Pre Assignment-Git Usage

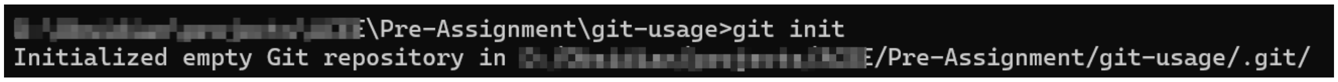
Group 14

March 22, 2025

1 Part 1

1.1 Task 1

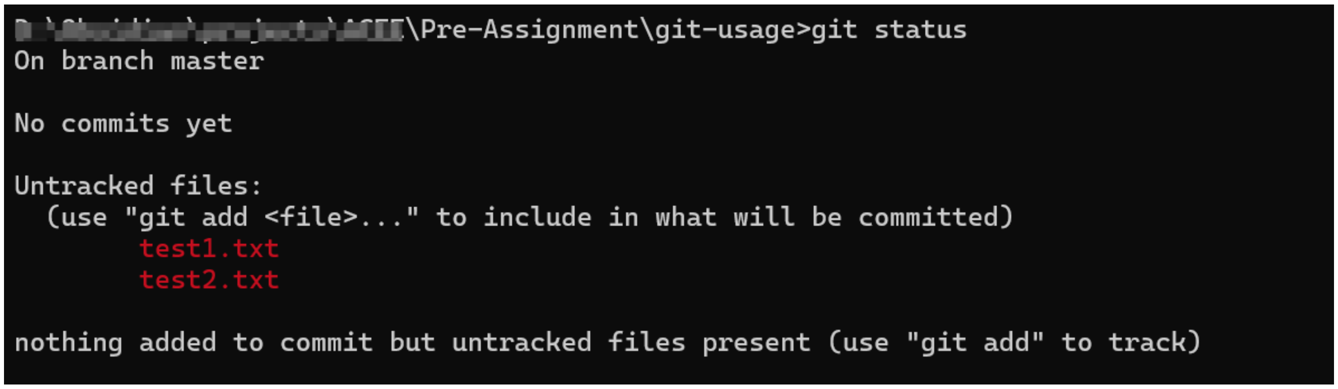
Initialize the repository.



```
[redacted]\Pre-Assignment\git-usage>git init
Initialized empty Git repository in [redacted]/Pre-Assignment/git-usage/.git/
```

