Experiment No. 06

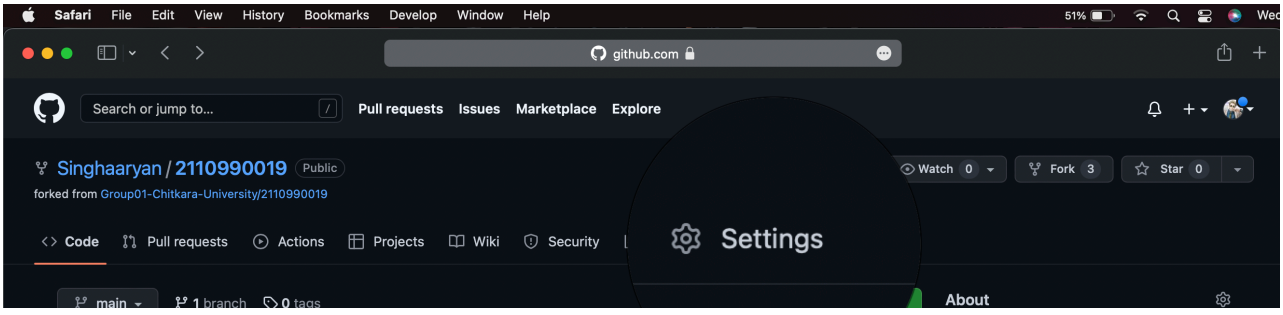
AIM: Add Collaborators on GitHub Repository

THEORY:
In GitHub, We can invite other GitHub users to become collaborators to our private repositories (which expires after 7 days if not accepted, restoring any unclaimed licenses). Being a collaborator, of a personal repository you can pull (read) the contents of the repository and push (write) changes to the repository. You can add unlimited collaborators on public and private repositories (with some per day limit restrictions). But, in a private repository, the owner of the repo can only grant write-access to the collaborators, and they can't have the read-only access.
GitHub also restricts the number of collaborators we can invite within a period of 24 hours. If we exceed the limit, then either we have to wait for 24-hours or we can also create an organization to collaborate with more people.

Actions that can be performed by collaborators:

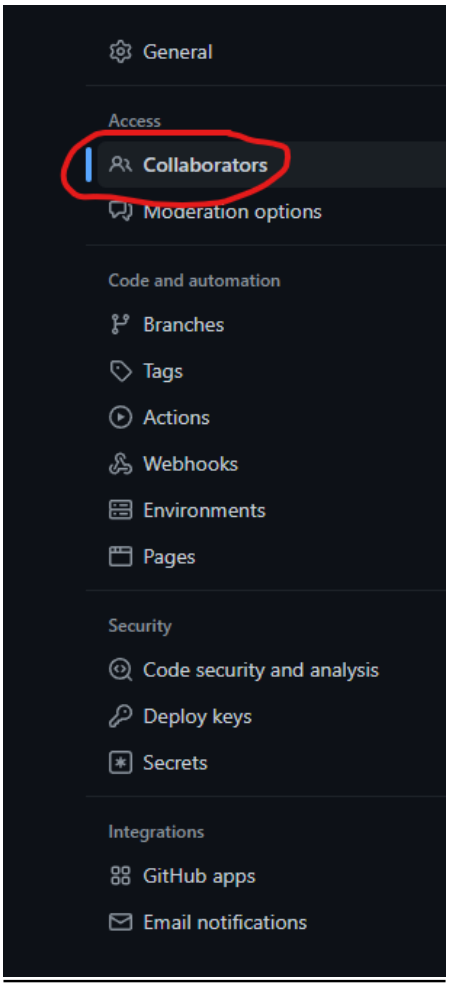
- Create, merge, and close pull requests in the repository
- Publish, view, install the packages
- Fork the repositories
- Make the changes on the repositories as suggested by the Pull requests.
- Mark issues or pull requests as duplicate
- Create, edit, and delete any comments on commits, pull requests, and issues in the repository
- Removing themselves as collaborators on the repositories.

Inviting Collaborators to your personal repositories

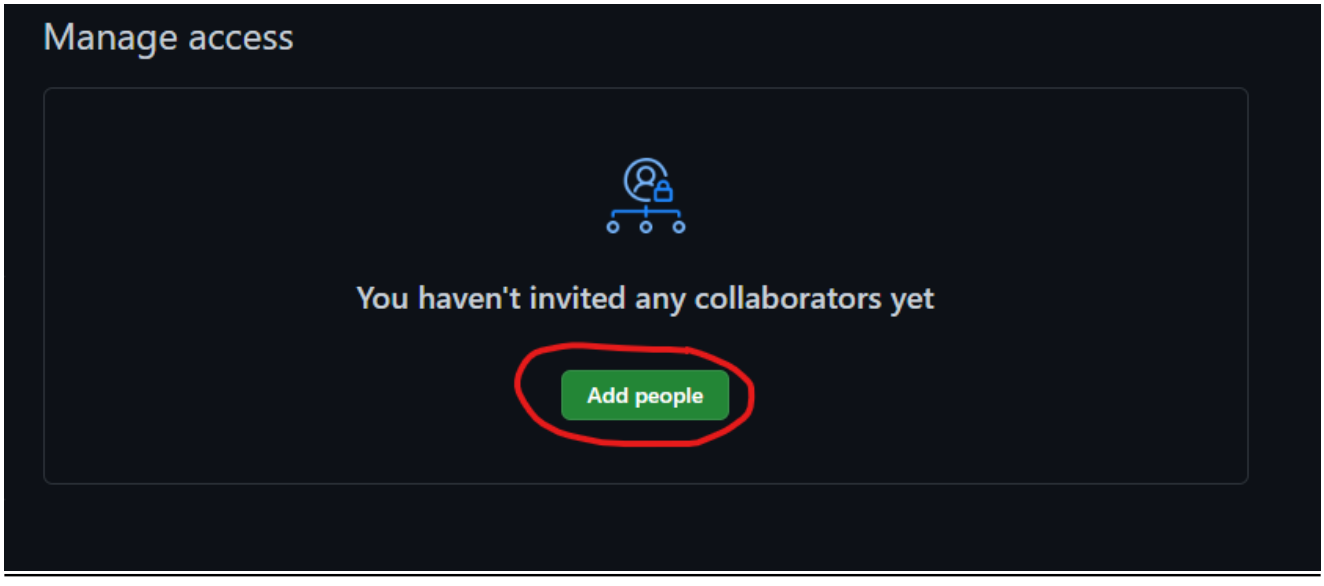


Step 1: Go to your repository.
Step 2: Click into the Settings.

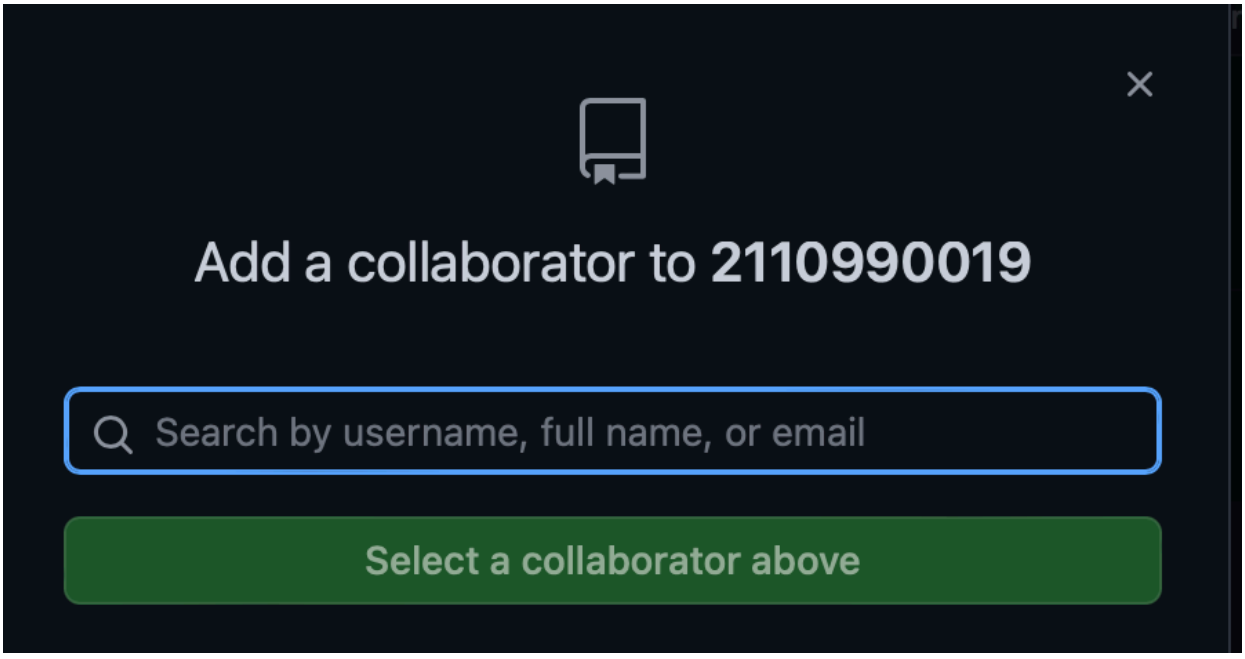
Step 3: In the left sidebar, click **Collaborators**.



