Concepts of GitHub
Create GitHub
Create, Add and Commit repository
File states: Committed, Modified, Staged
Add and Commit files in Git
Pushing and Pulling repository to GitHub
Using branches in Git

2.1 Concepts of GitHub

- GitHub is a code hosting platform for collaboration and version control.
- GitHub allows working together on projects.
- Git is not the same as GitHub.
- GitHub makes tools that use Git.
- GitHub is the largest host of source code in the world, owned by Microsoft since 2018.

2.1.1 Create GitHub

GitHub essentials are:

1. Repositories
2. Branches
3. Commits
4. Pull Requests
5. Git (the version control software GitHub is built on)

1. **Repository**
   - A GitHub **repository** can be used to store a development **project**.
   - It can contain **folders** and any type of **files** (HTML, CSS, JavaScript, Documents, Data, Images).
   - A GitHub repository should also include a **licence** file and a **README** file about the project.
   - A GitHub repository can also be used to store ideas, or any resources that you want to share.

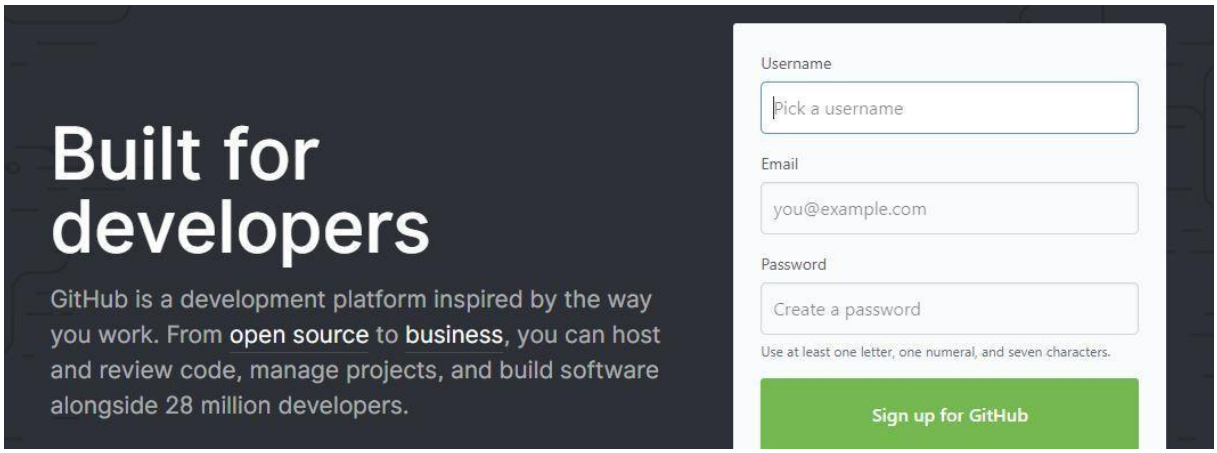
2. **Branch**
   - A GitHub branch is used to work with different **versions** of a repository at the same time.
   - By default, a repository has a **master** branch (a production branch).
   - Any other branch is a **copy** of the master branch (as it was at a point in time).
   - New Branches are for bug fixes and feature work separate from the master branch. When changes are ready, they can be merged into the master branch. If you make changes to the master branch while working on a new branch, these updates can be pulled in.

3. **Commits**
   - At GitHub, changes are called commits.
   - Each commit (change) has a description explaining why a change was made.

4. **Pull Requests**
   - Pull Requests are the heart of GitHub **collaboration**.
   - With a pull request you are **proposing** that your changes should be **merged** (pulled in) with the master.
   - Pull requests show content **differences**, changes, additions, and subtractions in **colors** (green and red).
   - As soon as you have a commit, you can open a pull request and start a discussion, even before the code is finished.

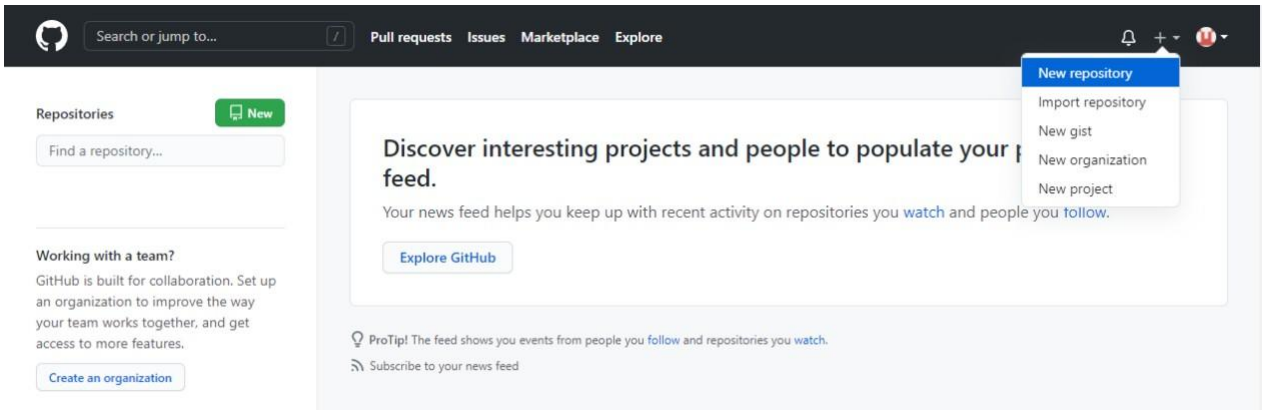
2.1.2 Create, Add and Commit repository

Step 1: Sign up for GitHub at https://github.com/:

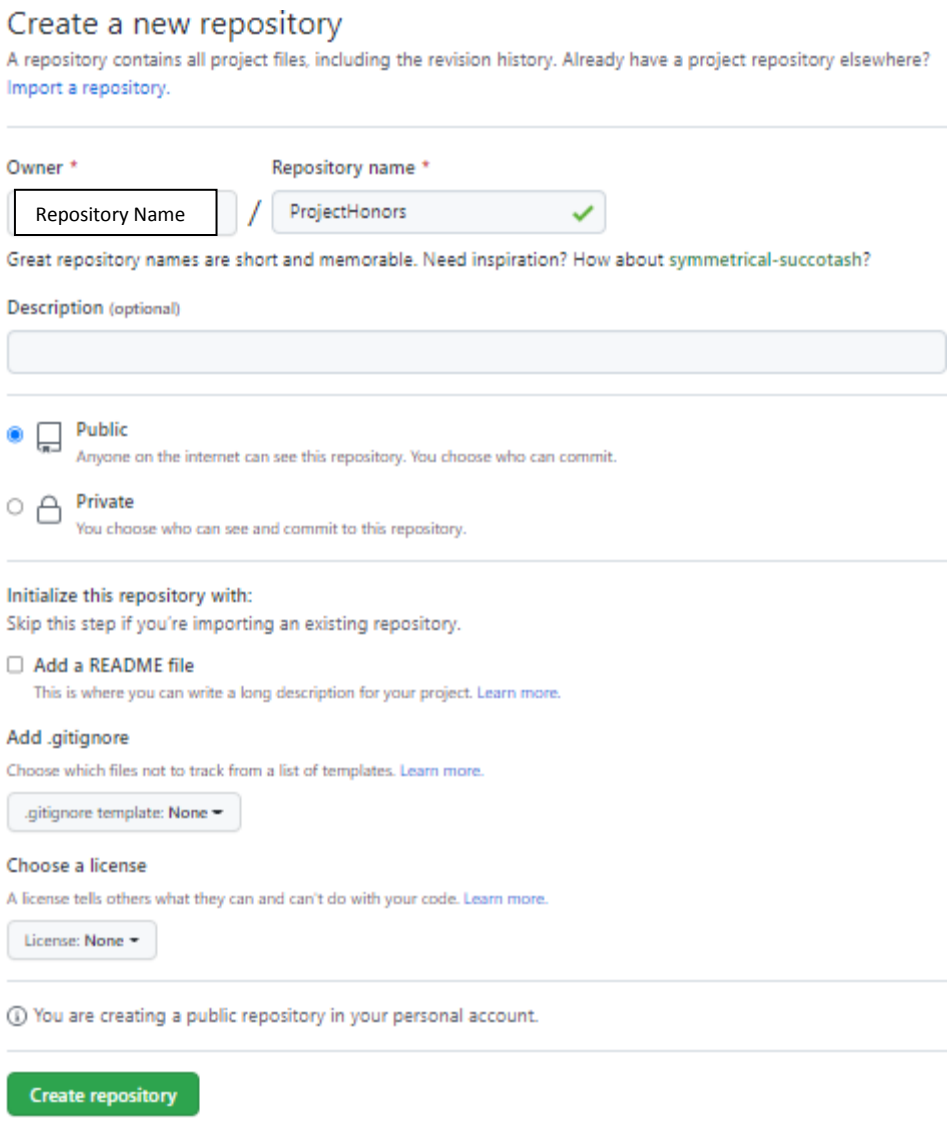


Step 2: Create a Repository on GitHub

Now that you have made a GitHub account, sign in, and create a new repository:



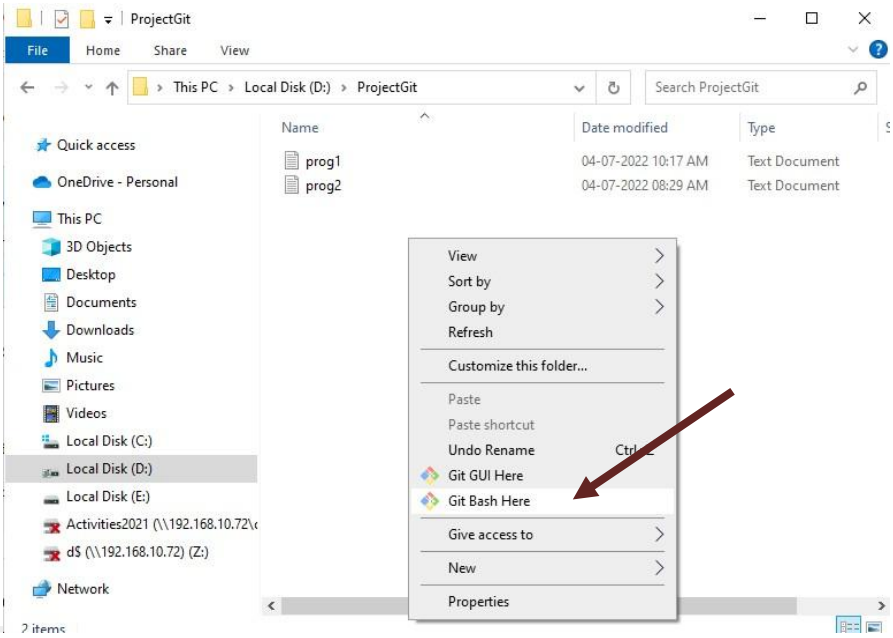
Step 3: And fill in the relevant details:



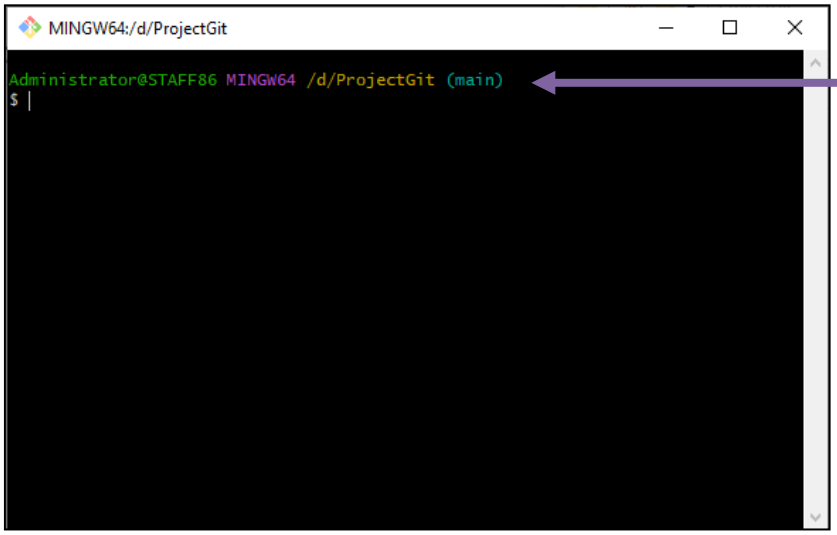
Choose Public (if the repo to be viewable for anyone) or Private (To restrict people who should be able to view the repo). Then click "Create repository".