Figure 1: use ‘git init’ to initialize the repository

Add the file ‘test1.txt’ ‘test2.txt’ as the new file.



```
[redacted]\Pre-Assignment\git-usage>git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test1.txt
        test2.txt

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

Figure 2: use ‘git status’ to check out status of all files

Question: What changes after tracking your file? Write down your comprehension of status of eachfile. (Untracked,Modified,Staged,Committed)

Answer: After tracking my file, the status is transformed from ‘Untracked’ to ‘Staged’.

- Untracked:

  The file is new and it’s not under the control of git. Git will not track its changes. If you want to stage it, you can use ‘git add <file name>’ to turn it into staged.

- Modified:

  The file is tracked, but modified. It differs from the latest version of the one in repository. You can use 'git add <file name>' to stage it.

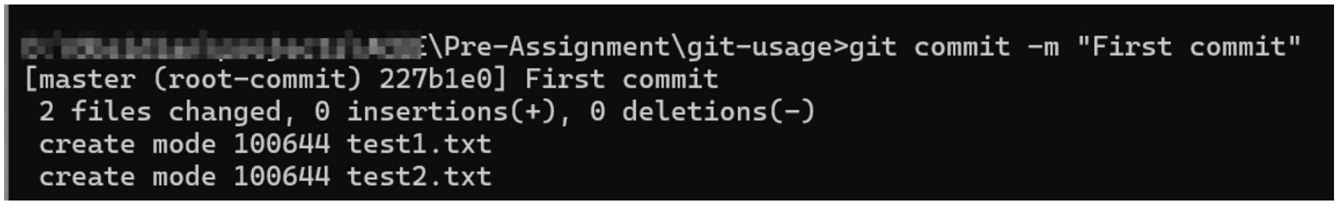
- Staged:

  The file is ready to commit and is storaged in the 'Staging Area'.

- Committed:

  The file permanently stored in the repository as a new commit in the history. You can navigate through the file's history by moving the HEAD pointer to a specific commit.

Commit the files.



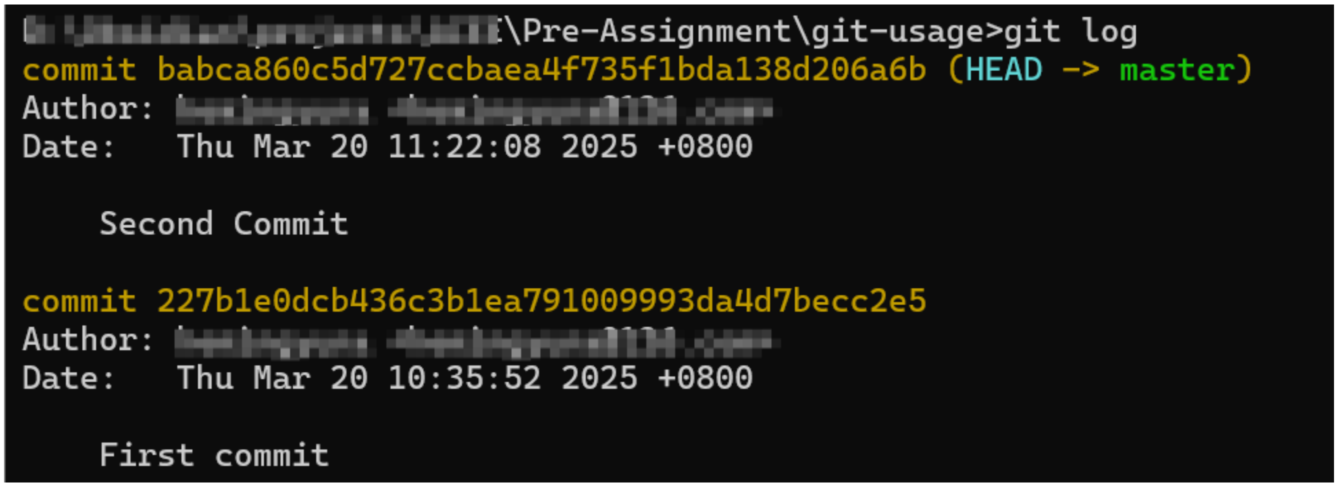
```
[redacted]\Pre-Assignment\git-usage>git commit -m "First commit"
[master (root-commit) 227b1e0] First commit
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 test1.txt
 create mode 100644 test2.txt
```

Figure 3: use 'git commit' to add commit to the repository

Question: What happens if you didn't track the file? Which status of file will be committed?

Answer: If a file is untracked, it will not be included in any commit. Only files in the staged state are committed when you run git commit.

use 'git log' to inspect all the commits



```
[redacted]\Pre-Assignment\git-usage>git log
commit babca860c5d727ccbaea4f735f1bda138d206a6b (HEAD -> master)
Author: [redacted]
Date:   Thu Mar 20 11:22:08 2025 +0800

    Second Commit

commit 227b1e0dcb436c3b1ea791009993da4d7becc2e5
Author: [redacted]
Date:   Thu Mar 20 10:35:52 2025 +0800

    First commit
```

Figure 4: use 'git log' to inspect the commits

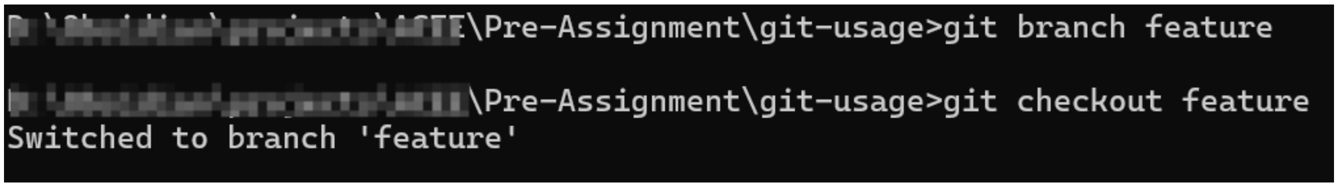
Question: What is the **HEAD** pointer?

Answer: The **HEAD** pointer points the current version of the repository. By moving the HEAD pointer to a specific commit, you can review the historical changes of the file.

2 Part 2

2.1 Task 2

Make a new branch



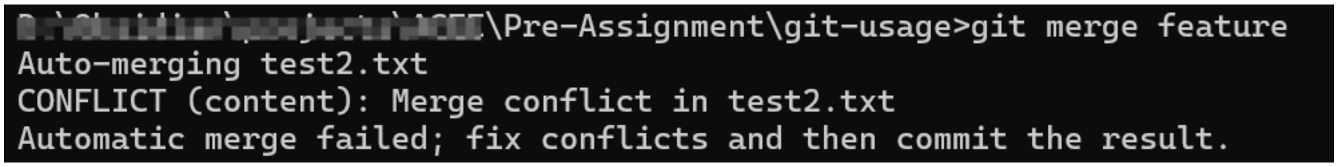
```
[redacted]\Pre-Assignment\git-usage>git branch feature