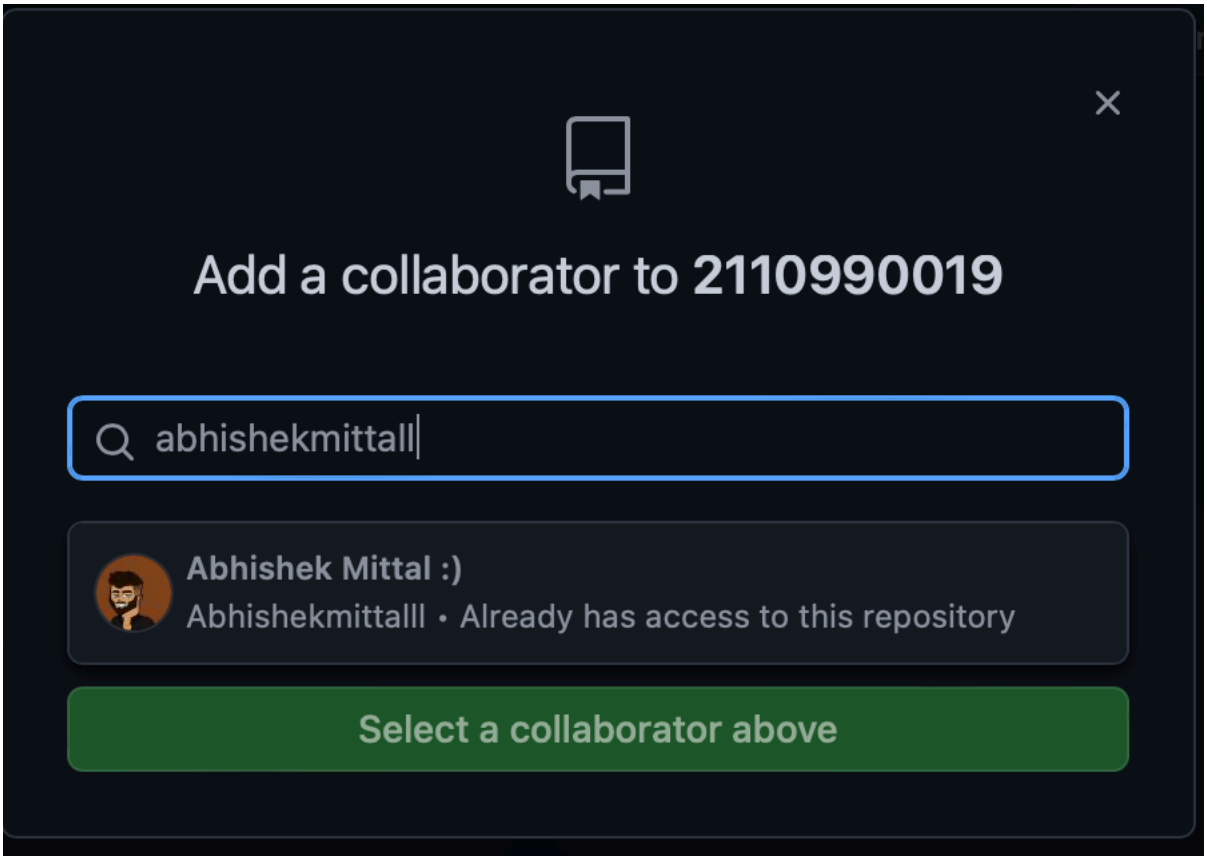
Step 4: Click on Add people.



Step 5: Then a search field will appear, where you can enter the username of the one's you want to add as collaborator.



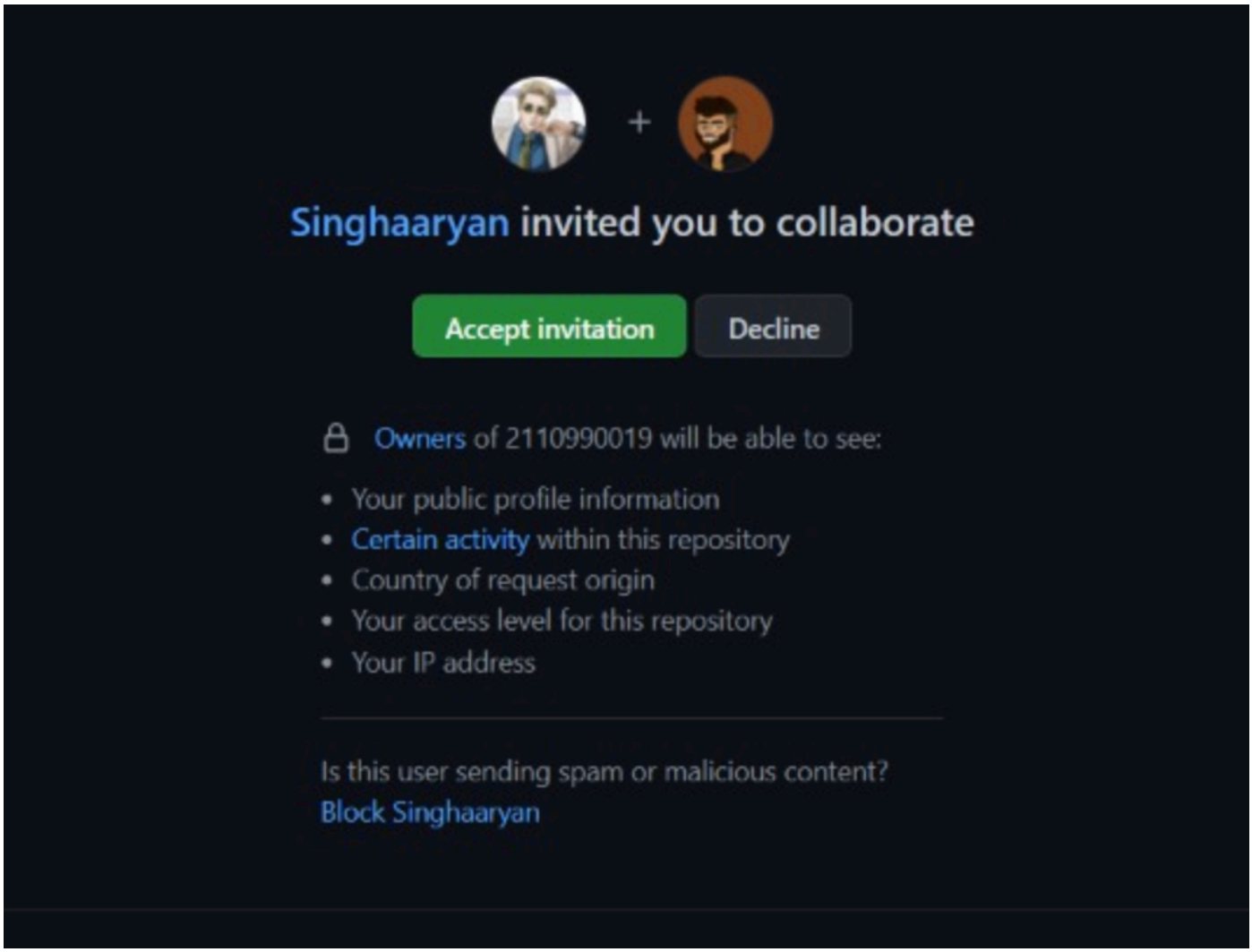
Step 6: Select the collaborator's username from the drop-down menu.