2.1.4 Add and Commit files in Git

Step 1: let's create a project in local drive of Computer, add some of the project files and now right in project folder and select **Git Bash Here**



Git Bash will be open as follows (already navigated to project folder's drive):



Step 2: To add Project files to the repository of git follow the below mentioned steps:

Step 2.1: check git version with command git -v.

Step 2.2: Initialize Git with command git init.

Step 2.3: Adding files to the repository using git add --a

Step 2.4: Check status about Commits and file using git status.

Step 2.5: Commit Repository with appropriate Message using command git commit -m "First Commit".

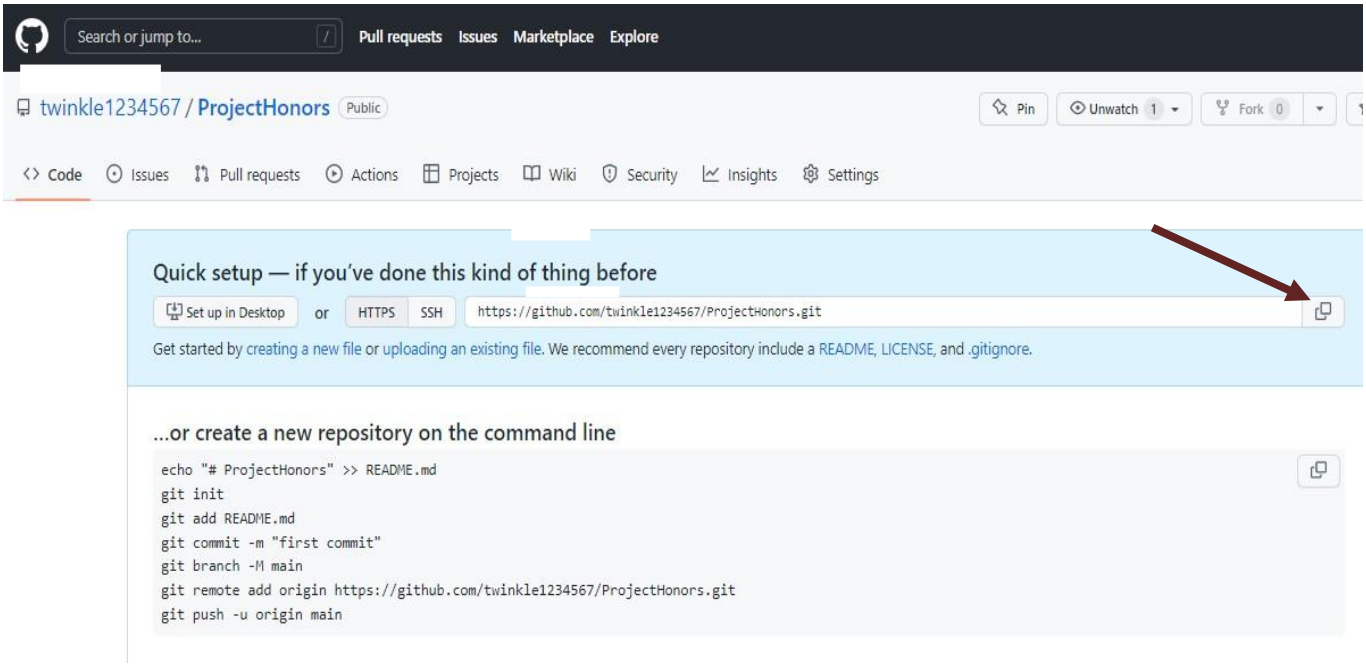
```
MINGW64:/d/GitProject

Administrator@STAFF86 MINGW64 /d/GitProject
$ git -v
git version 2.37.0.windows.1

Administrator@STAFF86 MINGW64 /d/GitProject
$ git init
Initialized empty Git repository in D:/GitProject/.git/

Administrator@STAFF86 MINGW64 /d/GitProject (master)
$ git add --a

Administrator@STAFF86 MINGW64 /d/GitProject (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   prog1.txt
        new file:   prog2.txt


Administrator@STAFF86 MINGW64 /d/GitProject (master)
$ git commit -m "First Commit"
[master (root-commit) 5824e13] First Commit
 2 files changed, 2 insertions(+)
 create mode 100644 prog1.txt
 create mode 100644 prog2.txt

Administrator@STAFF86 MINGW64 /d/GitProject (master)
$ |
```

Pushing and pulling repository to GitHub

Step 1: Push Code (local Repository) to Github from Git

Since the Git repository is already initialized and set, let's push the code to GitHub:

Copy the URL as shown with arrow, or click the clipboard marked in the image above and follow the below given steps to create a new repository.



Step 1.1 : Copy **"**git branch -M main "command and paste in Git Bash to set the main branch for git repository.

Step 1.2: Copy "git remote add origin https://github.com/repository_name /ProjectHonors.git "command and paste in Git Bash to add the remote origin of GitHub.

git remote add origin *URL* specifies that you are adding a remote repository, with the specified URL, as an origin to your local Git repo.

Step 1.3: To push main branch to the origin URL, and set it as the default remote branch:

Copy "git push -u origin main "command and paste in Git Bash.