[redacted]\Pre-Assignment\git-usage>git checkout feature
Switched to branch 'feature'
```

Figure 5: use 'git branch' to create a new branch

Question: What's the difference between committing to main branch and feature branch?

Answer: There's no obvious difference between them. Commits to the feature branch are isolated from the main branch. You can use 'git merge' to integrate your changes into main when ready.

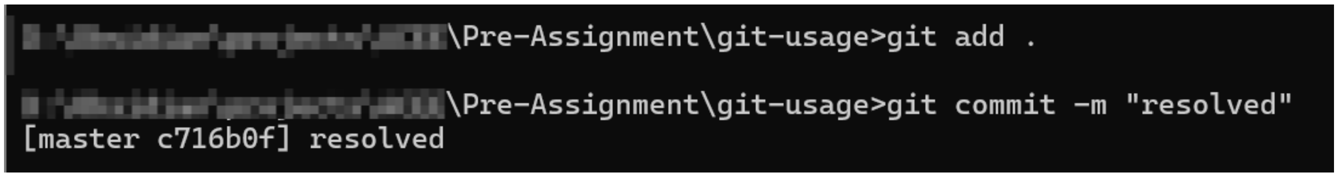
When merge the feature branch into main branch, Git reports an error.



```
[redacted]\Pre-Assignment\git-usage>git merge feature
Auto-merging test2.txt
CONFLICT (content): Merge conflict in test2.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Figure 6: the conflict when merging two branches

Resolve the conflict and complete the merge.



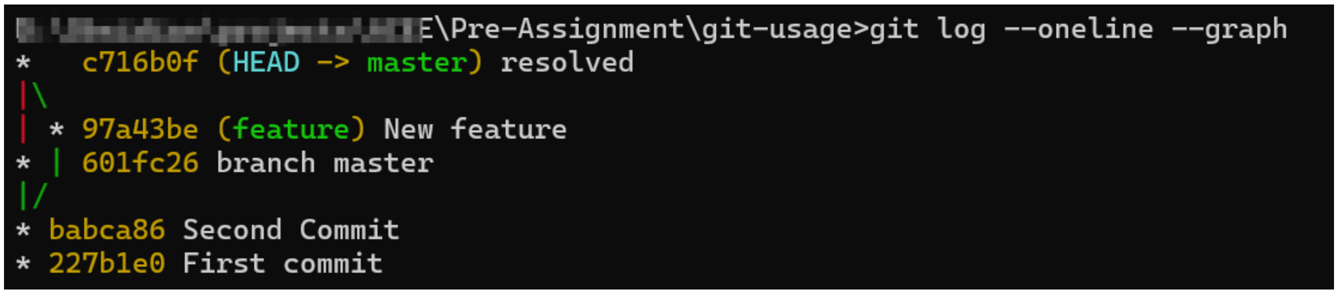
```
[redacted]\Pre-Assignment\git-usage>git add .

[redacted]\Pre-Assignment\git-usage>git commit -m "resolved"
[master c716b0f] resolved
```

Figure 7: resolve the conflict

Question: What causes conflicts? Why git didn't merge these conflict files automatically?

Answer: Conflicts happen when two branches modify the same part of a file in different ways, and Git cannot determine which version to keep. You can determine the version that you can keep and resolve the conflict, which makes the merge successful.