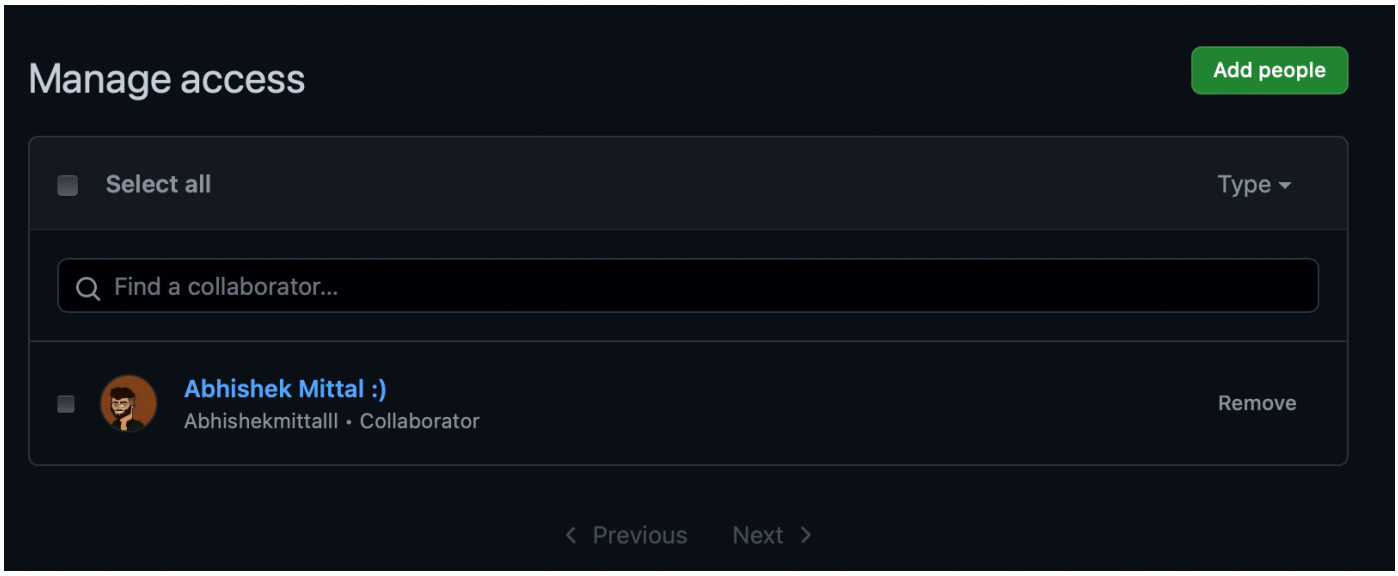
Step 7: Click Add collaborator (name) to this repository.

Step 8: We are done adding a single collaborator. Now, they will get a mail regarding invitation to your repository.

Step 9: Once they accept, they will have collaborator access to your repository. Till then it will be in pending invitation state.



Step 10: You can also add more collaborator and delete the existing one.



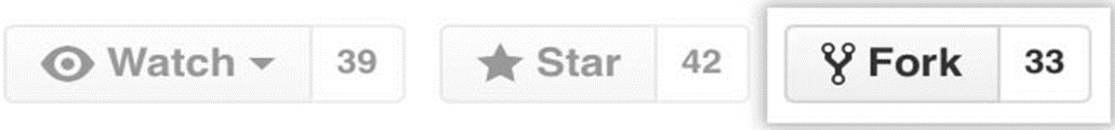
Removing collaborator permissions from a person contributing to a repository

Similar to above steps, we have to move to Your **Repository → Settings → Manage Access → Remove (**on the right side of collaborator username**)**

Experiment No. 07

AIM: Fork and Commit

THEORY:
A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. Propose changes to someone else's project: For example, we can use forks to propose changes related to fixing a bug.

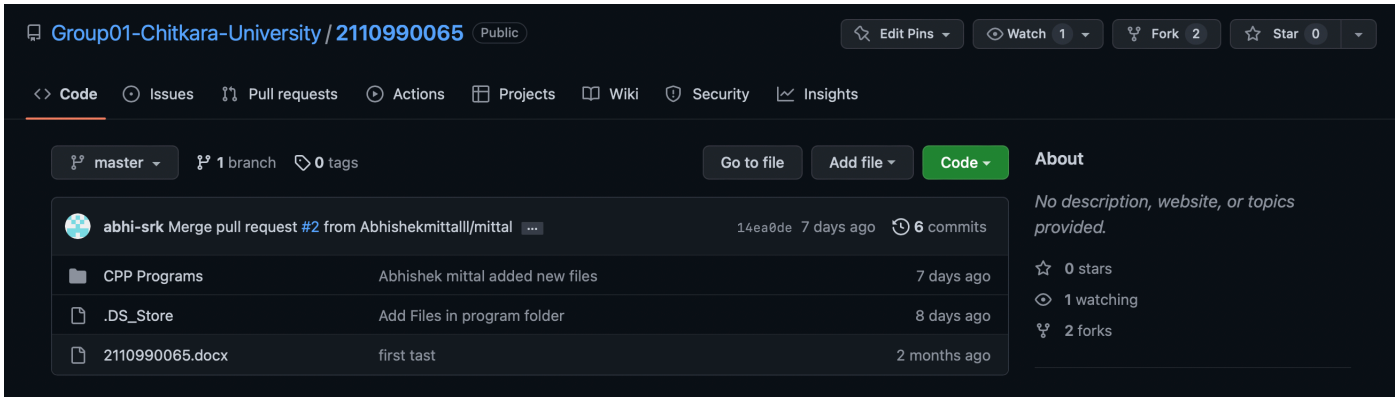


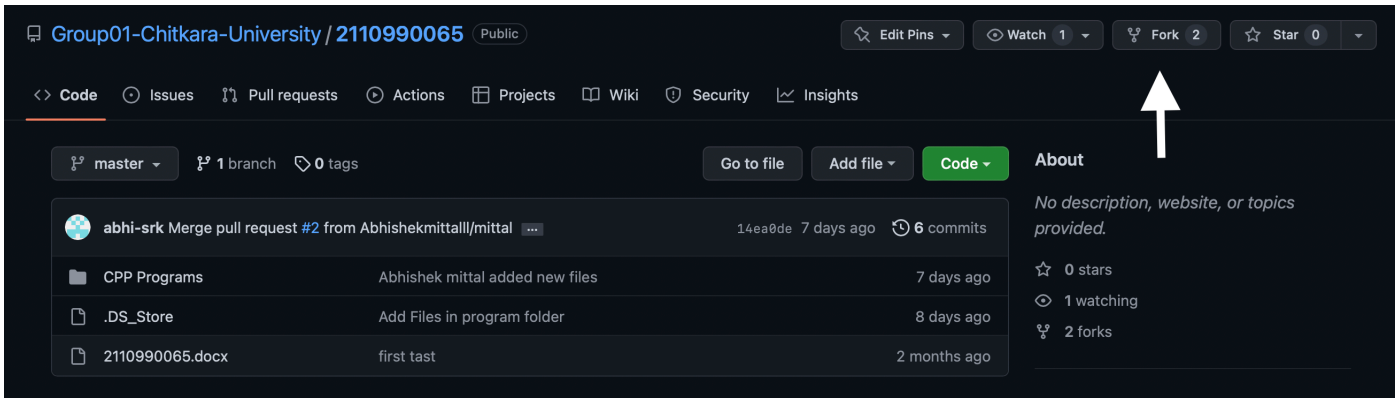
Rather than logging an issue for a bug you have found, you can:

- Fork the repository.
- Make the fix.
- Submit a pull request to the project owner.

