Work Integrated Learning Programmes

M.Tech Software Engineering

Introduction to Devops

Assignment 1 – Git Workflow

Submitted By

Pavithra.S

2022MT93172

INDEX

GIT WORKFLOW

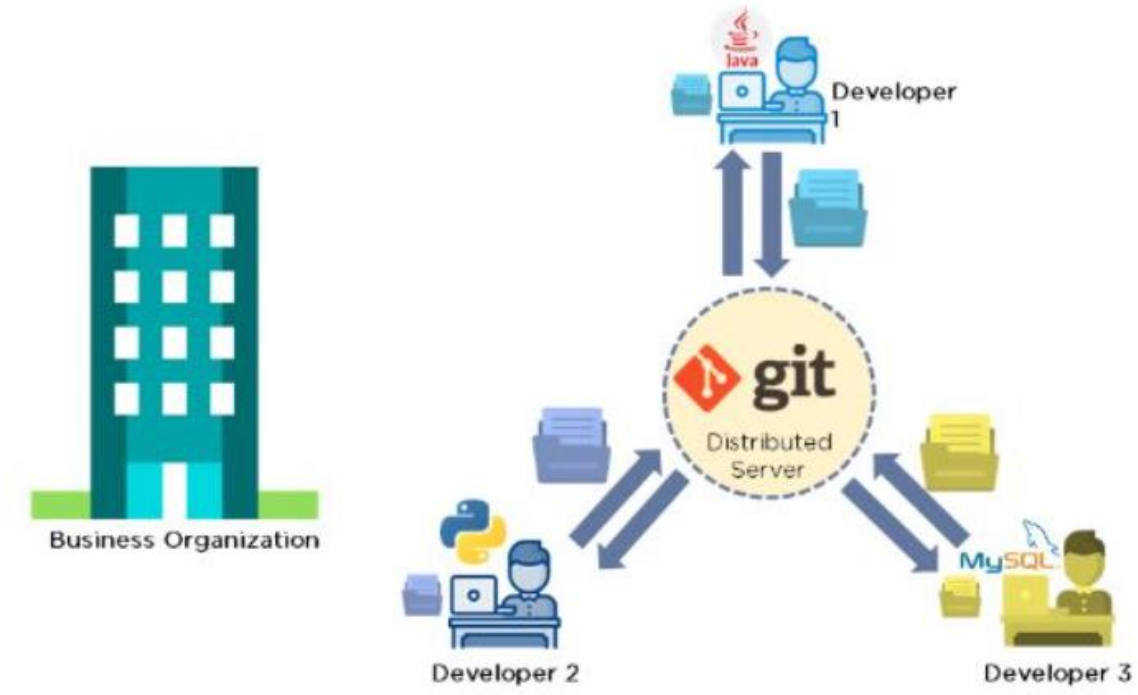
INTRODUCTION

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.

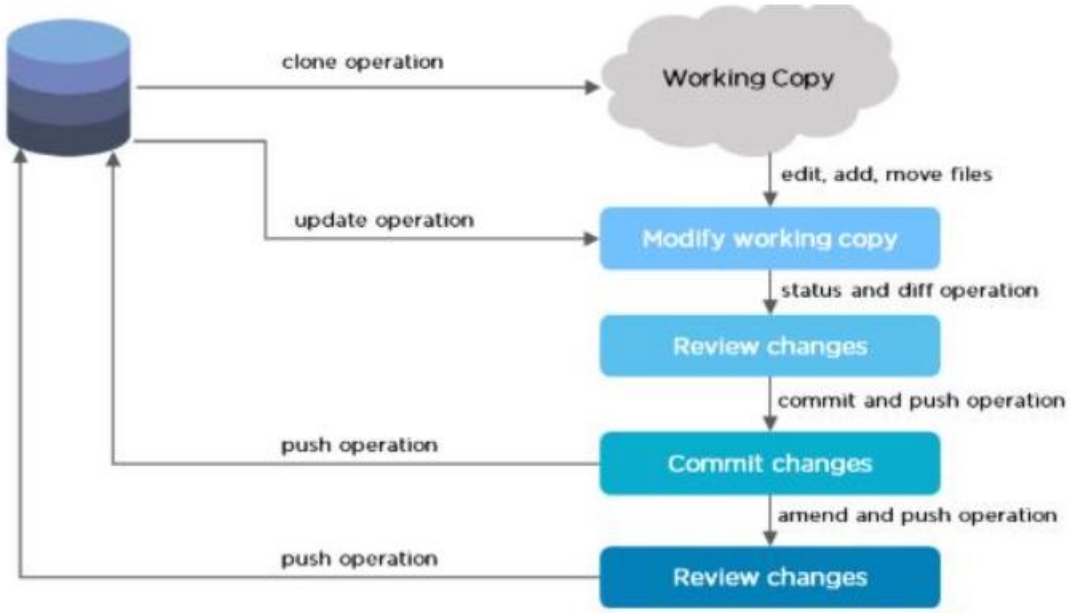
Git is used to tracking changes in the source code, enabling multiple developers to work together on non-linear development. Git makes collaborating on code easier.