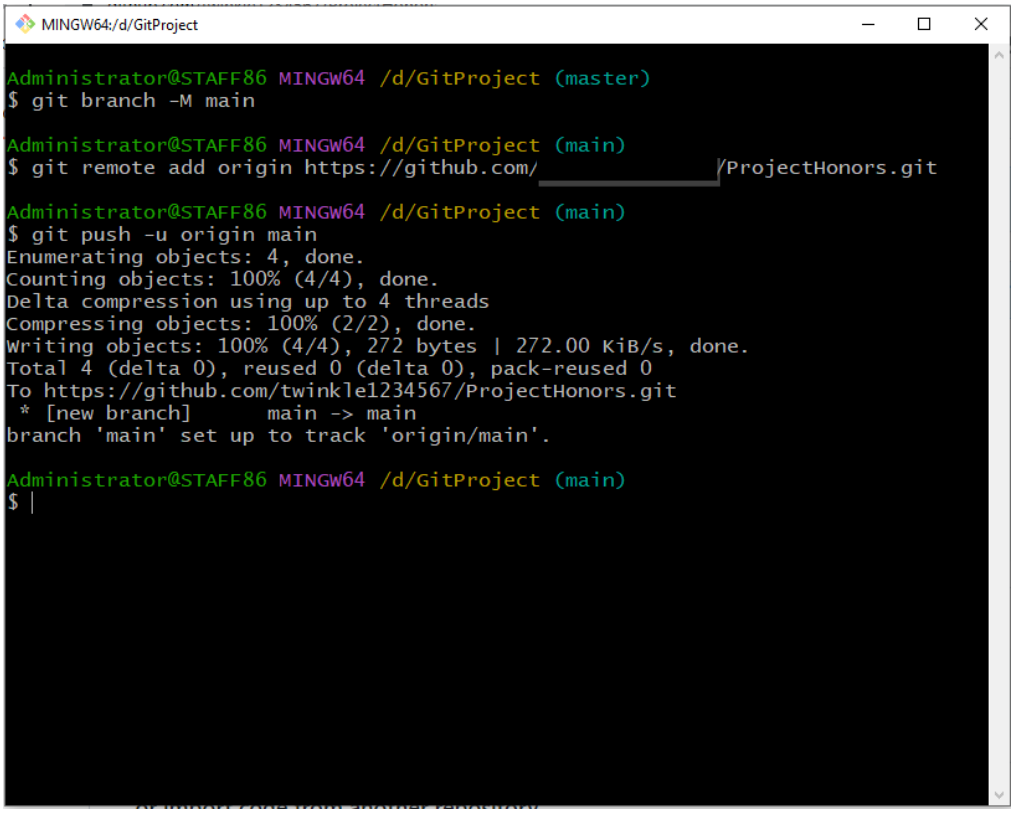
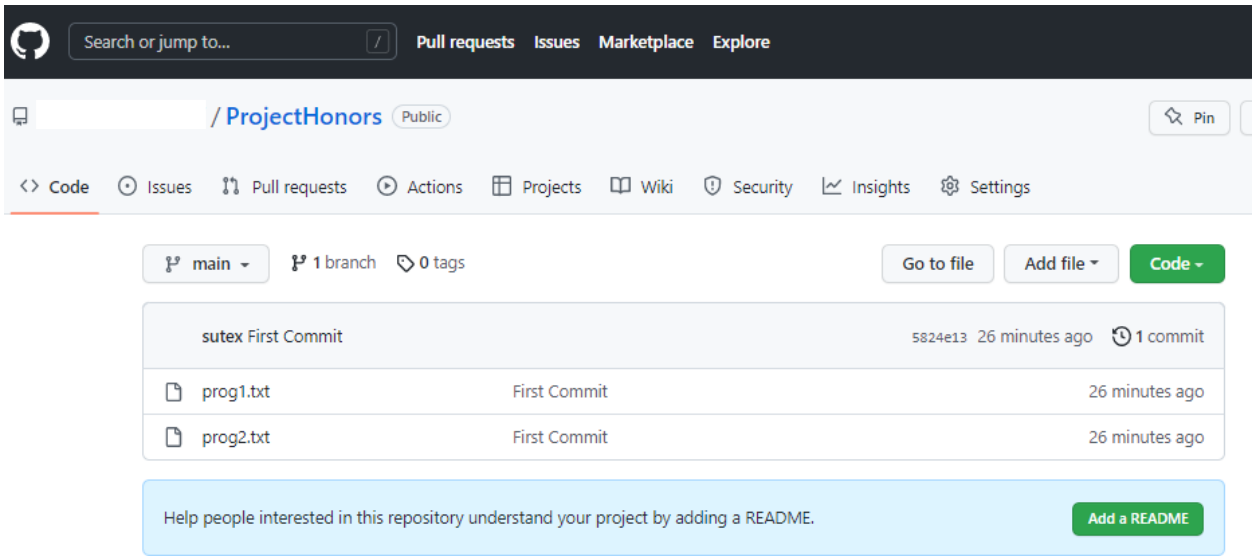


Image: Git Bash Screen to Push Code to Github

Note: Since this is the first time GitHub is connected, some kind of notification will be provided to authenticate this connection.

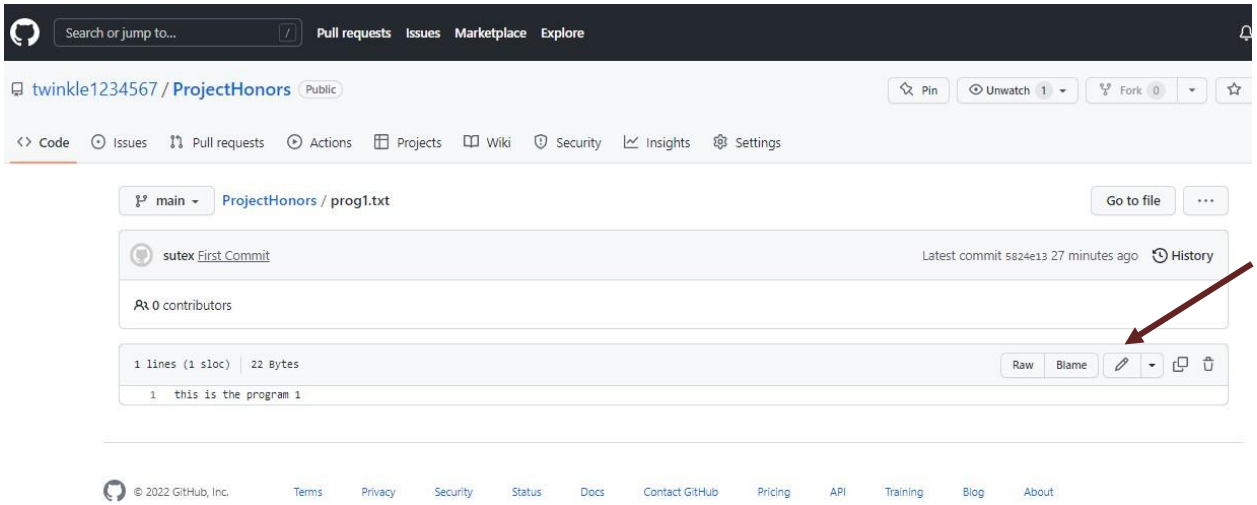
Now, go back into GitHub and check that the repository has been updated.