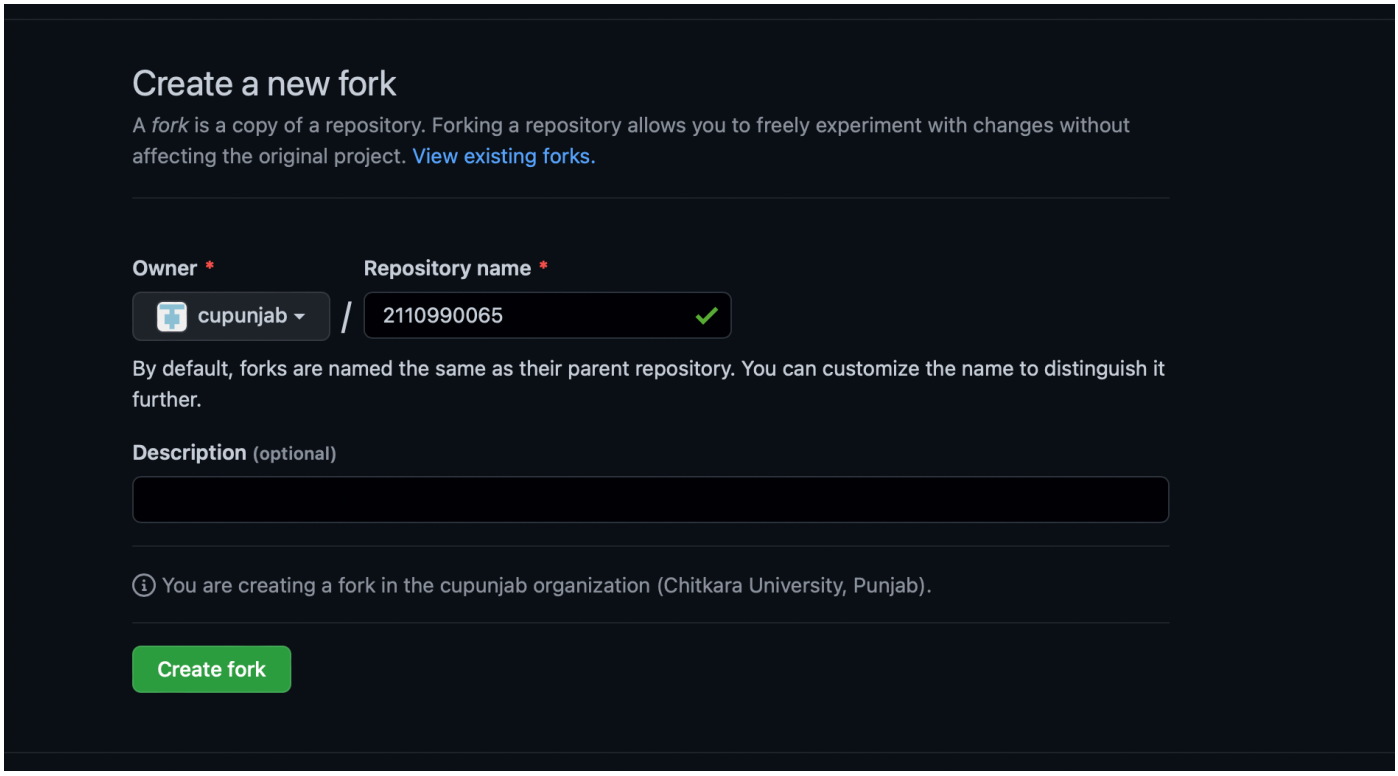
Forking a repository:

- On Github.com, navigate to the repository and search for the repo you want to fork.





- In the top-right corner of the page, click **Create Fork**.



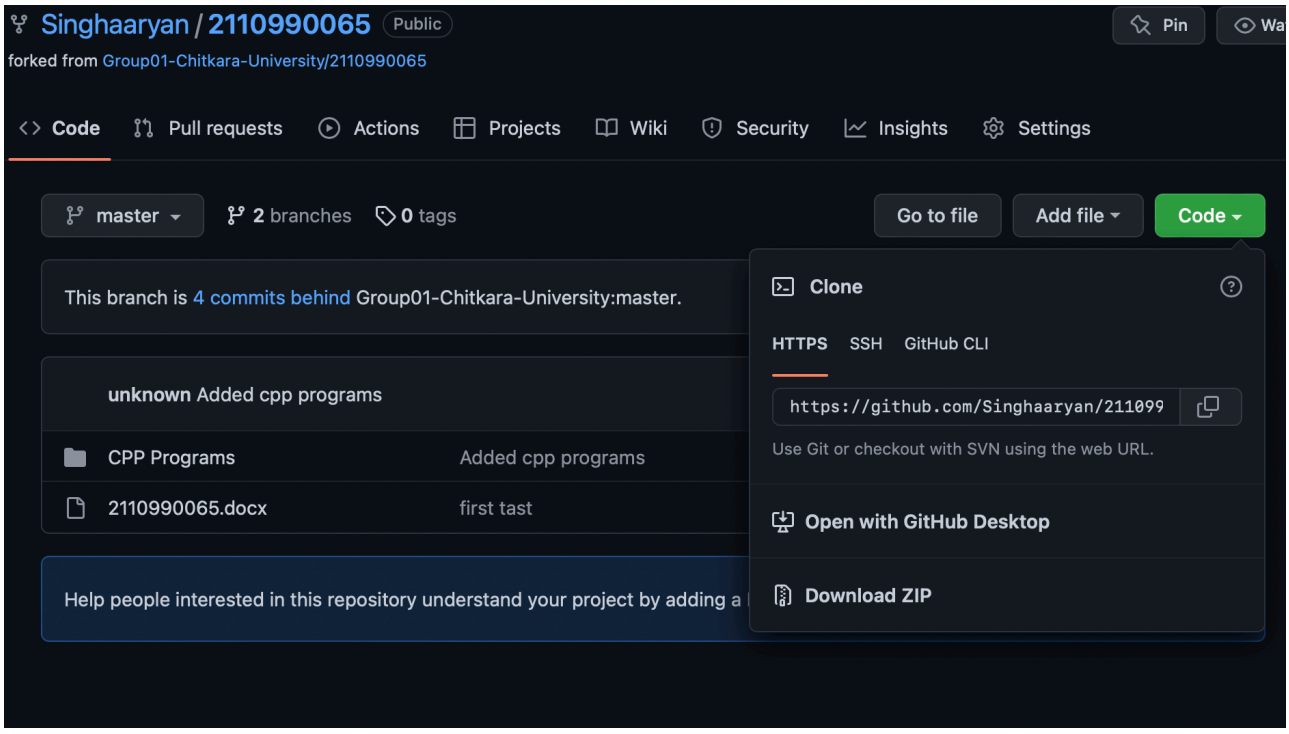
Cloning your forked repository:

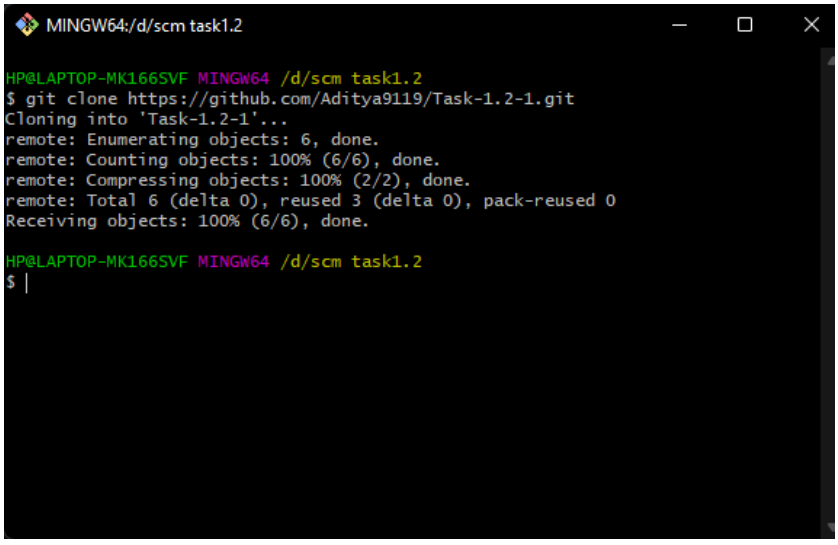
Right now, you have a fork of the Task-1.2 repository, but you do not have the files in that repository locally on your computer.

