GIT Assignment Questions

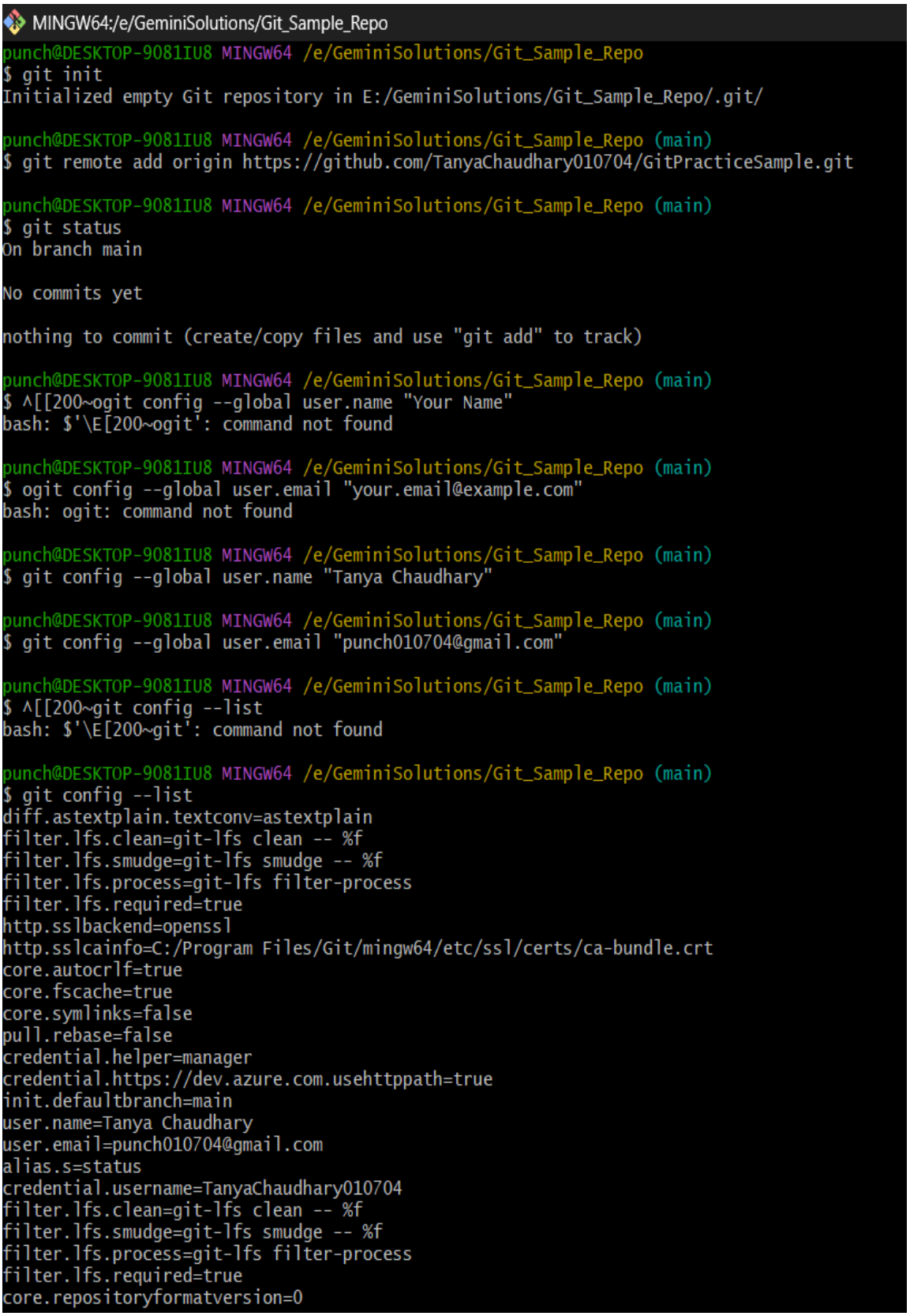
1. Setting Up Git:

- **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"`

- **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:


MINGW64:/e/GeminiSolutions/Git_Sample_Repo
punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo
$ git init
Initialized empty Git repository in E:/GeminiSolutions/Git_Sample_Repo/.git/
punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ git remote add origin https://github.com/TanyaChaudhary010704/GitPracticeSample.git
punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ git status
On branch main
No commits yet
nothing to commit (create/copy files and use "git add" to track)
punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ ^[[200~ogit config --global user.name "Your Name"
bash: $'\E[200~ogit': command not found
punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ ogit config --global user.email "your.email@example.com"
bash: ogit: command not found
punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ git config --global user.name "Tanya Chaudhary"
punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ git config --global user.email "punch010704@gmail.com"
punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ ^[[200~git config --list
bash: $'\E[200~git': command not found
punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=main
user.name=Tanya Chaudhary
user.email=punch010704@gmail.com
alias.s=status
credential.username=TanyaChaudhary010704
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
core.repositoryformatversion=0

```
punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=main
user.name=Tanya Chaudhary
user.email=punch010704@gmail.com
alias.s=status
credential.username=TanyaChaudhary010704
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
remote.origin.url=https://github.com/TanyaChaudhary010704/GitPracticeSample.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*

punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ git --version
git version 2.40.1.windows.1
```

---

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.

On branch main
This tells us which branch we are currently on. In this case, it's the `main` branch.

nothing to commit
This means that there are no changes in our working directory or staging area that need to be committed.

working tree clean
This indicates that our working directory is exactly in sync with the last commit — no untracked, modified, or staged files.

- **Q7:** How can you view the commit history of a repository? Use the `git log` command and describe what information it provides.

To view the git history of the repository we can use git log command.

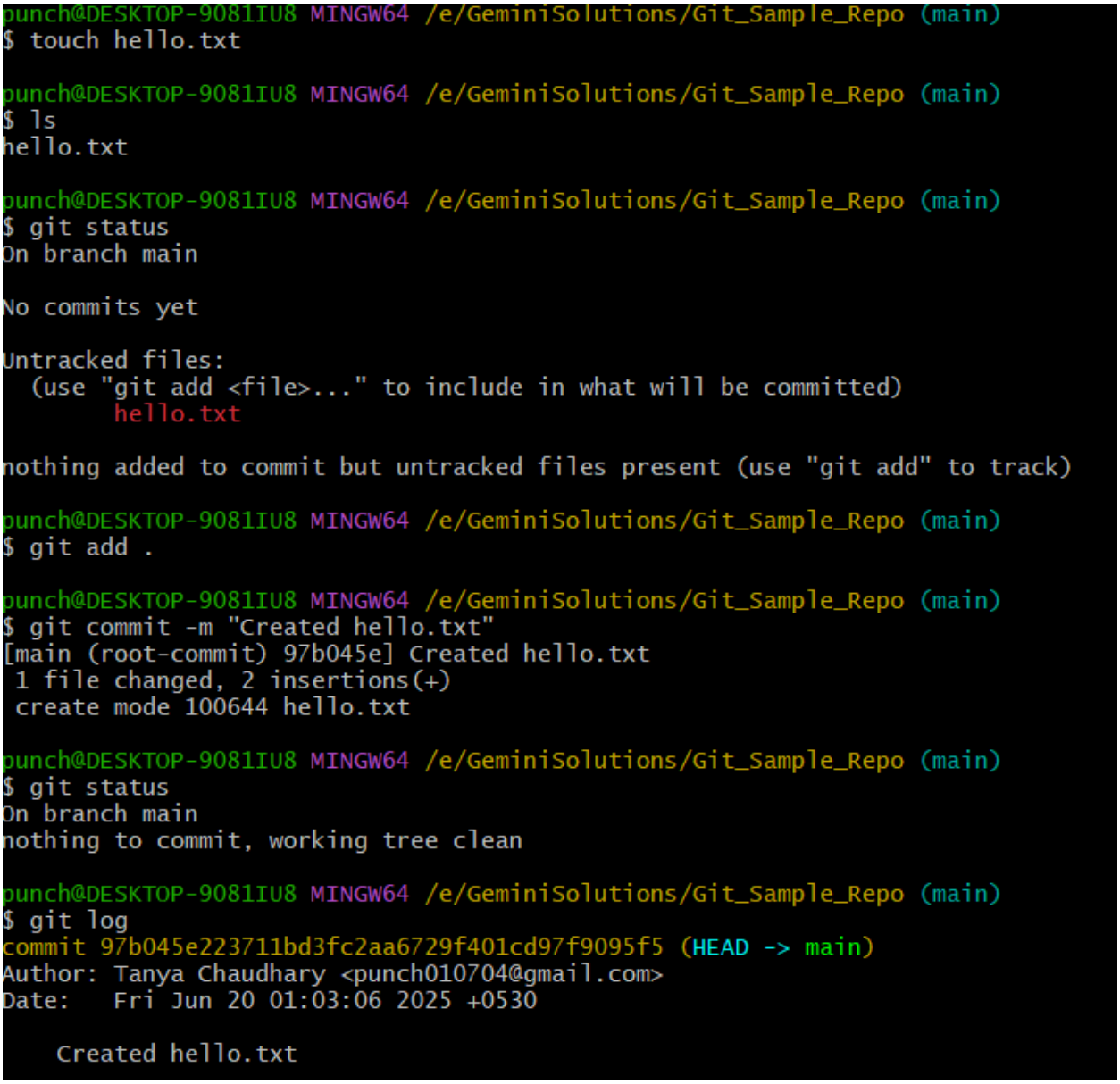
Explanation of git log:

- **commit**: The unique SHA-1 hash of the commit (e.g., `3f1aab2...`)
- **Author**: Name and email of the person who made the commit
- **Date**: The exact date and time of the commit
- **Commit message**: Description of what was done in that commit (written during `git commit`)

Optional Flags:

- `git log --oneline` — shows a concise summary (short hash + commit message)
- `git log --graph` — shows a visual graph of branches and merges
- `git log -p` — shows the actual changes (diffs) made in each 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:

Purpose of Branching in Git

Branching in Git allows developers to diverge from the main codebase (typically the main or master branch) to work on features, fixes, or experiments independently without affecting the main project**.**

How Branches Help in Software Development:

1. Isolated Development

- Each branch can represent a feature**,** bug fix**,** or task**.**
- Developers can work independently without interfering with others' work.

2. Parallel Workflows

- Multiple team members can work simultaneously on different branches (e.g., `feature/login`, `bugfix/navbar`).
- Encourages modular and scalable development.

3. Safe Experimentation

- You can test new ideas or features in a separate branch without risking the stability of the main codebase.

4. Easy Collaboration

- Teams use pull requests (PRs) or merge requests to review and merge code from branches into the main branch.

5. Release Management

- Use dedicated branches like `develop`, `release/v1.0`, or `hotfix/urgent-bug` to manage versions and production-ready code.

- **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?

