

# Work Integrated Learning Programmes

## M.Tech Software Engineering



### Introduction to Devops

### Assignment 1 – Git Workflow

Submitted By

Pavithra.S

2022MT93172

# INDEX

ASSIGNMENT 1 - GIT WORKFLOW .....	1
INTRODUCTION.....	1
Features of Git .....	1
GIT WORKFLOW.....	2
GITHUB .....	3
GitHub Features.....	3
Account Creation .....	4
GIT OPERATIONS.....	7
GitBash .....	7
Creating a Repository.....	8
Adding Collaborators .....	10
Notification Setup.....	15
Cloning a Repository .....	16
Creating a Branch.....	19
Code Commit .....	21
Push to Origin .....	23
Pull Requests .....	24
Deleting a Branch.....	27
Merge Conflict .....	28
ReadMe and Gitignore Files.....	32
Force Push and Reset Commit .....	37
Creating Tags .....	40
Releasing Features.....	41
CONCLUSION .....	50

# GIT WORKFLOW

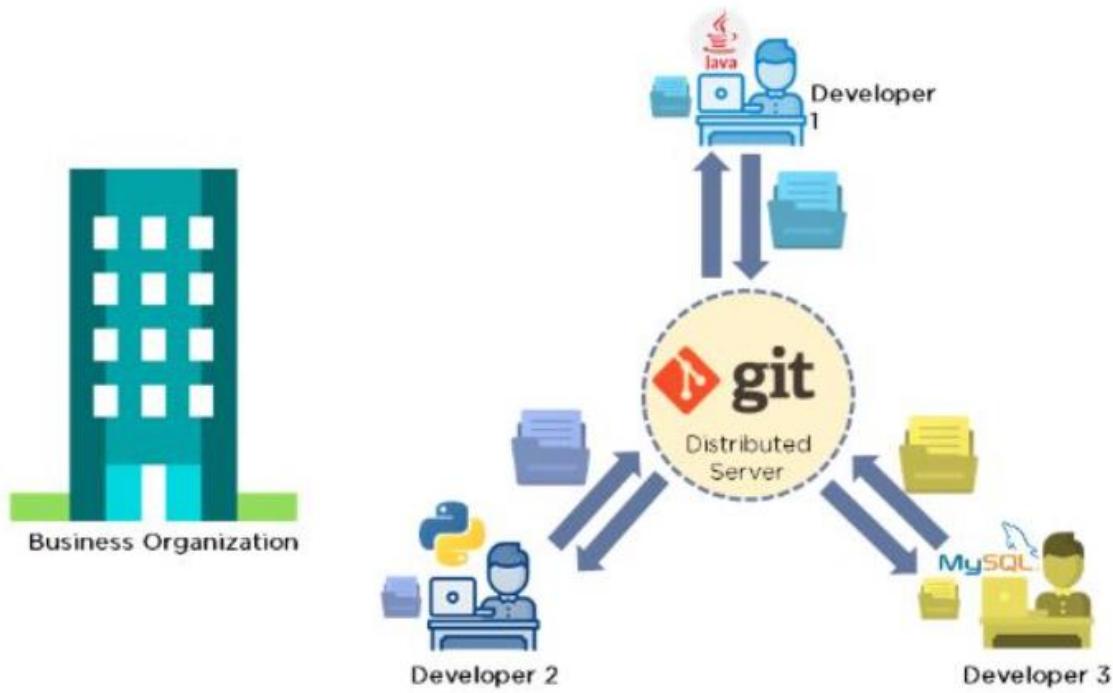
## INTRODUCTION

Git is a DevOps tool that is used for source code management. It is a free open-source version control system used to handle projects of all sizes efficiently.

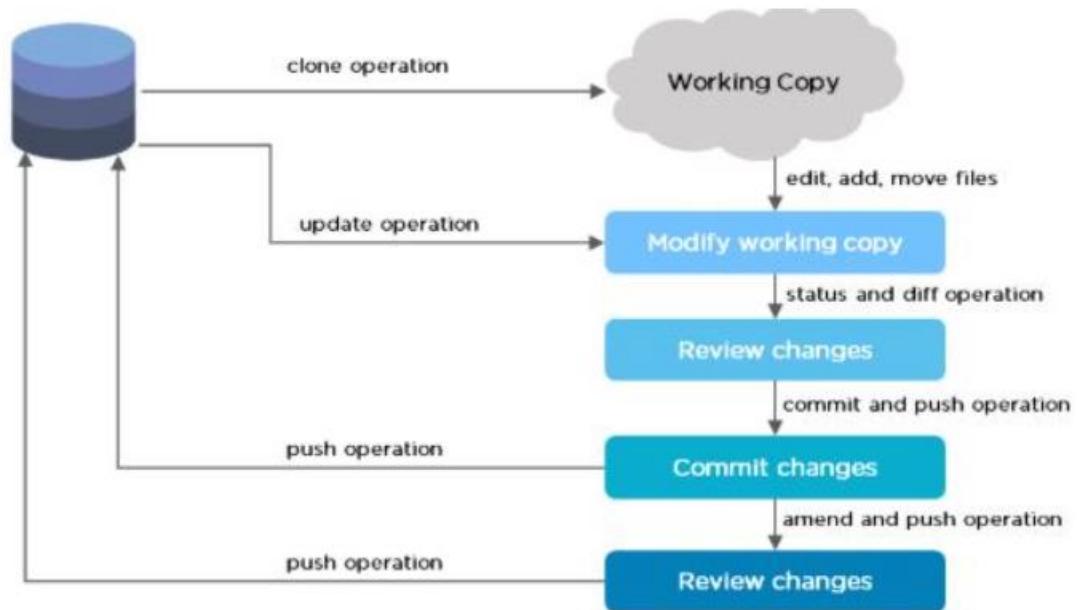
Git is used to tracking changes in the source code, enabling multiple developers to work together on non-linear development. Git makes collaborating on code easier.

## Features of Git

- Tracks history
- Free and open source
- Supports non-linear development
- Creates backups
- Scalable
- Supports collaboration
- Branching is easier
- Distributed development

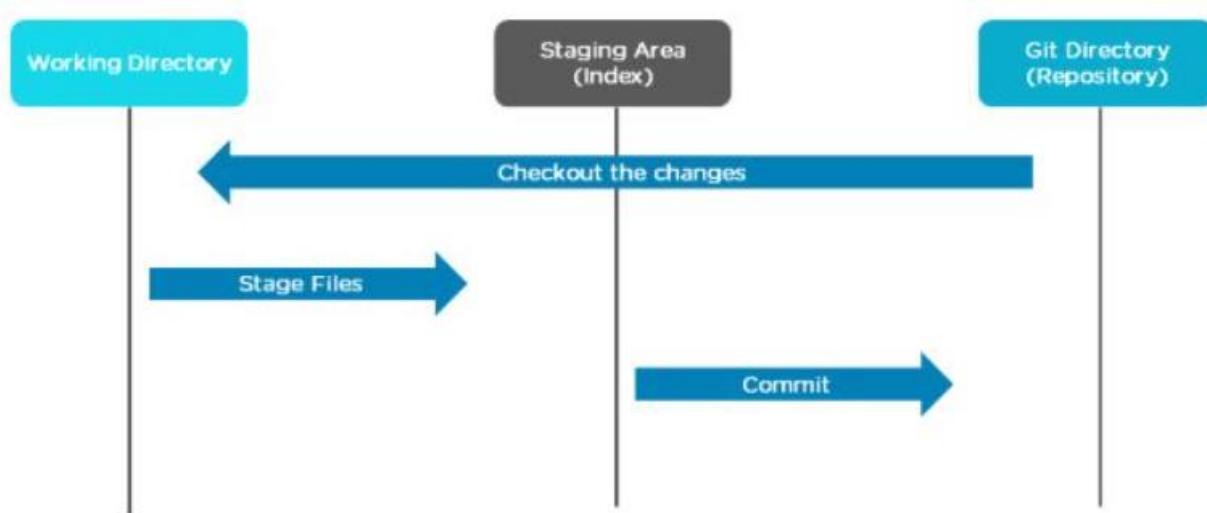


# GIT WORKFLOW



The Git workflow is divided into three states:

- **Working directory** - Modify files in your working directory
- **Staging area (Index)** - Stage the files and add snapshots of them to your staging area
- **Git directory (Repository)** - Perform a commit that stores the snapshots permanently to your Git directory. Checkout any existing version, make changes, stage them and commit.



# **GITHUB**

GitHub is a cloud-based Git repository hosting service that provides a web-based graphical interface. It is the world's largest coding community. Putting a code or a project into GitHub brings it increased, widespread exposure. Programmers can find source codes in many different languages and use the command-line interface, Git, to make and keep track of any changes.

GitHub helps every team member work together on a project from any location while facilitating collaboration. You can also review previous versions created at an earlier point in time.

## **GitHub Features**

### **1. Easy Project Management**

GitHub is a place where project managers and developers come together to coordinate, track, and update their work so that projects are transparent and stay on schedule.

### **2. Increased Safety with Packages**

Packages can be published privately, within the team, or publicly to the open-source community. The packages can be used or reused by downloading them from GitHub.

### **3. Effective Team Management**

GitHub helps all the team members stay on the same page and organized. Moderation tools like Issue and Pull Request Locking help the team to focus on the code.

### **4. Improved Code Writing**

Pull requests help the organizations to review, develop, and propose new code. Team members can discuss any implementations and proposals through these before changing the source code.

### **5. Increased Code Safety**

GitHub uses dedicated tools to identify and analyze vulnerabilities to the code that other tools tend to miss. Development teams everywhere work together to secure the software supply chain, from start to finish.

### **6. Easy Code Hosting**

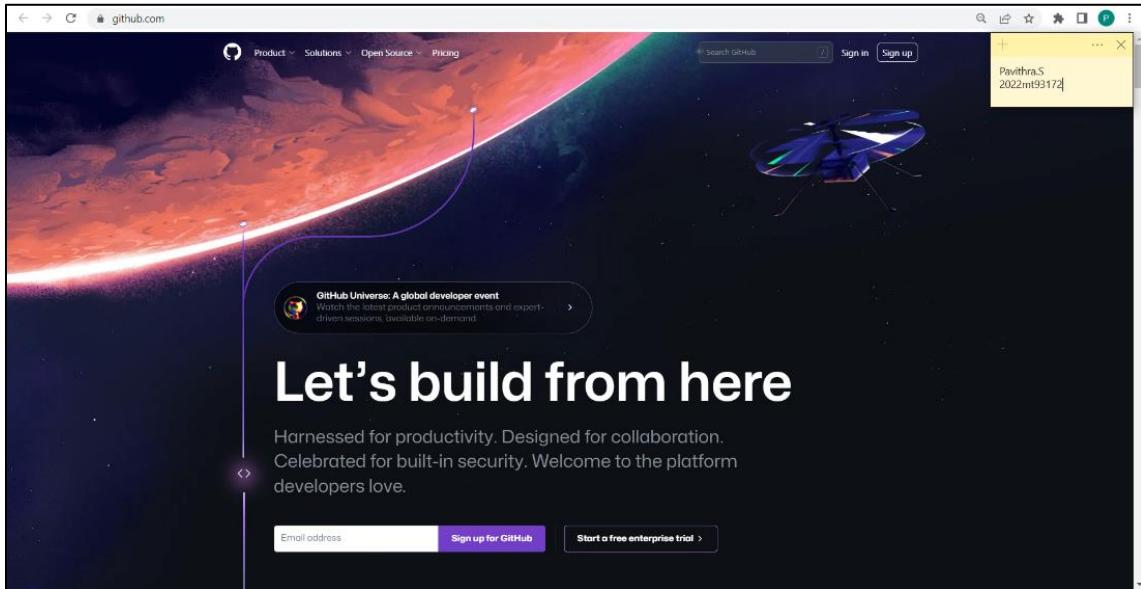
All the code and documentation are in one place. There are millions of repositories on GitHub, and each repository has its own tools to help you host and release code.

## Account Creation

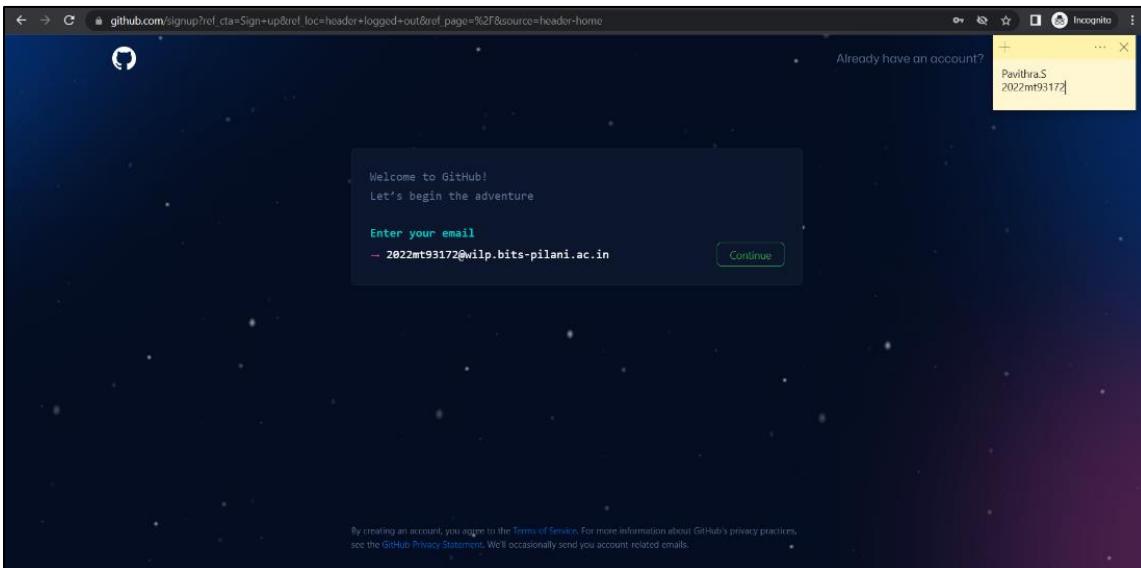
For our git demonstration we are going to need 2 GitHub accounts. One account will be the owner who will act as the manager and the other will be a collaborator or a team member.

Steps to create an account in GitHub are,

1. Visit [github.com](https://github.com) from chrome or any browser of your choice.



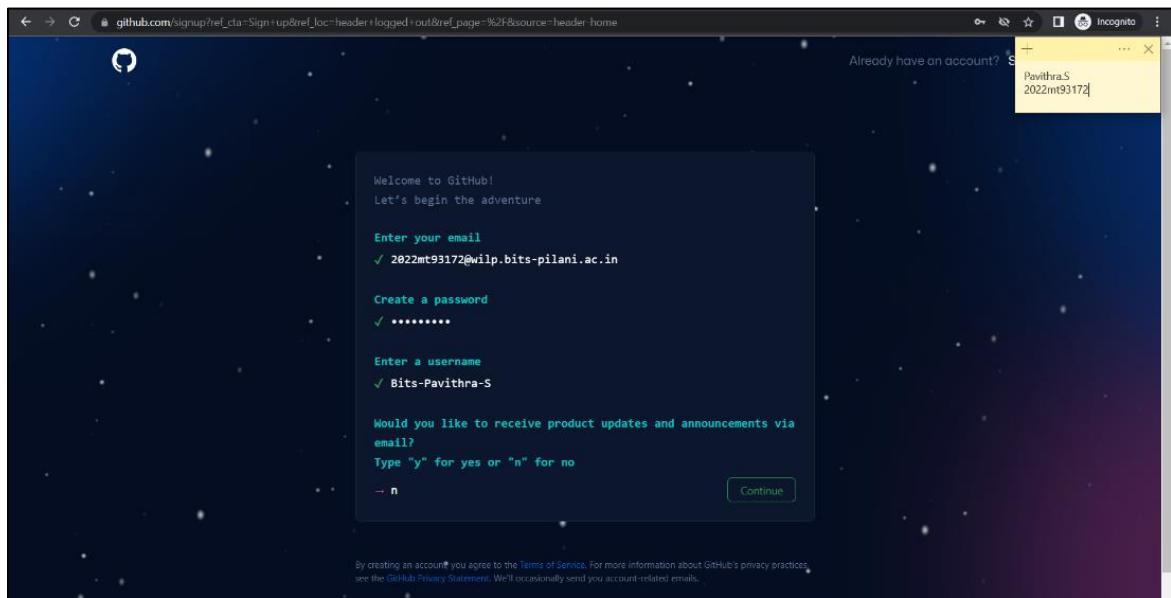
2. Click on the “Sign Up” button.



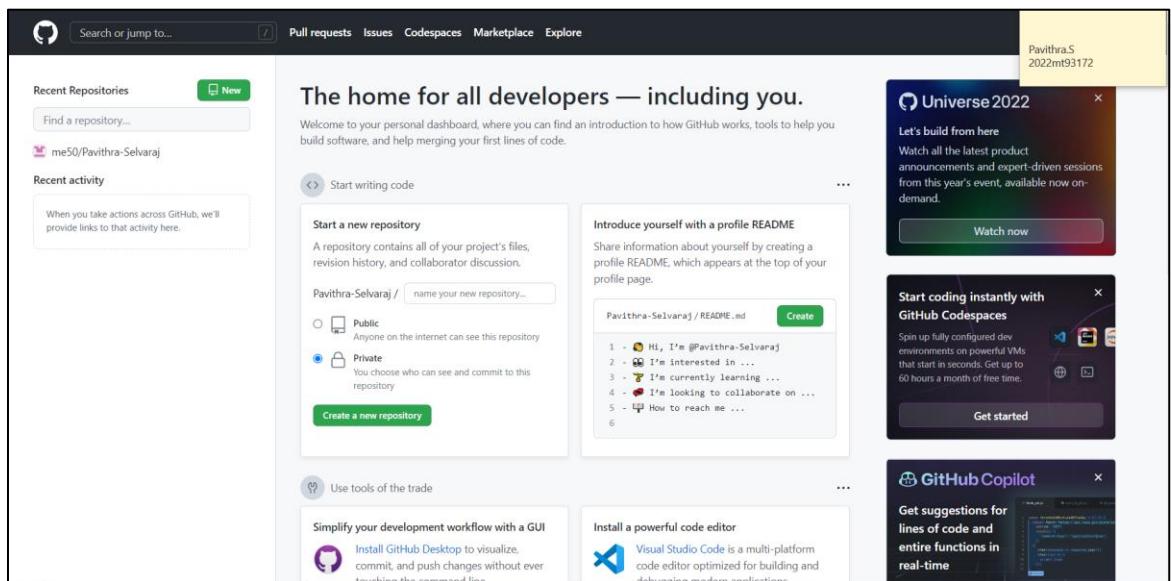
3. Fill the required details such as,

- Email ID,
- Username,
- Password etc.

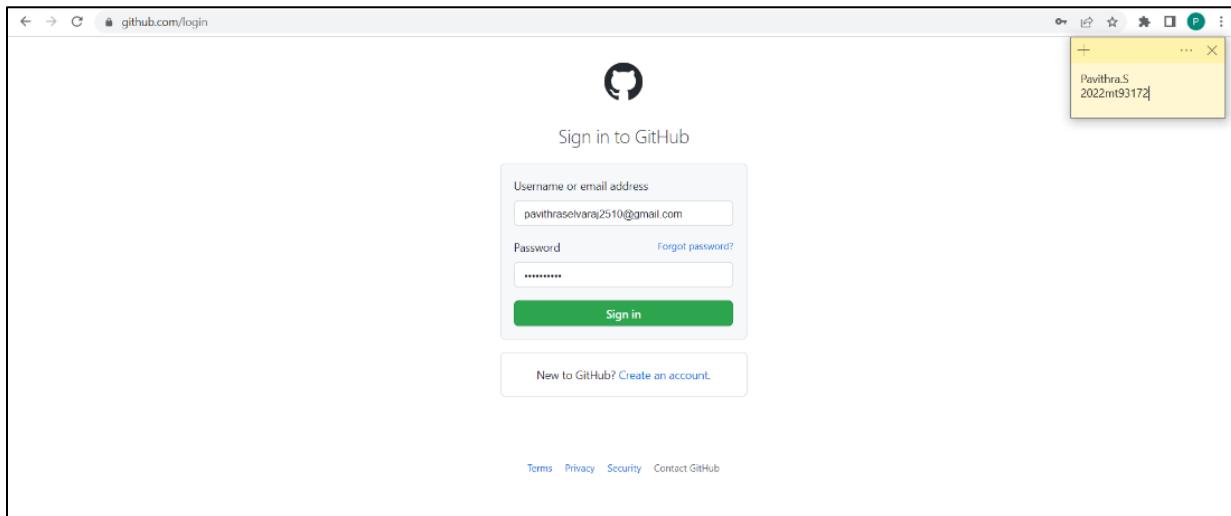
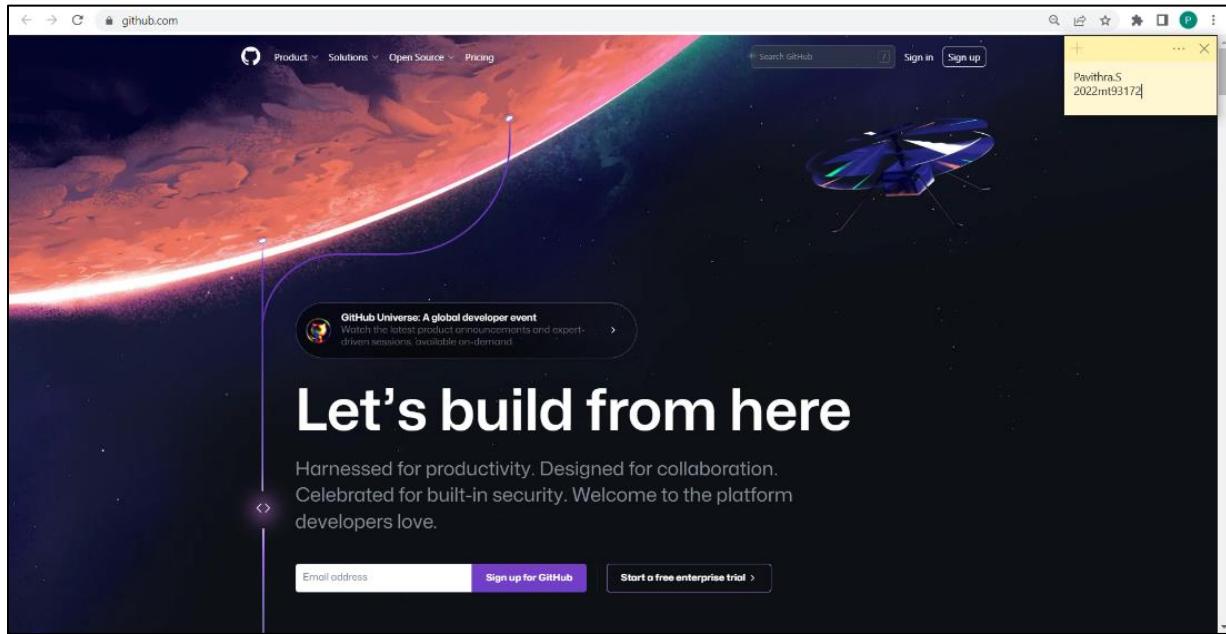
and click continue.



4. The account will then be created and we will be navigated to the landing page.



If already an account is created, we can directly click on the “Sign In” button and enter the credentials to login into our account.



These accounts can now be connected to our local git to share code.