Features of Git

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

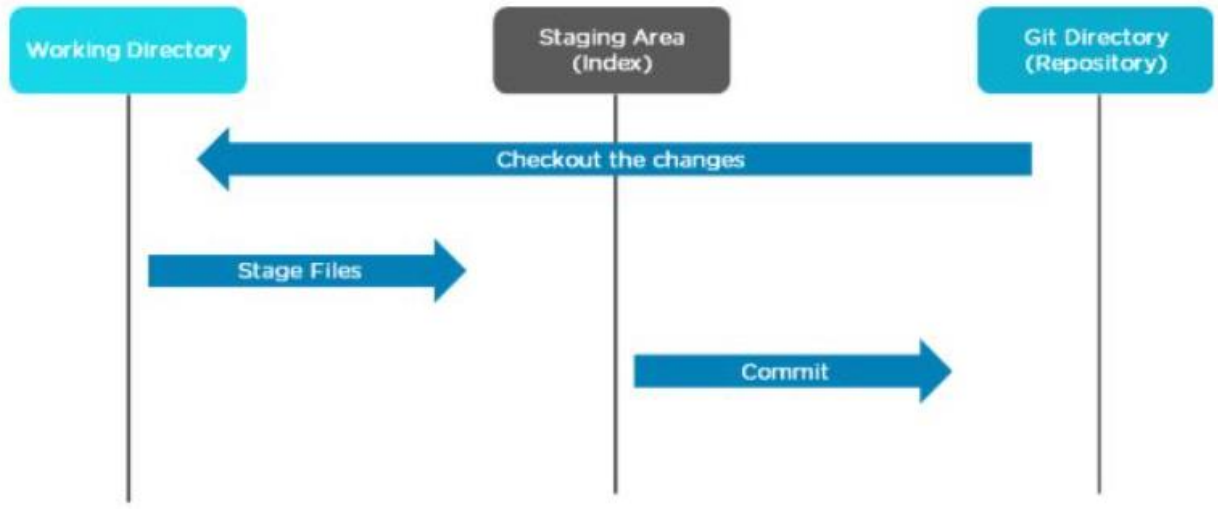


GIT WORKFLOW



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

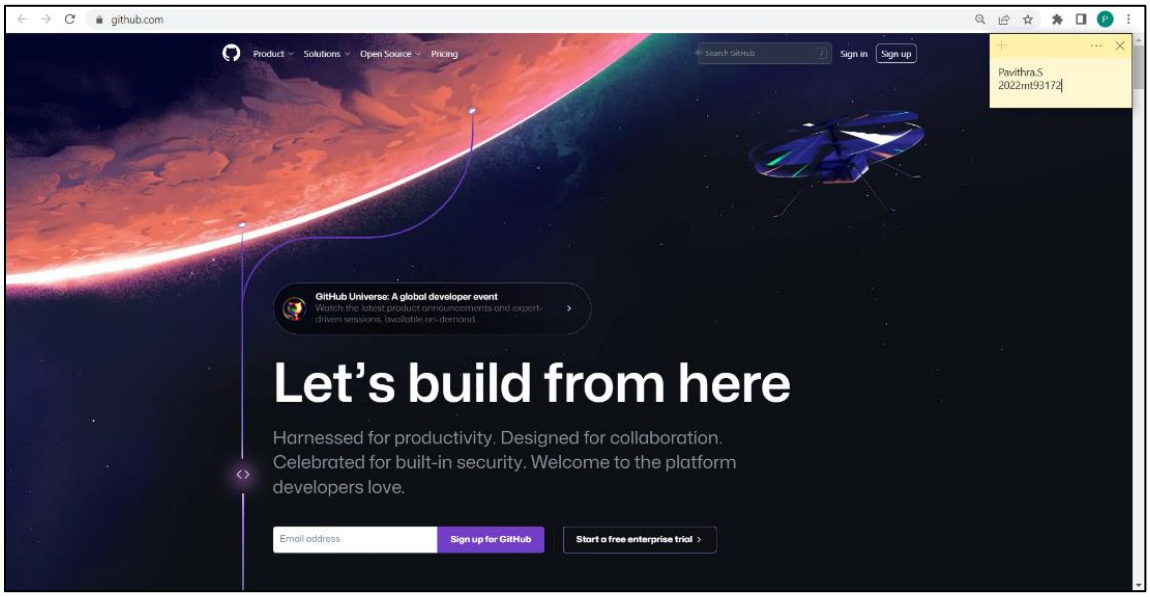
All the code and documentation are in one place. There are millions of repositories on GitHub, and each repository has its own tools to help you host and release code.

Account Creation

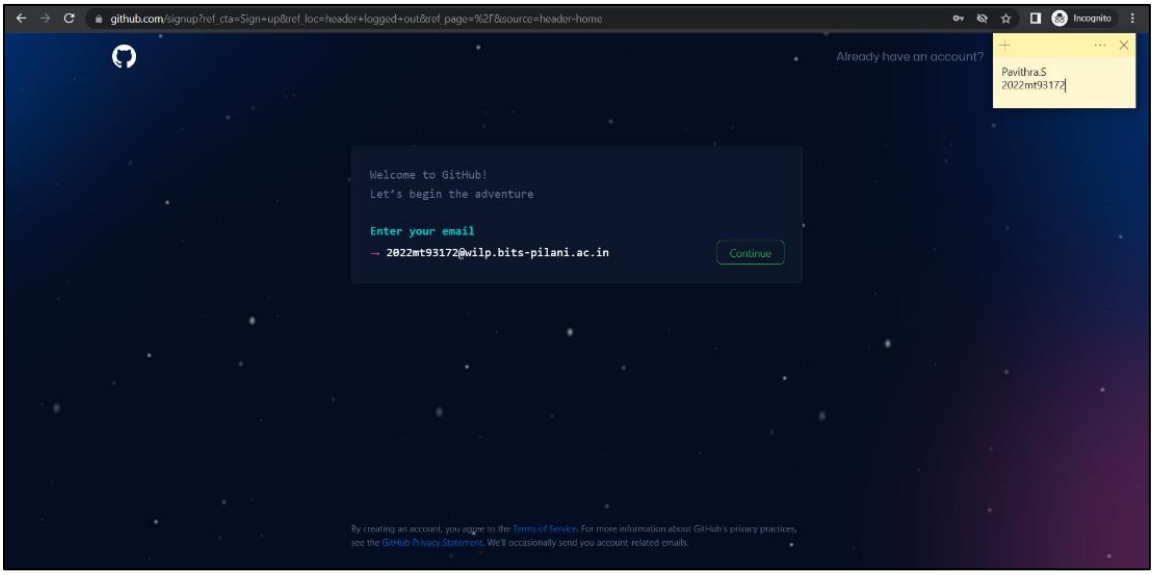
For our git demonstration we are going to need 2 GitHub accounts. One account will be the owner who will act as the manager and the other will be a collaborator or a team member.