1. On GitHub.com, navigate to **your fork** of the Task-1.2 repository.
2. Above the list of files, click **Code**.

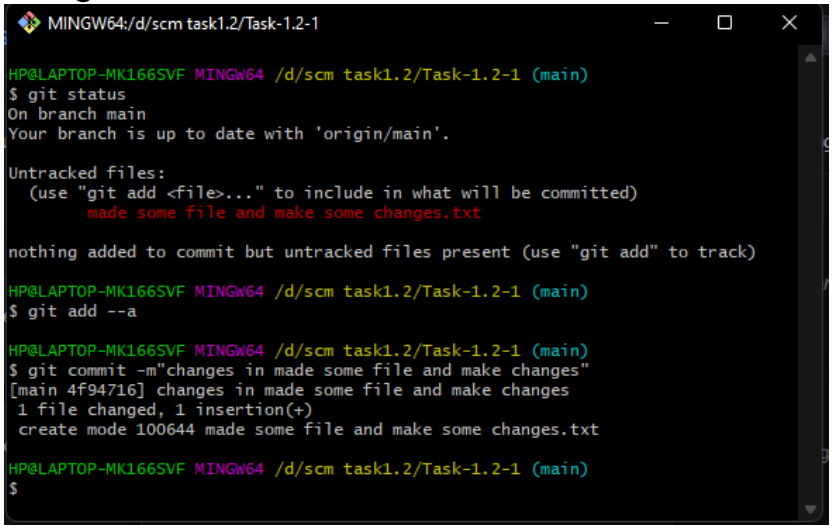
3. To clone a repository using HTTPS, click **HTTPS.** then copy the **URL**.

4. Create a new folder and open git bash from it.
5. Use the command **git clone <URL of repo>**. scm task1.2 will be clone in the folder.

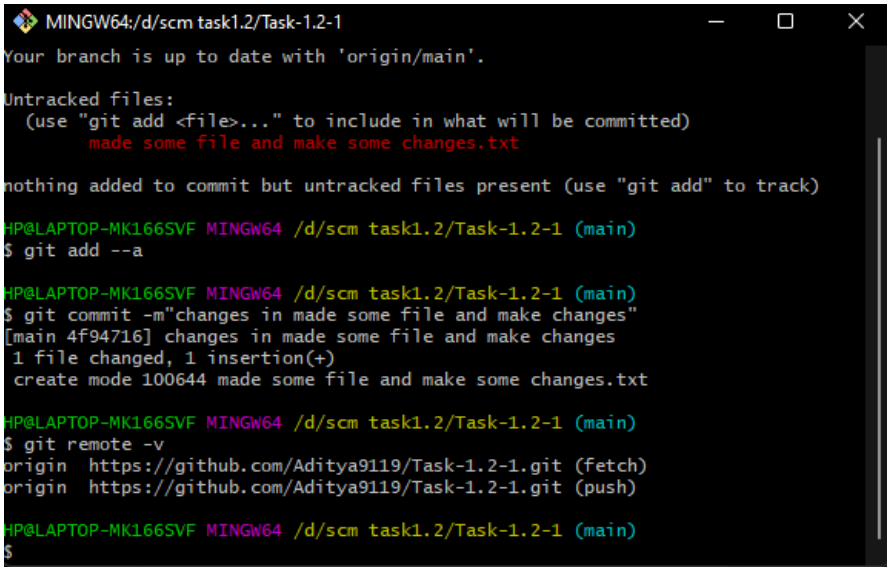




6. Now make some changes in .txt and open git bash from scm task 1.2 folder. Commit the changes made.



7. Check whether there is remote origin URL using **git remote -v command.**



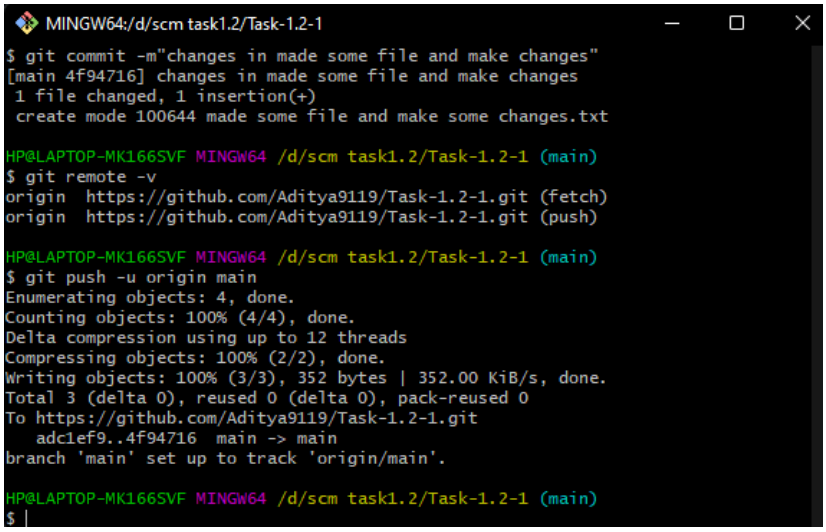
Git Commit Command:

The **"commit"** command is used to save your changes to the local repository.

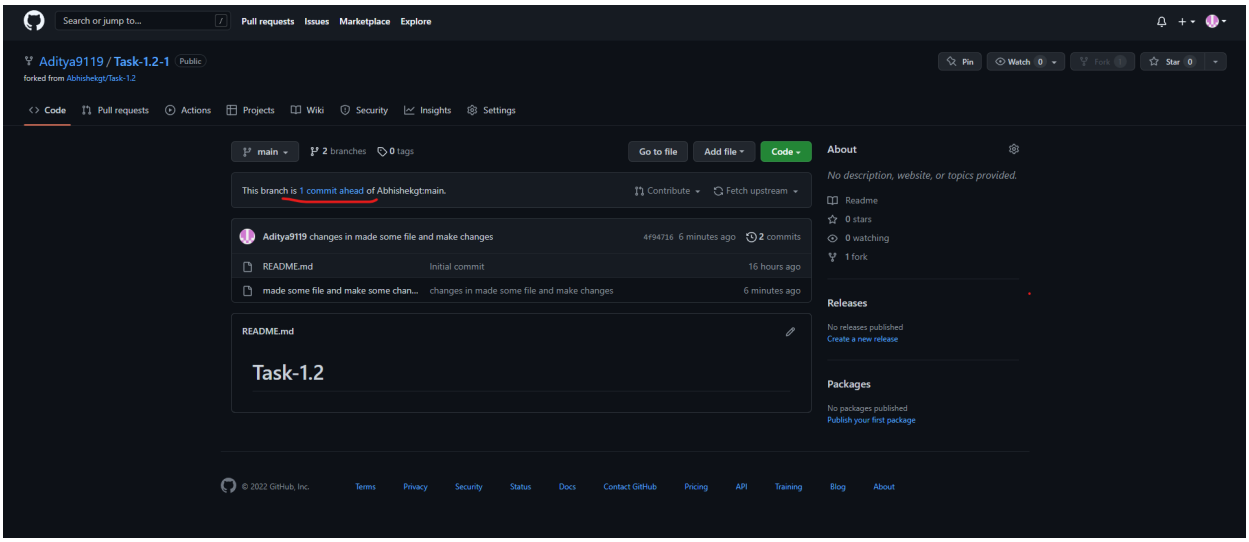
Note that you have to explicitly tell Git which changes you want to include in a commit before running the "git commit" command. This means that a file won't be automatically included in the next commit just because it was changed. Instead, you need to use the "git add" command to mark the desired changes for inclusion.

