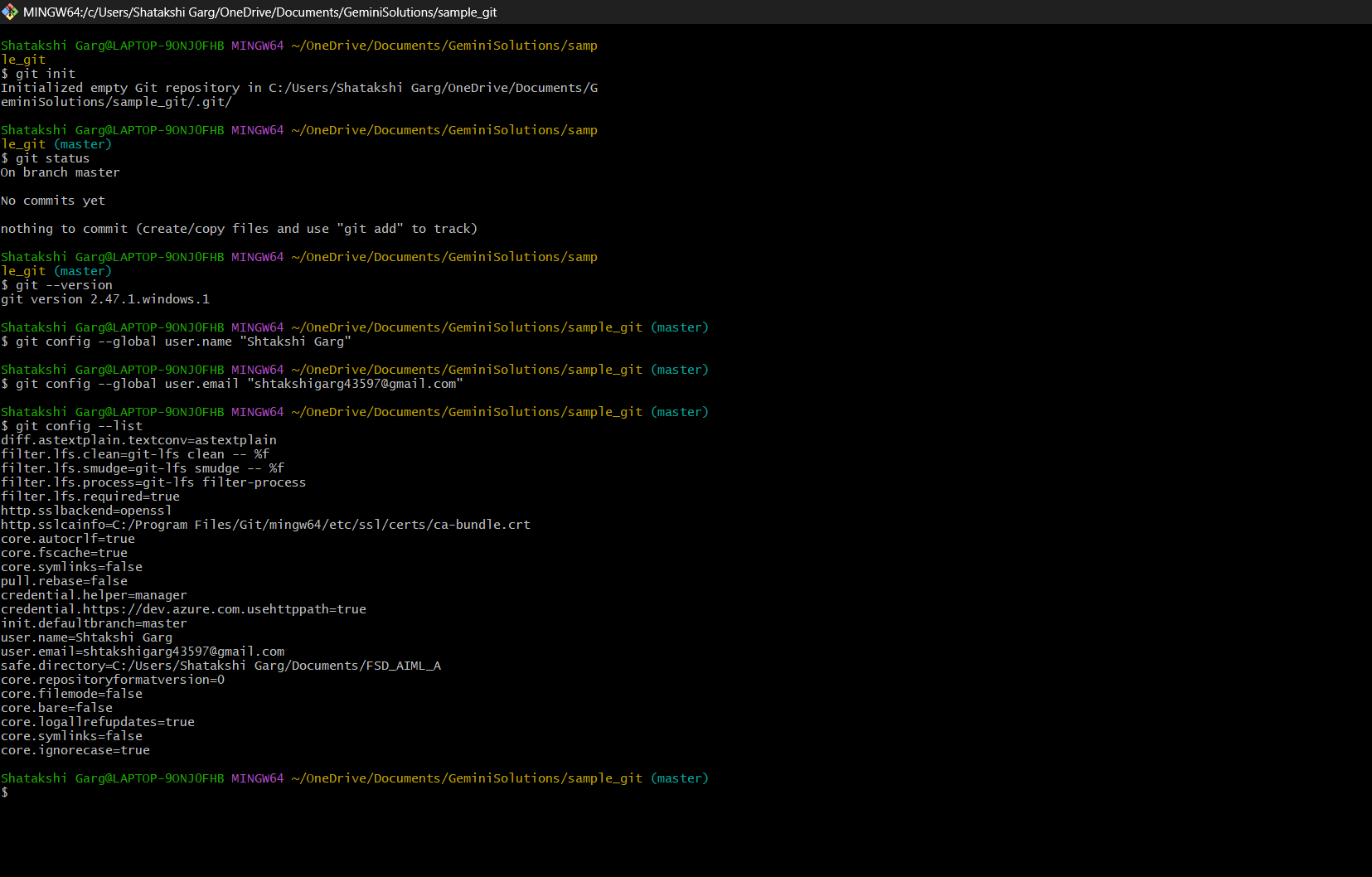
***GIT Assignment Questions***

### **1. Setting Up Git:**

* **Q1:** Install Git on your system and configure your name and email using the following commands:
  + git config --global user.name "Your Name"
  + git config --global user.email "[your.email@example.com](mailto:your.email@example.com)"
* **Q2:** How would you verify that Git has been installed and properly configured? Provide the command and the expected output.
* **Q3:** Initialize a new Git repository in an empty directory on your computer using git init.