Steps to create an account in GitHub are,

1. Visit github.com from chrome or any browser of your choice.

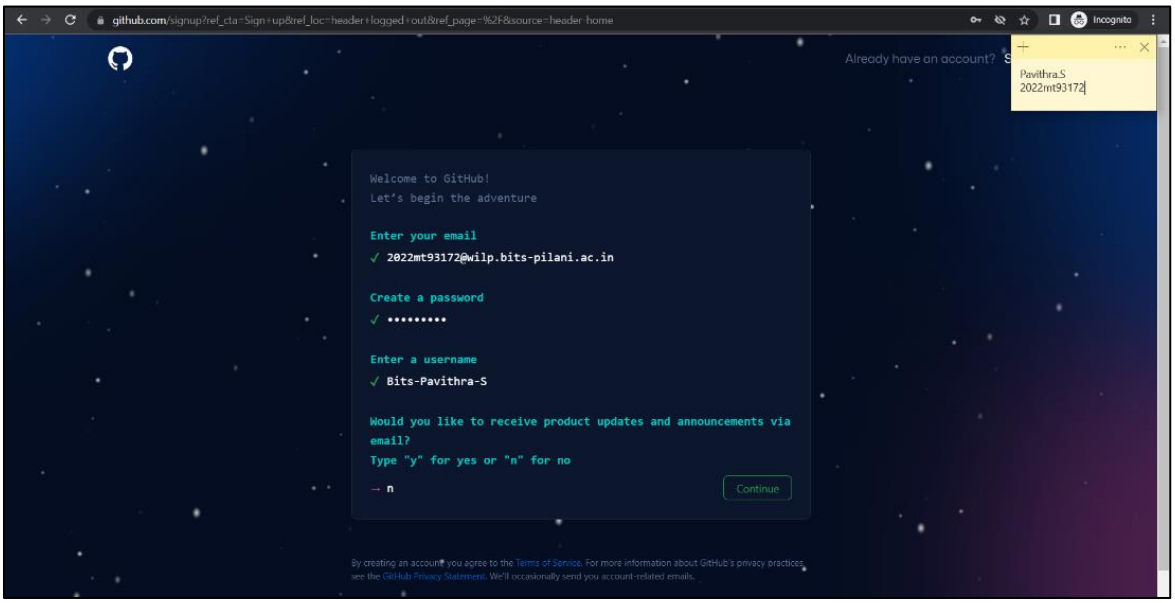


2. Click on the "Sign Up" button.

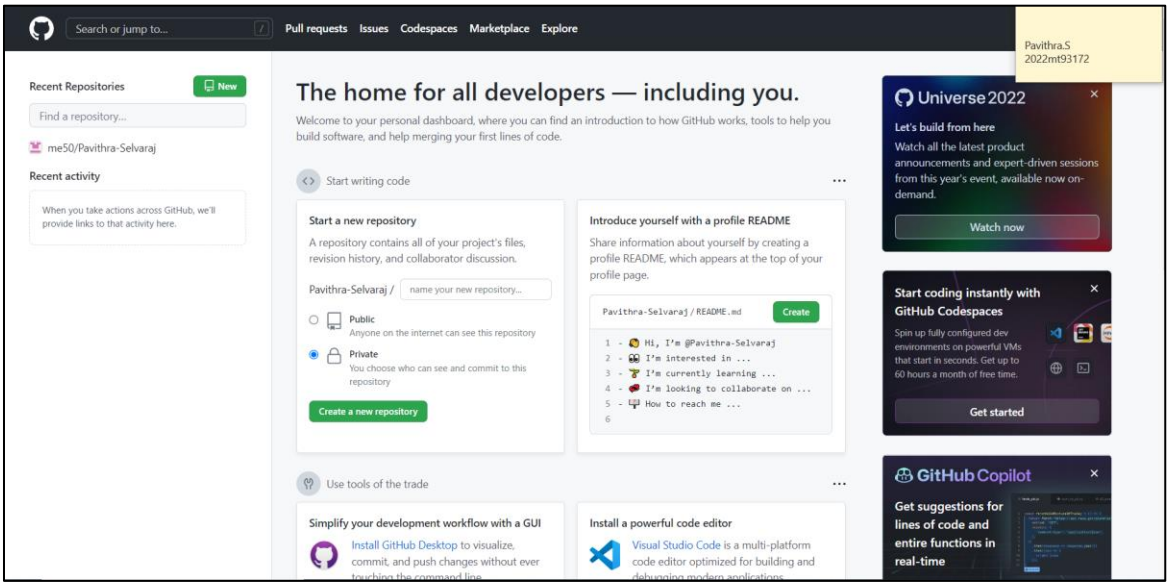


3. Fill the required details such as,
    - Email ID,