Using the "git commit" command only saves a new commit object in the local Git repository

8. Top push the commits on remote origin, use command **git push -u origin main.**



9. The commit will be reflected into your Git hub account.


Search or jump to...
Pull requests
Issues
Marketplace
Explore
Aditya9119 / Task-1.2-1
Public
forked from Abhishekgt/Task-1.2
Pin
Watch 0
Fork 1
Star 0
Code
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
main
2 branches
0 tags
Go to file
Add file
Code
This branch is 1 commit ahead of Abhishekgt:main.
Contribute
Fetch upstream
Aditya9119 changes in made some file and make changes
4f94716 6 minutes ago
2 commits
README.md
Initial commit
16 hours ago
made some file and make chan...
changes in made some file and make changes
6 minutes ago
README.md
Task-1.2
About
No description, website, or topics provided.
Readme
0 stars
0 watching
1 fork
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
© 2022 GitHub, Inc.
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About

Experiment No. 08

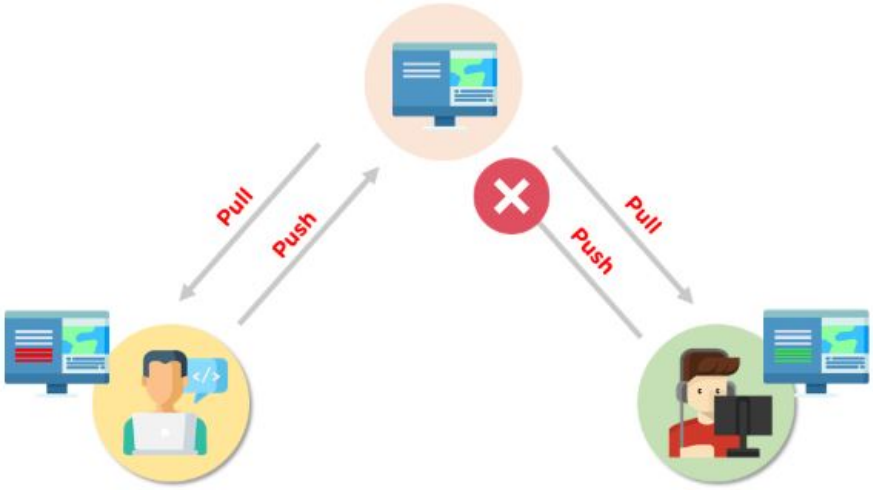
AIM: Merge and Resolve conflicts created due to own collaborators activity

THEORY:

What is a merge conflict?

A merge conflict is an event that takes place when Git is unable to automatically resolve differences in code between two commits. Git can merge the changes automatically only if the commits are on different lines or branches.

The following is an example of how a Git merge conflict works:



Let's assume there are two developers: Developer A and Developer B. Both of them pull the same code file from the remote repository and try to make various amendments in that file. After making the changes, Developer A pushes the file back to the remote repository from his local repository. Now, when Developer B tries to push that file after making the changes from his end, he is unable to do so, as the file has already been changed in the remote repository.

To prevent such conflicts, developers work in separate isolated branches. The Git merge command combines separate branches and resolves any conflicting edits.

Types of Git Merge Conflicts:

1. Starting the merge process: If there are changes in the working directory's stage area for the current project, merging won't start.

2. During the Merge Process: The failure during the merge process indicates that there is a conflict between the local branch and the branch being merged.

How to Resolve Merge Conflicts in Git?

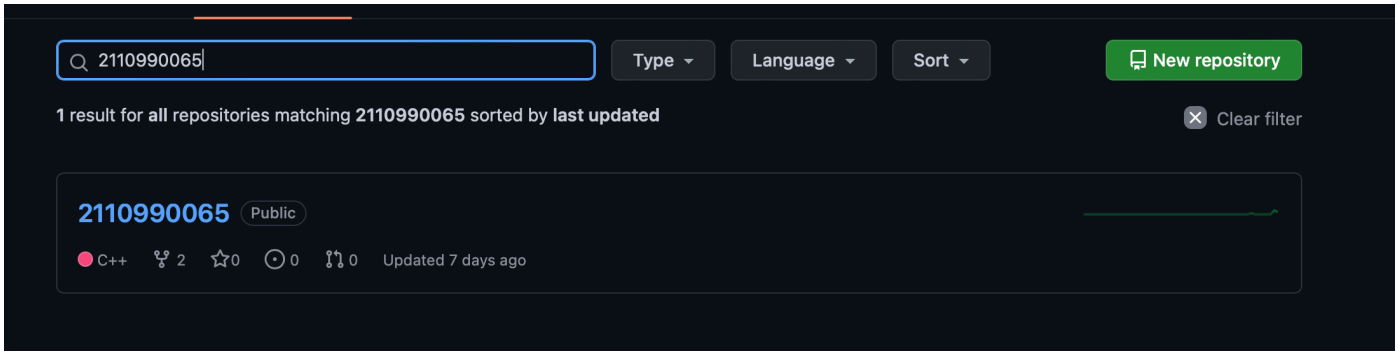
There are a few steps that could reduce the steps needed to resolve merge conflicts in Git.

- ❖ The easiest way to resolve a conflicted file is to open it and make any necessary changes
- ❖ After editing the file, we can use the git add a command to stage the new merged content
- ❖ The final step is to create a new commit with the help of the git commit command
- ❖ Git will create a new merge commit to finalize the merge.

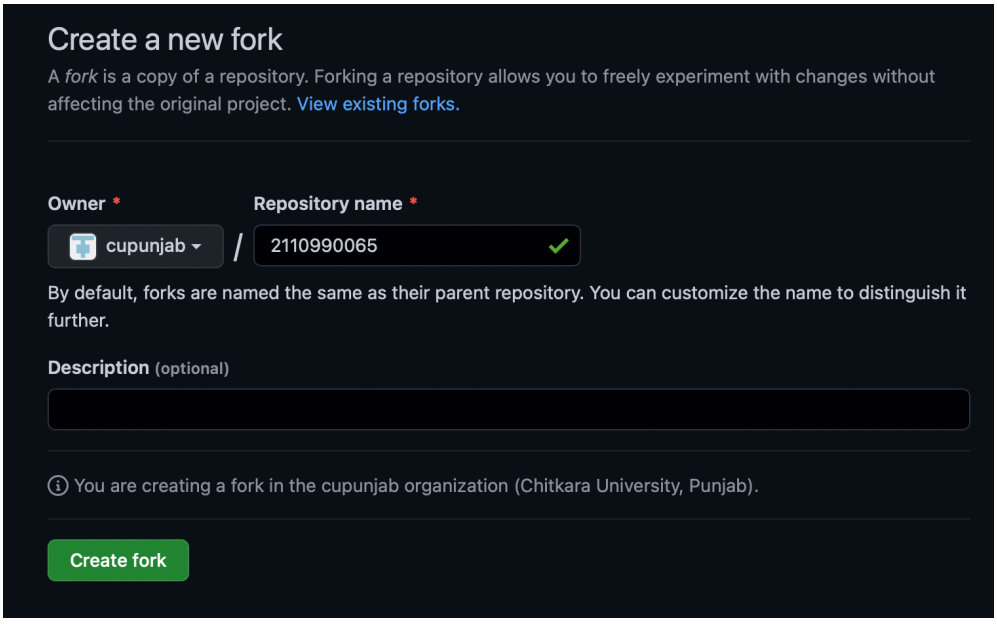
Demo: Resolving Git Merge Conflicts

There is a file named conflict1.txt which is commited in master branch in Abhishek's repo (id Abhishekgt).