Inspect the merge history

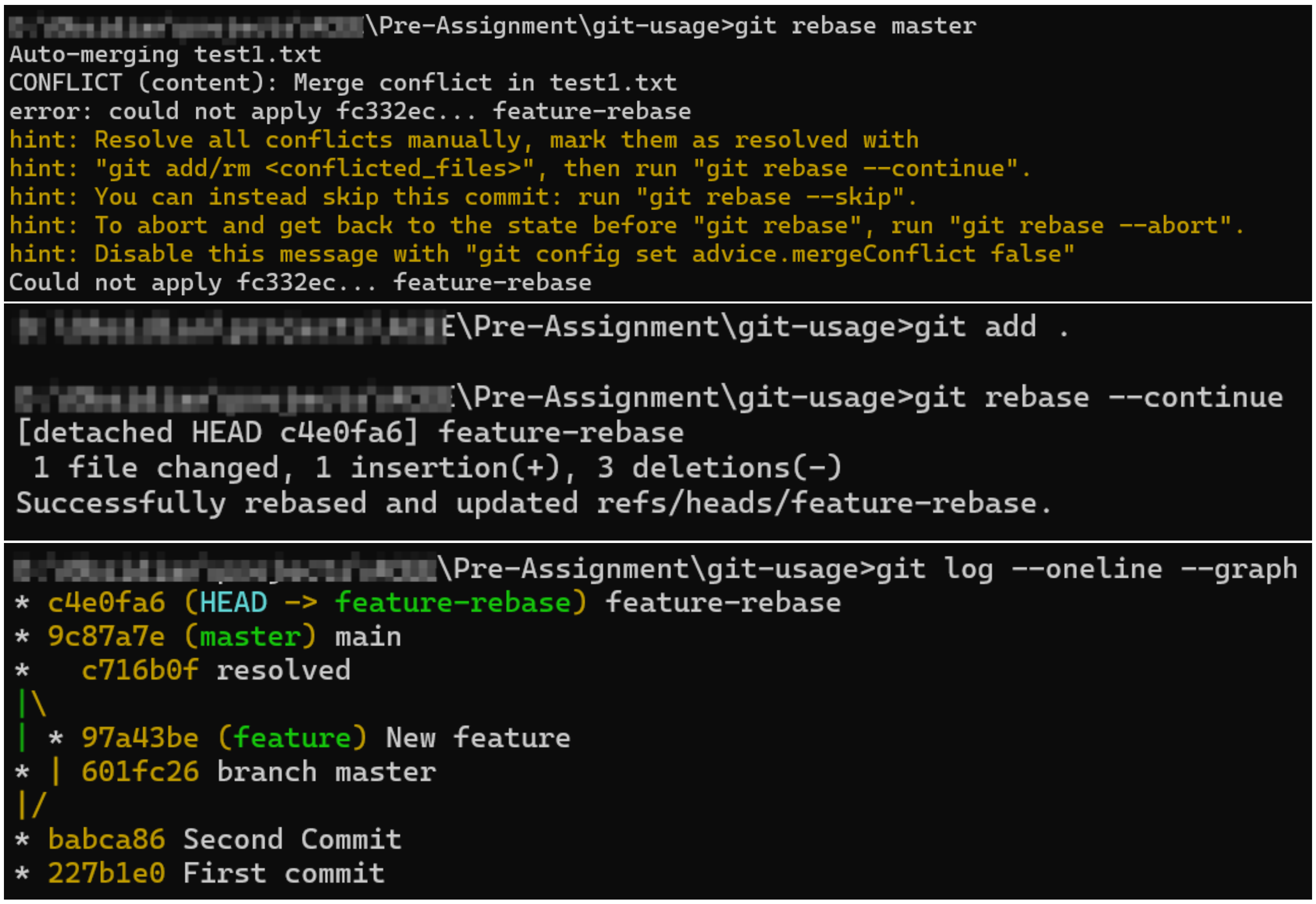


```
[redacted]E\Pre-Assignment\git-usage>git log --oneline --graph
*   c716b0f (HEAD -> master) resolved
|\
| * 97a43be (feature) New feature
* | 601fc26 branch master
|/
* babca86 Second Commit
* 227b1e0 First commit
```