    - Username,
    - Password etc.

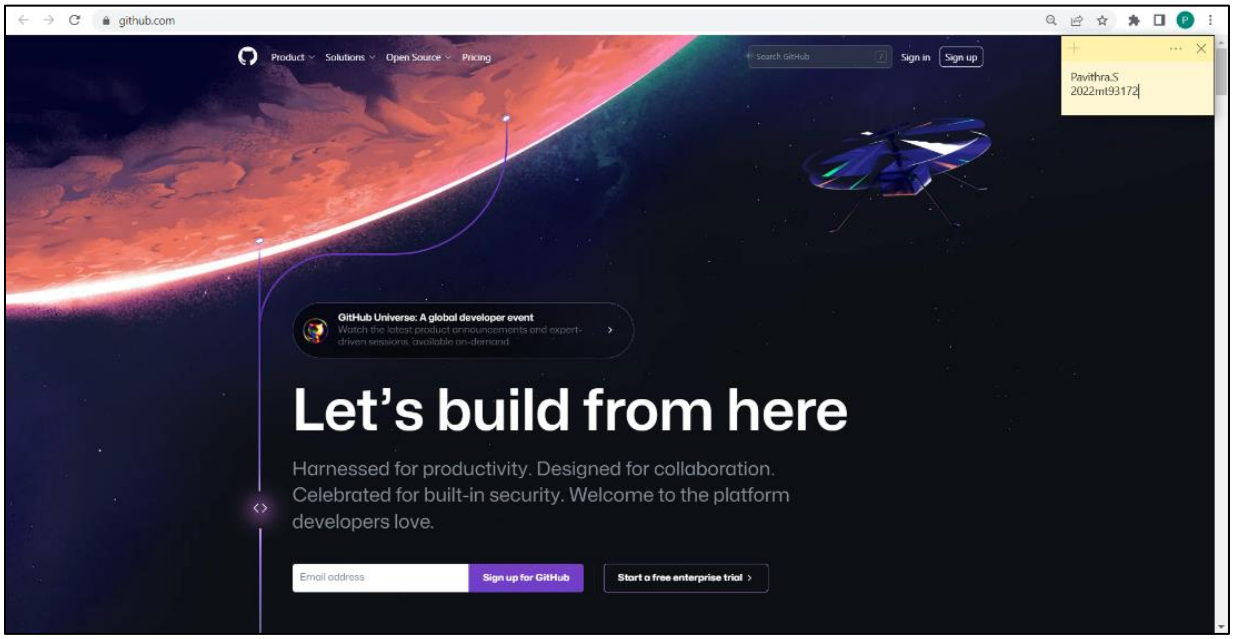
   and click continue.

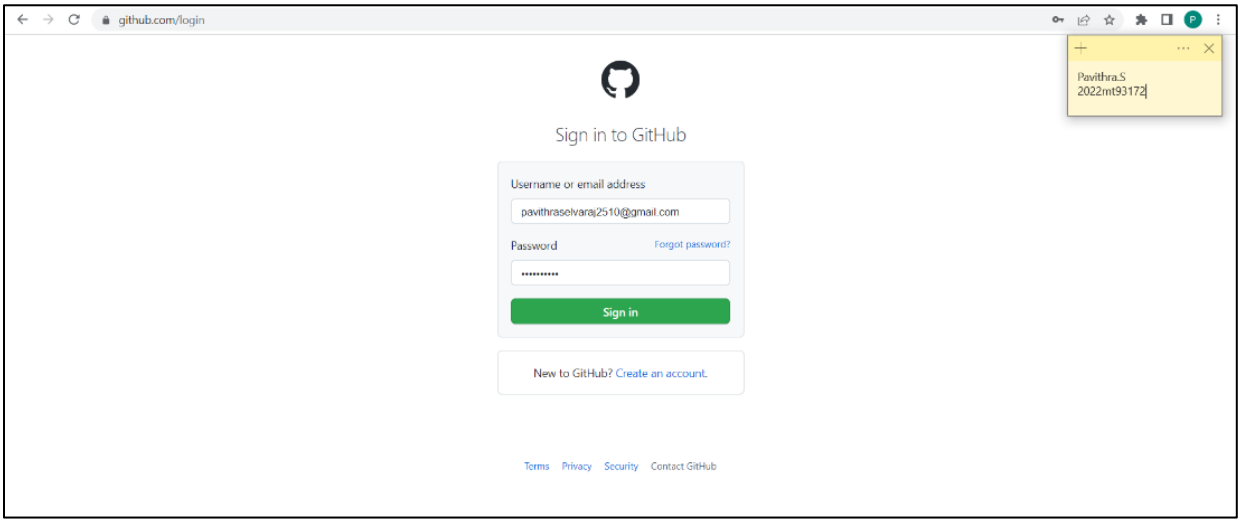


4. The account will then be created and we will be navigated to the landing page.



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.