Answer 1,2,3:



### **2. Basic Git Operations:**

* **Q4:** Create a new text file named hello.txt in your repository. Add some content to it. Then, stage the file for commit using the git add command.
* **Q5:** Commit the changes you made to the hello.txt file with a meaningful commit message. Provide the Git command to commit and the expected output.
* **Q6:** After committing your changes, use the git status command to check the state of your repository. Explain the output.

Answer: **File Creation**

bash

$ touch hello.txt

* Creates an empty file named hello.txt in the current directory.

**2. Open File in Editor**

bash

$ code .

* Opens the current folder (sample\_git) in **Visual Studio Code** for editing.

**3. Git Add**

bash

$ git add hello.txt

* Stages the file hello.txt to be committed.
* This tells Git to **track** the file and prepare it for a commit.

**4. Git Status**

bash

$ git status

* Displays the current state of the working directory and staging area.
* Output:
  + On branch master: You are on the master branch.
  + Changes to be committed: Shows hello.txt is staged for commit.

**5. Git Commit**

bash

$ git commit -m "Created hello.txt"

* Commits the staged file to the Git repository with a commit message "Created hello.txt".
* Output:
  + Shows the commit hash (e.g., 7cf63e4)
  + Confirms that 1 file (hello.txt) was added.

**6. Git Status (again)**

bash

$ git status

* Now shows: nothing to commit, working tree clean.
* This means there are no untracked or modified files—everything is committed.

**7. Git Log**

bash

$ git log

* Shows the commit history.
* Output includes:
  + Commit hash: 7cf63e4...
  + Branch position: HEAD -> master
  + Author and commit date
  + Commit message: "Created hello.txt"
* **Q7:** How can you view the commit history of a repository? Use the git log command and describe what information it provides.

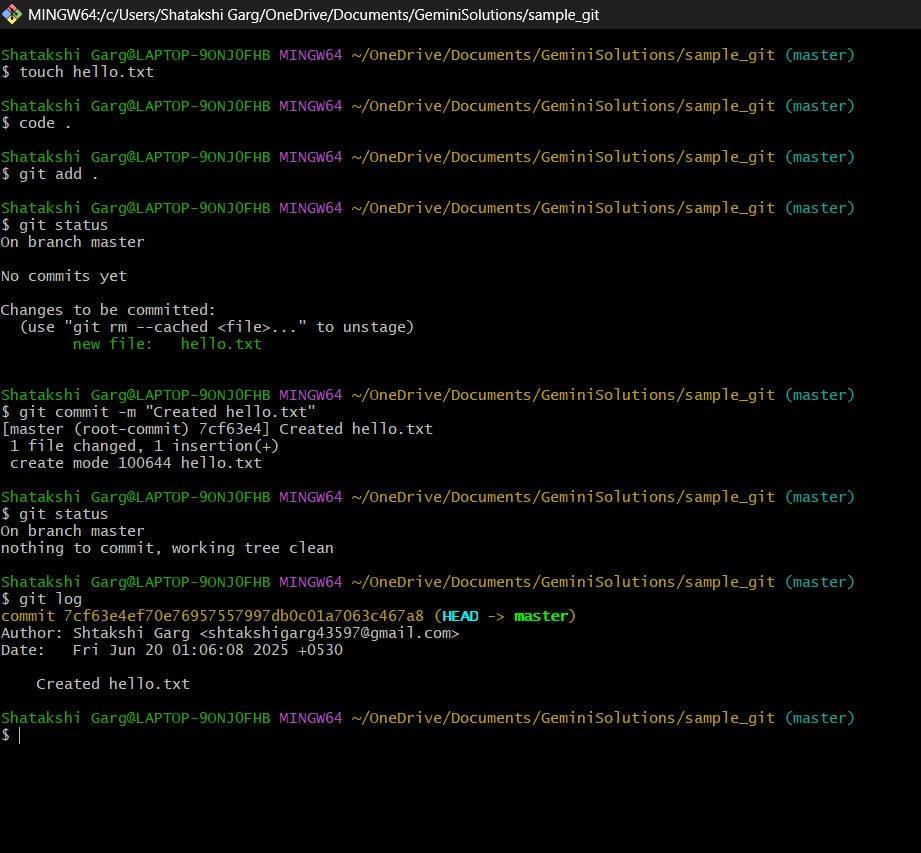
**Answer:** Explanation of Each Part

| Component | Description |
| --- | --- |
| commit hash | A unique 40-character identifier for the commit (7cf63e4...). Useful for referencing this specific change. |
| (HEAD -> master) | Shows the current position of the HEAD pointer (your current branch), indicating this commit is the latest on the master branch. |
| Author: | The name and email of the person who made the commit. |
| Date: | The exact timestamp of when the commit was made. |
| Commit Message | A short message describing what the commit does (e.g., "Created hello.txt"). This helps understand the purpose of the change. |