Figure 8: commit history

2.2 Task 3

Rebase the branch 'feature-rebase' onto main. And Git pauses due to the conflict.



```
[redacted]\Pre-Assignment\git-usage>git rebase master
Auto-merging test1.txt
CONFLICT (content): Merge conflict in test1.txt
error: could not apply fc332ec... feature-rebase
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --abort".
hint: Disable this message with "git config set advice.mergeConflict false"
Could not apply fc332ec... feature-rebase
```

```
[redacted]E\Pre-Assignment\git-usage>git add .

[redacted]\Pre-Assignment\git-usage>git rebase --continue
[detached HEAD c4e0fa6] feature-rebase
 1 file changed, 1 insertion(+), 3 deletions(-)
Successfully rebased and updated refs/heads/feature-rebase.
```

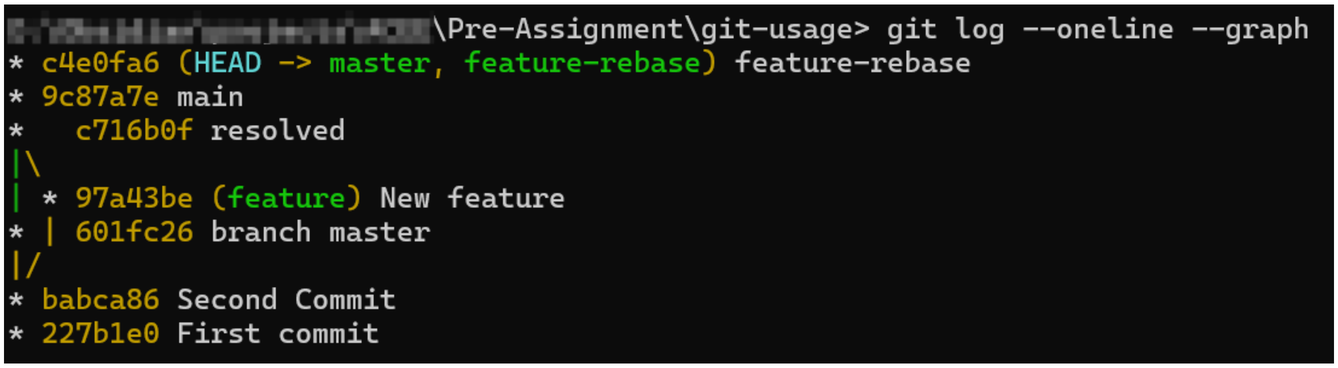
```
[redacted]\Pre-Assignment\git-usage>git log --oneline --graph
* c4e0fa6 (HEAD -> feature-rebase) feature-rebase
* 9c87a7e (master) main
*   c716b0f resolved
|\
| * 97a43be (feature) New feature
* | 601fc26 branch master
|/
* babca86 Second Commit
* 227b1e0 First commit
```

Figure 9: Rebase

Question: How does rebasing work? What's the difference between merge and rebase?

Answer: Rebase works by replaying the changes made in the feature branch onto the master branch, resulting in a linear and clean commit history. In contrast, merge preserves the original branch structure, which can lead to a more complex commit history. Generally, rebase is used only on private branches, while merge is used on public branches to clarify the division of work.

Merge the rebased branch



```
[redacted]\Pre-Assignment\git-usage> git log --oneline --graph
* c4e0fa6 (HEAD -> master, feature-rebase) feature-rebase
* 9c87a7e main
*   c716b0f resolved
|\
| * 97a43be (feature) New feature
* | 601fc26 branch master
|/
* babca86 Second Commit
* 227b1e0 First commit
```