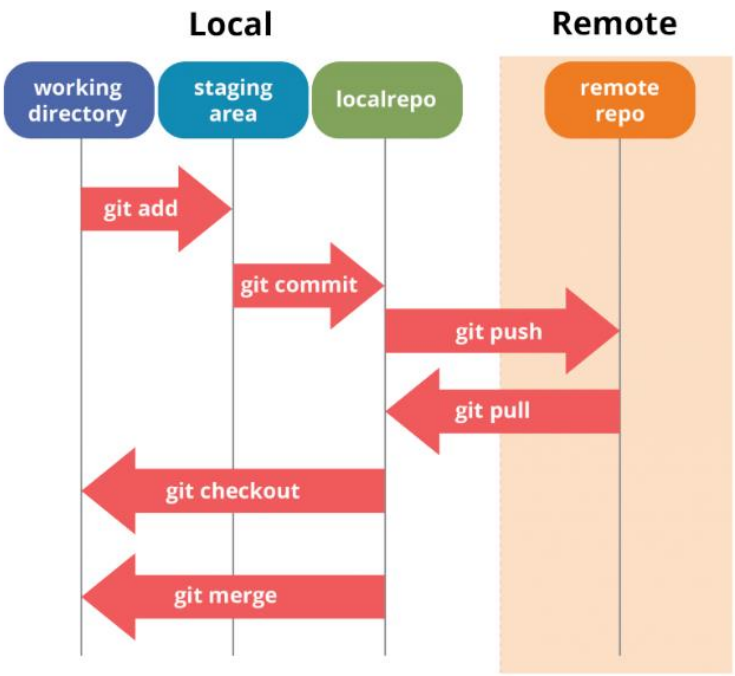
GIT OPERATIONS

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,

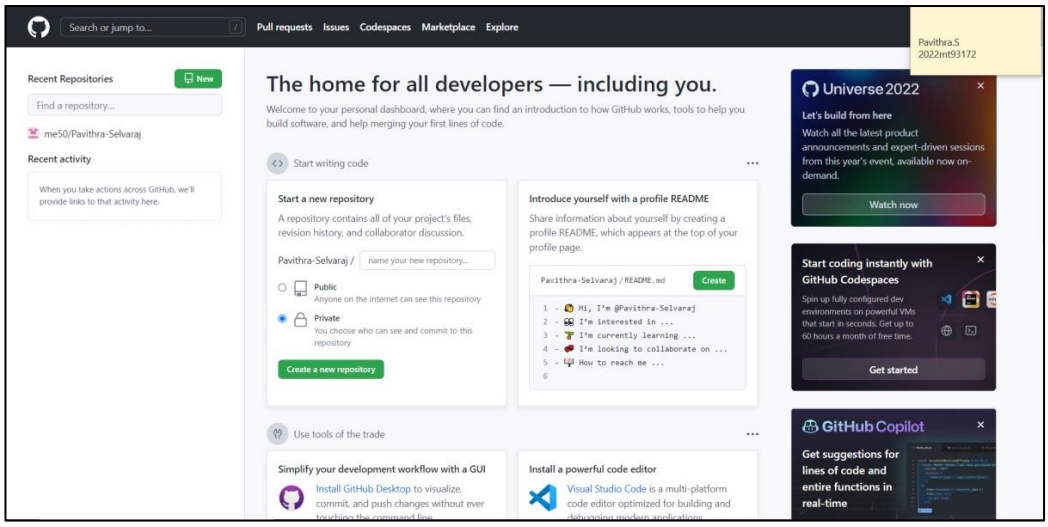
1. Using the command, “git init”

   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.

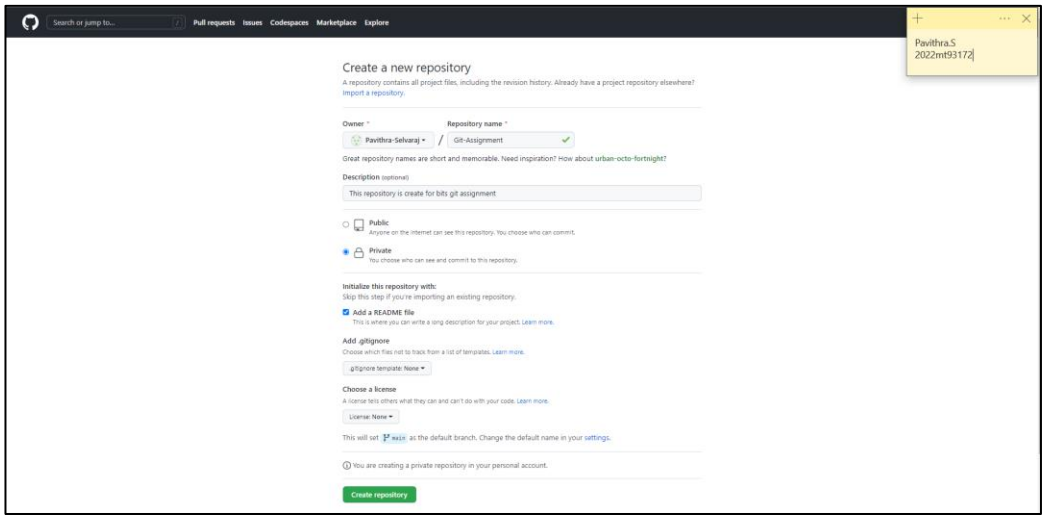
2. From the GitHub portal.