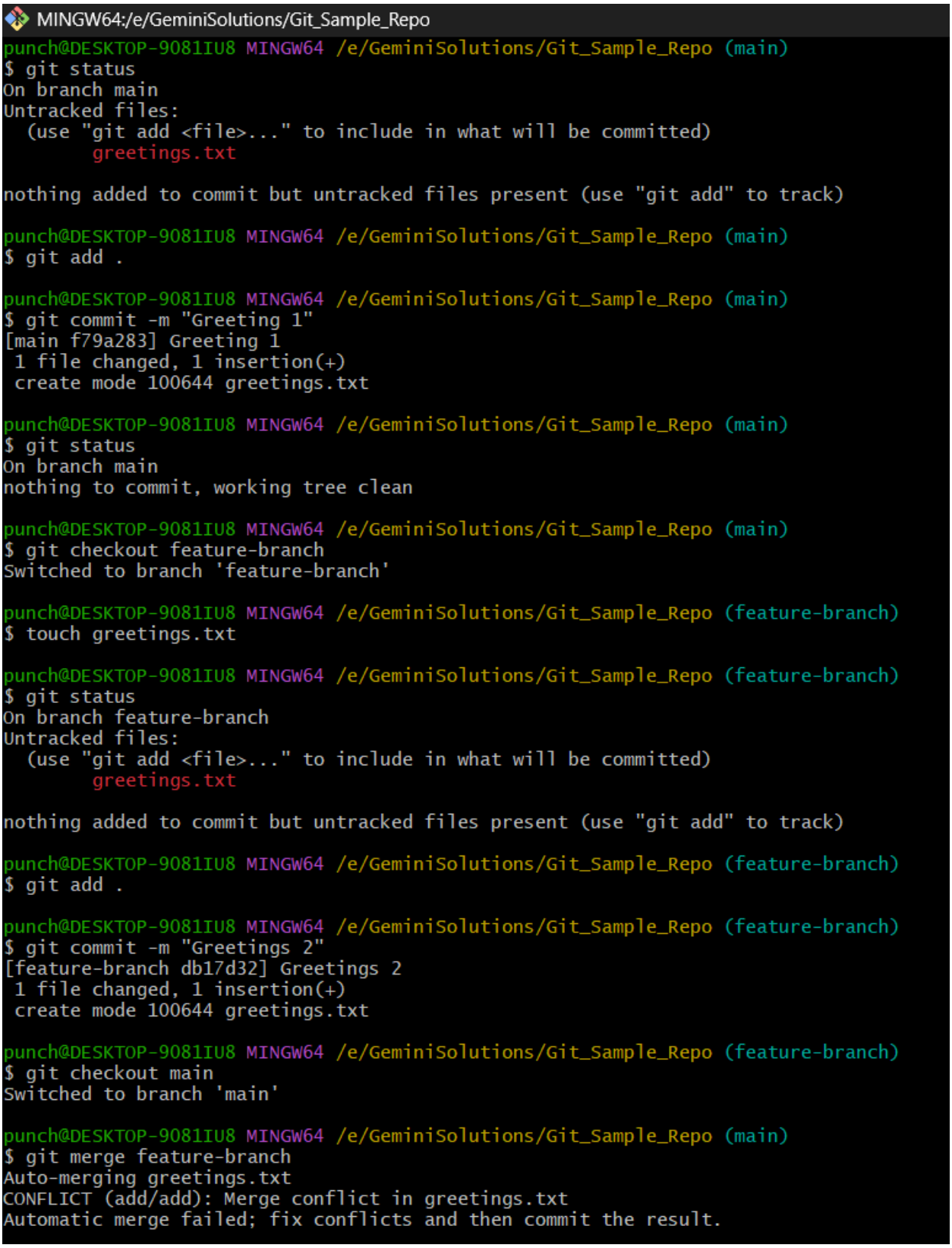
Answer 9, 10, 11:

If there are no merge conflicts, Git performs the merge automatically**.**

```
punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ git branch feature-branch

punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ git checkout feature-branch
Switched to branch 'feature-branch'

punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (feature-branch)
$ touch feature.txt

punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (feature-branch)
$ git status
On branch feature-branch
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        feature.txt

nothing added to commit but untracked files present (use "git add" to track)

punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (feature-branch)
$ git add .

punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (feature-branch)
$ git commit -m "Created new feature"
[feature-branch 3ad3ca7] Created new feature
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 feature.txt

punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (feature-branch)
$ git checkout main
Switched to branch 'main'

punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ git merge feature-branch
Updating 97b045e..3ad3ca7
Fast-forward
 feature.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 feature.txt
```

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

  Answer:


MINGW64:/e/GeminiSolutions/Git_Sample_Repo
punch@DESKTOP-9O81IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ git status
On branch main
Untracked files:
(use "git add <file>..." to include in what will be committed)
greetings.txt
nothing added to commit but untracked files present (use "git add" to track)
punch@DESKTOP-9O81IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ git add .
punch@DESKTOP-9O81IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ git commit -m "Greeting 1"
[main f79a283] Greeting 1
1 file changed, 1 insertion(+)
create mode 100644 greetings.txt
punch@DESKTOP-9O81IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ git status
On branch main
nothing to commit, working tree clean
punch@DESKTOP-9O81IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ git checkout feature-branch
Switched to branch 'feature-branch'
punch@DESKTOP-9O81IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (feature-branch)
$ touch greetings.txt
punch@DESKTOP-9O81IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (feature-branch)
$ git status
On branch feature-branch
Untracked files:
(use "git add <file>..." to include in what will be committed)
greetings.txt
nothing added to commit but untracked files present (use "git add" to track)
punch@DESKTOP-9O81IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (feature-branch)
$ git add .
punch@DESKTOP-9O81IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (feature-branch)
$ git commit -m "Greetings 2"
[feature-branch db17d32] Greetings 2
1 file changed, 1 insertion(+)
create mode 100644 greetings.txt
punch@DESKTOP-9O81IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (feature-branch)
$ git checkout main
Switched to branch 'main'
punch@DESKTOP-9O81IU8 MINGW64 /e/GeminiSolutions/Git_Sample_Repo (main)
$ git merge feature-branch
Auto-merging greetings.txt
CONFLICT (add/add): Merge conflict in greetings.txt
Automatic merge failed; fix conflicts and then commit the result.