Common Use Cases of git log

* Viewing the history of changes
* Finding specific changes using commit messages or hashes
* Debugging who introduced a bug (via blame & log)
* Reverting or cherry-picking a past commit

Answer: 4,5,6,7



### **3. Branching and Merging:**

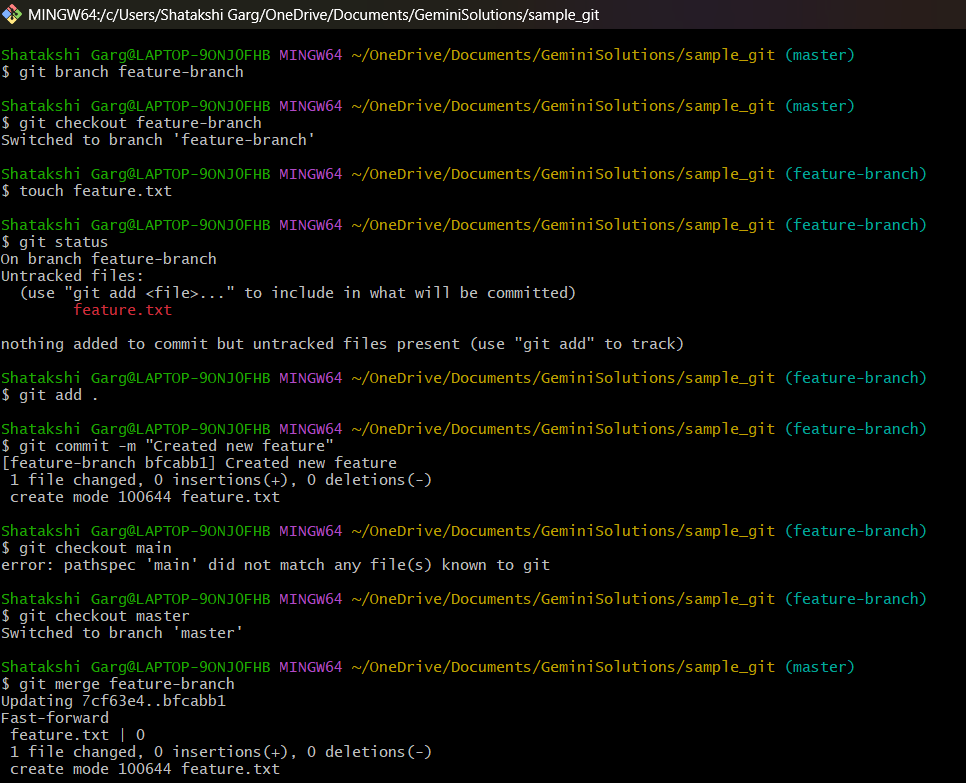
* **Q8:** What is the purpose of branching in Git? How do branches help in software development?

**Answer:** The purpose of branching in Git is to allow developers to work on different tasks, features, or bug fixes in isolation from the main codebase. A branch is essentially a separate line of development that diverges from the main project (usually the master or main branch) and can be worked on independently.

**Key Benefits of Branching in Software Development:**

1. **Isolated Development**  
   Branches enable developers to create isolated environments for new features, experiments, or bug fixes without affecting the stable version of the code.
2. **Parallel Workflows**  
   Multiple developers or teams can work on different branches simultaneously, allowing for faster and more organized collaboration.
3. **Safe Testing and Experimentation**  
   Branches allow developers to try out new ideas or technologies safely. If something breaks, it doesn't impact the main codebase.
4. **Code Review and Integration**  
   Branches are often used in pull requests or merge requests, allowing team members to review code before it's merged into the main branch. This ensures code quality and reduces bugs.
5. **Release Management**  
   Separate branches can be created for different stages of software development such as development, testing, and production. This helps manage deployments in a controlled and systematic way.
6. **Rollback and Recovery**  
   If a feature or fix introduced in a branch causes issues, it can be easily removed or modified without disturbing other ongoing work.

* **Q9:** Create a new branch called feature-branch and switch to it using the appropriate Git command.
* **Q10:** Create a new file named feature.txt on your new branch and commit the changes. Then, switch back to the main branch.
* **Q11:** Merge the feature-branch into the main branch. What command would you use to merge the changes, and what happens if there are no conflicts?

**Answer 9,10,11:** 

* **Q12:** What is a merge conflict? Create a scenario where a merge conflict occurs and explain how you would resolve it.