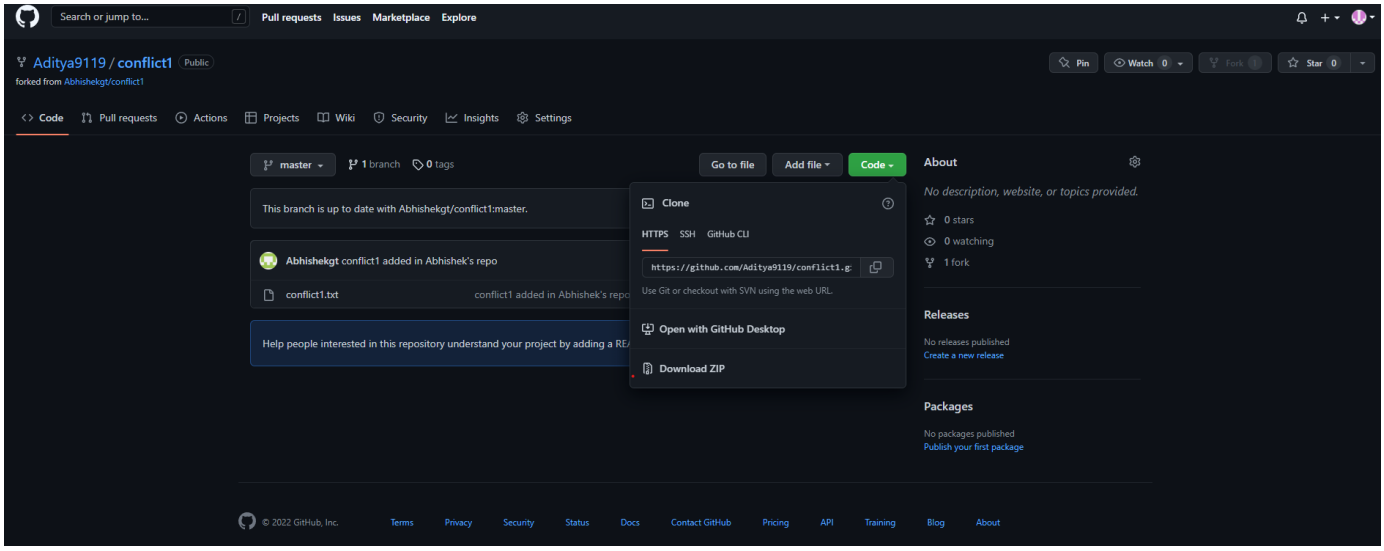
Step 1: Search for the repo to fork.



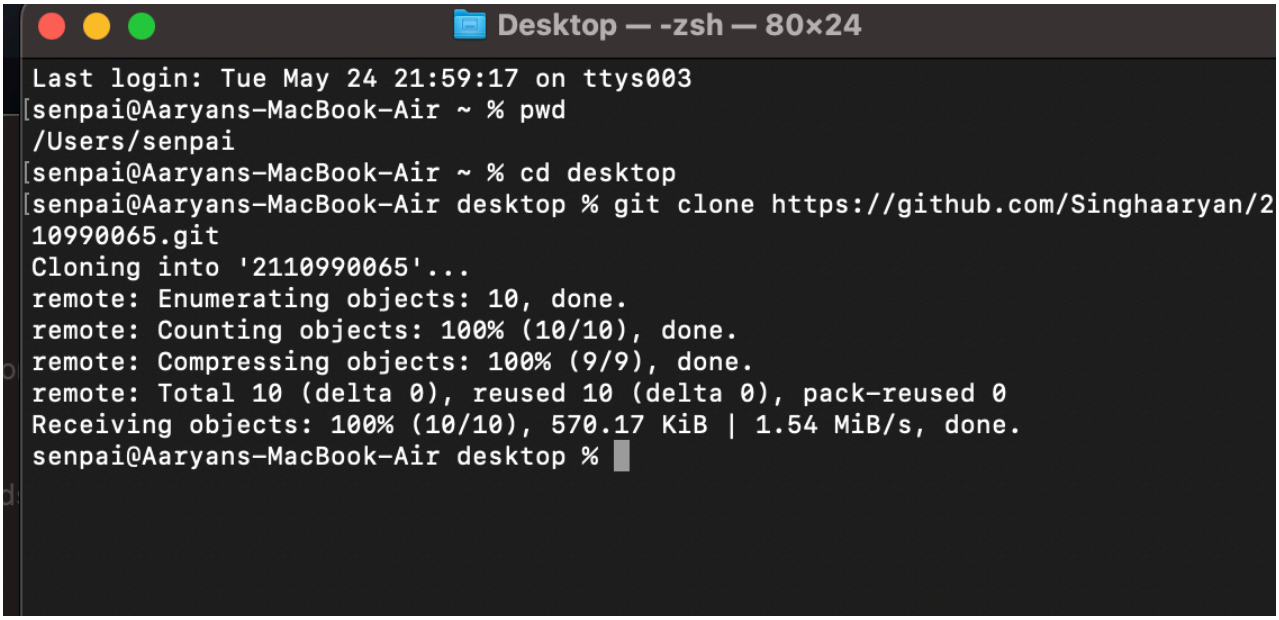
Step 2: Fork the repo by clicking on fork , and create the fork by clicking on create fork.



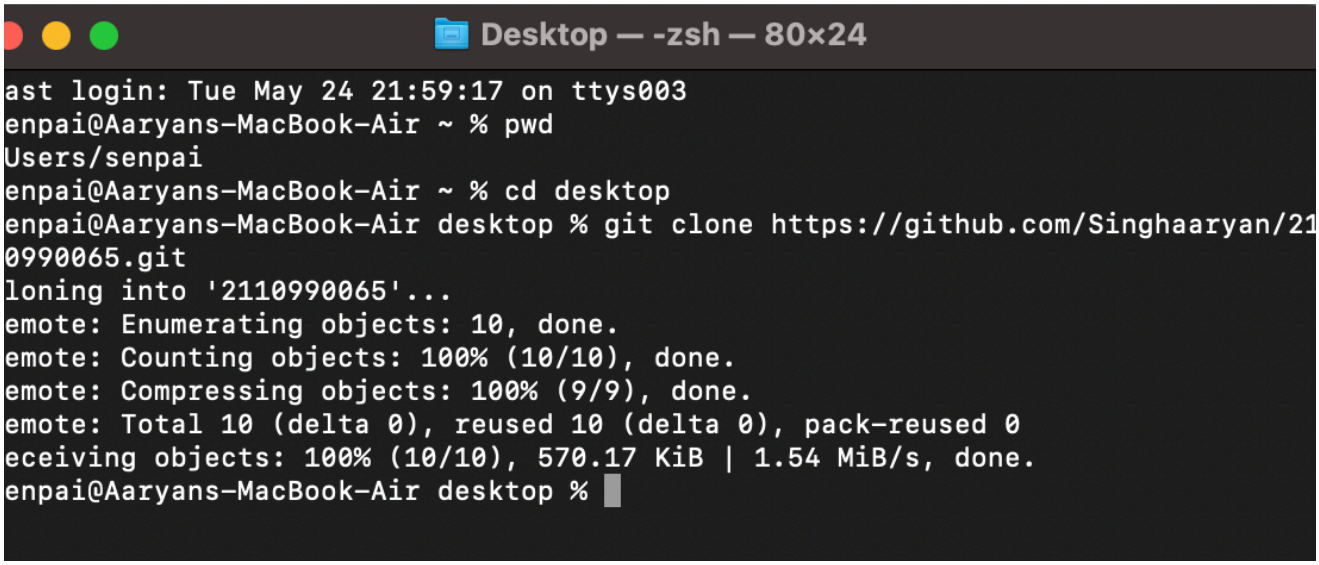
Step 3: Click on code and copy the link.