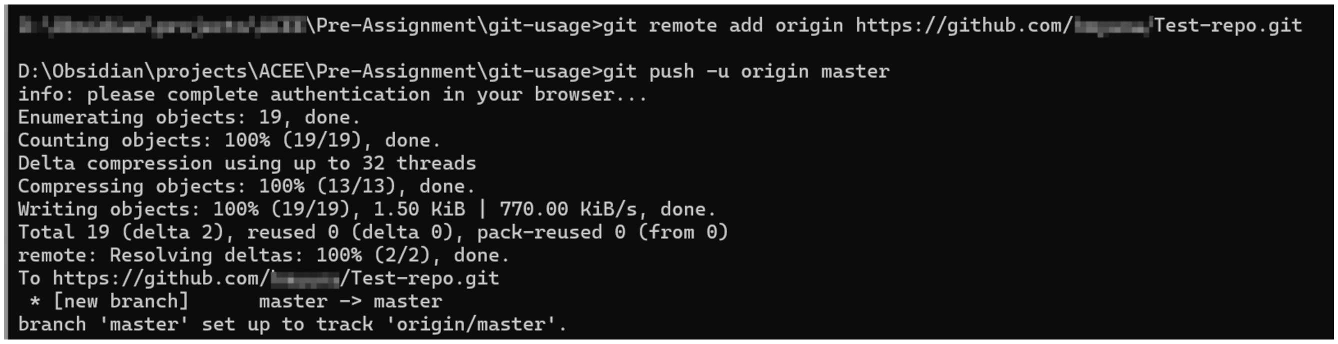
Figure 10: Merge the rebased branch

Question: In which condition could “Fast Forward”? What’s the advantage of rebasing compared to merge directly **Answer**: A fast-forward merge occurs when the target branch is the direct ancestor of the merging branch. Since all changes are already in a linear history, Git simply moves the branch pointer without creating a merge commit. Compared to merge directly, rebasing creates a linear history which is cleaner than the one created by merging directly.

3 Part 3

3.1 Task 4

Connect the local repository to the remote one, and push.



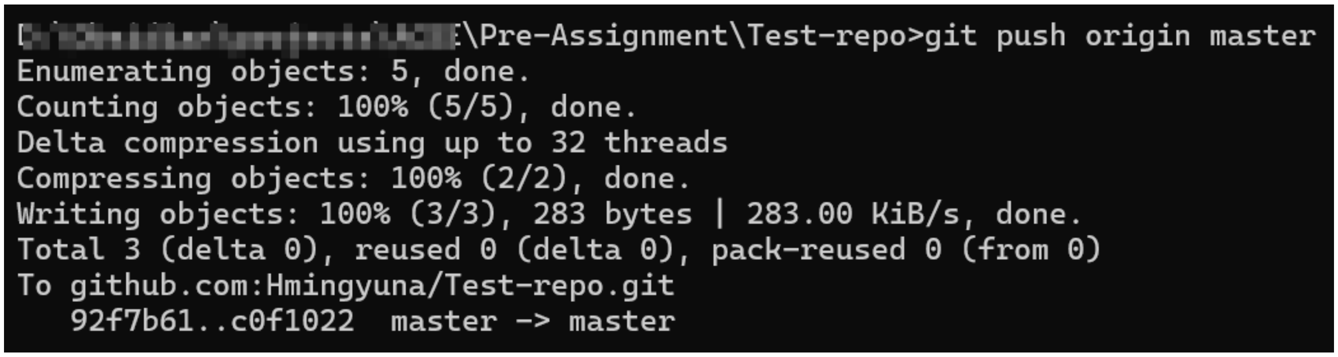
```
[redacted]\Pre-Assignment\git-usage>git remote add origin https://github.com/[redacted]Test-repo.git

D:\Obsidian\projects\ACEE\Pre-Assignment\git-usage>git push -u origin master
info: please complete authentication in your browser...
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 32 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (19/19), 1.50 KiB | 770.00 KiB/s, done.
Total 19 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/[redacted]/Test-repo.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

Figure 11: push the repository to the remote branch

Question: What does push do? Why might you need the -u flag?

Answer: Push synchronizes the local commits to the remote repository, and the -u flag links the local branch with the remote branch to simplify operations.

Push the local repository to the remote repository.



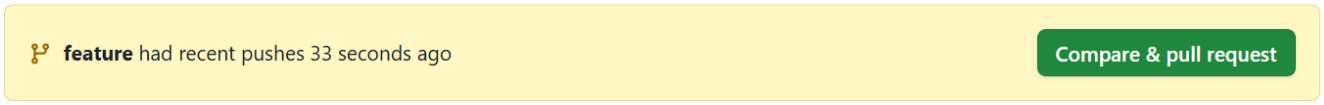
```
[redacted]\Pre-Assignment\Test-repo>git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 32 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 283 bytes | 283.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:Hmingyuna/Test-repo.git
   92f7b61..c0f1022  master -> master
```

Figure 12: push to the forked repo

Question: What's the relationship between forked repository and origin repository. Will changes in forked repository affect origin repository

Answer:The forked repository is a branch of the origin. Changes made in the forked repository won't affect the original repository, but you can synchronize the modified parts by submitting a pull request.

Create a pull request.



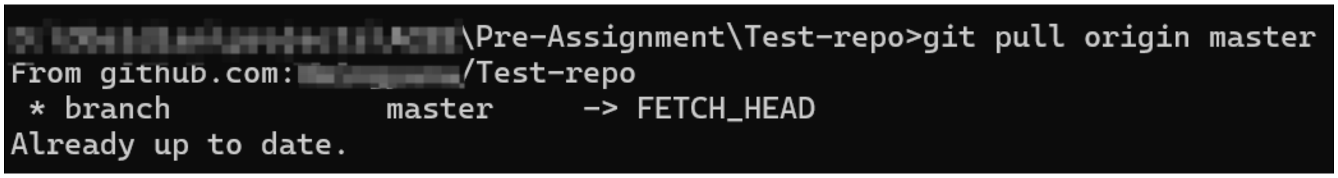
feature had recent pushes 33 seconds ago

Figure 13: PR

Question: What's the purpose of a pull request?

Answer:A pull request is used to propose and discuss changes before merging them into the target repository, helping code review and collaboration.

Pull changes from the remote.