**Answer:** A **merge conflict** occurs when Git cannot automatically combine changes from two branches because:

* The same part of the same file was modified differently in both branches.
* Git doesn’t know which version to keep, so it stops and asks you to resolve the conflict manually.

**Example Scenario**

* You and a teammate both edit the same line in a file on different branches.
* When you try to merge or pull, Git can’t decide which change to keep — **this is a merge conflict**.

**How to Resolve a Merge Conflict**

1. Git marks the conflict in the file like this:

plaintext

<<<<<<< HEAD

your code

=======

their code

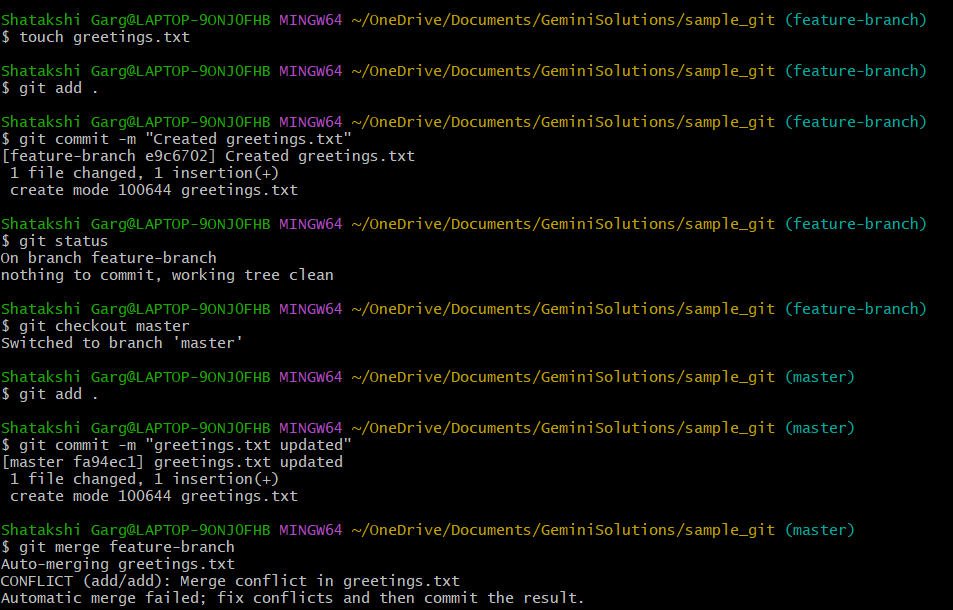
>>>>>>> branch-name

1. You manually edit the file to resolve the conflict.
2. Then run:

bash

git add <filename>

git commit

Answer: 

### **4. Working with Remote Repositories:**

* **Q13:** What is a remote repository in Git? How is it different from a local repository?

**Answer:** A **remote repository** in Git is a version of your project that is hosted on a network or internet server, such as GitHub, GitLab, or Bitbucket. It allows multiple users to collaborate on the same project by pushing and pulling code changes over the internet or a shared network.

**Difference Between Remote and Local Repository**

| **Aspect** | **Local Repository** | **Remote Repository** |
| --- | --- | --- |
| **Location** | Stored on your own computer | Stored on a remote server (e.g., GitHub, GitLab) |
| **Access** | Only you can access and modify it directly | Multiple users can access and contribute |
| **Usage** | Used for writing, committing, and testing changes | Used for sharing, syncing, and collaborating |
| **Commands Used** | git commit, git add, git status | git push, git pull, git fetch, git clone |
| **Connection** | Does not require internet | Usually requires internet to access |

**Example Use:**

* You do your work locally on your computer using a local repository.
* When ready to share or back up your work, you **push** your changes to a remote repository.
* Others can **pull** your changes or collaborate by **cloning** the remote repository.
* **Q14:** Clone a remote repository from GitHub to your local machine using the git clone command. Provide the URL of a public repository to clone.
* **Q15:** After cloning the repository, make a small change (e.g., edit README.md), and commit the changes to your local repository.
* **Q16:** Push your local commits to the remote repository. What Git command is used to push changes to a remote repository? Explain how you would use it.

Answer: The Git command used to **push changes to a remote repository** is:

bash

git push

**How to Use git push**

**Basic Syntax**

bash

git push <remote-name> <branch-name>

**Example**

bash

git push origin main

* **origin** is the default name for the remote repository.
* **main** is the branch you are pushing.

This command:

* Sends your **committed changes** from your **local repository** to the **remote repository**.
* Updates the remote branch to match your local branch.