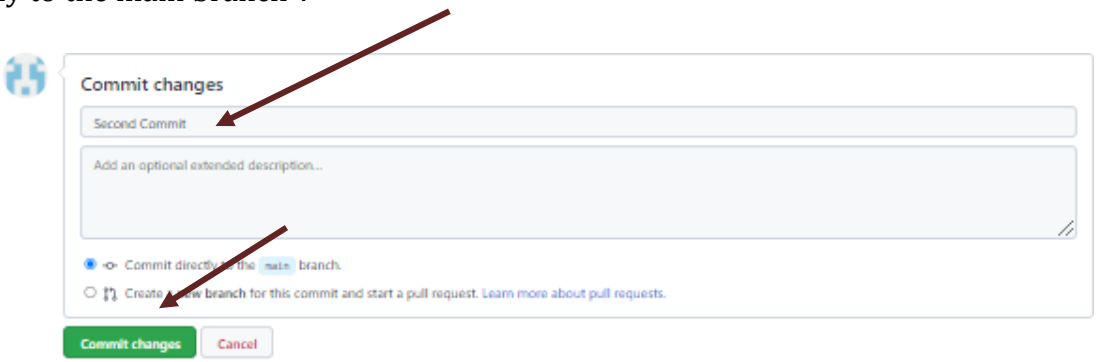
Step 2: Edit Code in GitHub

In addition to being a host for Git content, GitHub has a very good code editor.

Let's try to edit the file in GitHub. Just click the edit button:



Add some changes to the code, and then commit the changes. For now, we will "Commit directly to the main branch".



Note: Remember to add a description for the commit.

Pulling repository from GitHub

GitHub allows easy request to pull code to git, click the code option and copy the https request URL and use with git pull command as follows:

“git pull https://github.com/repository_name /ProjectHonors.git “and paste it in Git Bash