   The steps are as follows,

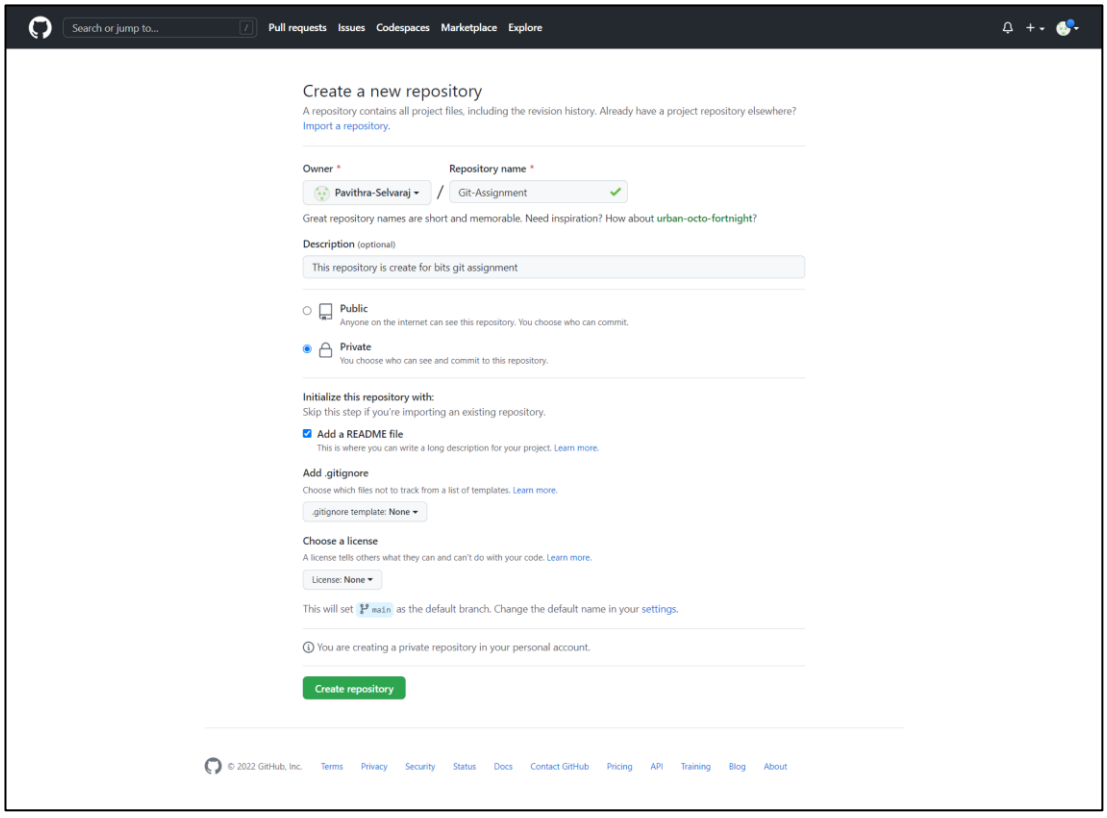
- Click on “Create a new repository” button.



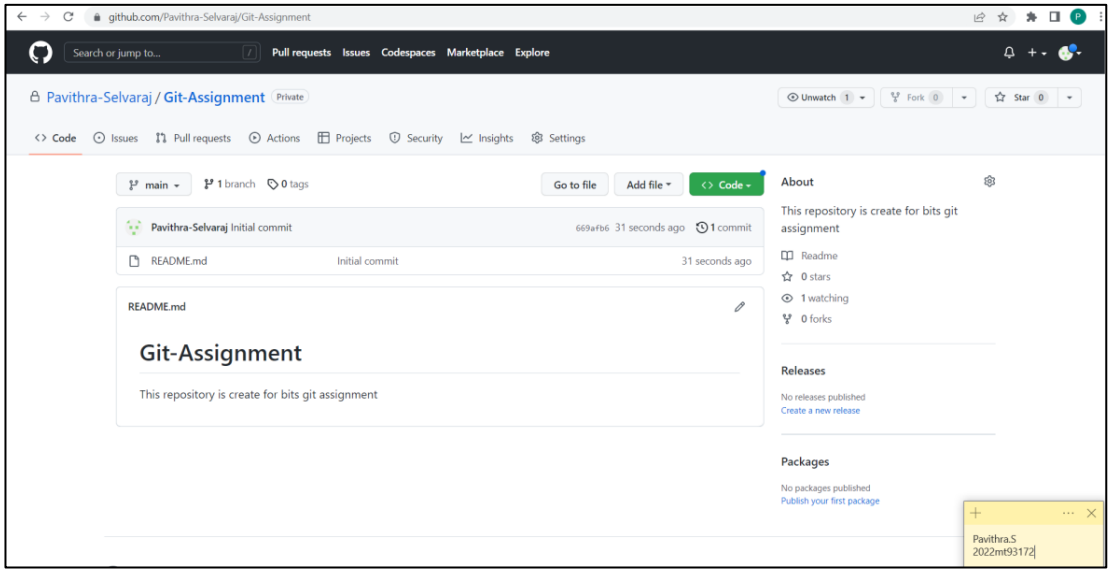
- Fill the required details.



- Click on “Create Repository” button.



- The repo is created.