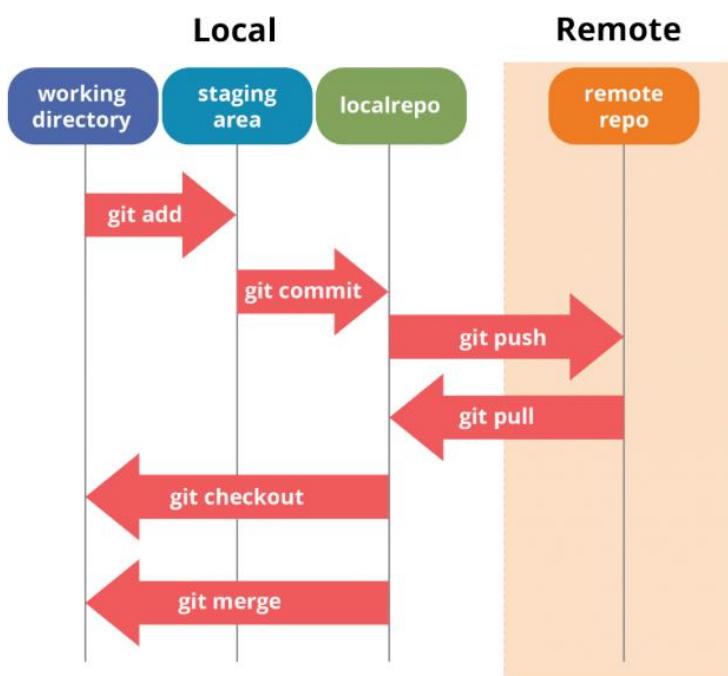
## GIT OPERATIONS

Some of the basic operations in Git are:

1. Initialize
2. Add
3. Commit
4. Pull
5. Push

Some advanced Git operations are:

1. Branching
2. Merging
3. Rebasing



These operations can be done by running commands on GitBash.

### GitBash

Git Bash is a Microsoft Windows application with a Git command-line shell experience and utilities, such as Secure Shell Protocol (SSH), Secure Copy Protocol (SCP), CAT (concatenate command), etc.

Git Bash emulates a bash environment on Windows, allowing users to use the Bash shell and most of the standard Unix commands on a Windows OS. Users can interact with a repository and Git elements by running commands in Git Bash.

## Creating a Repository

Repositories in GIT contain a collection of files of various different versions of a Project.

A repo, short for repository can be created in the following ways,

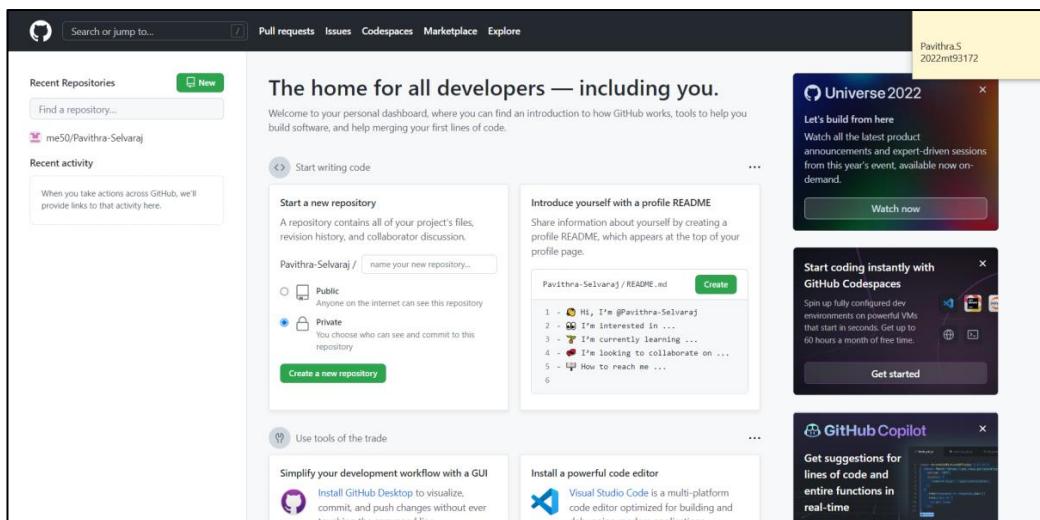
1. Using the command, “git init”

This command creates an empty Git repository. Basically a .git directory with subdirectories for objects, refs/heads, refs/tags, and template files. An initial branch without any commits will be created.

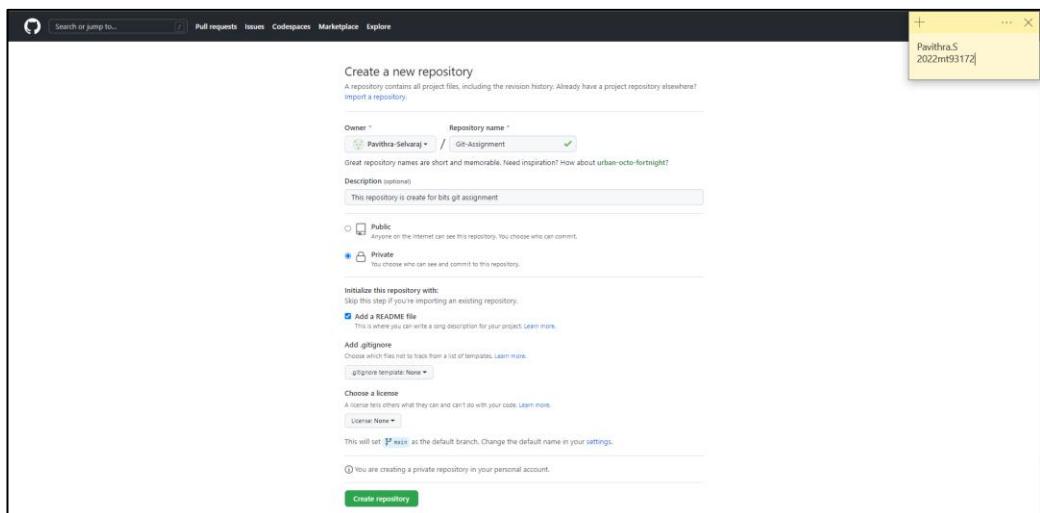
2. From the GitHub portal.

The steps are as follows,

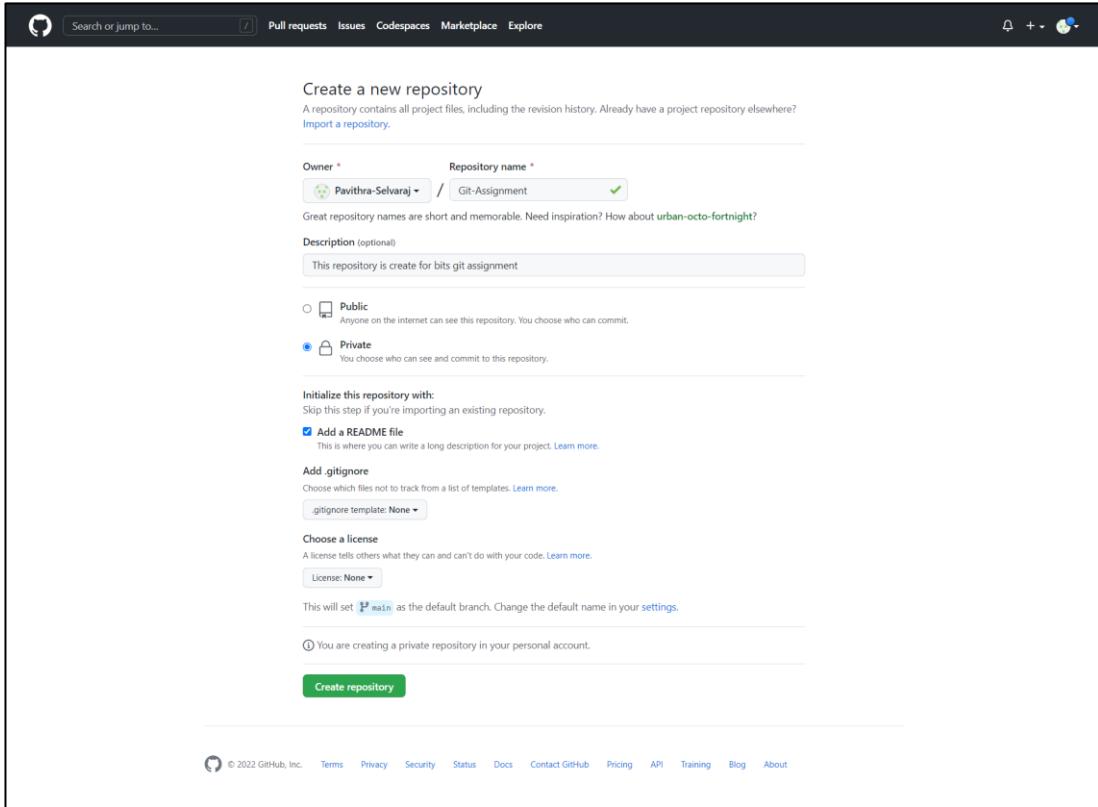
- Click on “Create a new repository” button.



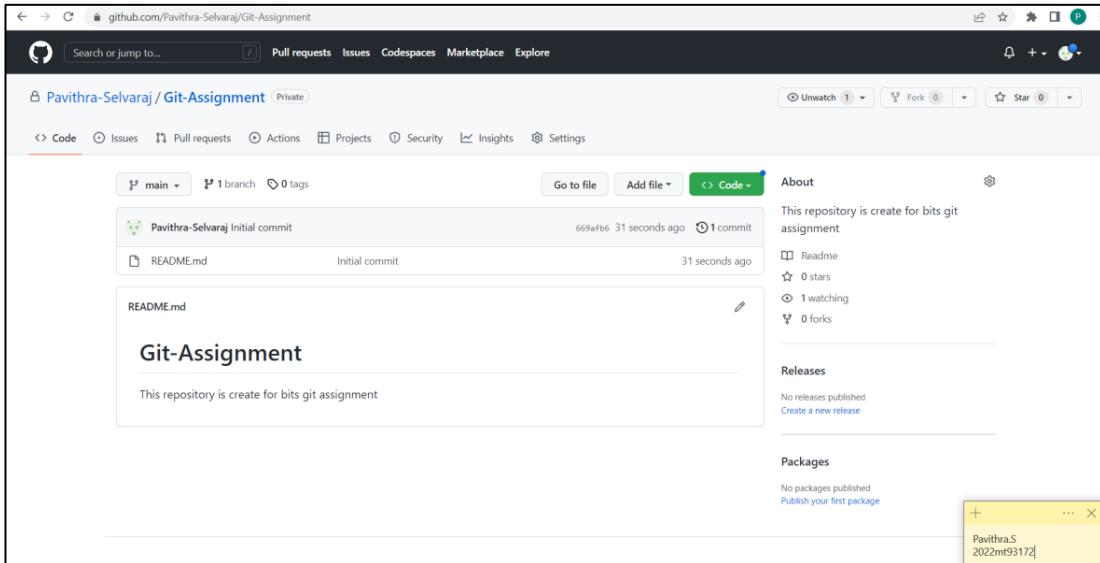
- Fill the required details.



- Click on “Create Repository” button.



- The repo is created.



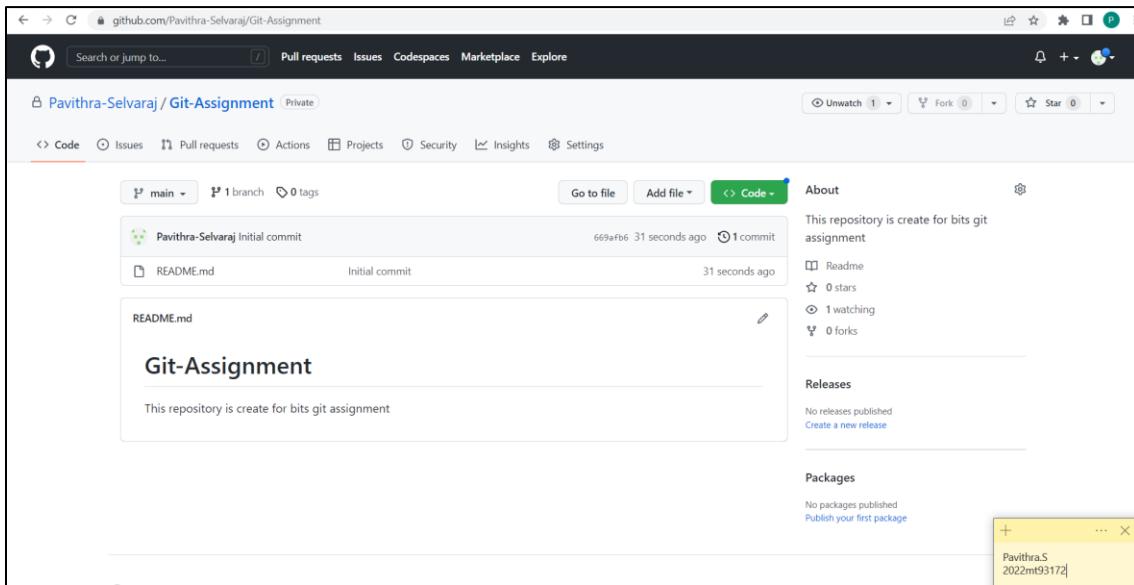
## Adding Collaborators

Typically, in a project we will have multiple team members working on the same repository. For enterprise account, we can add the team members with different roles and permissions.

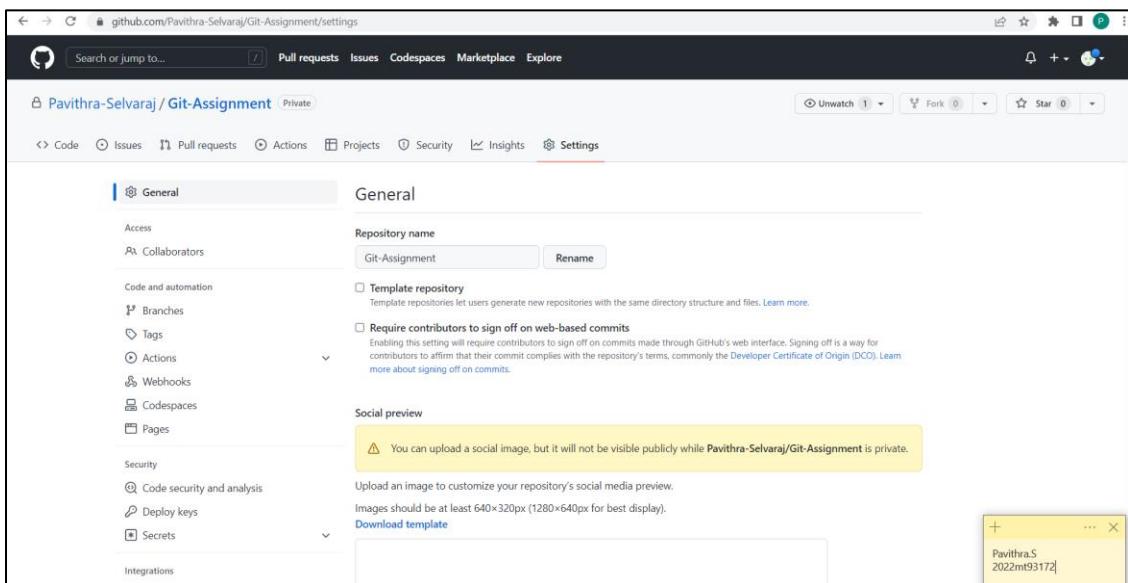
So, the account we are currently using “Pavithra-Selvaraj” is going to act as the manager account and we are going to add the other team members as collaborators.

The steps are as follows,

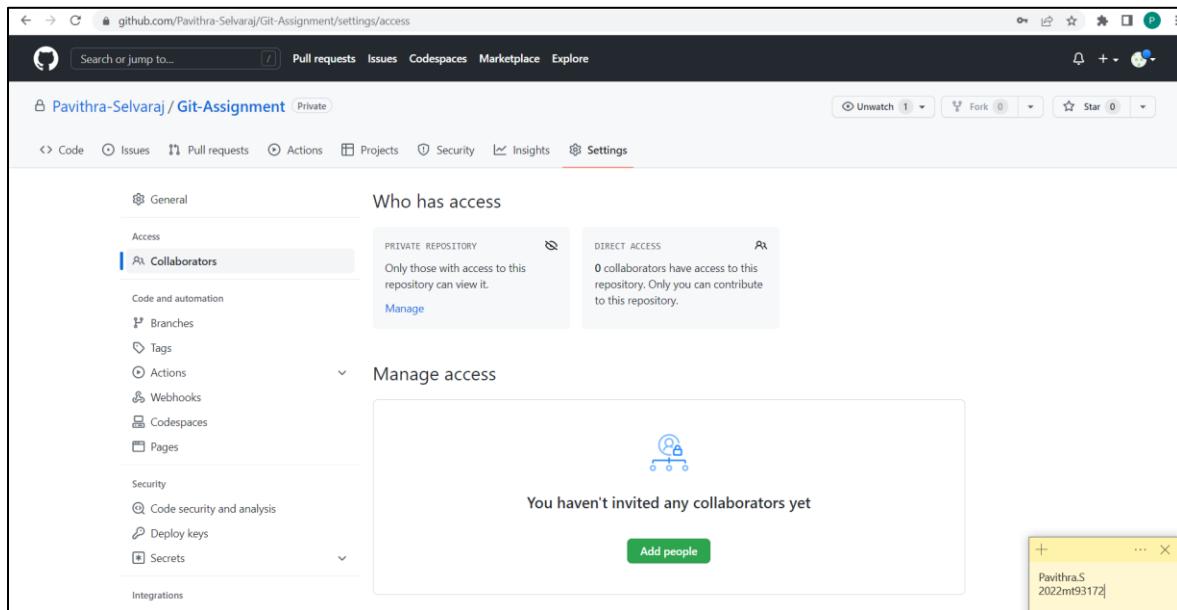
1. Click on “Settings”.



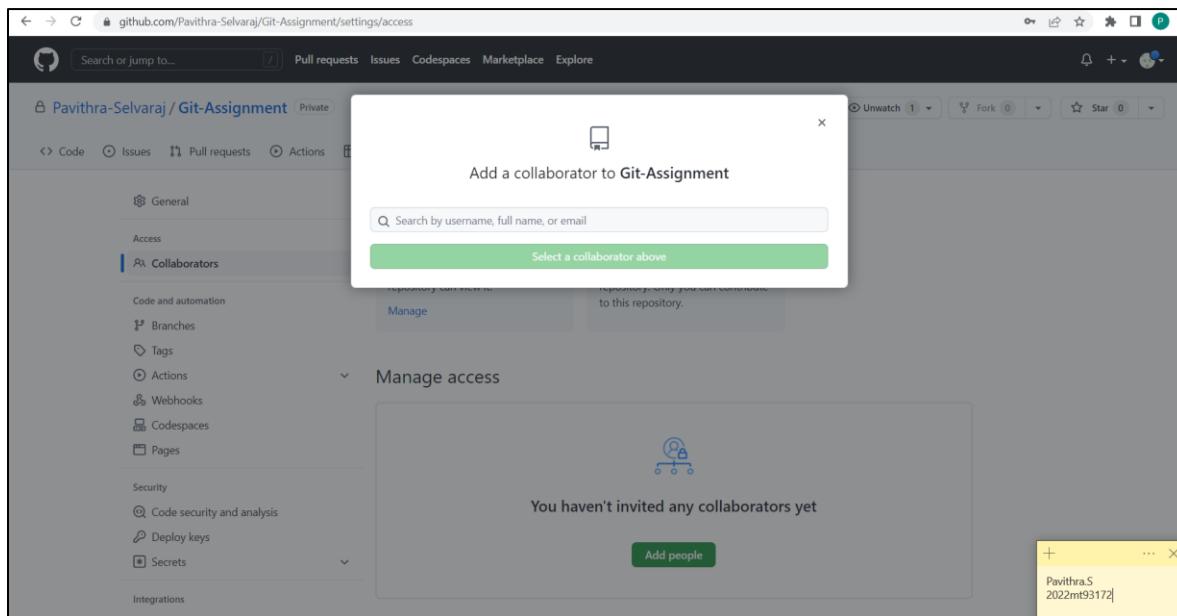
2. Click “Collaborators” option.



3. Click on “Add People” button.

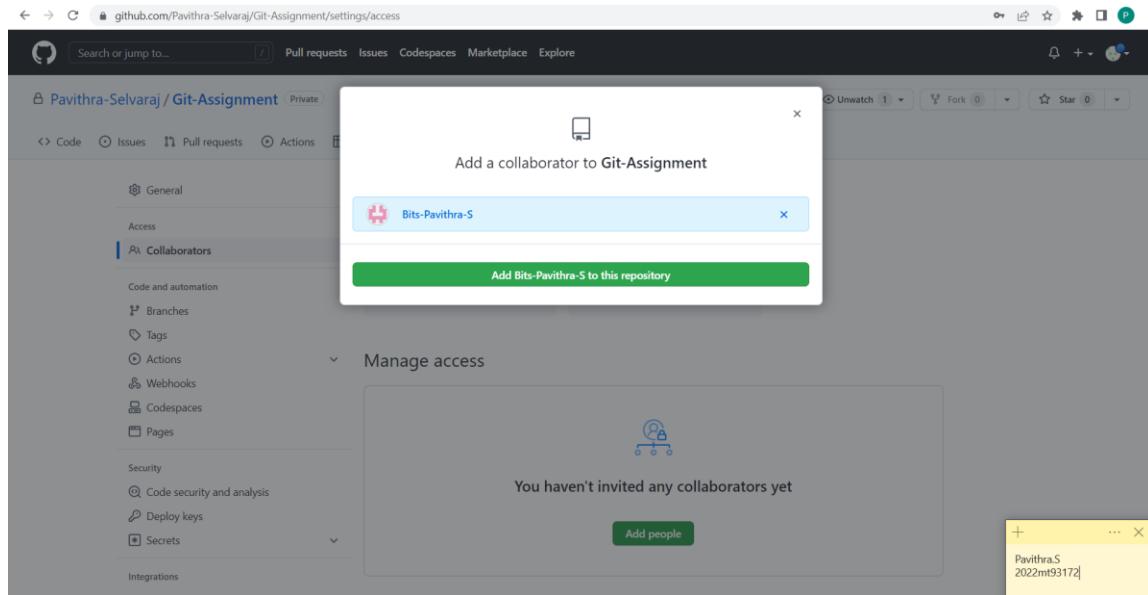


4. Search for the collaborator name.

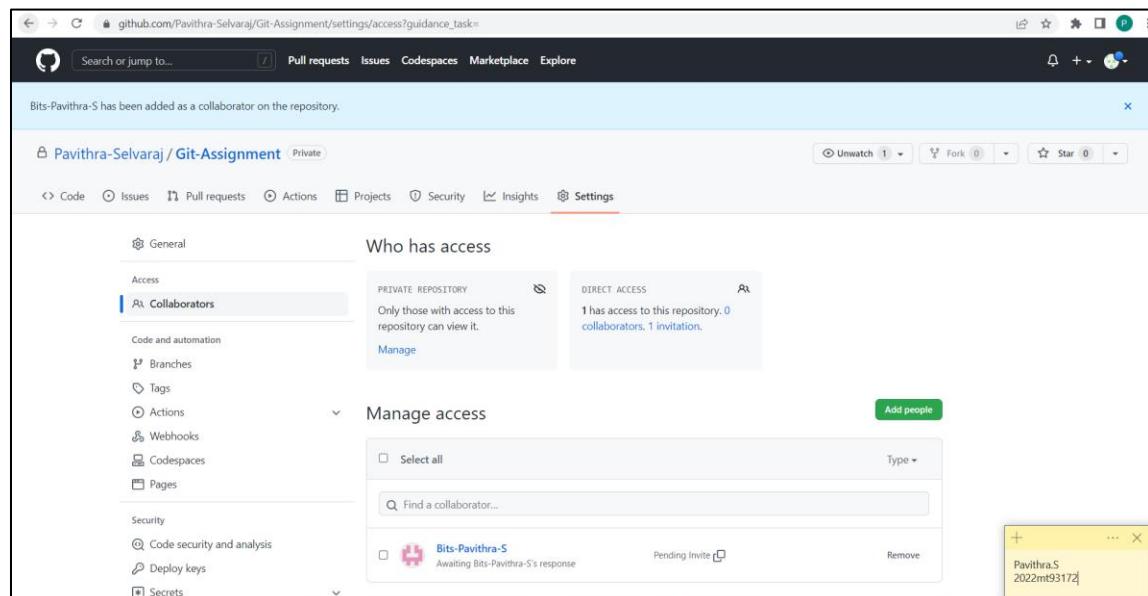


We are going to invite “Bits-Pavithra-S” (Dummy account for assignment purpose) into this repository.