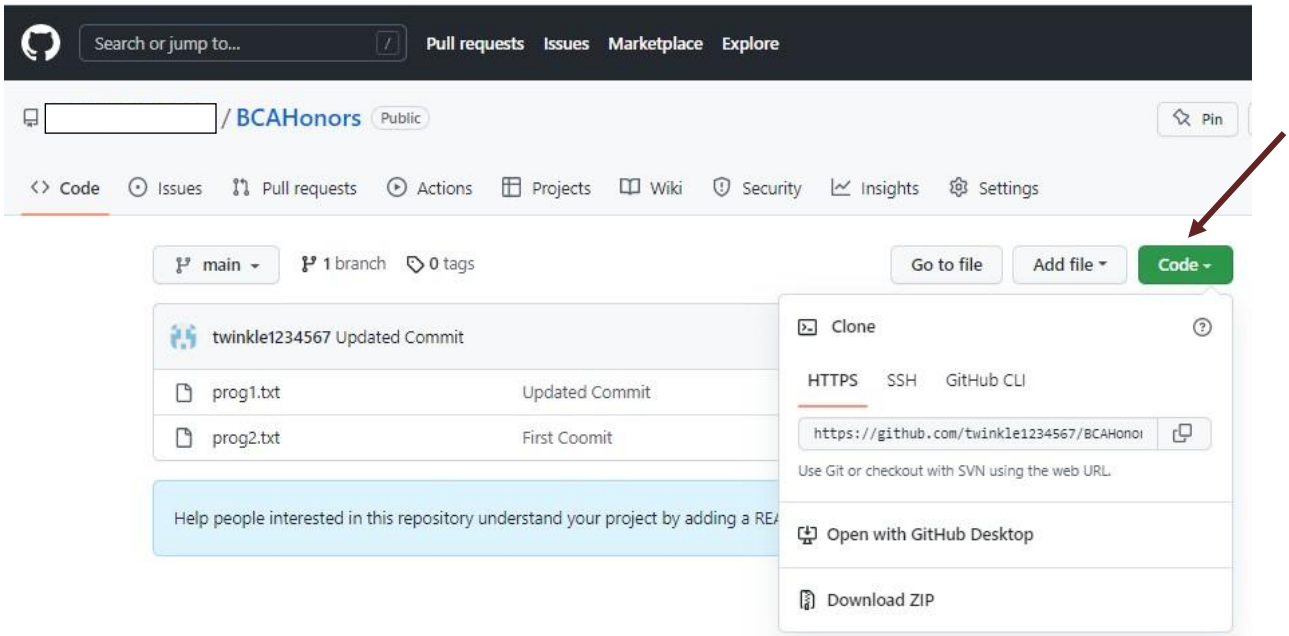


Image: GitHub Screen for Pull request

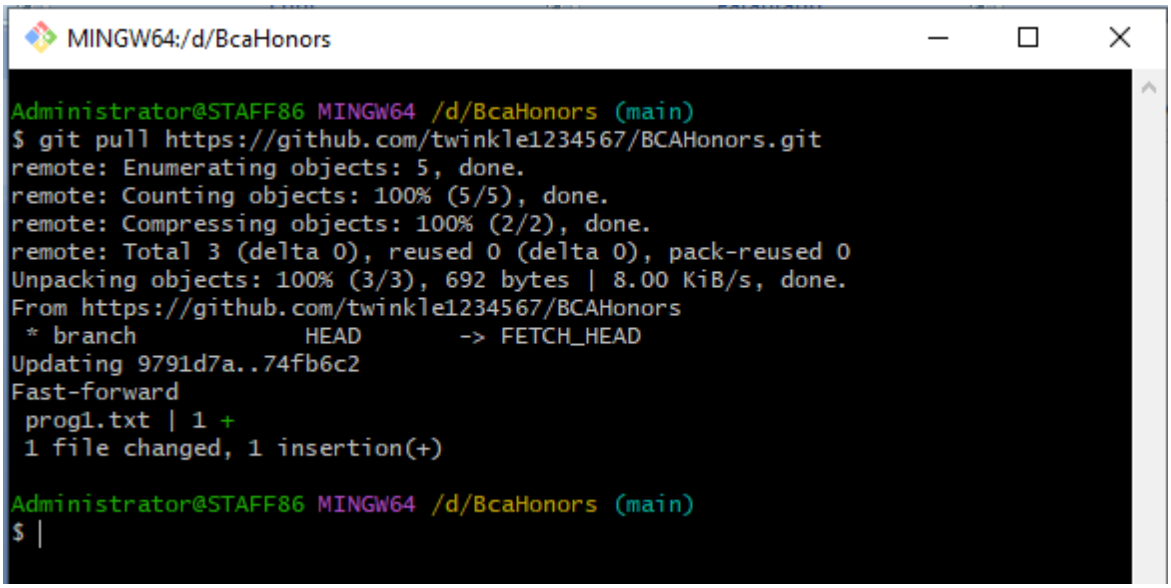


Image: Git Bash Screen to pull the code from GitHub to git

Code is successfully pulled from GitHub repository.

2.1.3 File states: Committed, Modified, Staged

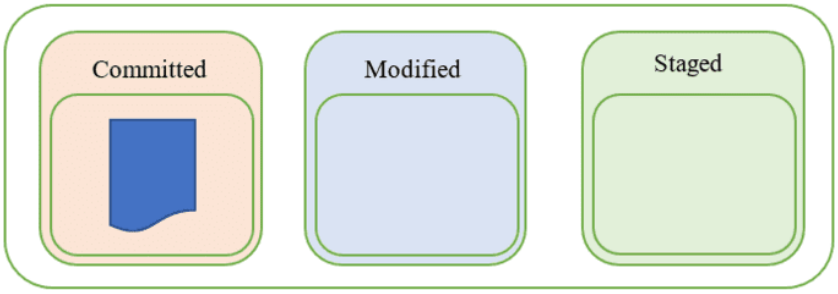
Git has three main states that your files can reside in:

1. ***Committed***
2. ***Modified***
3. ***Staged***

These three states make a system based on promotion. Each file can reside in one of these three states and change states depending on what was done to it.

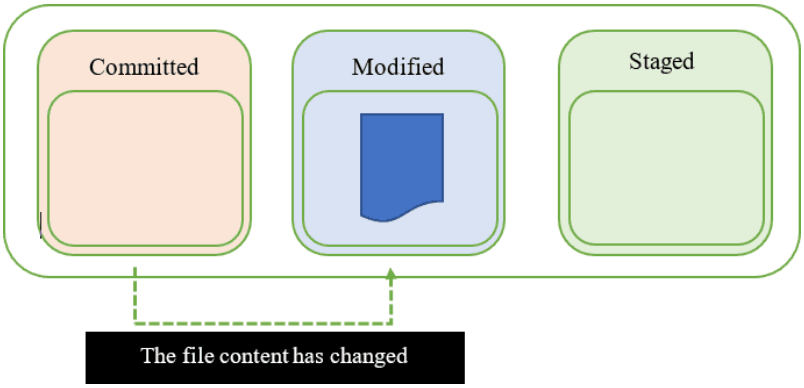
1. Committed

This state indicates that the file is safely stored in the local database.



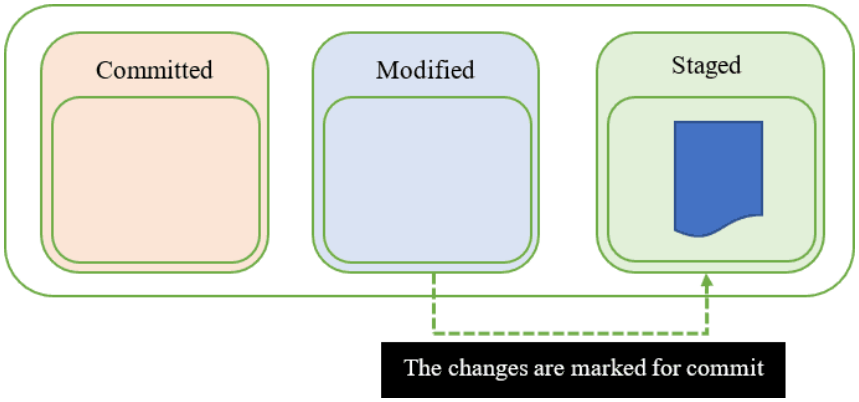
2. Modified

When any change to the file occurs, the state of the file changes from committed to modified. This means that the document has changed since its last committed version which is saved to our local database.

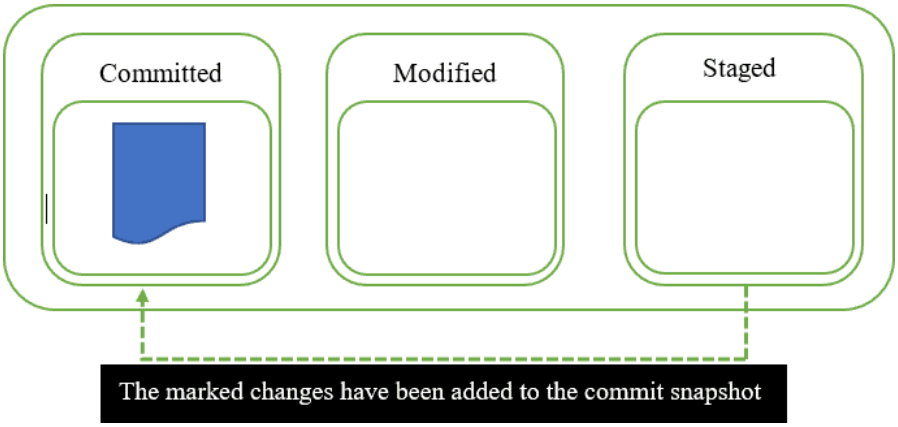


3. Staged

When modifications are completed in the file, it moves to the staged state. The file is now ready to be added to the local git database, file is marked and ready to go into next commit.



It's important to note that these three file states refer only to files tracked in a Git project. A file can be in a project but the changes to it are not tracked by Git. When we start tracking changes in Git for a file we haven't been tracking, it automatically goes into the staged state.