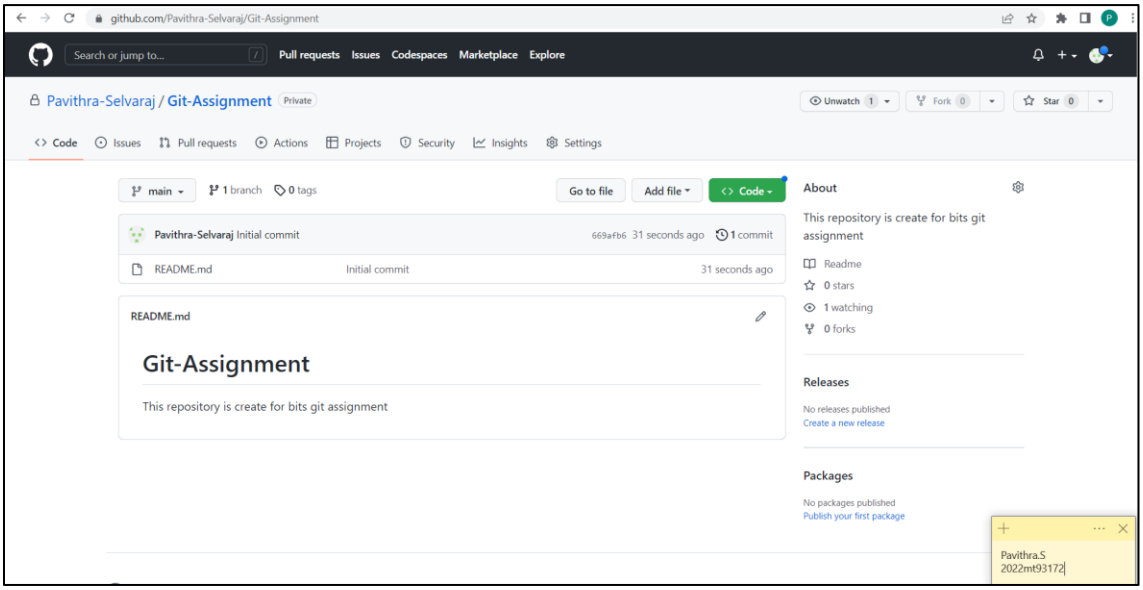
Adding Collaborators

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.

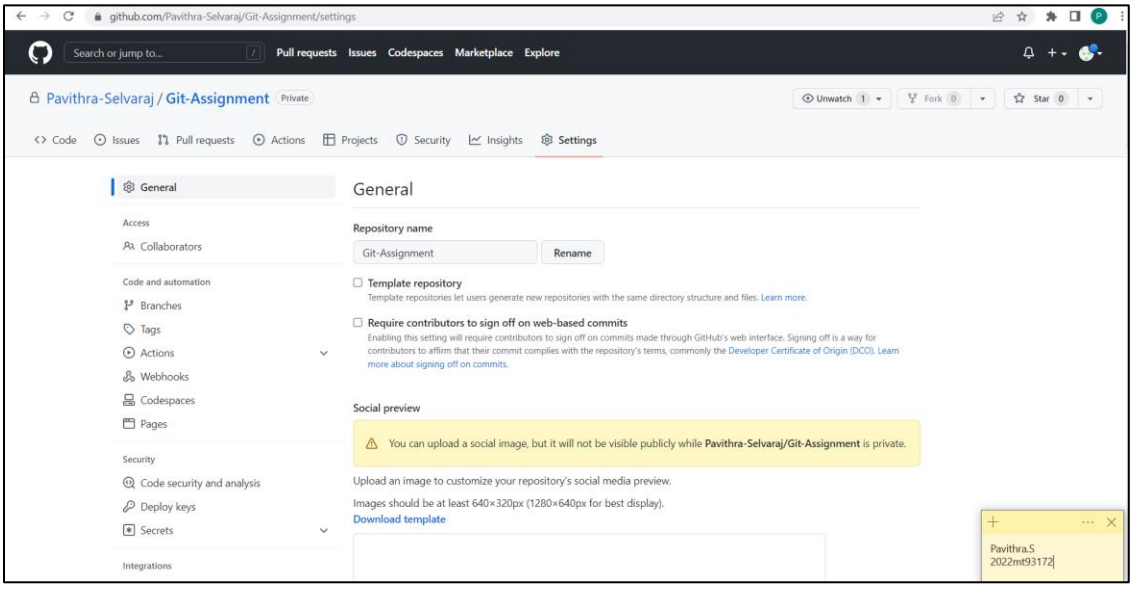
So, the account we are currently using “Pavithra-Selvaraj” is going to act as the manager account and we are going to add the other team members as collaborators.

The steps are as follows,

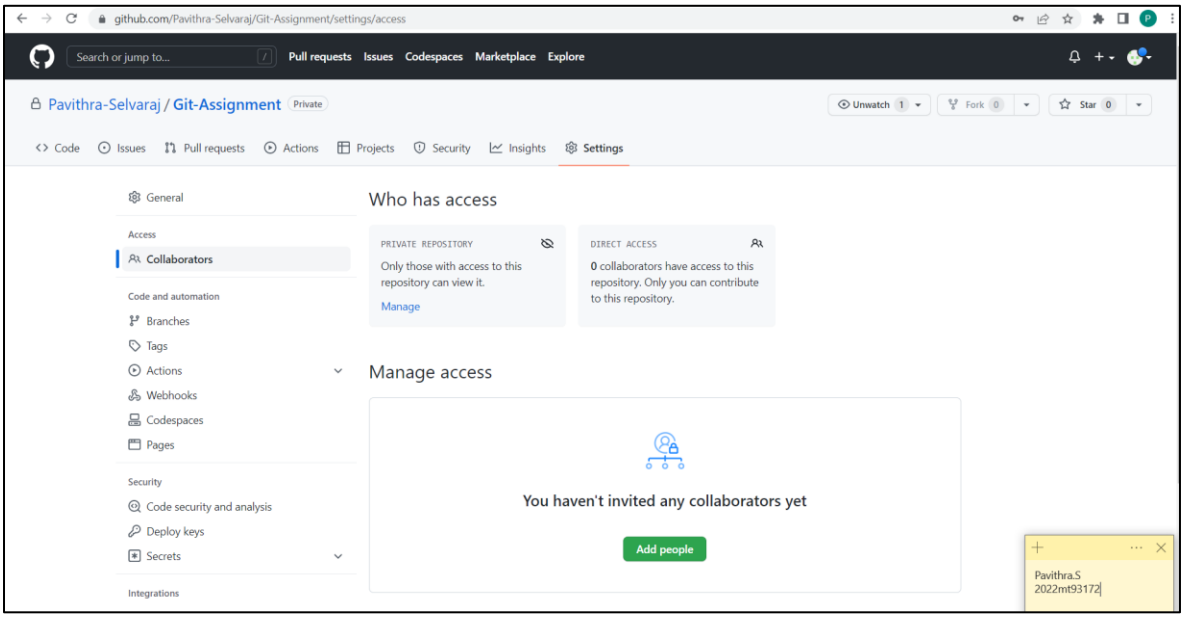
1. Click on “Settings”.