Search the name and click on the Add option.



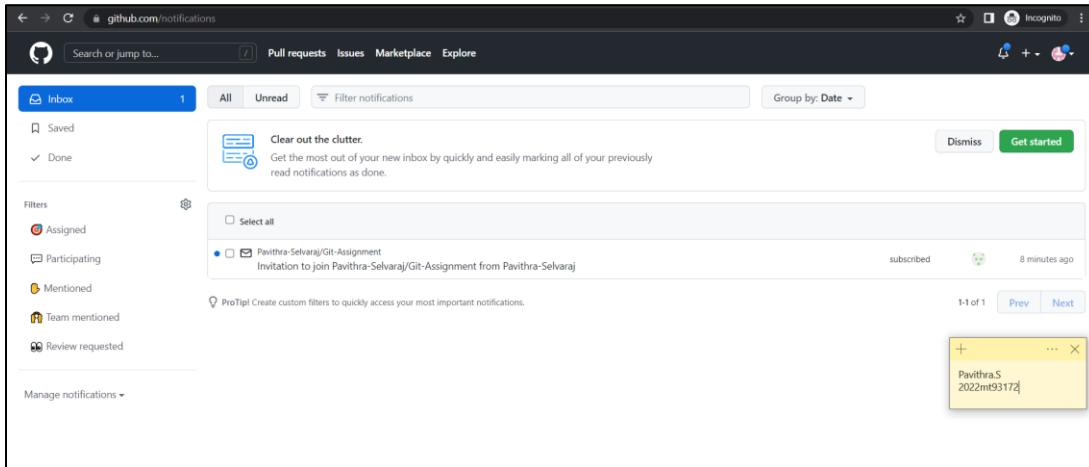
5. An invite is now sent to the collaborator.



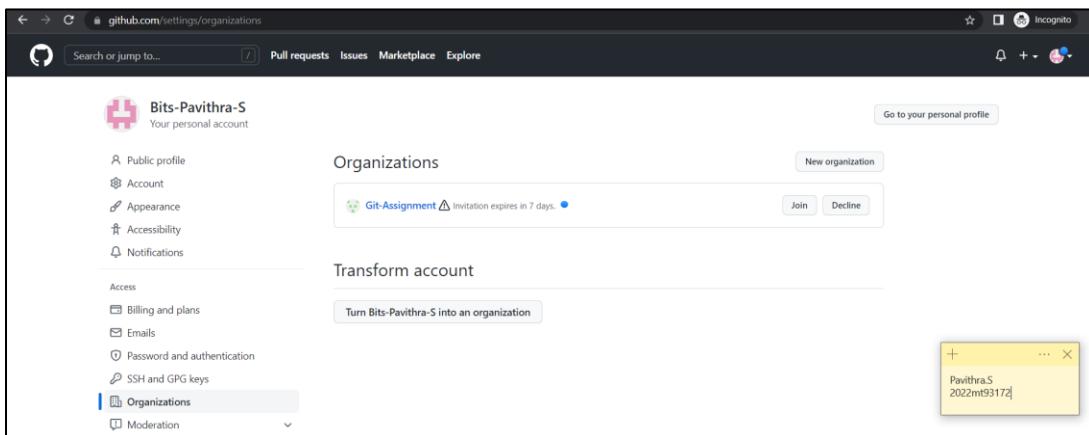
It now the collaborators work to accept the invite and be part of the repository.

## Accepting the Invite

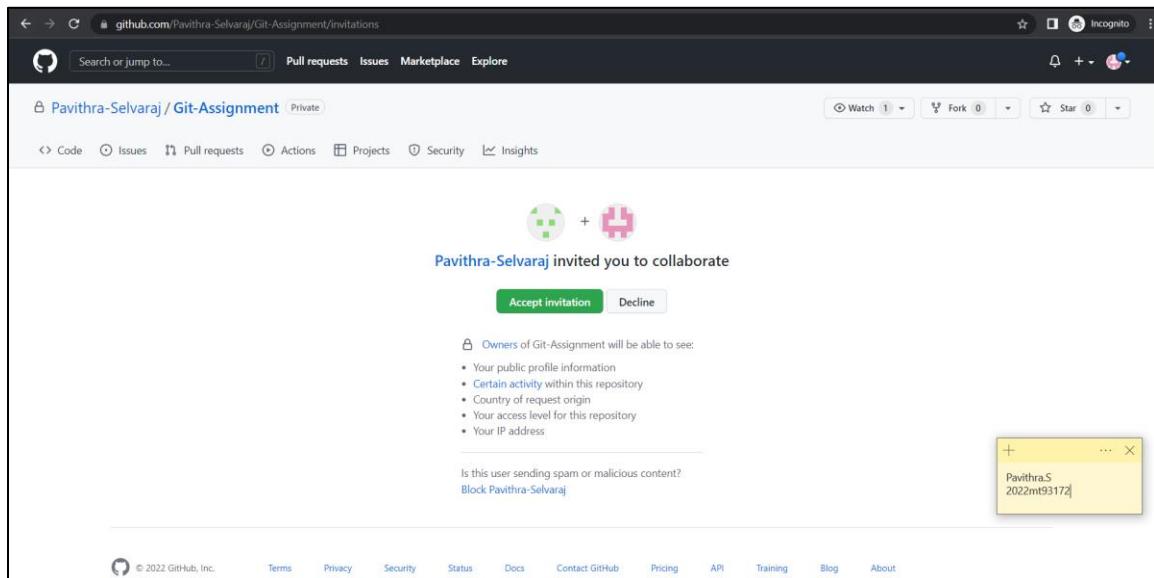
1. Once the invite is sent by the owner, a notification is received at the collaborator end.



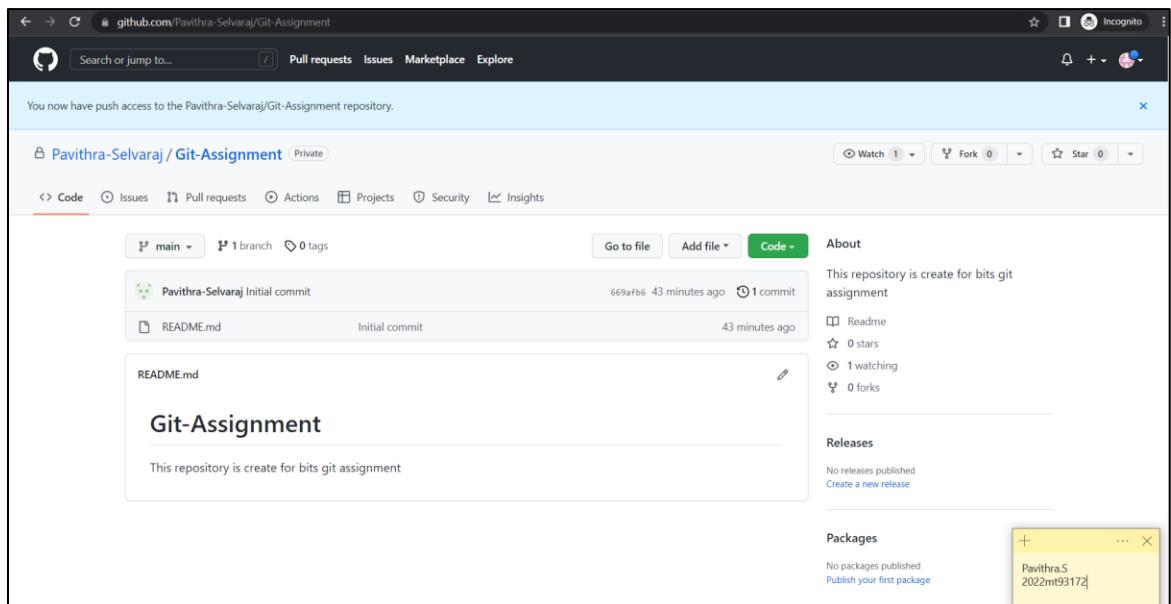
2. Open the notification to accept the request.



3. Click the “Join” button.



4. Click the “Accept Invitation” button.



The collaborator is now added to the repository.

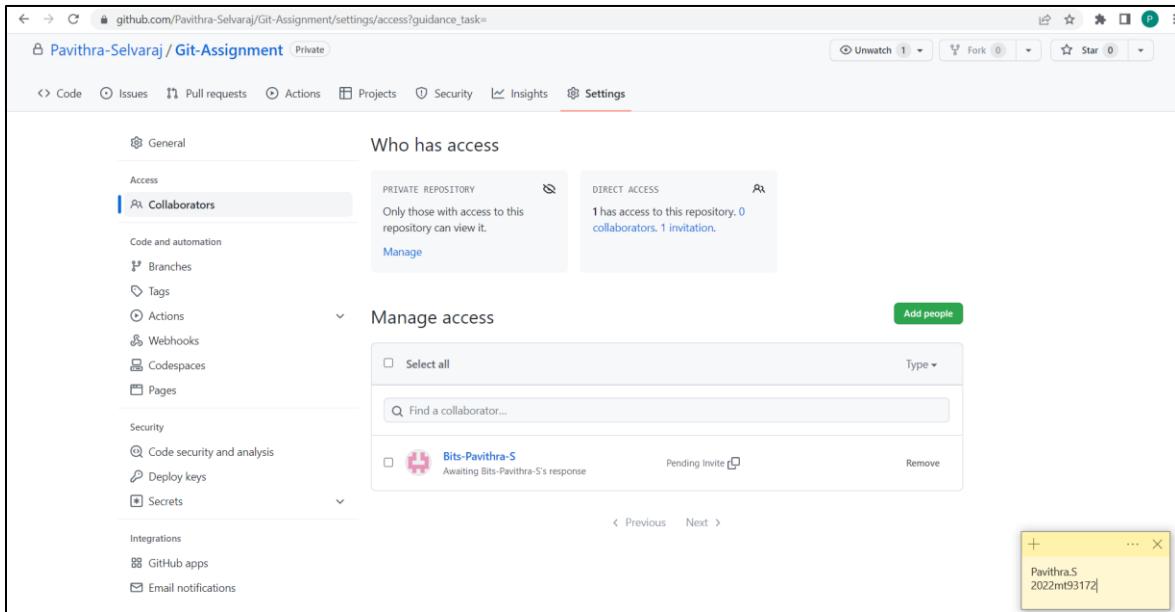
The permission levels for the collaborator and different branches can be setup in the github account directly.

## Notification Setup

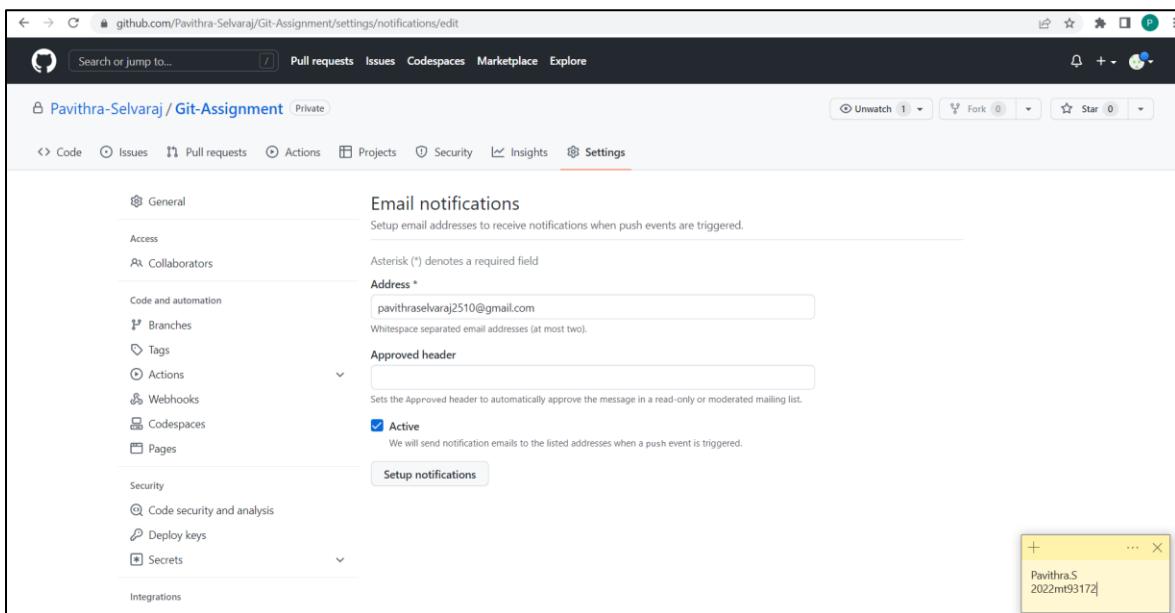
Every time a pull request is initiated, we can configure one or more email ids to send a email notification to.

The steps are,

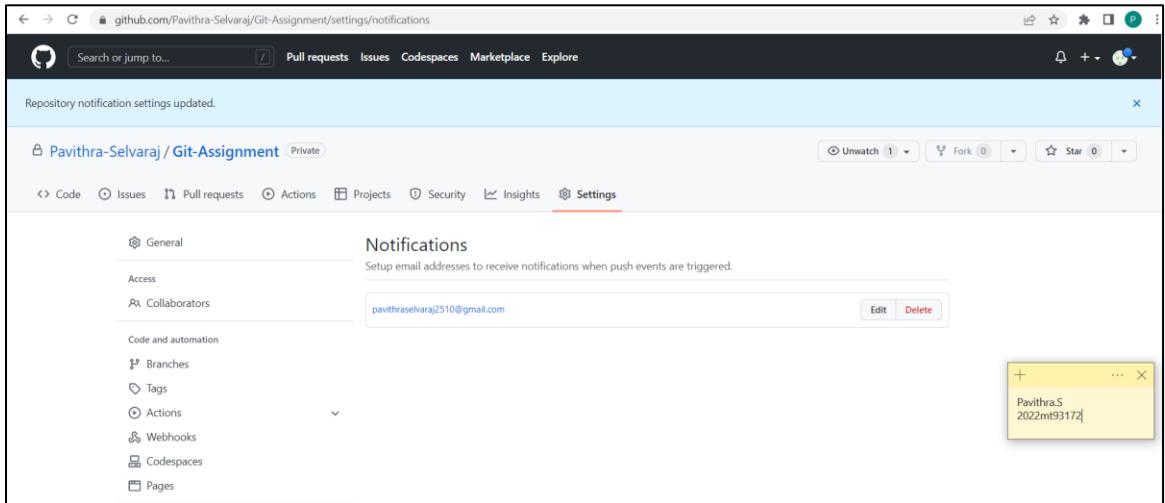
1. Open Settings.



2. Under Integrations, click on email notification.



3. Enter the email address and click on “Setup Notifications”.



With this setup completed, when a pull request is initiated, an email will be sent to the configured email addresses.

## Cloning a Repository

To copy an existing git repository to collaborate, we can clone that repository in our local machine and work.

The command for cloning is,

```
git clone <repository-link>
```

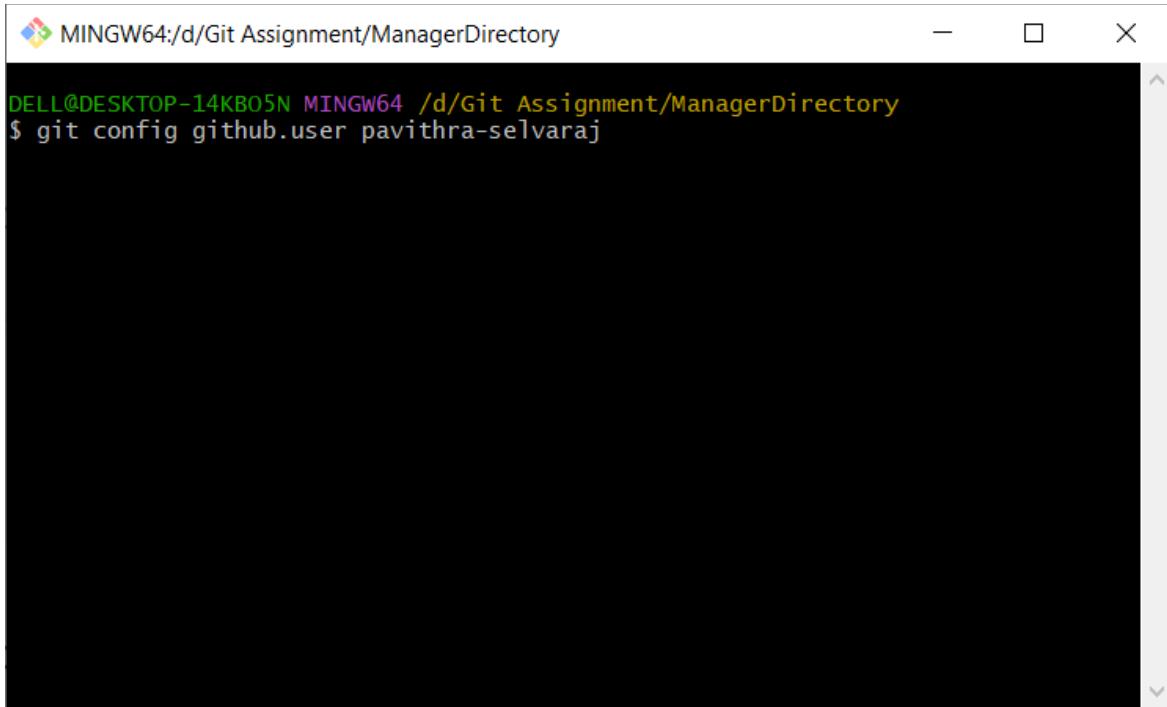
To clone our “Git-Assignment” repository, let us first create a folder in the required location.

For demonstration purpose, we are going to be using the same laptop for manager and collaborator account. So, two different folders are created to clone each of their versions.



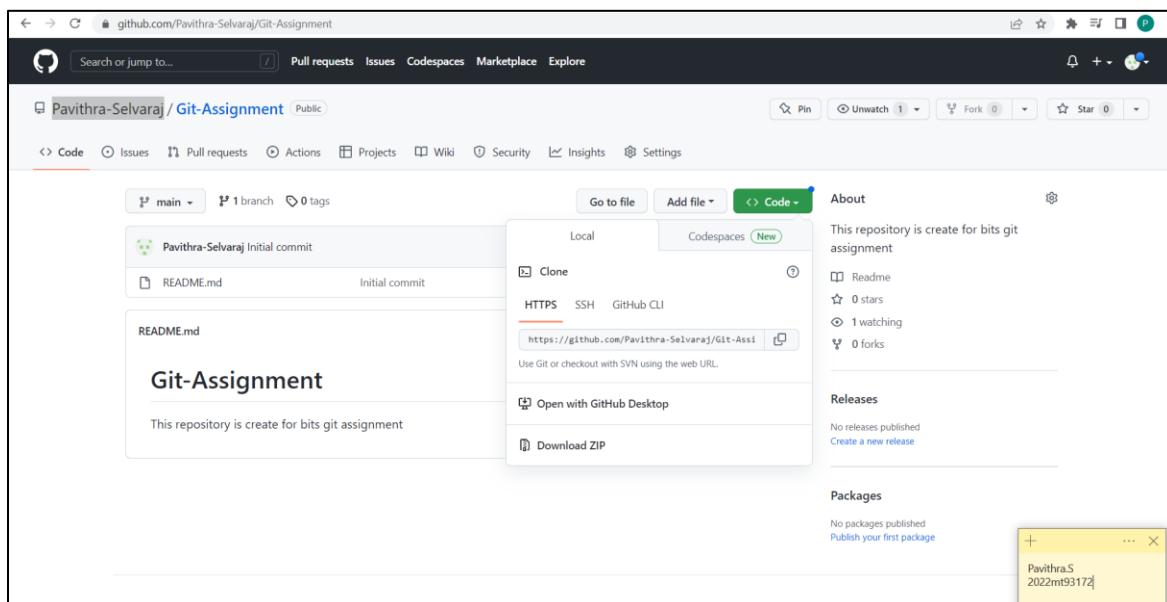
Steps to clone the repository are,

1. Open GitBash from Manager Directory and login into the manager GitHub account using the git config command.



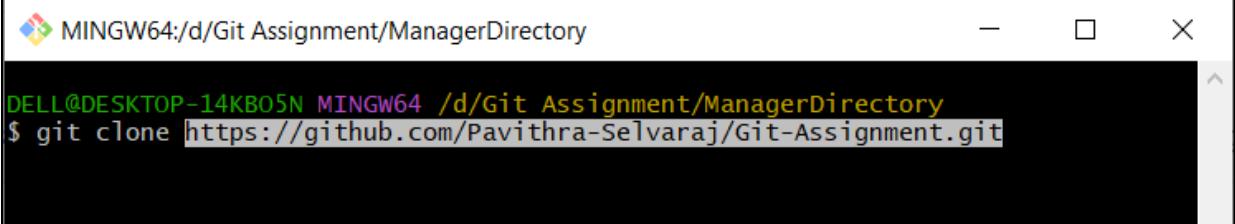
```
MINGW64:/d/Git Assignment/ManagerDirectory
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory
$ git config github.user pavithra-selvaraj
```

2. Copy the HTTPS link of the repository to clone from GitHub.



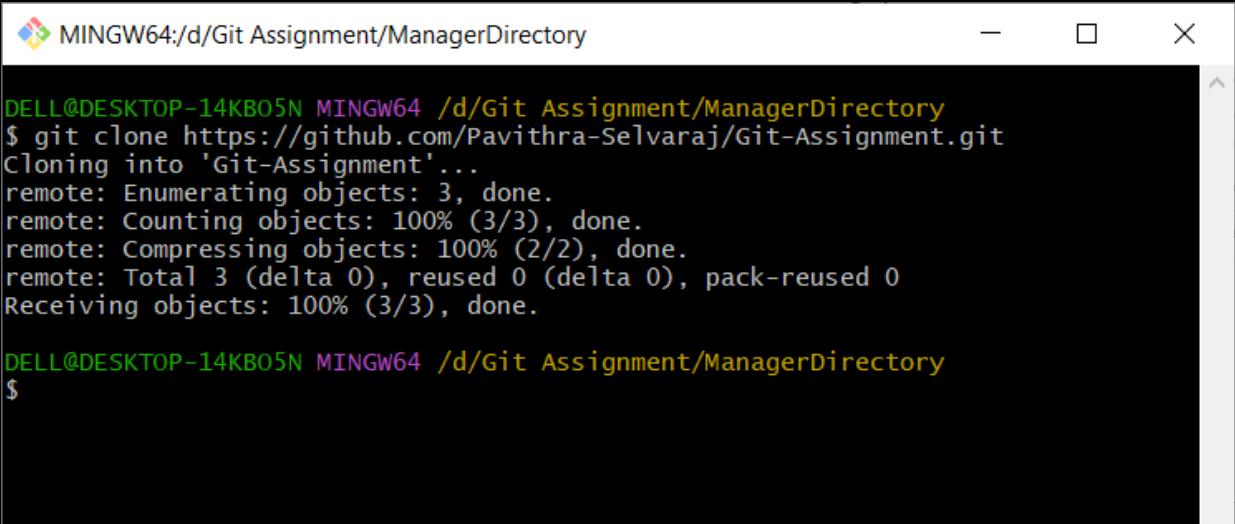
3. Use the git clone command to clone the repository.

```
git clone <https link>
```



MINGW64:/d/Git Assignment/ManagerDirectory

```
DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory
$ git clone https://github.com/Pavithra-Selvaraj/Git-Assignment.git
```

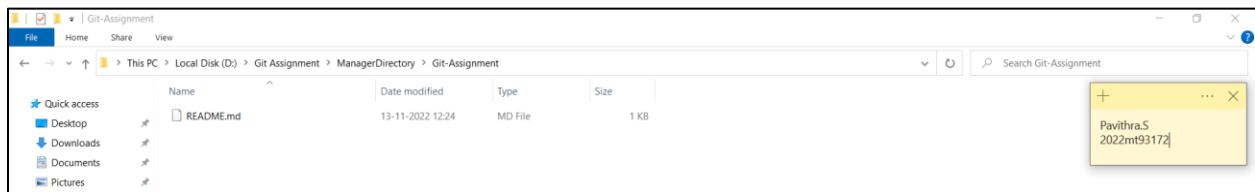


MINGW64:/d/Git Assignment/ManagerDirectory

```
DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory
$ git clone https://github.com/Pavithra-Selvaraj/Git-Assignment.git
cloning into 'Git-Assignment'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory
$
```

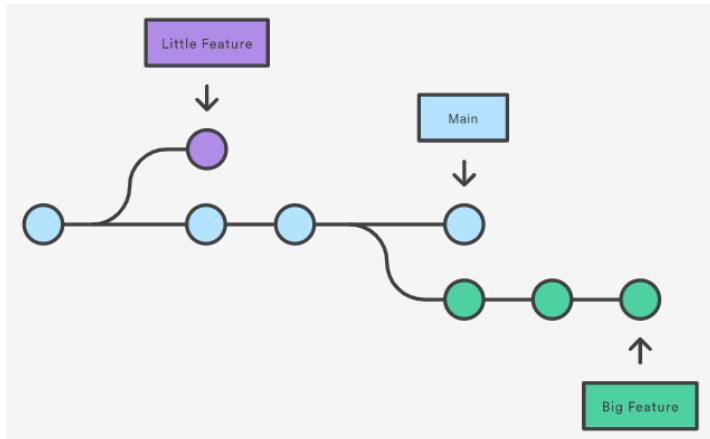
The repository is now cloned and the files are retrieved in the local folder.