Three Sections of a Git Project

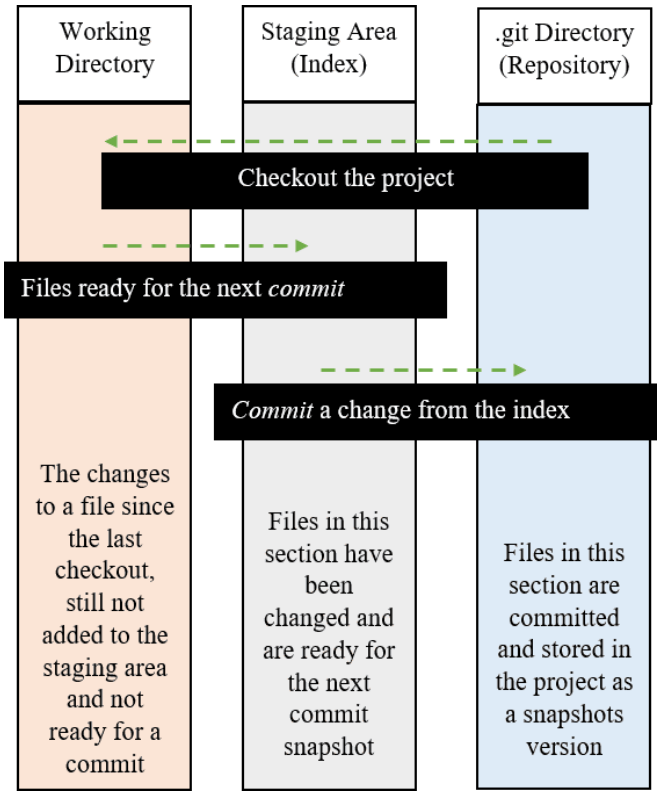
Similar to files, a Git project consists of three different sections.

The first section is the .git directory, also known as the **repository**. This is where Git stores the metadata and object database for your project.

The next section is the working **directory**. This is a single checkout of one version of the project. This is where you can modify files.

The third section is the **staging area**, also known as the index. It's the area between the working directory and the .git directory. All the files which are ready for a commit are stored here.

When commit is made, files in the staging area moves to the new version of the repository. This excludes what is in the working directory. This allows us to change files however we like, but only what we move to the staging area will be committed. Everything we changed but do not want to put in a repository stays in the working directory.



2.1.6 Using branches in Git

In Git, a branch is a new/separate version of the main repository.

Branches allows to work on different parts of a project without impacting the main branch.

When the work is complete, a branch can be merged with the main project.

switch between branches and work on different projects without them interfering with each other is one of the advantages of branching in git. Branching in Git is very lightweight and fast!

Create New Git Branch

To check information regarding branch following command is used:

git branch

To create new branch and make it current branch following command is used:

git checkout -b branch

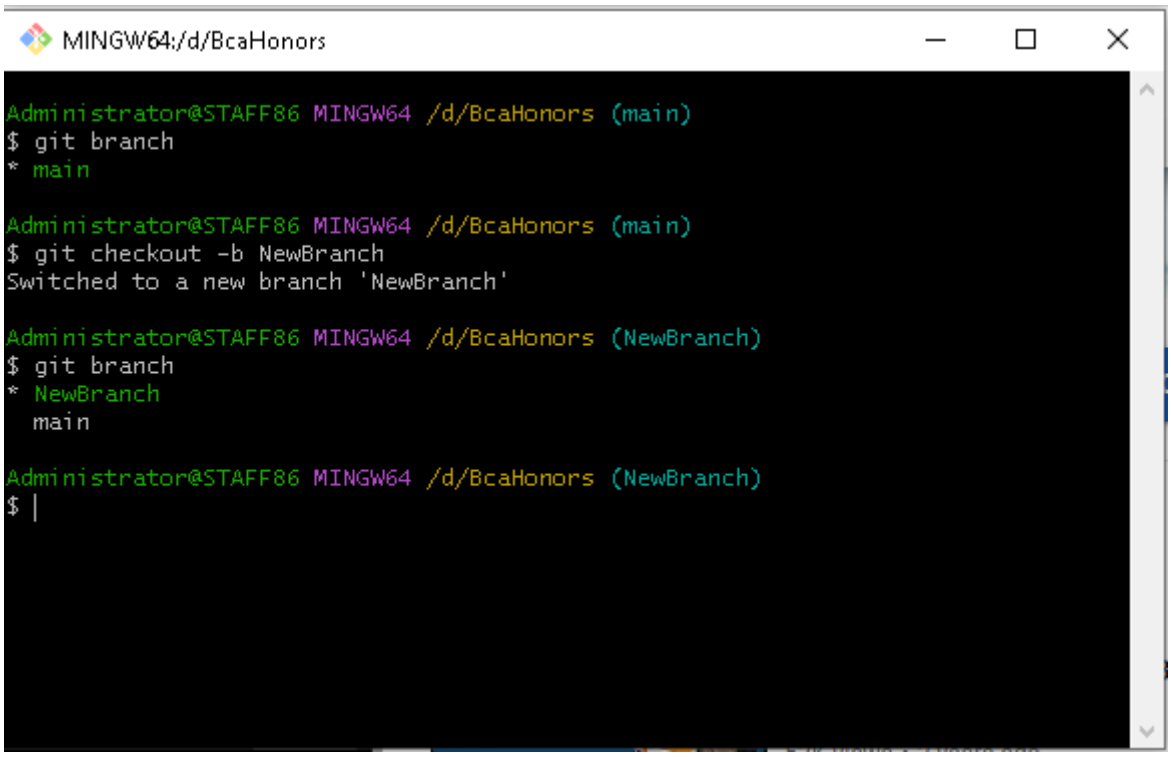


Image: Git Bash Screen for Branch Creation

Note: git branch command provides the information regarding branches and indicate current branch with asterisk mark (*). So, here *NewBranch is current working branch