How Do We Resolve Merge Conflicts ?

1. **Git detects the conflict:**
   When we run a merge, Git will pause the process and inform us that conflicts exist.
2. **Conflicted files are marked:**
   Git marks the areas of conflict inside the file using special markers like `<<<<<<<`, `=======`, and `>>>>>>>`.
3. **Developer reviews and edits the file manually:**
   We read the conflicting changes and decide what the final version should be keeping one side, both, or rewriting the content.
4. **Mark the conflict as resolved:**
   After editing, we tell Git that we have resolved the conflict by staging the file.
5. **Finish the merge with a commit:**
   Finally, we create a new commit to complete the merge process**.**

---

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 the project that is hosted on a server, such as GitHub, GitLab, or Bitbucket. It is used to share the code with others and collaborate with team members over a network or the internet.

A **local repository**, on the other hand, is stored on your personal computer. It contains your project files and a hidden `.git` folder where Git tracks the full history of changes.

Differences between Local and Remote Repositories:

Aspect	Local Repository	Remote Repository
Location	On your own computer	On a remote server (e.g., GitHub)
Accessibility	Accessible only to you	Can be accessed by multiple collaborators
Internet Required	No	Yes, to push or pull changes
Purpose	Personal development and testing	Collaboration, backup, and code sharing
Common Commands	`git commit`, `git status`, `git branch`	`git push`, `git pull`, `git fetch`

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

Difference Between git fetch and git pull

Both git fetch and git pull are used to get updates from a remote repository, but they behave differently.

git fetch

- Downloads the latest changes from the remote repository.
- Does not merge the changes into your current branch automatically.
- Allows you to review changes before applying them.

Use when: We want to see what's new on the remote without affecting our local code immediately.

git pull

- Combines two actions: `git fetch` + `git merge`.
- Automatically updates your current branch with the latest changes from the remote.
- Merges remote changes directly into your working branch.

Use when: We want to bring our local branch up to date with the remote branch immediately.

Answer 14, 15, 16, 17 :

```
punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions
$ git clone https://github.com/TanyaChaudhary010704/GitPracticeSample.git
Cloning into 'GitPracticeSample'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.

punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions
$ ls
 ClonePractice/                    GratuityNominationForm.pdf
 Form-11.pdf                      'Re_ Freshers -2025 Pre Training Material QA (Review)'/
 Gemini_Employment_agreement.pdf   Tanya_Chaudhary_OfferLetter.pdf
 Gemini_Solutions_agreement.pdf    Training_Assignments_1/
 GitPracticeSample/                form11PDF.pdf
 Git_Sample_Repo/

punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions
$ cd GitPracticeSample/

punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/GitPracticeSample (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/GitPracticeSample (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/GitPracticeSample (main)
$ git add .

punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/GitPracticeSample (main)
$ git commit -m "Modified Readme.md"
[main 3a5928d] Modified Readme.md
 1 file changed, 2 insertions(+)

punch@DESKTOP-9081IU8 MINGW64 /e/GeminiSolutions/GitPracticeSample (main)
$ git push -u origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 301 bytes | 301.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/TanyaChaudhary010704/GitPracticeSample.git
   337541a..3a5928d  main -> main
branch 'main' set up to track 'origin/main'.
```

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:

  If you want to edit the message of the most recent commit**,** and we haven't pushed it yet to a remote repository, we can use:

  ```
  git commit --amend
  ```

  ### What it does:

  - Opens our default editor with the last commit message.
  - We can edit the message, save, and close the editor.
  - The commit content stays the same unless you also stage new changes before running the command.

  ### Important:

  - Only use **`--amend`** if the commit has not been pushed to a shared remote, or we'll need to force-push (which can affect collaborators).

  If we have already pushed the commit, you'd need to run:

  ```
  git commit --amend
  git push origin branch-name --force
  ```

  Use this with caution on shared branches. Let me know if you want to edit older commits or work with `rebase`.