## Creating a Branch

In Git, branches are a part of the development process. Git branches are a pointer to a snapshot of the code changes.

When we want to add a new feature, a fix to a bug or a hotfix we create a new branch to encapsulate the changes. This makes it harder for unstable code to get merged into the main code base. It also gives us the chance to clean up the code before merging it into the main branch.



A new branch can be created by using the command,

```
git checkout -b <branch-name>
```

To move or switch to an existing branch, we can use the command,

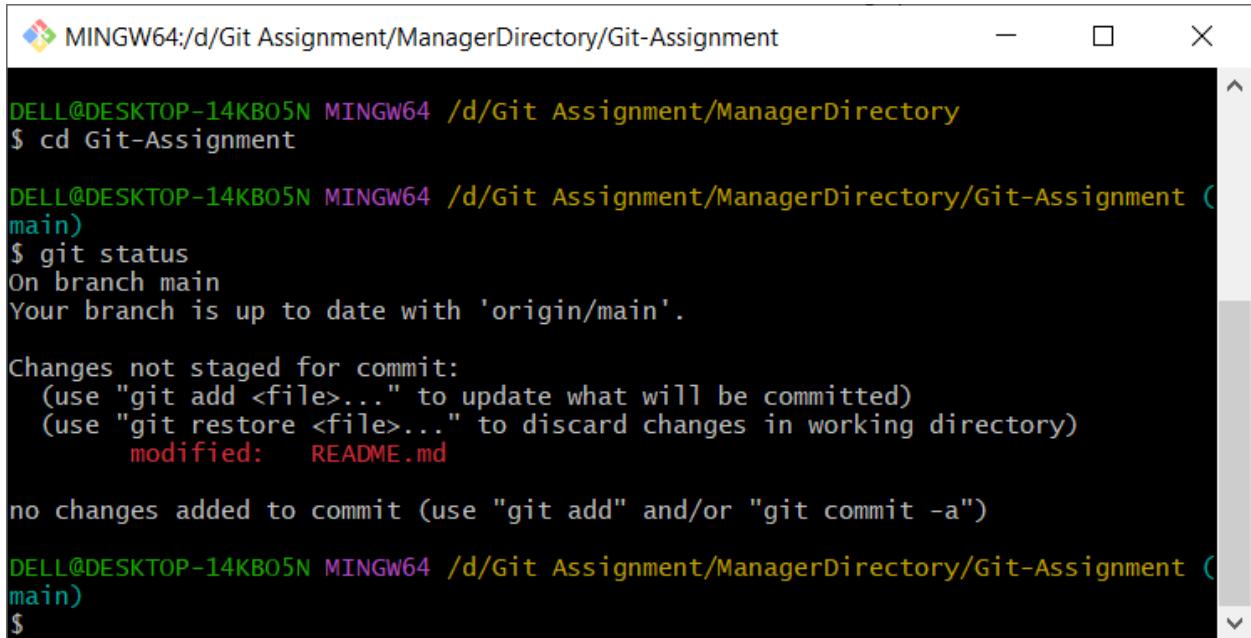
```
git checkout <branch-name>
```

Let's now add some new content to the `readme.md` file.

```
1 # Git-Assignment
```

The git status command shows the state of the working directory and the staging area.

It lets us to see what changes have been staged, what haven't, and the files that aren't being tracked by Git.



```
MINGW64:/d/Git Assignment/ManagerDirectory/Git-Assignment
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory
$ cd Git-Assignment

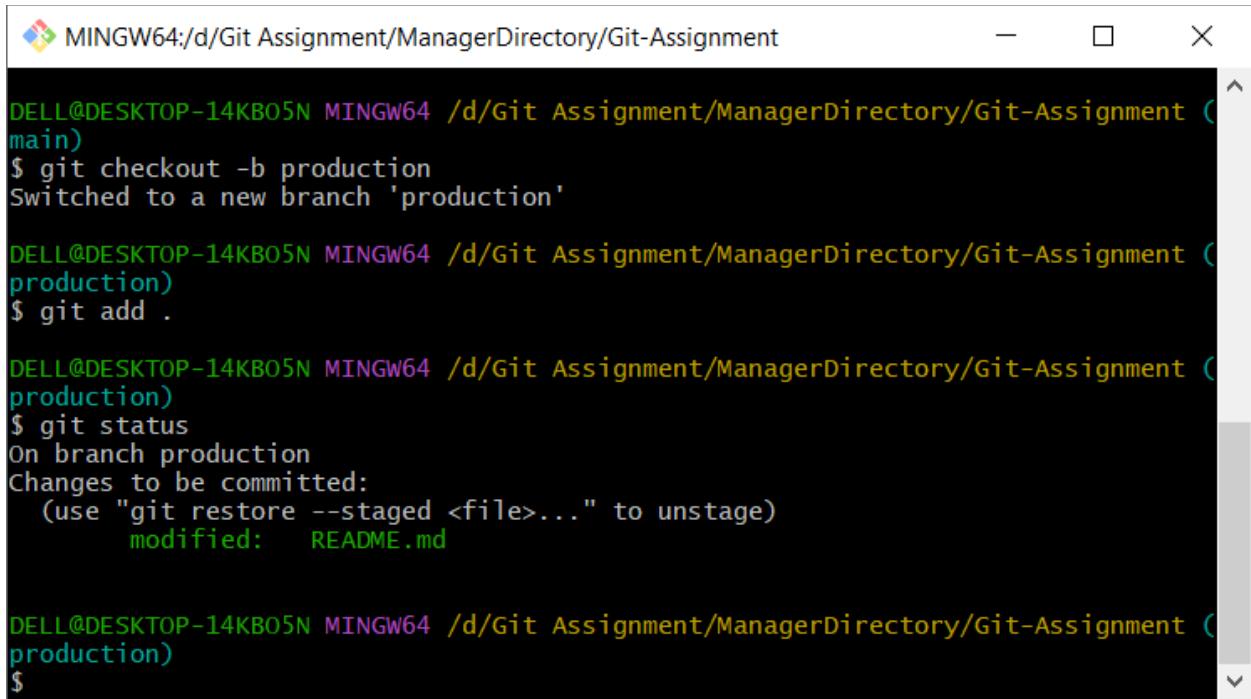
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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")

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$
```

Let's create a new branch named “production” branch.



```
MINGW64:/d/Git Assignment/ManagerDirectory/Git-Assignment
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$ git checkout -b production
Switched to a new branch 'production'

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$ git add .

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$ git status
On branch production
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   README.md

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$
```

## Code Commit

Code commits keep track of our progress and code changes as we work.

Git considers each commit as a change point or "save point".

It is a point in the project that you can go back to if you find a bug, or want to make a change.

To commit the code, stage the changes to be committed and then commit with a message using the commands.

To stage files, use,

```
git add .
```

This will add all the files to the staging area.

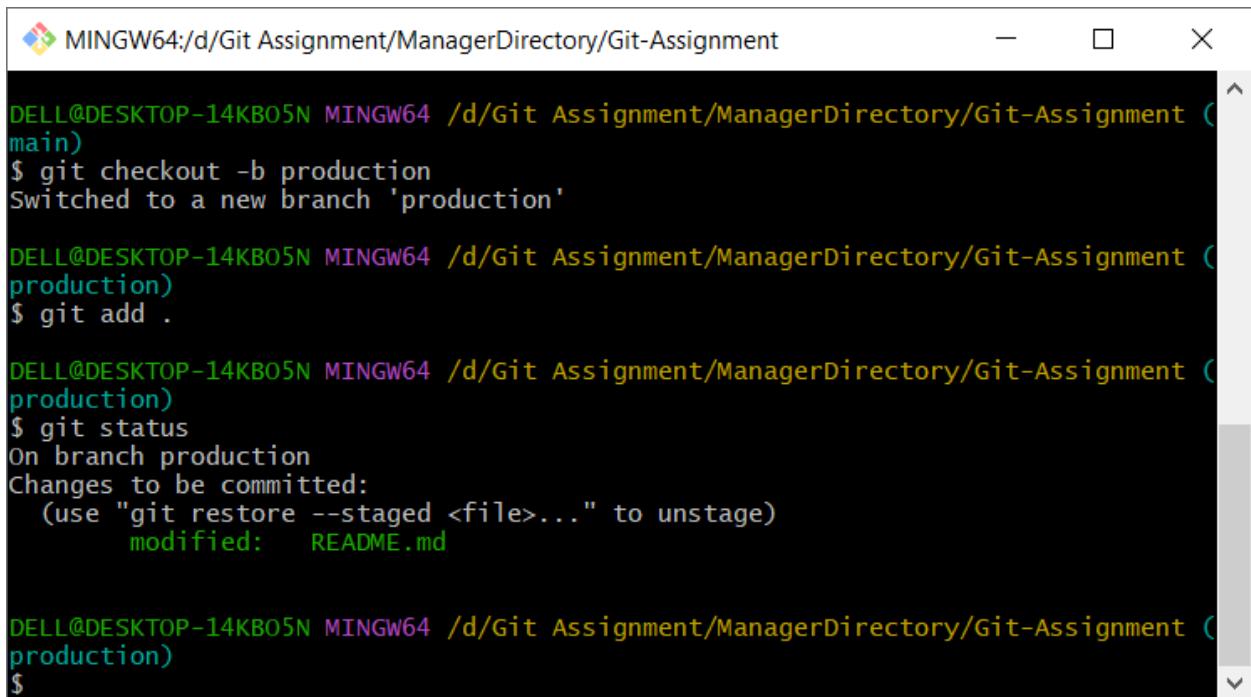
Or, use,

```
git add <file-name>
```

This can be used to add individual files.

Then commit using,

```
git commit -m <commit-message>
```



The screenshot shows a terminal window with the following session:

```
MINGW64:/d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$ git checkout -b production
Switched to a new branch 'production'

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$ git add .

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$ git status
On branch production
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified: README.md

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$
```

```
MINGW64:/d/Git Assignment/ManagerDirectory/Git-Assignment
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$ git commit -m "Updating readme file"
Author identity unknown

*** Please tell me who you are.

Run

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

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'DELL@DESKTOP-14KB05N.(none)')

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$ git config user.email "pavithraselvaraj2510@gmail.com"

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$ git config user.name "Pavithra"

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
```

```
MINGW64:/d/Git Assignment/ManagerDirectory/Git-Assignment
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$ git commit -m "Updating readme file"
[production 995d52d] Updating readme file
 1 file changed, 2 insertions(+), 1 deletion(-)

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$ git status
On branch production
nothing to commit, working tree clean

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$ |
```

## Push to Origin

The git push command is used to upload local repository content to a remote repository. Pushing is the way through which we transfer commits from our local repository to a remote repository.

The git push command takes two arguments:

- A remote name, for example, origin
- A branch name, for example, main

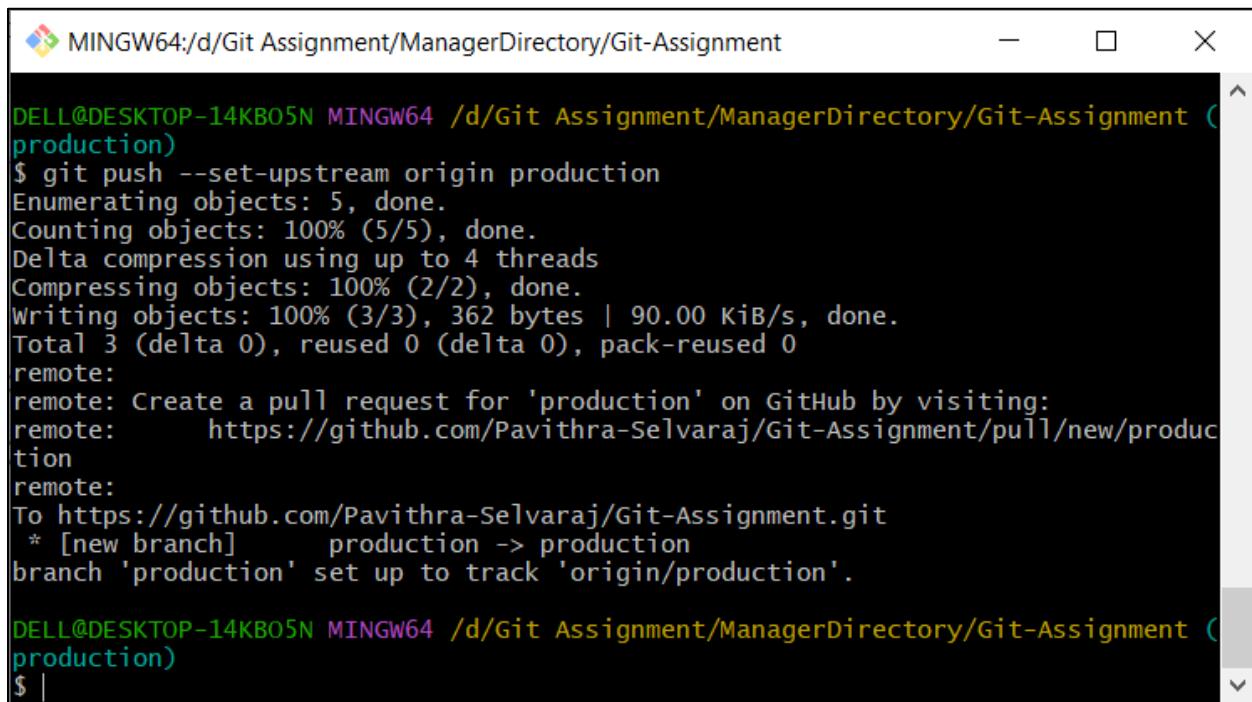
For example:

```
git push <REMOTE-NAME> <BRANCH-NAME>
```

In our case we run,

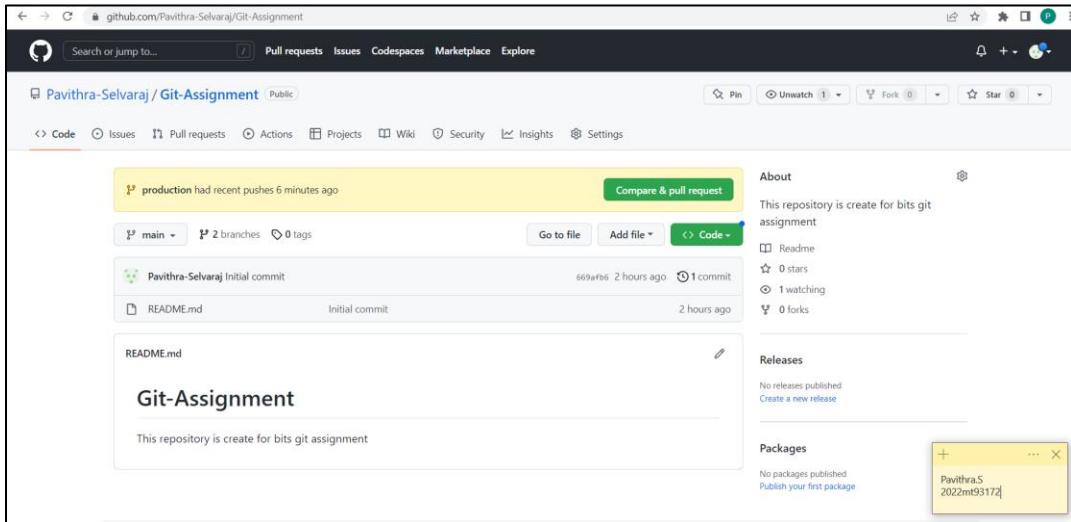
```
git push --set-upstream origin production.
```

The git set-upstream allows us to set the default remote branch for your current local branch.



```
MINGW64:/d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$ git push --set-upstream origin production
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), 362 bytes | 90.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'production' on GitHub by visiting:
remote:     https://github.com/Pavithra-Selvaraj/Git-Assignment/pull/new/production
remote:
To https://github.com/Pavithra-Selvaraj/Git-Assignment.git
 * [new branch]      production -> production
branch 'production' set up to track 'origin/production'.
```

The production branch is now pushed to the origin.



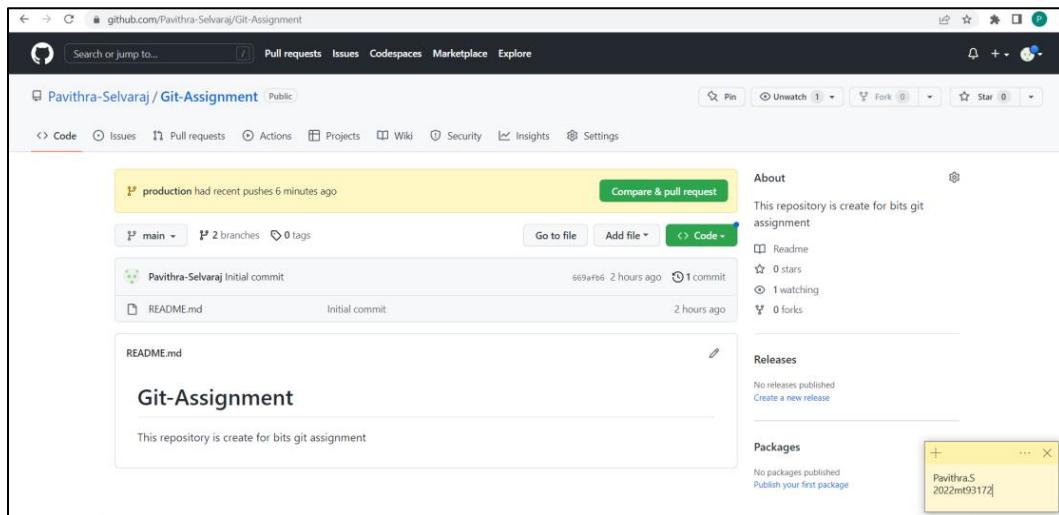
## Pull Requests

A pull request in Git is where a contributor asks a maintainer of a Git repository to review code they want to merge into a project.

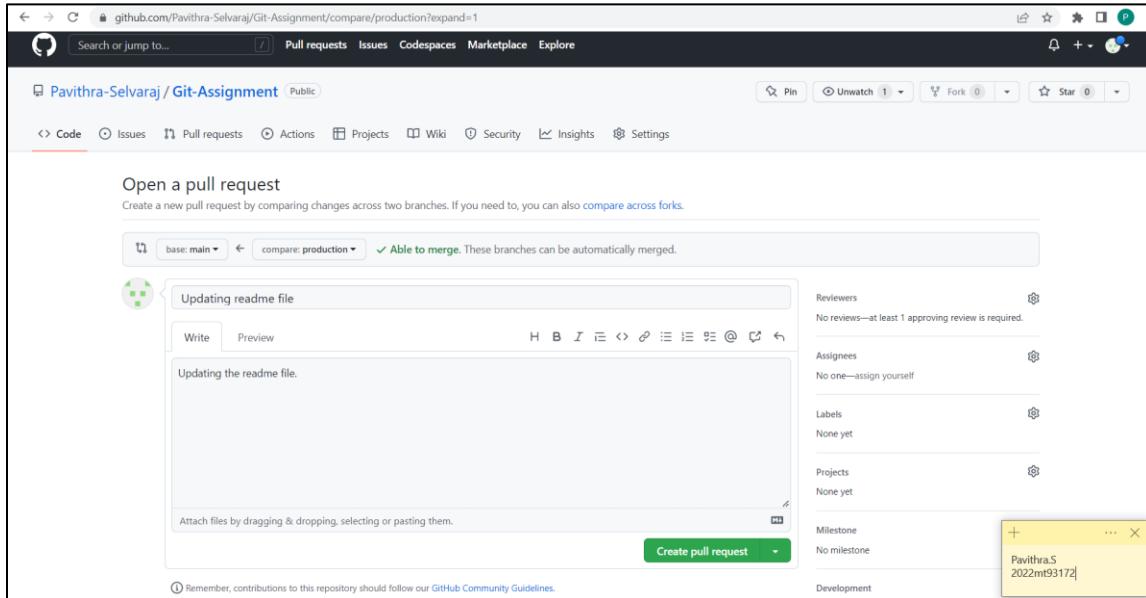
A pull request approval in Git will involve getting the project maintainer(s) to review the code part of the pull request. After code review, they will provide comments or approve the request. Approval will merge all the code changes directly into the requested repository.

Let's initiate a pull request for the production branch we created and pushed to the origin to be merged into the main branch.

Click on “Compare & pull request”. Or navigate to the pull request module to create one.



Give a description for the pull request, add assignee and click on “Create pull request”.



base: main ▾ compare: production ✓ Able to merge. These branches can be automatically merged.

Updating readme file

Updating the readme file.

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Remember, contributions to this repository should follow our GitHub Community Guidelines.

Reviewers: No reviews—at least 1 approving review is required.

Assignees: No one—assign yourself

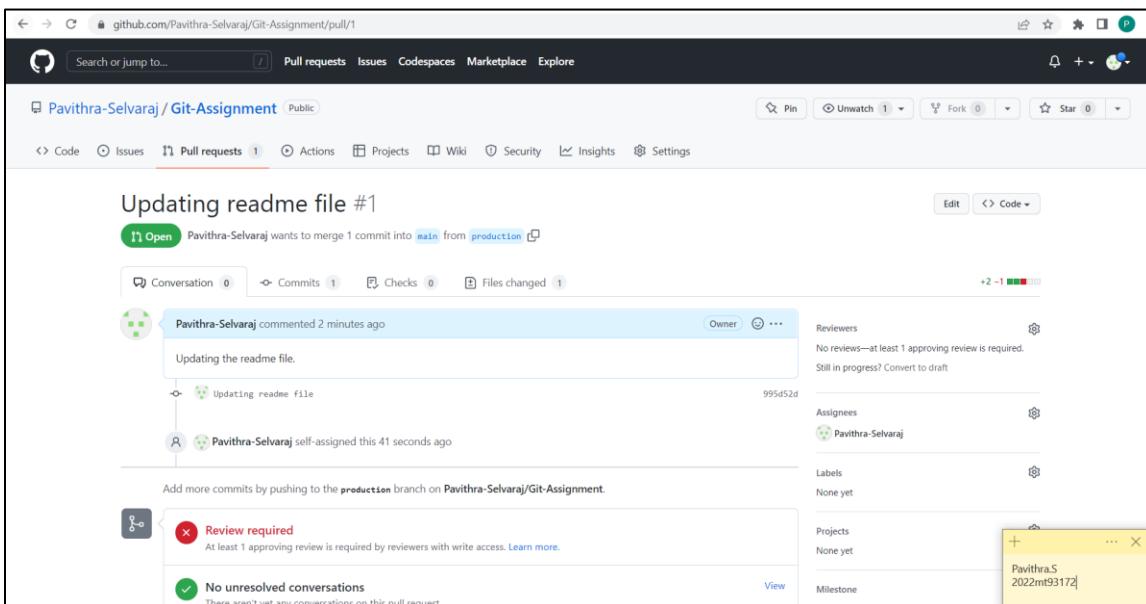
Labels: None yet

Projects: None yet

Milestone: No milestone

Pavithra.S 2022mt93172

Development



Updating readme file #1

Pavithra-Selvaraj wants to merge 1 commit into [main](#) from [production](#)

Conversation 0 Commits 1 Checks 0 Files changed 1

Pavithra-Selvaraj commented 2 minutes ago

Updating the readme file.