Reflecting Changes into New Branch

Now u can make some changes in the local folder and check for status as depicted in git bash prompt.

```
MINGW64:/d/BcaHonors

Administrator@STAFF86 MINGW64 /d/BcaHonors (NewBranch)
$ git status
On branch NewBranch
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   prog1.txt
        deleted:    prog2.txt

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

Administrator@STAFF86 MINGW64 /d/BcaHonors (NewBranch)
$ git add --a

Administrator@STAFF86 MINGW64 /d/BcaHonors (NewBranch)
$ git commit -m "commit made for new branch"
[NewBranch 8ad3f04] commit made for new branch
 2 files changed, 2 insertions(+), 1 deletion(-)
 delete mode 100644 prog2.txt

Administrator@STAFF86 MINGW64 /d/BcaHonors (NewBranch)
$ |
```

Image: Git Bash Screen to commit the changes into new branch

Pushing changes to github

To push the changes of NewBranch move to newly created branch using following command and execute both the commands:

git checkout NewBranch

git push origin NewBranch

```
MINGW64:/d/BcaHonors

Administrator@STAFF86 MINGW64 /d/BcaHonors (main)
$ git checkout NewBranch
Switched to branch 'NewBranch'

Administrator@STAFF86 MINGW64 /d/BcaHonors (NewBranch)
$ git push origin NewBranch
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 307 bytes | 6.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'NewBranch' on GitHub by visiting:
remote:      https://github.com/twinkle1234567/BCAHonors/pull/new/NewBranch
remote:
To https://github.com/twinkle1234567/BCAHonors.git
 * [new branch]      NewBranch -> NewBranch

Administrator@STAFF86 MINGW64 /d/BcaHonors (NewBranch)
$ |
```

Merge and Delete Operations for NewBranch:

Before deleting any branch, Merging the branch is required so that the final changes will be reflected to the repository.

Follow the below steps to merge and delete branch.

Step 1: move to main branch using "git checkout main"

Step 2: To merge the branch use following command: "git merge NewBranch" which will merge the changes of NewBranch to Main Branch.

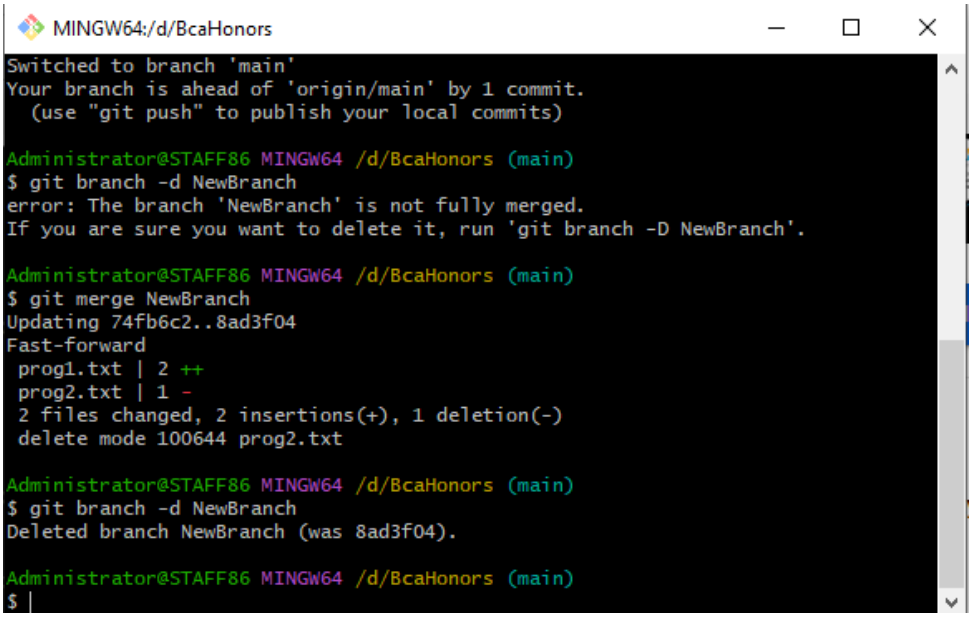


Image: Git Screen to delete and merge the branch

Step 3: As the merge is successfully done, deletion operation can be done by using following command: "git branch -d NewBranch"

```
MINGW64:/d/BcaHonors
Administrator@STAFF86 MINGW64 /d/BcaHonors (main)
$ git branch -d NewBranch
error: The branch 'NewBranch' is not fully merged.
If you are sure you want to delete it, run 'git branch -D NewBranch'.

Administrator@STAFF86 MINGW64 /d/BcaHonors (main)
$ git merge NewBranch
Updating 74fb6c2..8ad3f04
Fast-forward
 prog1.txt | 2 ++
 prog2.txt | 1 -
 2 files changed, 2 insertions(+), 1 deletion(-)
 delete mode 100644 prog2.txt

Administrator@STAFF86 MINGW64 /d/BcaHonors (main)
$ git branch -d NewBranch
Deleted branch NewBranch (was 8ad3f04).

Administrator@STAFF86 MINGW64 /d/BcaHonors (main)
$ git branch
* main

Administrator@STAFF86 MINGW64 /d/BcaHonors (main)
$ |
```

Image: Git Bash Screen for Delete and Merge Operation.

Reflecting Merge and Delete Branch at GitHub

To reflect the merge and delete operation on branch in GitHub follow the below steps:

Step 1: Move to main branch using command “git checkout main”
Step 2: Push the changes to GitHub using command “git push origin main”
Step 3: Delete the branch from github using command: “git push -d origin NewBranch”

```
MINGW64:/d/BcaHonors
 2 files changed, 2 insertions(+), 1 deletion(-)
 delete mode 100644 prog2.txt

Administrator@STAFF86 MINGW64 /d/BcaHonors (main)
$ git branch -d NewBranch
Deleted branch NewBranch (was 8ad3f04).

Administrator@STAFF86 MINGW64 /d/BcaHonors (main)
$ git branch
* main

Administrator@STAFF86 MINGW64 /d/BcaHonors (main)
$ git push origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/twinkle1234567/BCAHonors.git
   74fb6c2..8ad3f04  main -> main

Administrator@STAFF86 MINGW64 /d/BcaHonors (main)
$ git push -d origin NewBranch
To https://github.com/twinkle1234567/BCAHonors.git
 - [deleted]         NewBranch
```

Image: Git Bash Screen to Reflecting Merge and Delete Branch at GitHub