Step 4: open git bash on your window and clone the repo by command **git clone <link of the repo>.**



Step 5: Make a new branch and checkout in it .



Step 6: Make changes in the file(s) in there , stage and commit it. And then checkout in

```
senpai@Aaryans-MacBook-Air 2110990065 % git push origin aaryan
Username for 'https://github.com': Singhaaryan
Password for 'https://Singhaaryan@github.com':
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 5.42 KiB | 5.42 MiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
remote:
remote: Create a pull request for 'aaryan' on GitHub by visiting:
remote:      https://github.com/Singhaaryan/2110990065/pull/new/aaryan
remote:
To https://github.com/Singhaaryan/2110990065.git
 * [new branch]      aaryan -> aaryan
senpai@Aaryans-MacBook-Air 2110990065 %
```

master branch.

```
2110990065 — -zsh — 141×50
senpai@Aaryans-MacBook-Air 2110990065 % git commit -m "Add Files in program folder"
[aaryan c95635c] Add Files in program folder
 Committer: Aaryan Singh <senpai@Aaryans-MacBook-Air.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 .DS_Store
 create mode 100755 CPP Programs/exceptionhandling
senpai@Aaryans-MacBook-Air 2110990065 % git push origin aaryan
Username for 'https://github.com': Singhaaryan
Password for 'https://Singhaaryan@github.com':
remote: Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.
remote: Please see https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information.
fatal: Authentication failed for 'https://github.com/Singhaaryan/2110990065.git/'
senpai@Aaryans-MacBook-Air 2110990065 % $ git config --global user.name
zsh: command not found: $
senpai@Aaryans-MacBook-Air 2110990065 %  git config --global user.name
senpai@Aaryans-MacBook-Air 2110990065 % git config --global user.name "Singhaaryan"
senpai@Aaryans-MacBook-Air 2110990065 % git push origin aaryan
Username for 'https://github.com': Singhaaryan
Password for 'https://Singhaaryan@github.com':
remote: Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.
remote: Please see https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information.
fatal: Authentication failed for 'https://github.com/Singhaaryan/2110990065.git/'
senpai@Aaryans-MacBook-Air 2110990065 % git push origin aaryan
Username for 'https://github.com': Singhaaryan
Password for 'https://Singhaaryan@github.com':
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 5.42 KiB | 5.42 MiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
remote:
remote: Create a pull request for 'aaryan' on GitHub by visiting:
remote:      https://github.com/Singhaaryan/2110990065/pull/new/aaryan
remote:
To https://github.com/Singhaaryan/2110990065.git
 * [new branch]      aaryan -> aaryan
senpai@Aaryans-MacBook-Air 2110990065 %
```

Step 9: Merge secondary branch with master branch by command **git merge <branch name>**. A merge conflict will occur here.

```
Last login: Thu Jun  2 13:26:02 on ttys000
senpai@Aaryans-MacBook-Air 2110990019 % git branch
* main
senpai@Aaryans-MacBook-Air 2110990019 % git branch aaryan
senpai@Aaryans-MacBook-Air 2110990019 % git checkout aaryan
M       .DS_Store
Switched to branch 'aaryan'
senpai@Aaryans-MacBook-Air 2110990019 % ls
2110990019.pdf  CPP-Prog
senpai@Aaryans-MacBook-Air 2110990019 % cd CPP-Prog
senpai@Aaryans-MacBook-Air CPP-Prog % ls
BookAllocation.cpp      PROG!                   exceptionhandling
FUNCtion                Prog2                   prog3
FindPeak.cpp            bubblesort.cpp
senpai@Aaryans-MacBook-Air CPP-Prog % git branch
* aaryan
  main
senpai@Aaryans-MacBook-Air CPP-Prog % git status
On branch aaryan
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:   ../.DS_Store
        modified:   BookAllocation.cpp

no changes added to commit (use "git add" and/or "git commit -a")
senpai@Aaryans-MacBook-Air CPP-Prog % git add .
senpai@Aaryans-MacBook-Air CPP-Prog % git commit -m "modified program"
[aaryan 5938869] modified program
 Committer: Singhaaryan <senpai@Aaryans-MacBook-Air.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
```

Step 10: Press **"I"** button to enter in the insert mode.

Step 11: Remove all the unwanted data by clicking on backspace (or delete) and leave the data which you want there.

```
senpai@Aaryans-MacBook-Air CPP-Prog % git merge tool
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.
senpai@Aaryans-MacBook-Air CPP-Prog % git mergetool

This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
opendiff tortoisemerge emerge vimdiff nvimdiff
Merging:
CPP-Prog/BookAllocation.cpp

Normal merge conflict for 'CPP-Prog/BookAllocation.cpp':
  {local}: modified file
  {remote}: modified file
Hit return to start merge resolution tool (opendiff):
xcode-select: error: tool 'opendiff' requires Xcode, but active developer directory '/Library/Developer/CommandLineTools' is a command line tools instance
CPP-Prog/BookAllocation.cpp seems unchanged.
Was the merge successful [y/n]? y
senpai@Aaryans-MacBook-Air CPP-Prog % git commit -m
error: switch `m' requires a value
senpai@Aaryans-MacBook-Air CPP-Prog % git commit -m "conflict resolved"
[main 6eb69ff] conflict resolved
 Committer: Singhaaryan <senpai@Aaryans-MacBook-Air.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author
```

Step 12: Press Esc button and type :wq to save and exit. Repeat this process till you come out of Mergetool interface.

Step 13: Now commit (that now conflict is resolved) and check the status it will show an extra (unwanted) unstaged file ()remove it by command **rm -rf <file name>.**

```
senpai@Aaryans-MacBook-Air CPP-Prog % git log --oneline
6eb69ff (HEAD -> main) conflict resolved
917ac5c added comments to program
5938869 (aaryan) modified program
e7ad028 (origin/main, origin/HEAD) Merge pull request #1 from abhi-srk/abhishek
71f766d Added some new cpp files in the Cpp folder
907f34a Adding files to repo
df6c87f added cpp files to the repository
ae0ae28 Add files via upload
senpai@Aaryans-MacBook-Air CPP-Prog %
```

Now the merge conflict has been resolved.

Experiment No. 09

Aim: Reset and Revert

THEORY:

Git Reset Command :

git reset is used when we want to unstage a file and bring our changes back to the working directory. git reset can also be used to remove commits from the local repository.

git reset HEAD <filename>

Let's check how it works:

Make a new folder and name it demo and then create a file and then commit it, repeat this process three times , so there will be three commits . Run the command git log. It will show three commits. Now run the command git reset <1st commit's checksum>. It will reset the branch to its first commit. Run the command git log. It will only show first commit now.

Now run the command git status. The branch is now reset to the first commit.

There are three flags in reset (i.e. --hard, --soft, --mixed).

1. **git reset --hard.**

   This is the most direct, dangerous, and frequently used option. When passed –hard the Head ref pointer is pointed to the specified commit. Then, the Staging Area and Working Directory are reset to match that of the specified commit. This means that any pending work that was hanging out in the Staging Area and Working Directory

will be lost.

2. **git reset --mixed.**
   This is the default option of reset. When we use this option, the files in committed state
   are reset and they go under untracked files (red colored). The ref pointers are updated. The Staging Area is reset to the state of the specified commit. Any changes that have been undone from the Staging Area are moved to the Working Directory.

3. **Git reset - - soft.**
   In this option of reset, the committed files are reset to Staging Area (green colored). The Staging Area and the Working Directory are left untouched.

Git Revert: It reverts the existing commits.

Git revert simply creates a new commit that is the opposite of an existing account.
It leaves the files in the same state as if the commit that has been reverted never existed.

1. Run command git log to see all the commits.

2. Type command **git revert <checksum of the commit>**. It reverts the changes that done before commit. Use **:wq** to exit the interface.

git revert HEAD~1
Reverts the changes specified by the second last commit in HEAD and create a new commit with the reverted changes.