Pavithra-Selvaraj self-assigned this 41 seconds ago

Add more commits by pushing to the [production](#) branch on [Pavithra-Selvaraj/Git-Assignment](#).

Review required At least 1 approving review is required by reviewers with write access. [Learn more](#).

No unresolved conversations There aren't yet any conversations on this pull request.

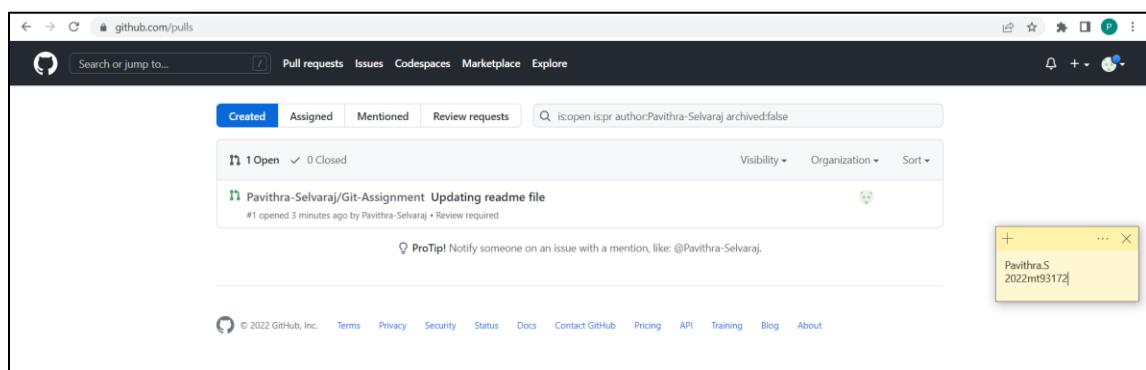
Reviewers: No reviews—at least 1 approving review is required. Still in progress? Convert to draft.

Assignees: Pavithra-Selvaraj

Labels: None yet

Projects: None yet

Milestone: Pavithra.S 2022mt93172



Created Assigned Mentioned Review requests

1 Open 0 Closed

Pavithra-Selvaraj/Git-Assignment Updating readme file

#1 opened 3 minutes ago by Pavithra-Selvaraj • Review required

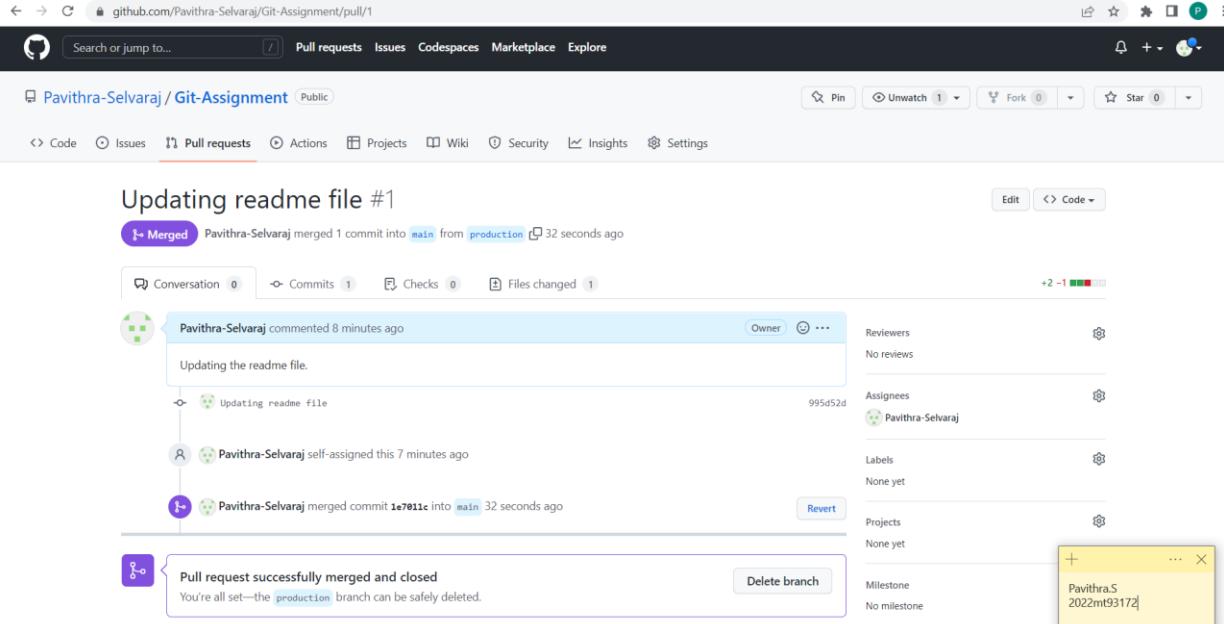
ProTip! Notify someone on an issue with a mention, like: @Pavithra-Selvaraj.

Visibility Organization Sort

Since this is a manager account, we force accept the pull request.

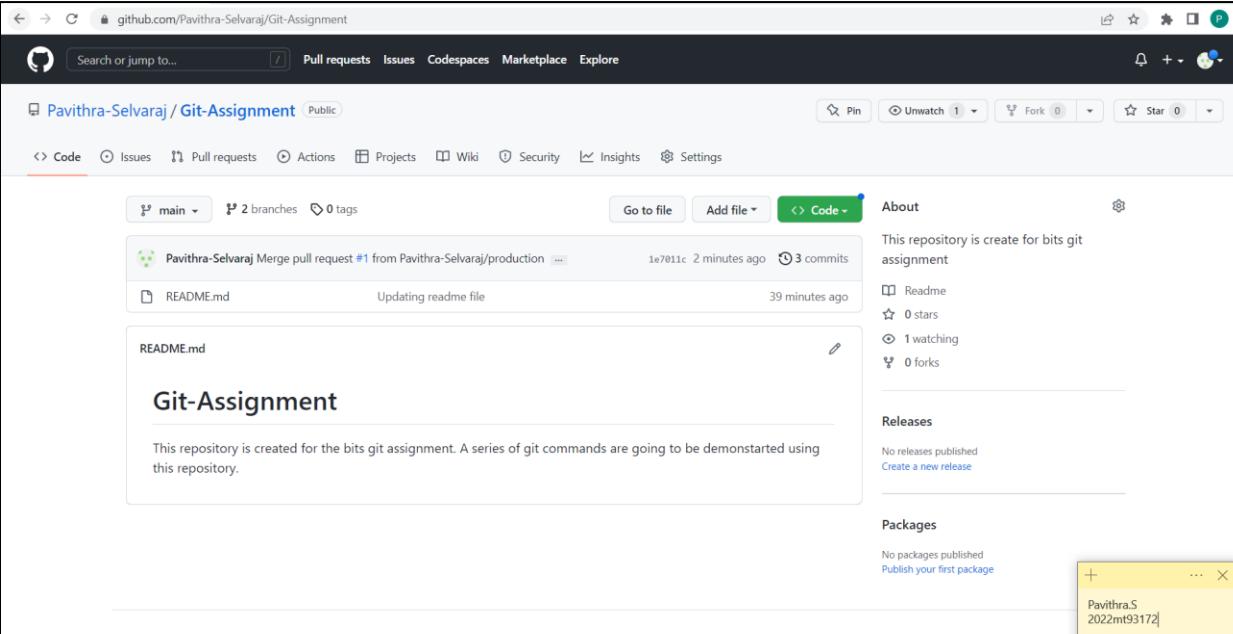
In a project situation, the approval request will be sent to the manager or the assigned person for approval.

Once approved the code will be merged.



The screenshot shows a GitHub pull request merge page for a repository named 'Git-Assignment'. The pull request has been merged, as indicated by the 'Merged' button and the message 'Pavithra-Selvaraj merged 1 commit into main from production'. The commit message 'Updating the readme file.' is visible. The pull request has 1 commit, 0 checks, and 1 file changed. The repository owner is Pavithra-Selvaraj. The merge was self-assigned by Pavithra-Selvaraj 7 minutes ago. The merge commit '1e7011c' was merged into the 'main' branch 32 seconds ago. A note at the bottom says 'Pull request successfully merged and closed. You're all set—the production branch can be safely deleted.' A 'Delete branch' button is present. The repository page shows 2 branches and 0 tags. The README.md file has been updated. The repository description is 'This repository is created for bits git assignment'. The repository has 0 stars, 1 watching, and 0 forks. It has 3 commits and 2 branches. A yellow sticky note in the bottom right corner of the page area contains the text 'Pavithra.S 2022mt93172'.

The changes are now merged and updated.



The screenshot shows the GitHub repository page for 'Git-Assignment'. The repository has 2 branches and 0 tags. The README.md file contains the text 'Git-Assignment' and 'This repository is created for bits git assignment. A series of git commands are going to be demonstrated using this repository.' The repository description is 'This repository is created for bits git assignment'. It has 0 stars, 1 watching, and 0 forks. It has 3 commits and 2 branches. A yellow sticky note in the bottom right corner of the page area contains the text 'Pavithra.S 2022mt93172'.

## Deleting a Branch

Once a branch code is merged to the main branch, the branch we created in the local and the remote repository can be deleted.

In our example, we merged the code from production branch to main branch. So, now the production branch has redundant code only and can be removed.

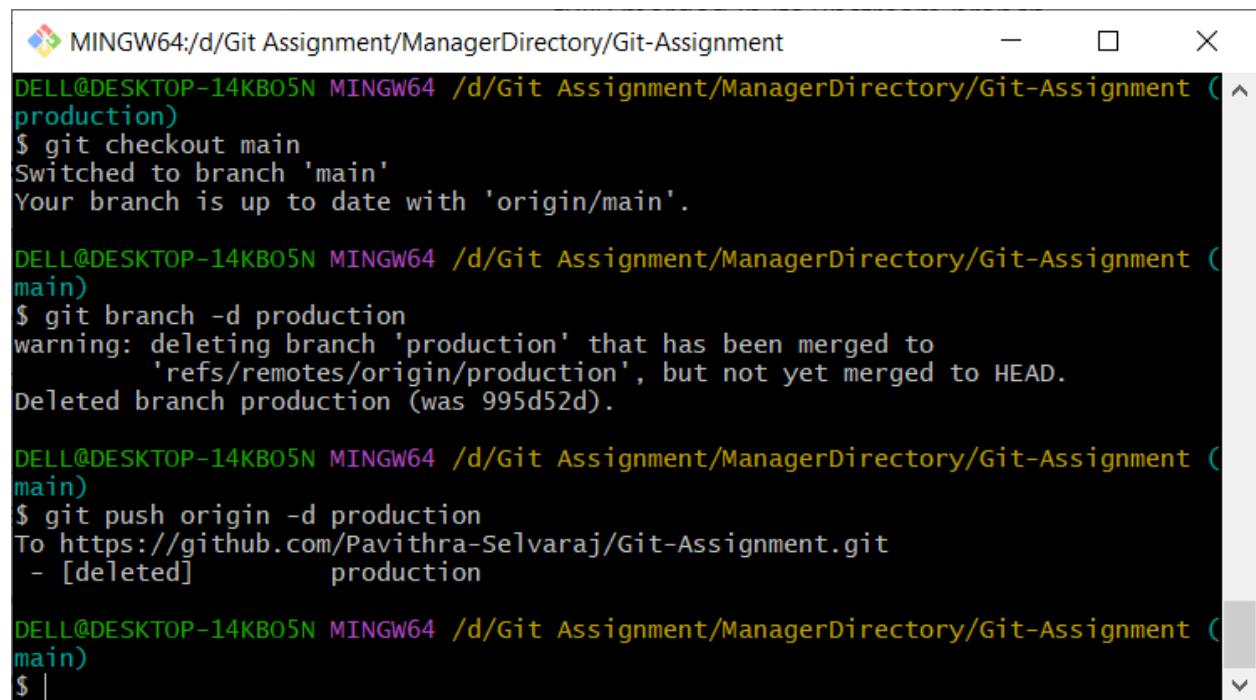
We cannot delete the branch in which we are currently in. So, move to another branch, here it's main and then delete the required branch.

The command to delete branch in local repository is,

```
git branch -d <branch-name>
```

The command to delete branch in the origin is,

```
git push origin -d <branch-name>
```



The screenshot shows a terminal window with the following session:

```
MINGW64:/d/Git Assignment/ManagerDirectory/Git-Assignment
DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$ git branch -d production
warning: deleting branch 'production' that has been merged to
      'refs/remotes/origin/production', but not yet merged to HEAD.
Deleted branch production (was 995d52d).

DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$ git push origin -d production
To https://github.com/Pavithra-Selvaraj/Git-Assignment.git
 - [deleted]      production

DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$ |
```

## Merge Conflict

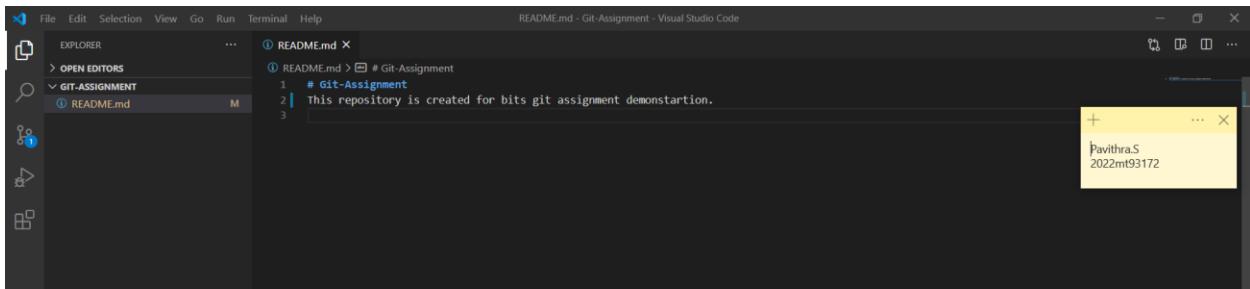
Conflicts arise when two people have changed the same lines in a file, or if one developer deleted a file while another developer was modifying it.

In these cases, Git cannot automatically determine which change is correct. So, Git will mark the file as being conflicted and halt the merging process. It is then the developer's responsibility to resolve the conflict.

In our example, the main branch is now merged with production branch and the latest code is updated the origin.

But the repository in the local is not updated yet.

Let's now edit this file again and try to push the changes to the origin.



Now, commit the changes.

```
MINGW64:/d/Git Assignment/ManagerDirectory/Git-Assignment
DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$ git add .

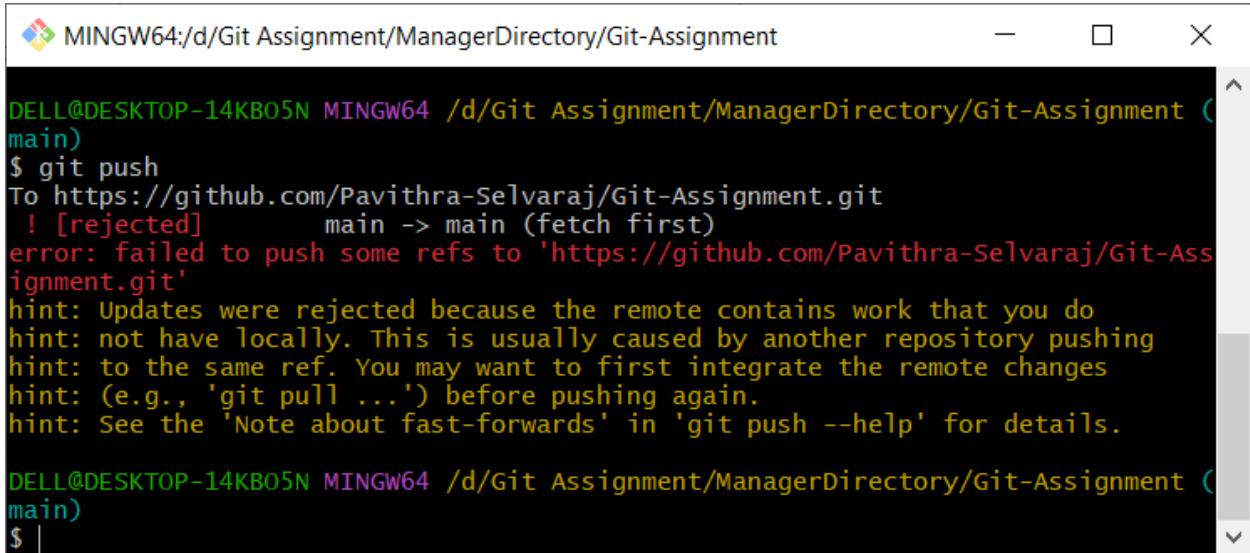
DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified: README.md

DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$ git commit -m "Re updating readme"
[main 5b44d36] Re updating readme
  1 file changed, 1 insertion(+), 1 deletion(-)

DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$
```

Push the changes to the origin.

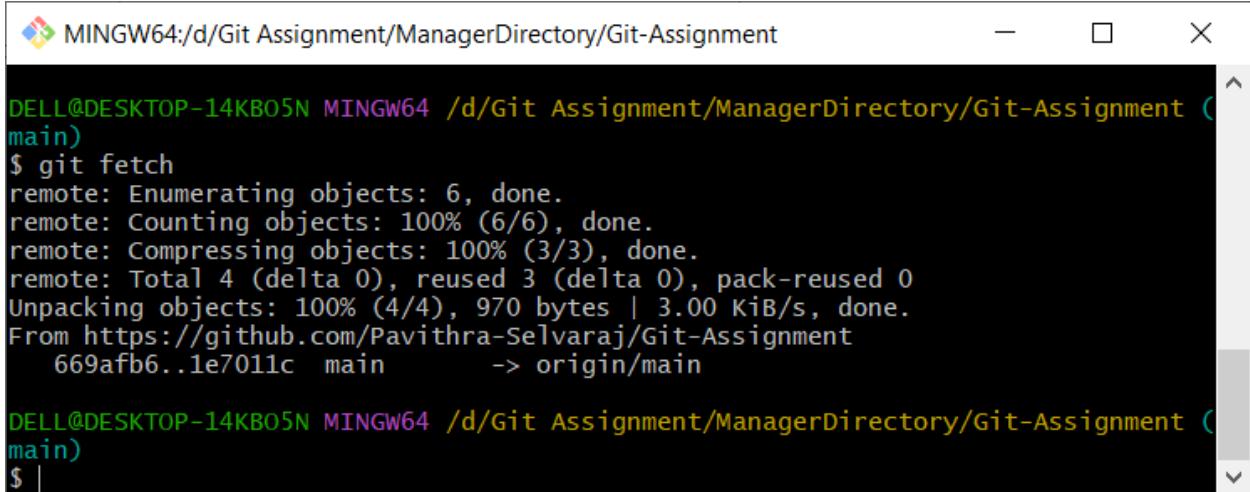


```
MINGW64:/d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$ git push
To https://github.com/Pavithra-Selvaraj/Git-Assignment.git
  ! [rejected]      main -> main (fetch first)
error: failed to push some refs to 'https://github.com/Pavithra-Selvaraj/Git-Assignment.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$ |
```

The push got rejected since there are new changes available in the origin.

Let's fetch the changes.



```
MINGW64:/d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$ git fetch
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), 970 bytes | 3.00 KiB/s, done.
From https://github.com/Pavithra-Selvaraj/Git-Assignment
  669afb6..1e7011c  main      -> origin/main

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$ |
```

The “git fetch” command will fetch all the changes from the origin to the local repository, but not merge it to the working directory.

Use the “git merge” command to merge the changes to the working directory.

git fetch and git merge used together is similar to using git pull.

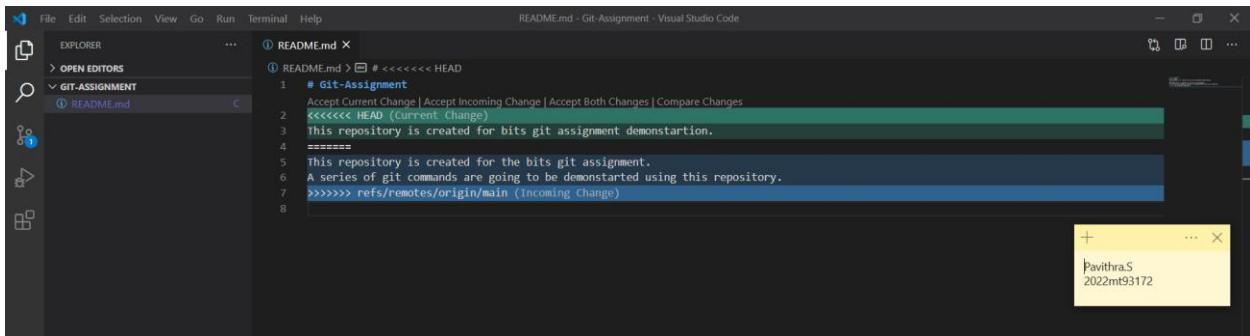
```
MINGW64:/d/Git Assignment/ManagerDirectory/Git-Assignment
DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$ git merge
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.

DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main|MERGING)
$
```

The merge is not successful. A merge conflict had occurred.

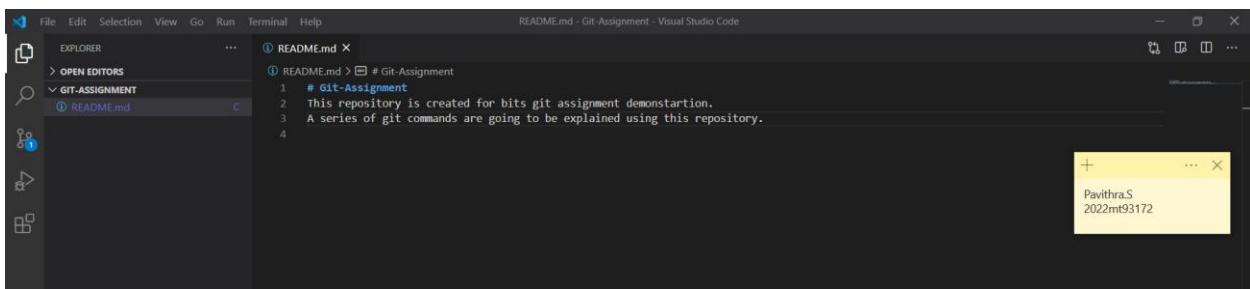
This happened since we changed the readme.md file which already had another change in the origin.

We need to resolve this in order to continue further.



```
README.md - Git-Assignment - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
GIT-ASSIGNMENT
README.md
① README.md X
① README.md > ↵ # <<<<< HEAD
1 # Git-Assignment
2 Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
3 <<<<< HEAD (Current Change)
4 This repository is created for bits git assignment demonstration.
5 =====
6 This repository is created for the bits git assignment.
7 >>>> refs/remotes/origin/main (Incoming Change)
8
```

Merge conflicts can be resolved from our ide and then we commit the changes again.



```
README.md - Git-Assignment - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
GIT-ASSIGNMENT
README.md
① README.md X
① README.md > ↵ # Git-Assignment
1 # Git-Assignment
2 This repository is created for bits git assignment demonstration.
3 A series of git commands are going to be explained using this repository.
4
```

We have now resolved the conflict by updating the content in the file.