**Typical Workflow**

bash

git add .

git commit -m "Added new feature"

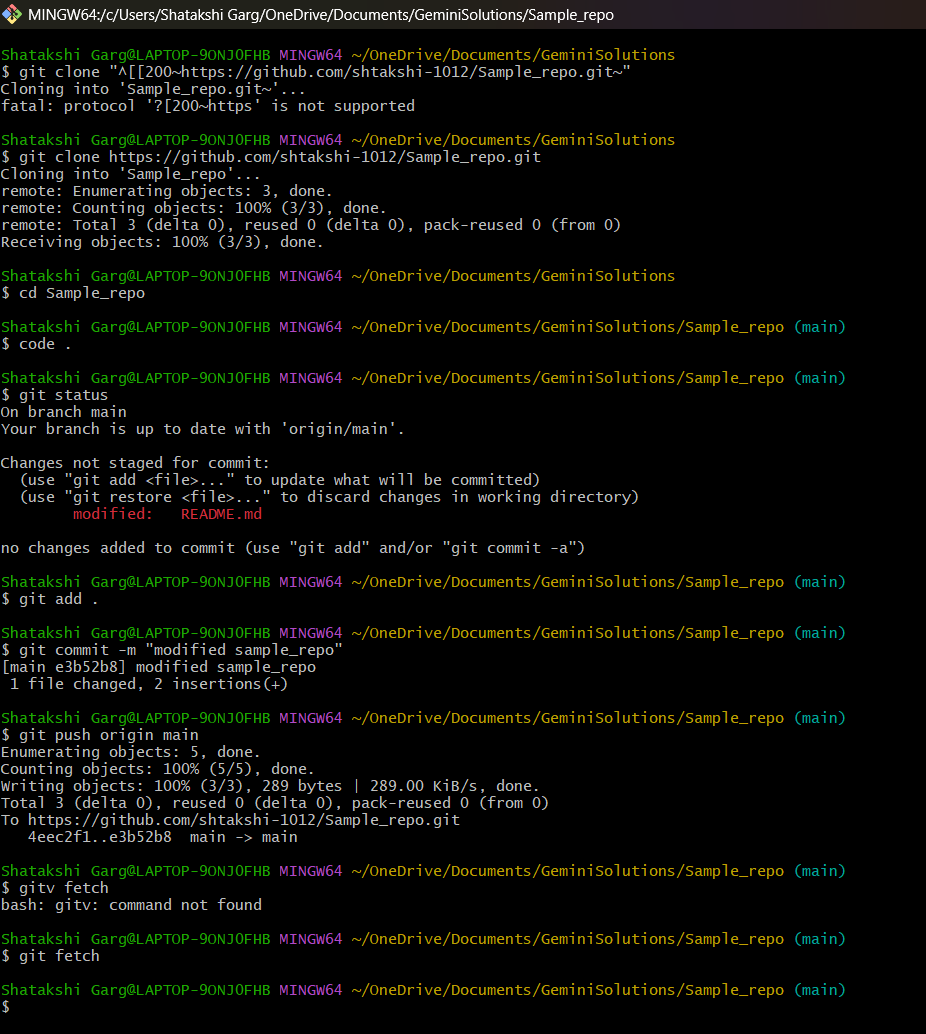
git push origin main

* **Q17:** Fetch the latest changes from the remote repository using the git fetch command. What is the difference between git fetch and git pull?

Answer: **Difference between git fetch and git pull** in tabular form:

| **Feature** | **git fetch** | **git pull** |
| --- | --- | --- |
| **Definition** | Downloads updates from the remote repository | Downloads and merges updates into the current branch |
| **Merge Operation** | Does **not** merge changes automatically | Automatically merges remote changes into your local branch |
| **Local Branch Affected** | No | Yes |
| **Risk of Conflicts** | None (since no merge happens) | Possible, if local and remote have diverged |
| **Use Case** | When you want to review or compare changes first | When you want to update your local branch immediately |
| **Control Level** | More control over when and how to merge | Less control, automatic merge |
| **Command Combination** | Standalone command | Combination of git fetch + git merge |
| **Common Syntax** | git fetch origin | git pull origin main |

Answer 14,15,16,17:



### **5. Undoing Changes in Git:**

* **Q18:** After making several commits, you realize that a commit message needs to be changed. How can you edit the last commit message using Git?

**Answer:** To edit the **last commit message** in Git, you can use the following command:

bash

git commit --amend

**What This Does:**

* Opens your default text editor with the previous commit message.
* You can edit the message, save, and close the editor.
* It **replaces the last commit** with the new message **without creating a new commit**.

**Example:**

bash

git commit --amend -m "Updated commit message with correct info"