```
[redacted]\Pre-Assignment\Test-repo>git pull origin master
From github.com:[redacted]/Test-repo
 * branch            master     -> FETCH_HEAD
Already up to date.
```

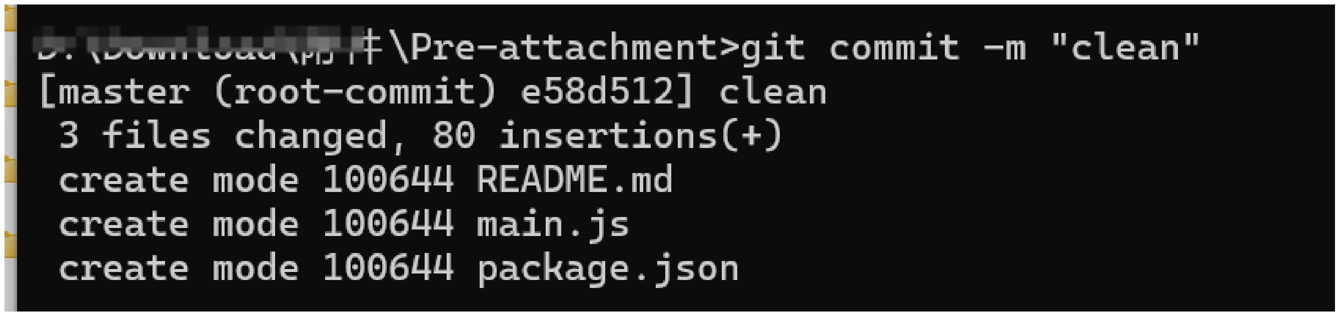
Figure 14: pull the changes

Question: What does pull do? Explain how pull request contributes in team cooperation?

Answer: Git pull fetches changes from a remote repository and automatically merges them into the local branch. A pull request, makes team cooperation easier by allowing team members to review, discuss, and approve proposed changes before they are merged into the main branch, ensuring the code quality.

4 Bonus

Write a gitignore.

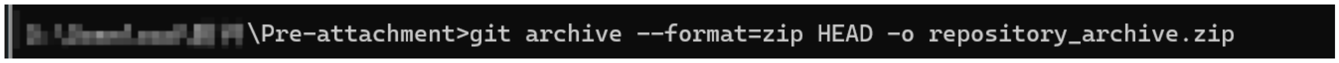


```
[redacted]\Pre-attachment>git commit -m "clean"
[master (root-commit) e58d512] clean
 3 files changed, 80 insertions(+)
 create mode 100644 README.md
 create mode 100644 main.js
 create mode 100644 package.json
```

Figure 15: use gitignore to get a clean git

Question: What's the contents of .gitignore? Why do we need .gitignore? What's the difference between *.tmp and /tmps/*.tmp in a .gitignore file? How can you archive a clean repository

Answer: The contents of gitignore means what files git will ignore and don't track. *.tmp → Ignores all files ending in .tmp in any directory. /tmps/*.tmp → Ignores only .tmp files inside the tmps/ root directory but not in subdirectories like tmps/subdir/file.tmp. You can use gitignore to get a clean archive.



```
[redacted]\Pre-attachment>git archive --format=zip HEAD -o repository_archive.zip
```

Figure 16: get a clean archive