After resolving, we can now commit the new changes and push it to the origin again.

```
MINGW64:/d/Git Assignment/ManagerDirectory/Git-Assignment
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main|MERGING)
$ git add .

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main|MERGING)
$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 2 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:
  modified: README.md

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main|MERGING)
$ git commit -m "Resolving Conflicts"
[main eab660e] Resolving Conflicts

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$
```

```
MINGW64:/d/Git Assignment/ManagerDirectory/Git-Assignment
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$ git push
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 622 bytes | 311.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/Pavithra-Selvaraj/Git-Assignment.git
  1e7011c..eab660e main -> main

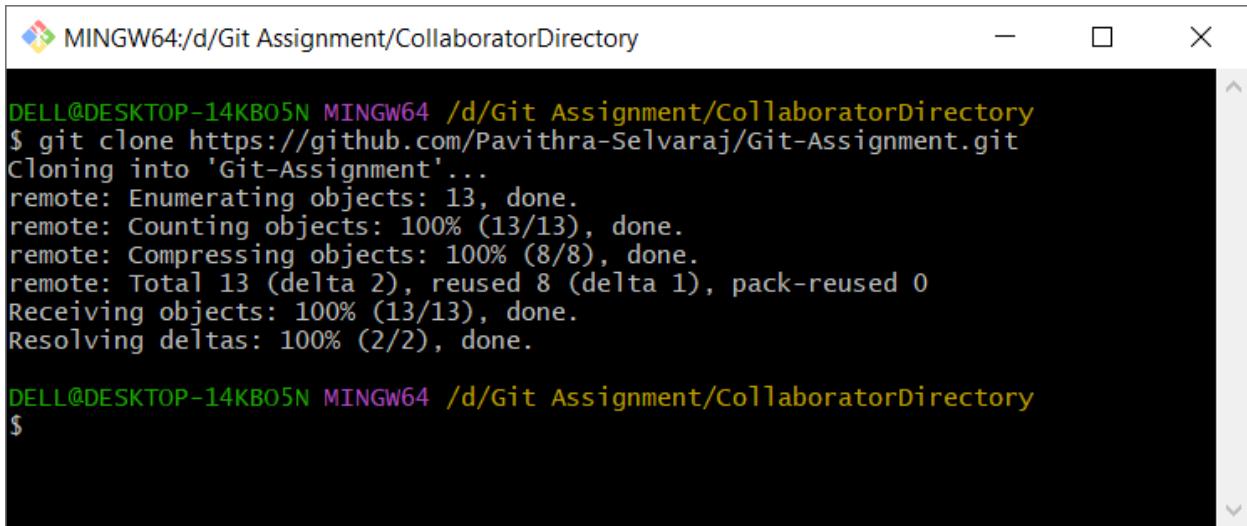
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (main)
$
```

The conflicts are now resolved and changes are successfully pushed to the origin.

## ReadMe and Gitignore Files

The collaborator of our git assignment repository is now going to add the code for a simple todo application using react in the development branch.

So, we are first cloning the repo.

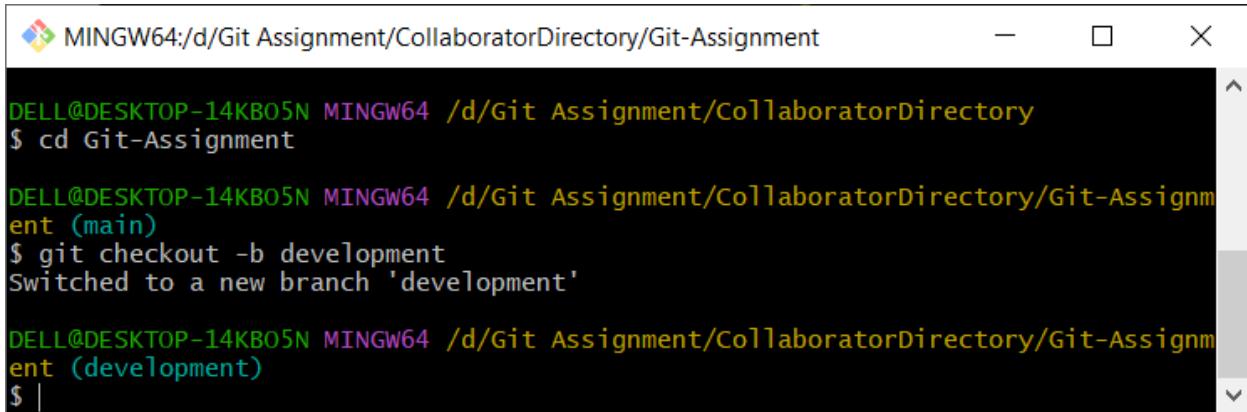


MINGW64:/d/Git Assignment/CollaboratorDirectory

```
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory
$ git clone https://github.com/Pavithra-Selvaraj/Git-Assignment.git
cloning into 'Git-Assignment'...
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 13 (delta 2), reused 8 (delta 1), pack-reused 0
Receiving objects: 100% (13/13), done.
Resolving deltas: 100% (2/2), done.

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory
$
```

Then, we create a branch named “development”.



MINGW64:/d/Git Assignment/CollaboratorDirectory/Git-Assignment

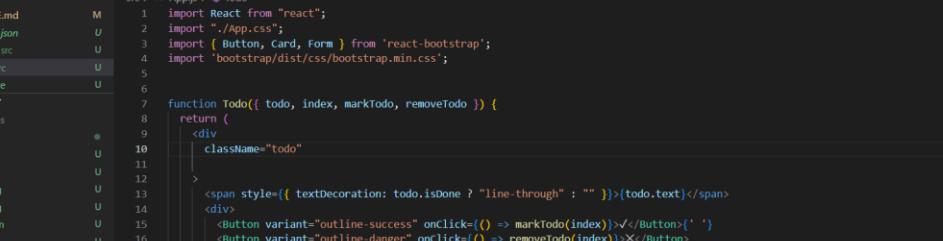
```
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory
$ cd Git-Assignment

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (main)
$ git checkout -b development
Switched to a new branch 'development'

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$
```

Now, add the required react code in the created branch.

Then, commit and push to the origin as earlier.



File Edit Selection View Go Run Terminal Help

App.js - Git-Assignment - Visual Studio Code

EXPLORER

OPEN EDITORS

- README.md M
- package.json U
- App.css U
- App.js U
- .gitignore U

GIT-ASSIGNMENT

- node\_modules
- public
  - favicon.ico
  - index.html
  - logo192.png
  - logo512.png
  - manifest.json
  - robots.txt
- src
  - App.css
  - App.js
  - App.test.js
  - index.css
  - index.js
  - logo.svg
  - reportWebVitals.js
  - setupTests.js
- .gitignore U
- package-lock.json U
- package.json U
- README.md M

src > App.js > Todo

```
1 import React from "react";
2 import "./App.css";
3 import { Button, Card, Form } from 'react-bootstrap';
4 import "bootstrap/dist/css/bootstrap.min.css";
5
6
7 function Todo({ todo, index, markTodo, removeTodo }) {
8   return (
9     <div
10       className="todo"
11     >
12       <span style={({ textDecoration: todo.isDone ? "line-through" : "" })}>{todo.text}</span>
13       <div>
14         <Button variant="outline-success" onClick={() => markTodo(index)}>✓</Button>{' '}
15         <Button variant="outline-danger" onClick={() => removeTodo(index)}>X</Button>
16       </div>
17     </div>
18   );
19 }
20
21
22 function FormTodo({ addTodo }) {
23   const [value, setValue] = React.useState("");
24 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Compiled successfully!

You can now view **todo** in the browser.

Local: <http://localhost:3000>  
On Your Network: <http://192.168.1.11:3000>

Note that the development build is not optimized.  
To create a production build, use `npm run build`.

webpack compiled successfully

Pavithra.S  
2022mrt93172

Here, notice that we have added files named `.gitignore` and `readme.md`.

## Readme.md

The Readme file is often the first file that the users read. It is a text file that contains information for the user about the software, project, code, or game, or it might contain instructions, help, or details about the patches or updates.

When sharing code with the world, a problem that might occur is that they may not particularly understand how to use the code or even understand it. So that is where the readme file helps. The readme file is used to explain what is uploaded and how we can install or use it.

A good readme file should be,

- Up to date
  - Brief and clear
  - Detailed
  - Self-explanatory



```
File Edit Selection View Go Run terminal Help README.md - Git-Assignment - Visual Studio Code README.md x package.json # App.css U JS App.js U gignore U README.md x # Git-Assignment 1 # Git-Assignment 2 This repository is created for bits git assignment demonstartion. 3 A series of git commands are going to be explained using this repository. 4 5 # Getting Started with Create React App 6 7 This project was bootstrapped with [Create React App](https://github.com/facebook/create-react-app). 8 9 ## Available Scripts 10 11 In the project directory, you can run: 12 13 *** npm start 14 15 Runs the app in the development mode. 16 Open [http://localhost:3000](http://localhost:3000) to view it in your browser. 17 18 PavithraS 2022m93172
```

## .gitignore

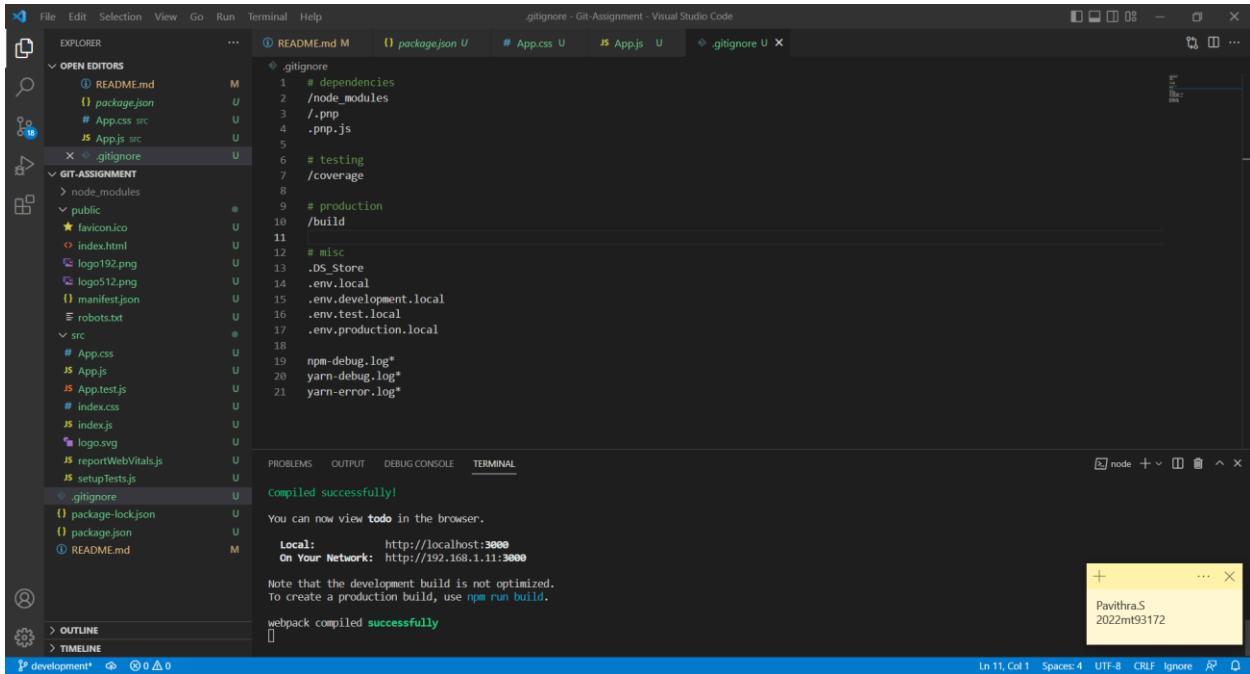
It's a file created inside the project folder that ignores/prevents files from getting committed to the local and remote repositories.

A .gitignore file is a plain text file where each line contains a pattern for files/directories to ignore.

Using this file, we can ignore the files that we never want to commit.

What Kind of Files Should You Ignore?

- Log files
- Files with API keys/secrets, credentials, or sensitive information
- Useless system files like .DS\_Store on macOS
- Generated files like dist folders
- Dependencies which can be downloaded from a package manager. Eg: node\_modules.
- And there might be other reasons (maybe you make little todo.md files)



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists files and folders, including a .gitignore file under the 'GIT-ASSIGNMENT' folder. The terminal tab at the bottom shows the following output:

```
Compiled successfully!
You can now view todo in the browser.
  Local:      http://localhost:3000
  On Your Network: http://192.168.1.11:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
webpack compiled successfully
```

Gitignore files are something that will be part of almost every project. It's important to ignore the correct files.

Now commit the code. The collaborator is not logged in. So login is required.

```
MINGW64:/d/Git Assignment/CollaboratorDirectory/Git-Assignment
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ git add.

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ git add .
warning: in the working copy of 'package-lock.json', LF will be replaced by CRLF
the next time Git touches it
warning: in the working copy of 'package.json', LF will be replaced by CRLF the
next time Git touches it
warning: in the working copy of 'public/index.html', LF will be replaced by CRLF
the next time Git touches it
warning: in the working copy of 'public/manifest.json', LF will be replaced by CRLF
the next time Git touches it
warning: in the working copy of 'public/robots.txt', LF will be replaced by CRLF
the next time Git touches it
warning: in the working copy of 'src/App.css', LF will be replaced by CRLF the
next time Git touches it
warning: in the working copy of 'src/App.js', LF will be replaced by CRLF the
next time Git touches it
warning: in the working copy of 'src/App.test.js', LF will be replaced by CRLF
the next time Git touches it
warning: in the working copy of 'src/index.css', LF will be replaced by CRLF the
next time Git touches it
warning: in the working copy of 'src/index.js', LF will be replaced by CRLF the
next time Git touches it
warning: in the working copy of 'src/reportWebVitals.js', LF will be replaced by
CRLF the next time Git touches it
warning: in the working copy of 'src/setupTests.js', LF will be replaced by CRLF
the next time Git touches it

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ git commit -m "todo application initial commit"
Author identity unknown

*** Please tell me who you are.

Run

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

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'DELL@DESKTOP-14KB05N.(none)')

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
```

```
MINGW64:/d/Git Assignment/CollaboratorDirectory/Git-Assignment
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ git config user.email "2022MT93172@wilp.bits-pilani.ac.in"

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ git config user.name Bits-Pavithra-S

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ git commit -m "todo application initial commit"
[development 834d56e] todo application initial commit
18 files changed, 17515 insertions(+)
  create mode 100644 .gitignore
  create mode 100644 package-lock.json
  create mode 100644 package.json
  create mode 100644 public/favicon.ico
  create mode 100644 public/index.html
  create mode 100644 public/logo192.png
  create mode 100644 public/logo512.png
  create mode 100644 public/manifest.json
  create mode 100644 public/robots.txt
  create mode 100644 src/App.css
  create mode 100644 src/App.js
  create mode 100644 src/App.test.js
  create mode 100644 src/index.css
  create mode 100644 src/index.js
  create mode 100644 src/logo.svg
  create mode 100644 src/reportWebVitals.js
  create mode 100644 src/setupTests.js

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ |
```

Now, push the code to origin development branch.

```
MINGW64:/d/Git Assignment/CollaboratorDirectory/Git-Assignment
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ git push origin development
Enumerating objects: 24, done.
Counting objects: 100% (24/24), done.
Delta compression using up to 4 threads
Compressing objects: 100% (22/22), done.
Writing objects: 100% (22/22), 171.19 KiB | 4.08 MiB/s, done.
Total 22 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'development' on GitHub by visiting:
remote:     https://github.com/Pavithra-Selvaraj/Git-Assignment/pull/new/development
remote:
To https://github.com/Pavithra-Selvaraj/Git-Assignment.git
 * [new branch]      development -> development

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ |
```

## Force Push and Reset Commit

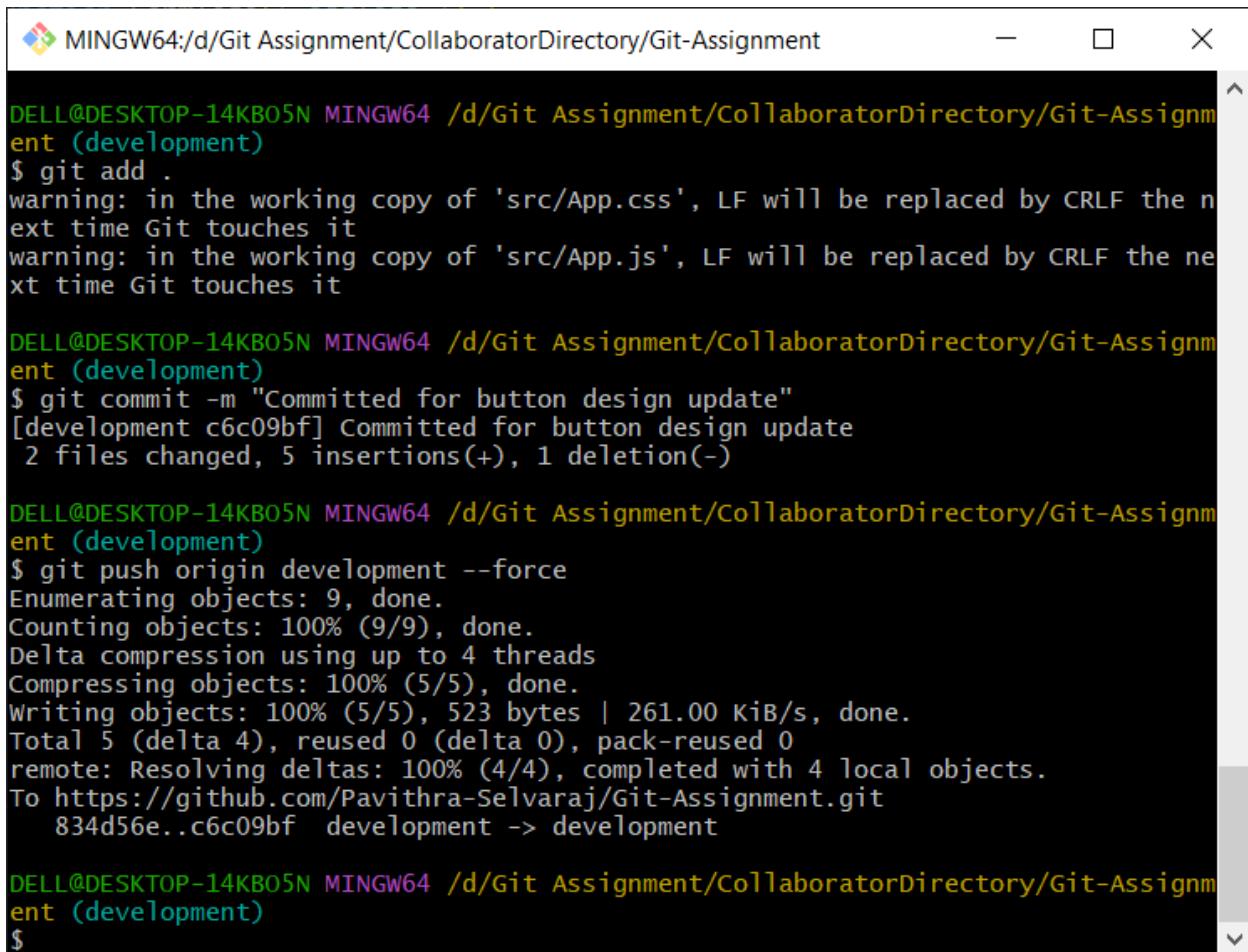
Git will normally only allow us to push the changes if we have previously updated the local branch with the latest commits from its remote counterpart. Only when we are up-to-date, we will be able to push new commits to the remote.

The --force option for git push allows you to override this rule. The commit history on the remote will be forcefully overwritten with your own local history.

To demonstrate this, lets add a new commit to our code and force push it.

The command will be,

```
git push origin <Branch-Name> --force
```



The screenshot shows a terminal window titled 'MINGW64:/d/Git Assignment/CollaboratorDirectory/Git-Assignment'. The terminal output is as follows:

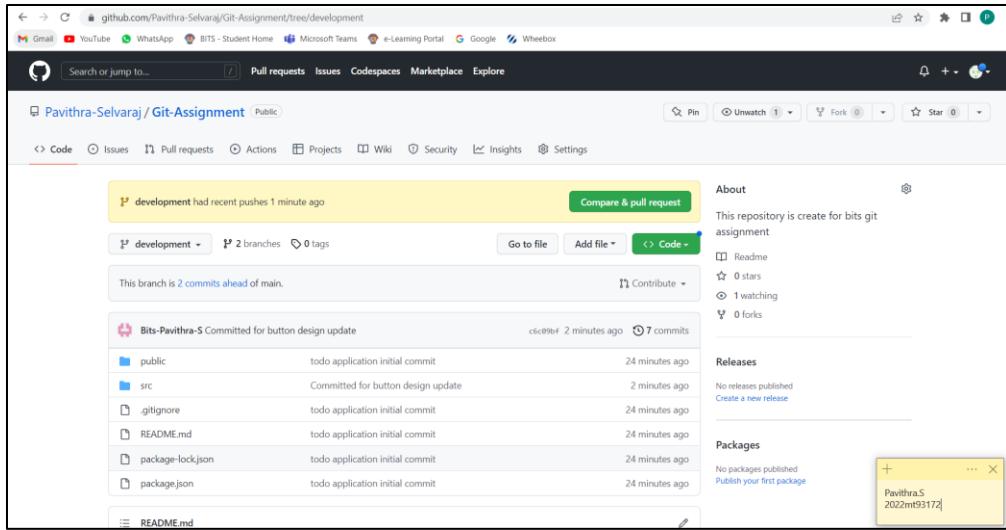
```
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ git add .
warning: in the working copy of 'src/App.css', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'src/App.js', LF will be replaced by CRLF the next time Git touches it

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ git commit -m "Committed for button design update"
[development c6c09bf] Committed for button design update
 2 files changed, 5 insertions(+), 1 deletion(-)

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ git push origin development --force
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 4 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 523 bytes | 261.00 KiB/s, done.
Total 5 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To https://github.com/Pavithra-Selvaraj/Git-Assignment.git
 834d56e..c6c09bf development -> development

DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$
```

The code changes are updated in the origin.



Let's now assume that the code change we just pushed is not necessary and need to be removed.

That can be done by using the command,

```
git reset
```

git reset is a powerful command that is used to undo local changes to the state of a Git repo.

Git reset operates on "The Three Trees of Git".

These trees are the Commit History (HEAD), the Staging Index, and the Working Directory. There are three command line options that correspond to the three trees.

To undo the last commit locally, we can use,

```
git reset HEAD^
```

To force push the local commit which was reverted to the remote git repository, we can use,

```
git push origin +HEAD
```

Let's now run the commands in gitbash from the branch in which we need to remove the last commit.

```
MINGW64:/d/Git Assignment/CollaboratorDirectory/Git-Assignment
834d56e..c6c09bf  development -> development
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ git reset HEAD^
Unstaged changes after reset:
M      src/App.css
M      src/App.js
DELL@DESKTOP-14KB05N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment
```

```
DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment
$ git push origin +HEAD
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Pavithra-Selvaraj/Git-Assignment.git
  + c6c09bf...834d56e HEAD -> development (forced update)

DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment
```

The last commit is now removed.

Pavithra-Selvaraj / Git-Assignment (Public)

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

development had recent pushes 2 minutes ago

development 2 branches 0 tags

This branch is 1 commit ahead of main.

Bits-Pavithra-S todo application initial commit 834d56e 36 minutes ago 6 commits

- public todo application initial commit 36 minutes ago
- src todo application initial commit 36 minutes ago
- .gitignore todo application initial commit 36 minutes ago
- README.md todo application initial commit 36 minutes ago
- package-lock.json todo application initial commit 36 minutes ago
- package.json todo application initial commit 36 minutes ago

README.md

About

This repository is created for bits git assignment

Readme 0 stars 1 watching 0 forks

Releases

No releases published Create a new release

Packages

No packages published Publish your first package

The changes are now reverted to our working directory.

```
DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment
$ git status
On branch development
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:   src/App.css
      modified:   src/App.js

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

DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment
```

## Creating Tags

Git tags are used as reference points in our development workflow.

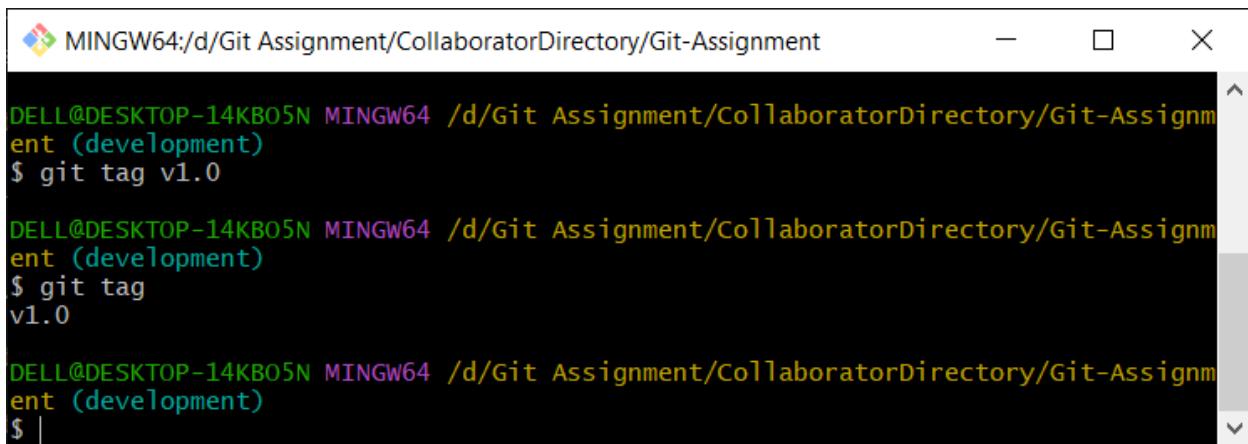
We create new Git tags in order to have a reference to a given release of our software.

Let's create a tag named v1.0 for our todo application.

In order to create a new tag, we can use the command,

```
git tag <tag-name>
```

To check if the tag is created successfully, we can use the “git tag” which will list all available tags.



MINGW64:/d/Git Assignment/CollaboratorDirectory/Git-Assignment

```
DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ git tag v1.0

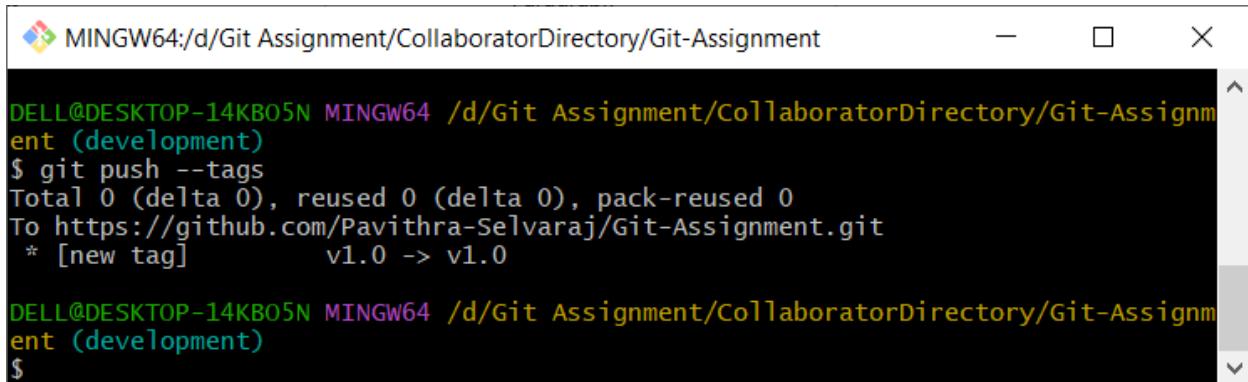
DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ git tag
v1.0

DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ |
```

The “git push” command does not automatically push our tags to the remote repository.

We need to explicitly mention as follows,

```
git push--tags
```

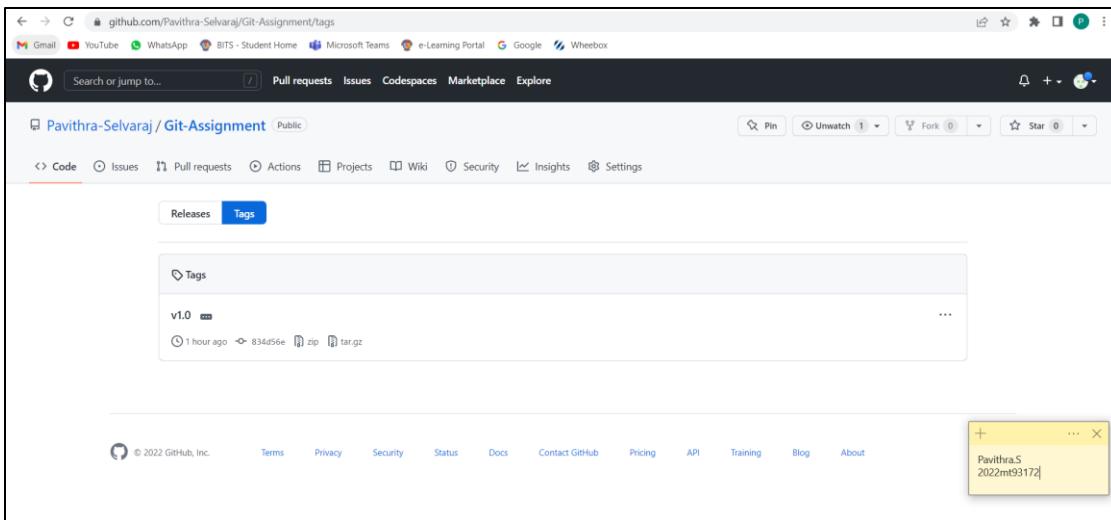


MINGW64:/d/Git Assignment/CollaboratorDirectory/Git-Assignment

```
DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ git push --tags
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Pavithra-Selvaraj/Git-Assignment.git
 * [new tag]           v1.0 -> v1.0

DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/CollaboratorDirectory/Git-Assignment (development)
$ |
```

The tag is now pushed to the origin.



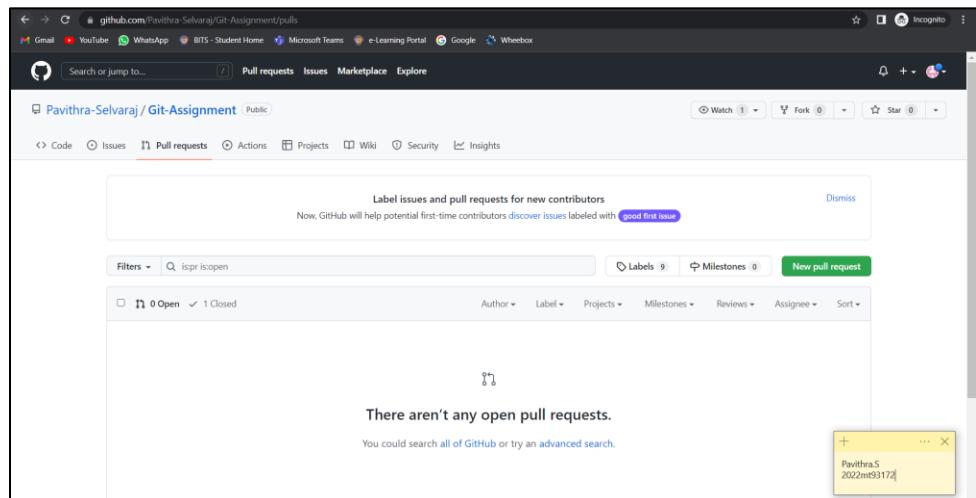
## Releasing Features

In a project, to release the features, the code from the development branch will be moved to the production branch in steps. Some projects may have multiple environments between development and production such as qa, staging, uat etc. Once the code is moved to the production branch which in some cases can be the master/main branch, the main production release will be done.

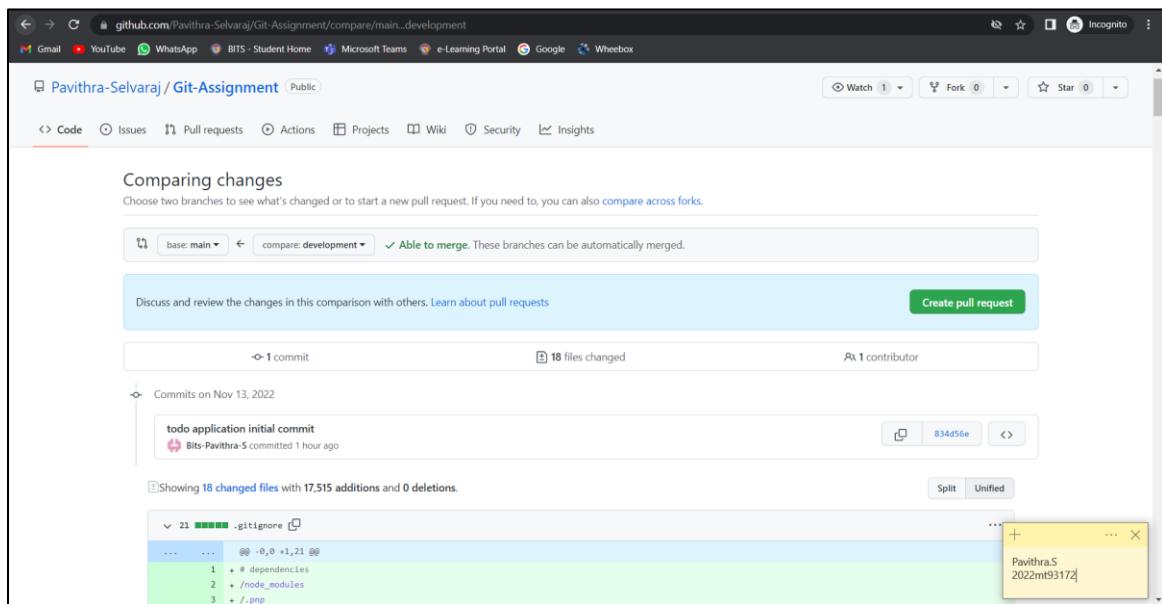
In our example here, lets consider that we are going to merge the changes from development branch to the main branch (which is the master branch in our case).

### Pull Request:

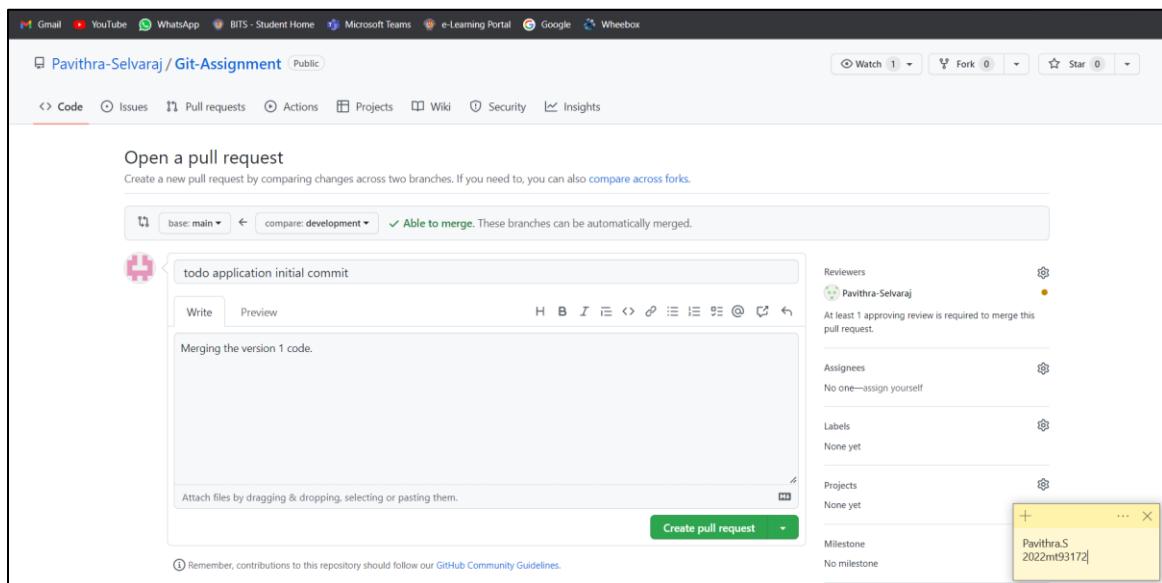
1. Click on “Pull Requests” tab in the navigation.



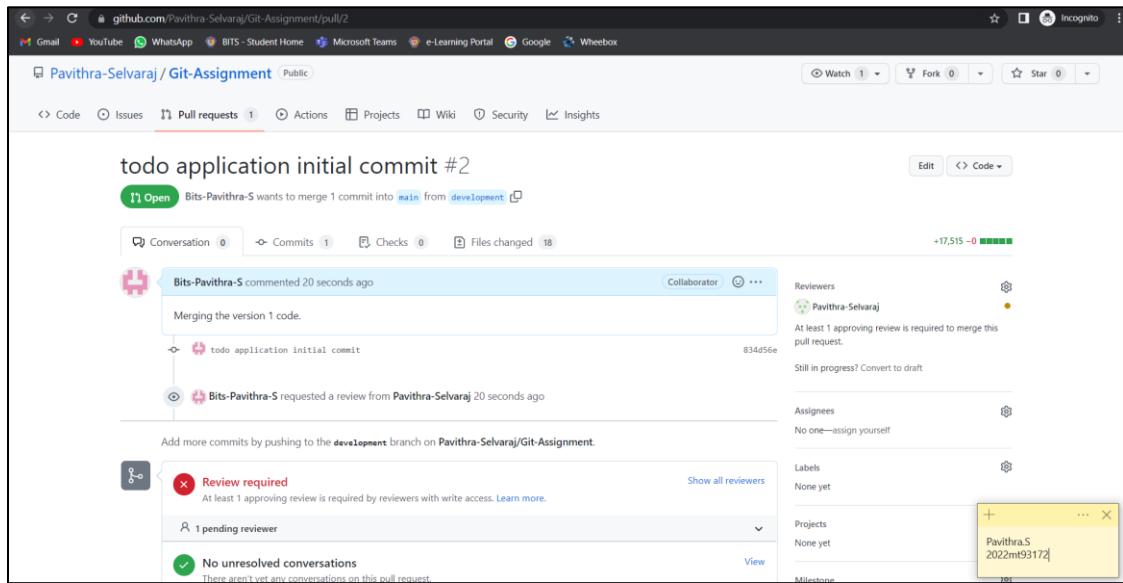
2. Click “New pull request” button. Select the branches to compare. Check the commits and click “Create pull request”.



3. Give the pull request message and add reviewer and click “Create pull request”.



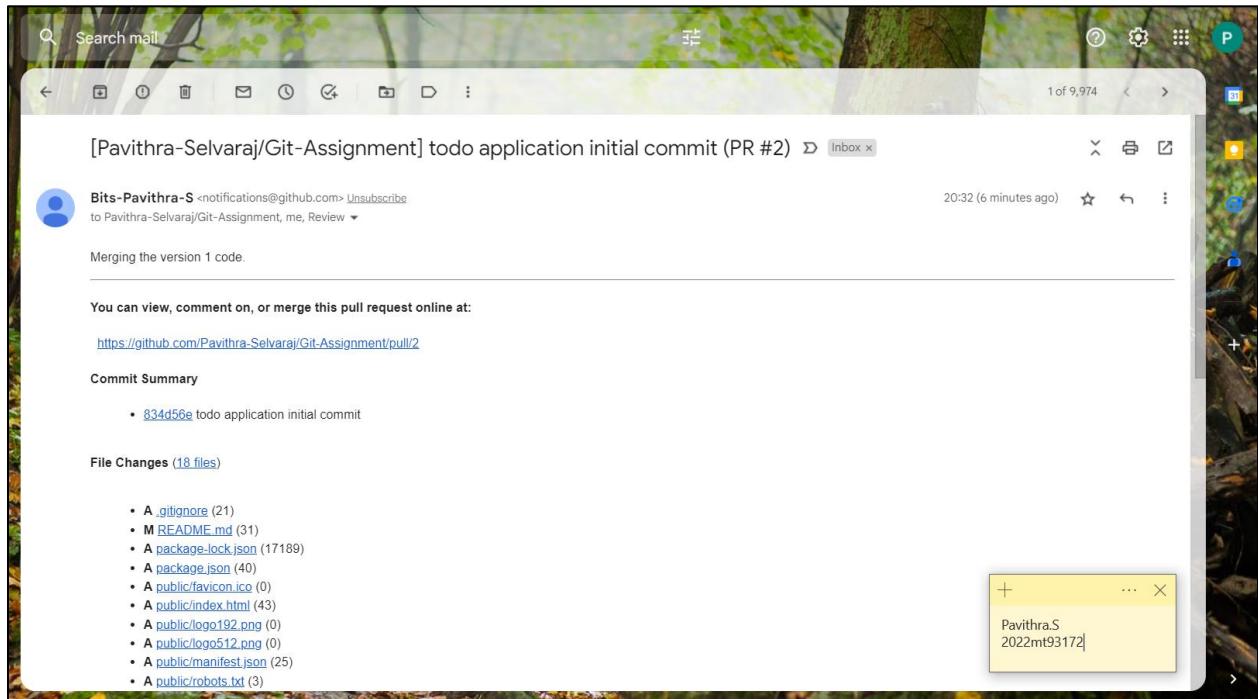
#### 4. The pull request is created.

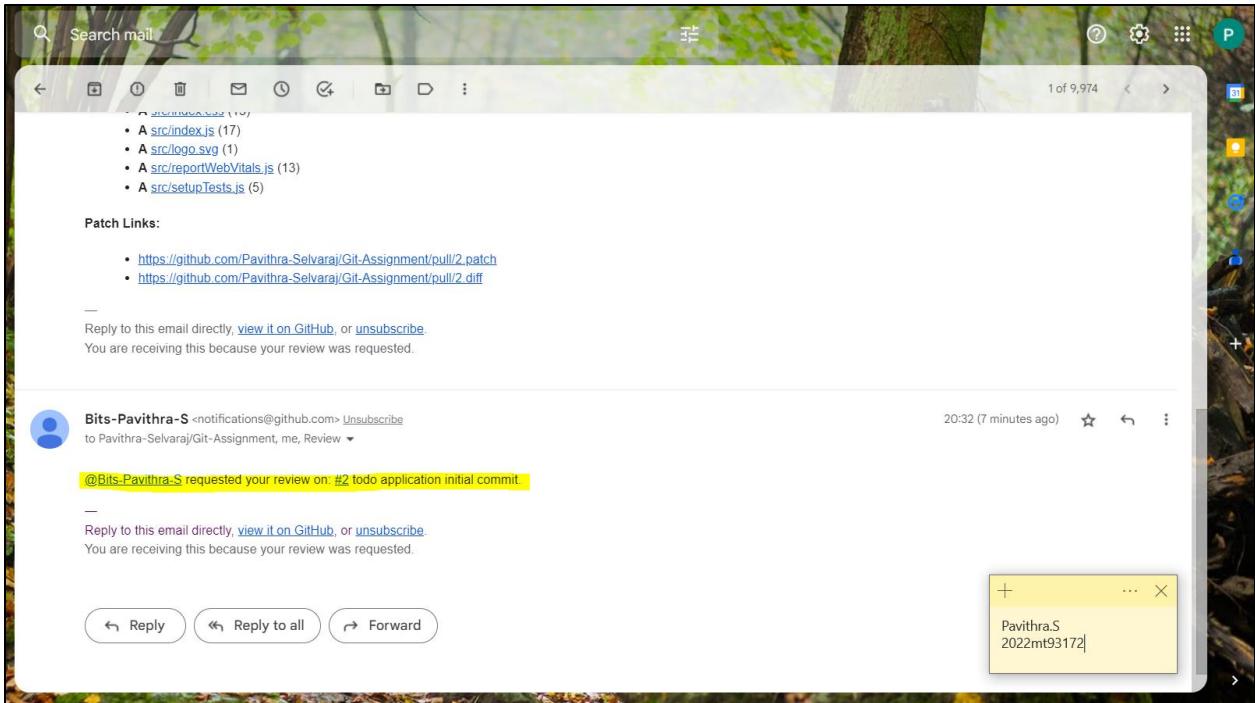


The review request is now sent to the manager (the person chosen as reviewer).

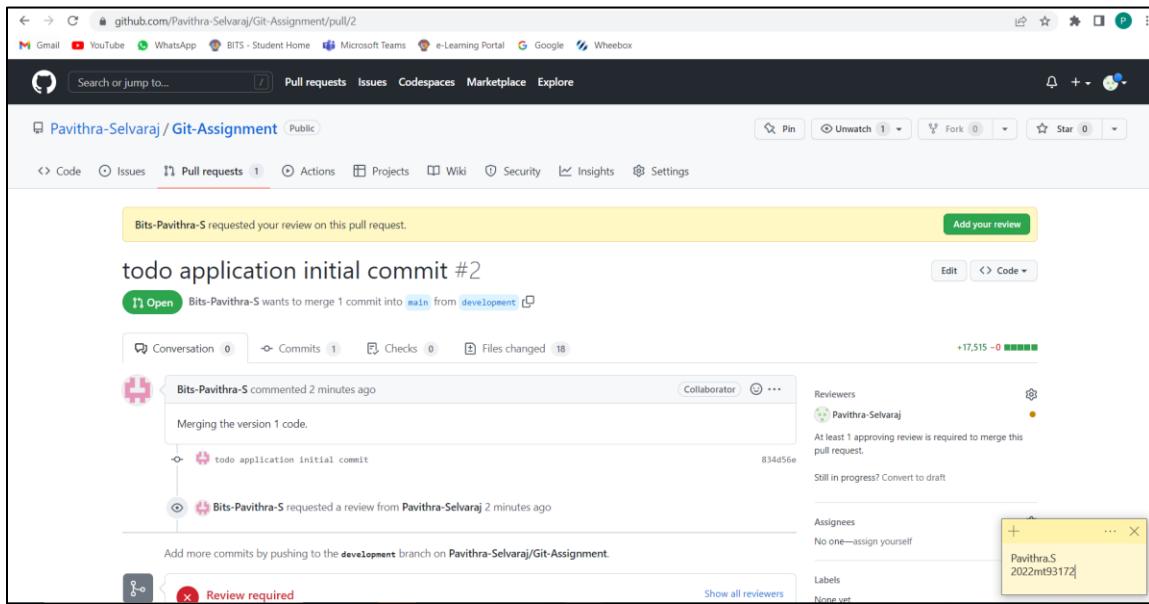
### Request Approval

An email is sent to the reviewer as we had done the setup earlier.





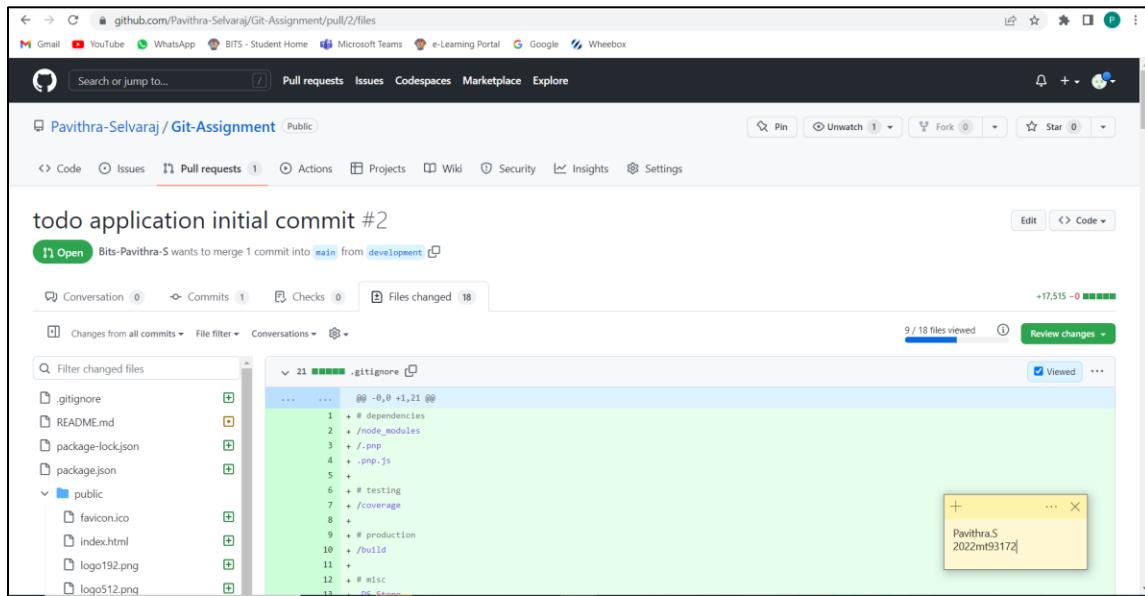
The reviewer can also see the request in the pull request tab.



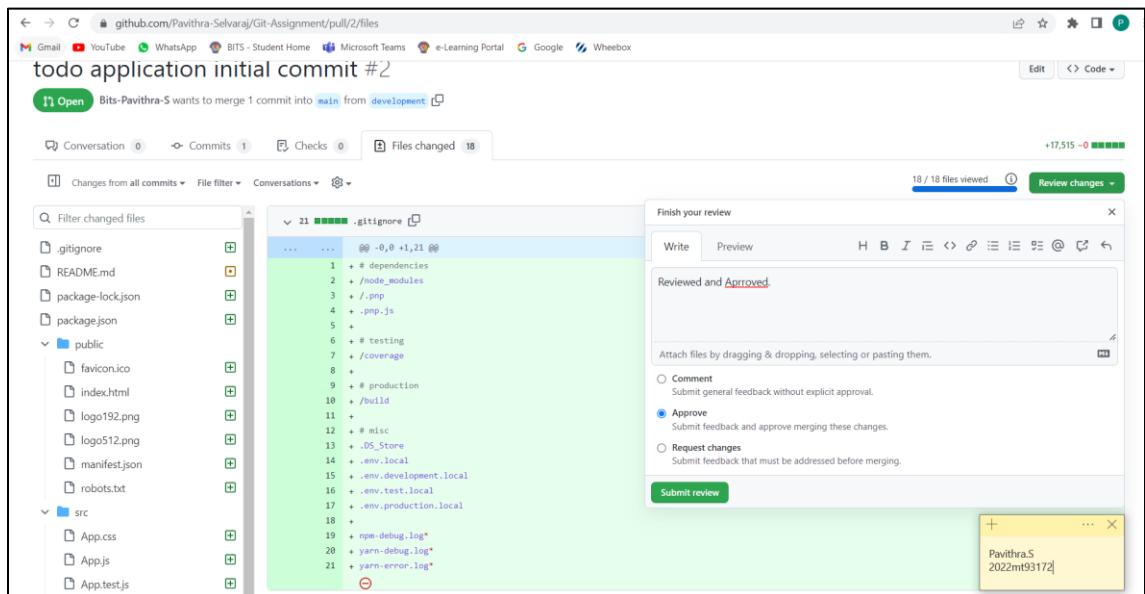
The reviewer can add review and approve the code once reviewed. If there are changes required, comments can be added and the code can be merged after the comments are resolved.

Follow the steps below to review and merge code.

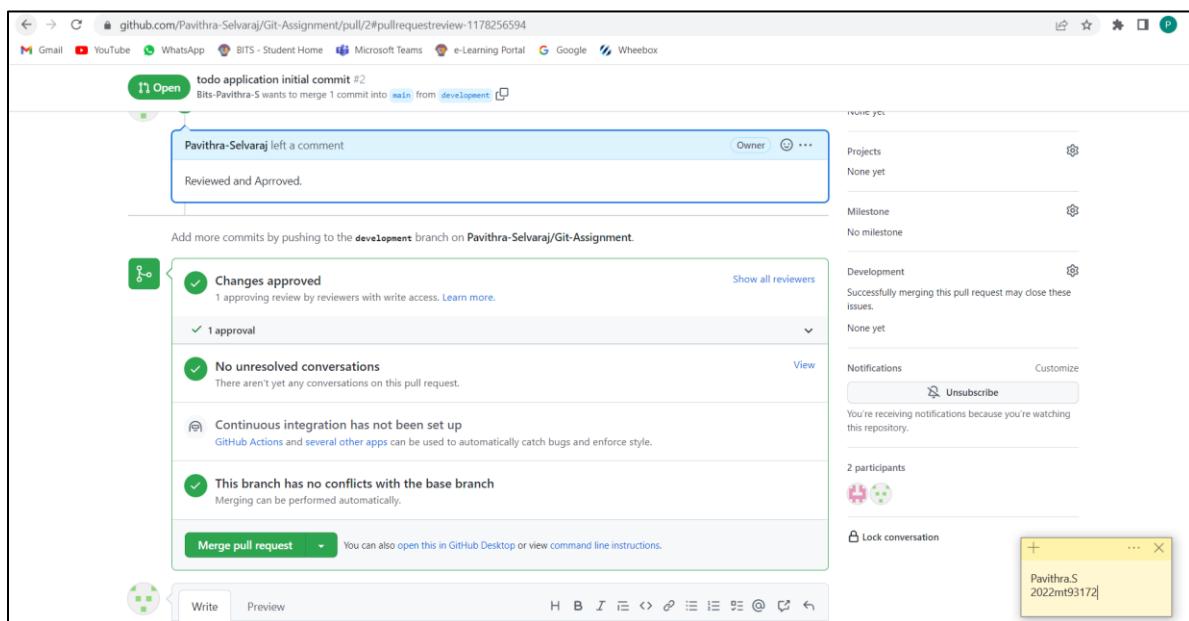
1. Click on “Add your review”.



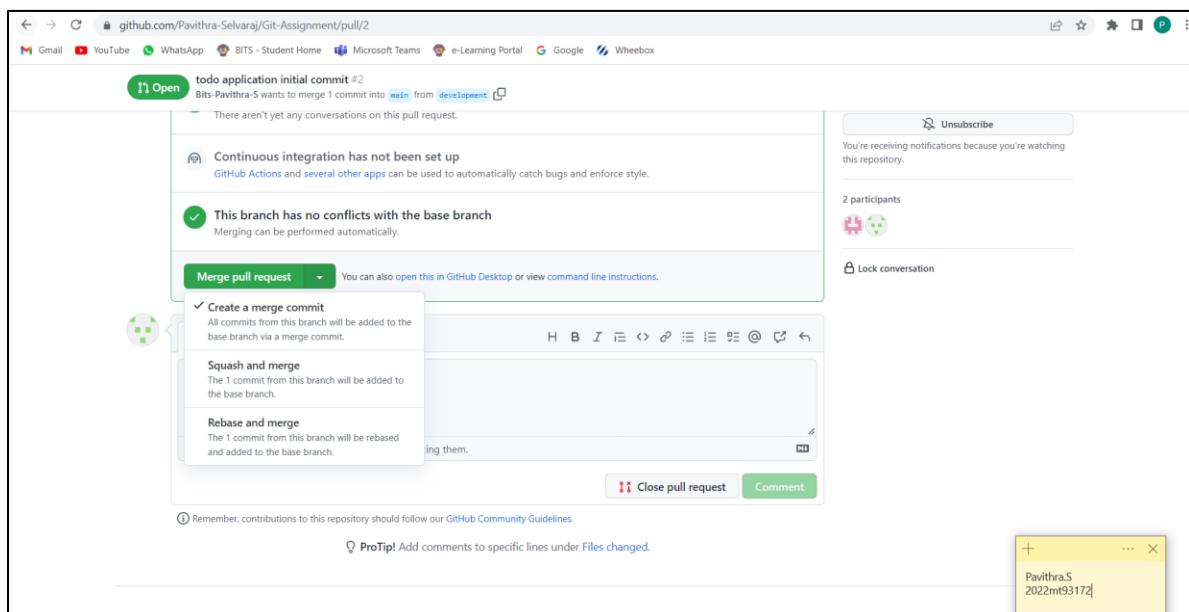
2. Review the changes and add comments if any. Else check all and click “Review changes”.



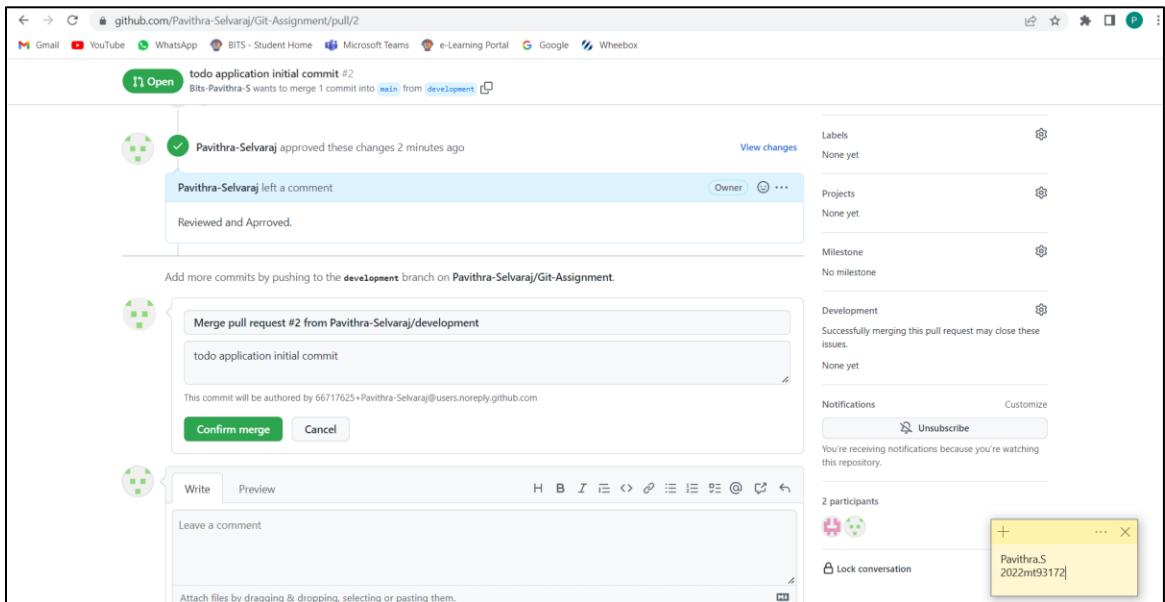
3. Click “Submit Review” with the option that suits. Here lets directly select approve and click the button.



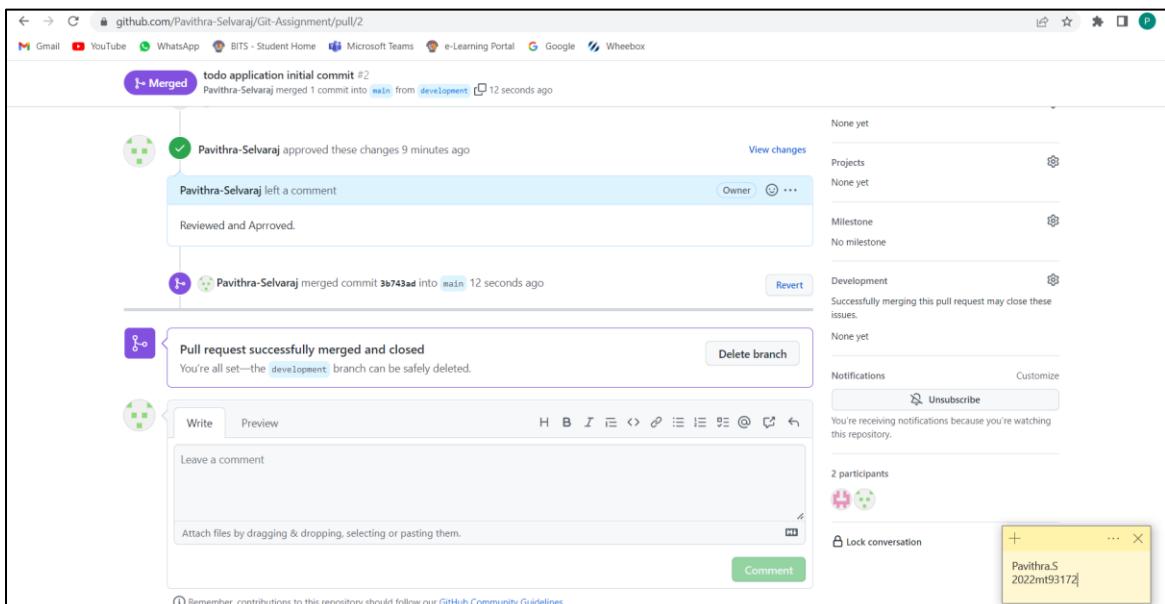
4. The code can be now merged in any one of the ways below. Select an option and click on “Merge pull request” button.



## 5. Click “Confirm merge”.



## 6. The code is successfully merged.

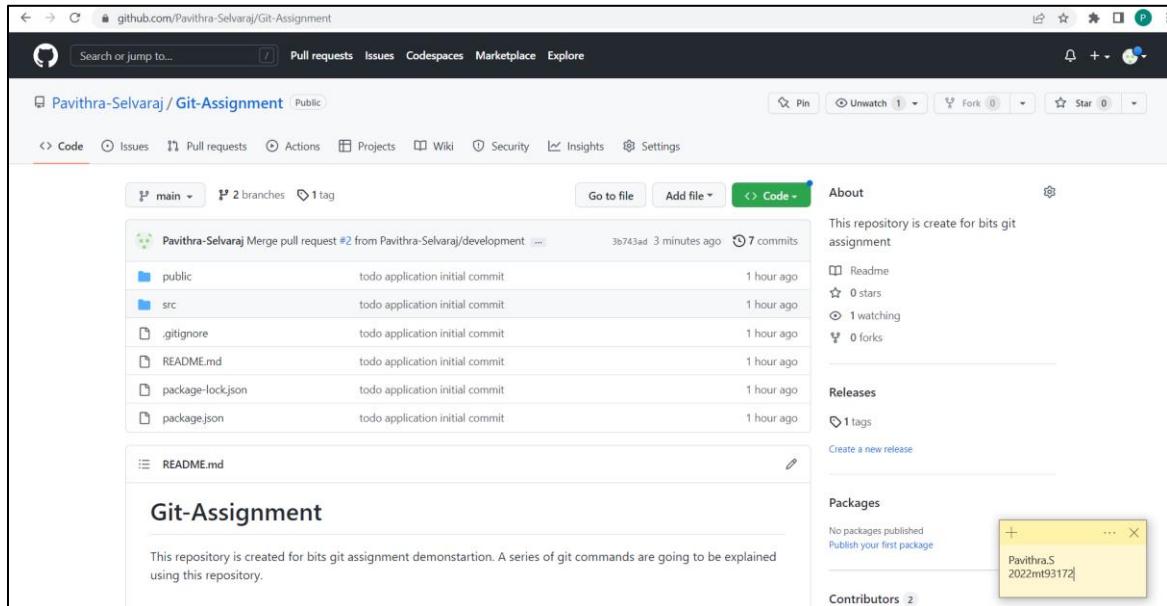


## Creating a Release

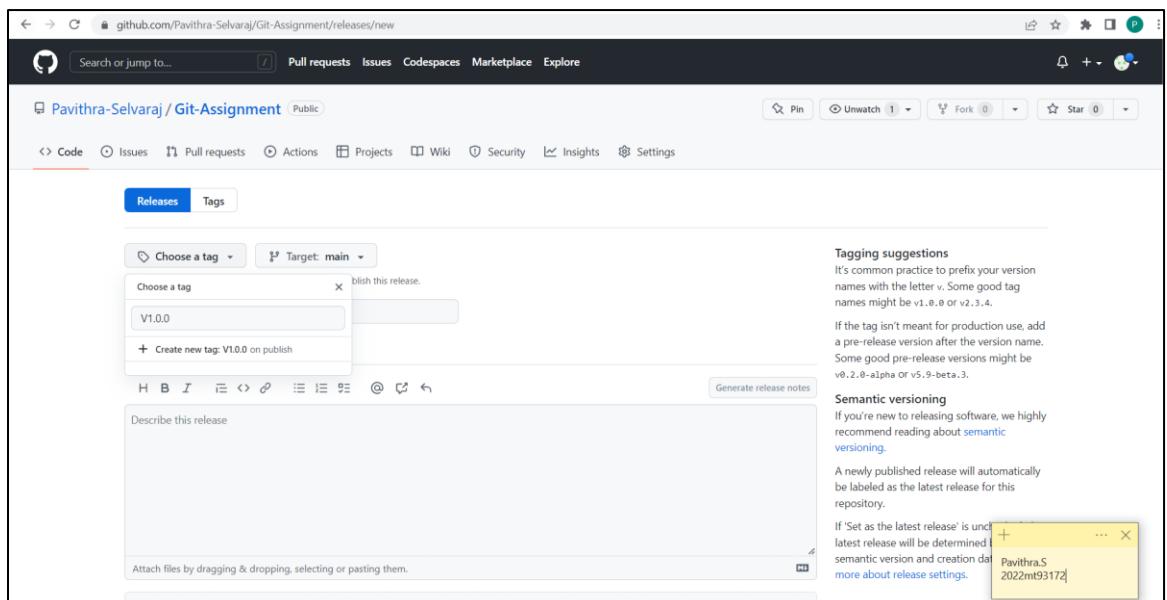
We can make a release using the release module in GitHub.

Follow the steps below,

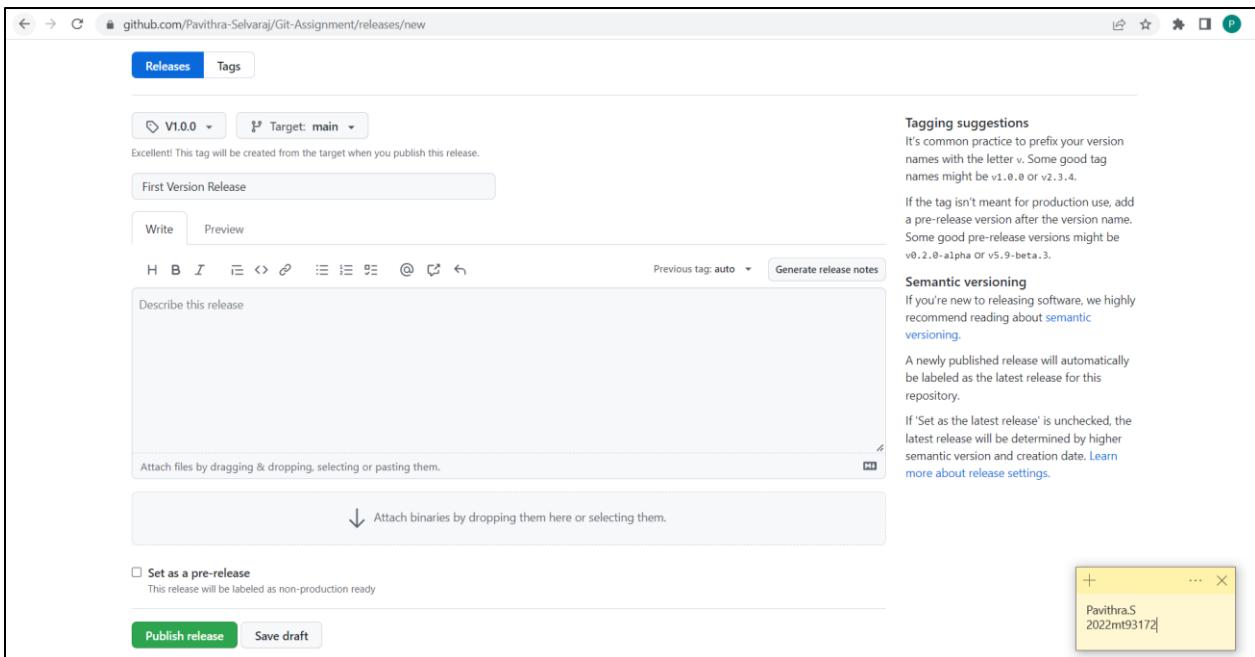
1. Click on “Create a new release” in the right option pane.



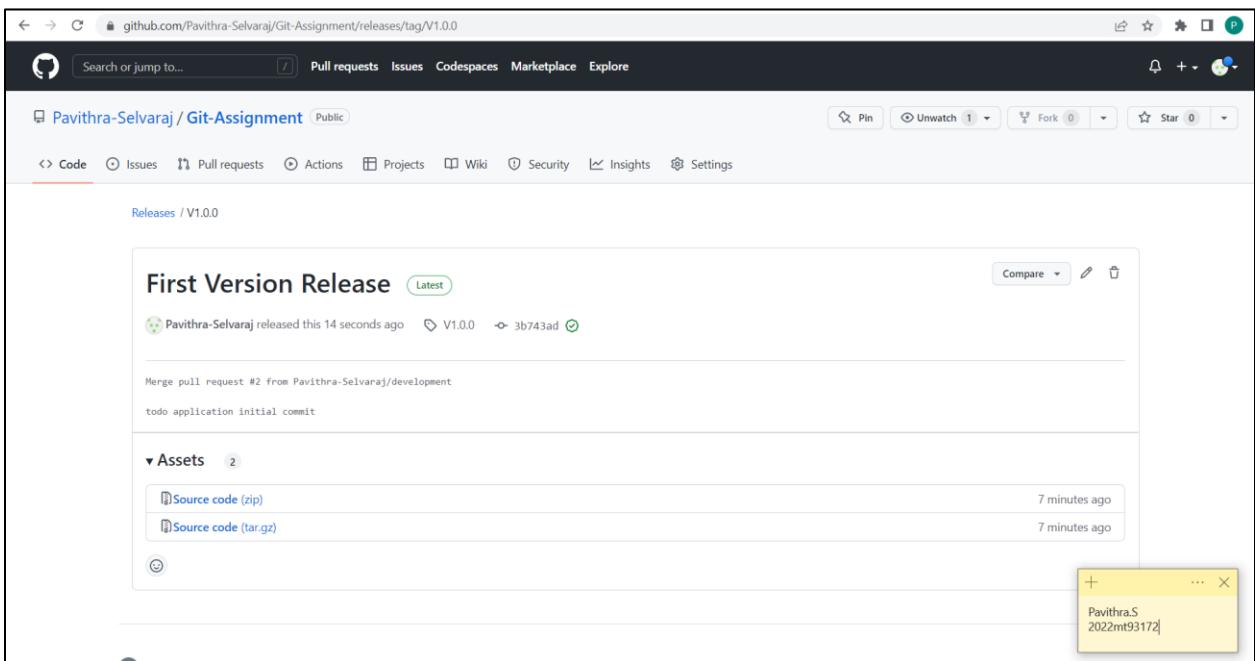
2. Click on choose a tag and select a new tag or an existing one.



3. Add the other details also and click on “Publish Release”.



This will create the first published version of our code.



## CONCLUSION

In this assignment, with the help of one manager and one collaborator account, I have demonstrated various git workflow related operations.

To summarize, the below topics are covered,

- Git basics and how to use GitHub and gitbash.
- Creating a repository.
- Adding collaborators to a repository.
- Adding notification integration for manager.
- Cloning a repository.
- Creating a branch.
- Committing the code in local repository.
- Pushing the code to the origin.
- Creating and approving pull requests.
- Deleting a branch in local and in remote repository.
- Resolving a merge conflict.
- Need of `readme.md` and `.gitignore` files.
- Force push and reset commit process.
- Creating tags and pushing them to origin.
- Releasing features with versions.