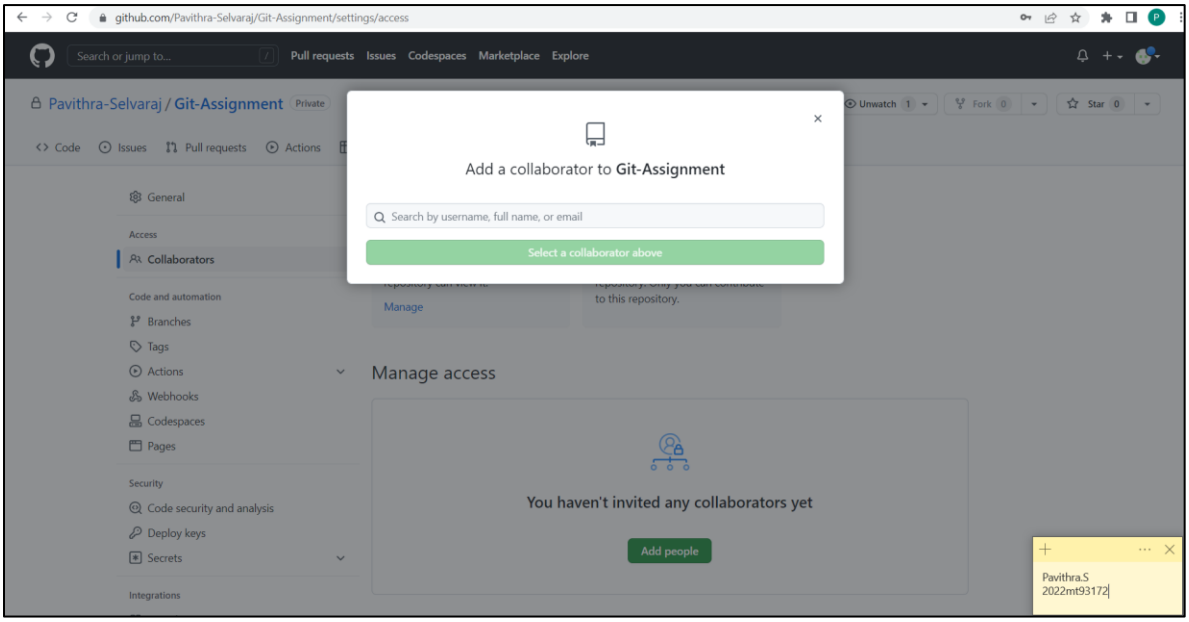
2. Click “Collaborators” option.



3. Click on “Add People” button.

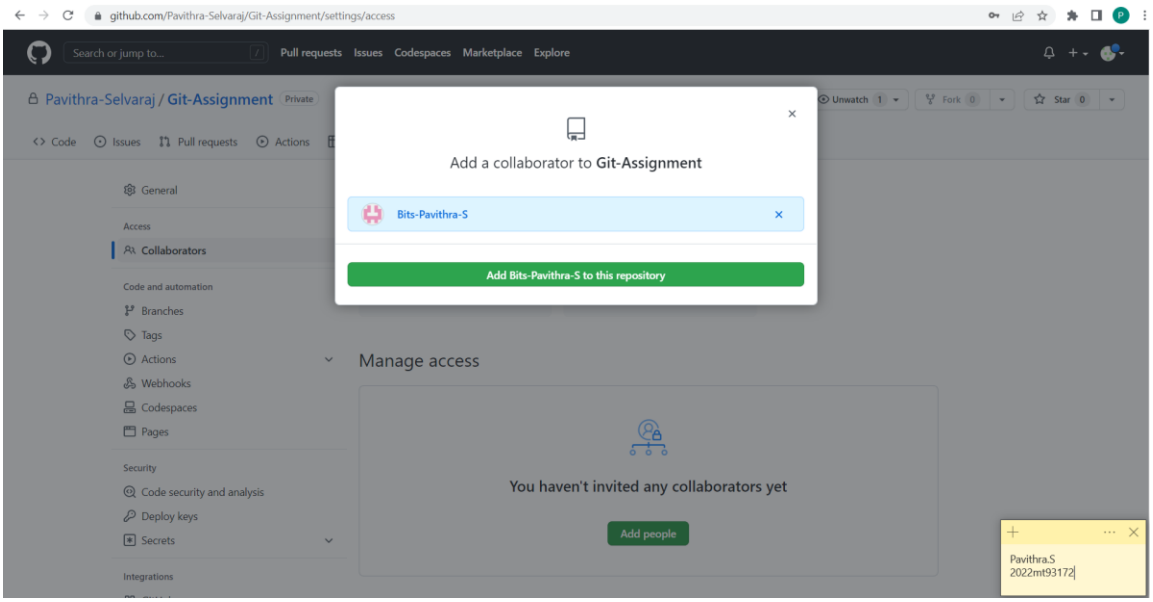


4. Search for the collaborator name.

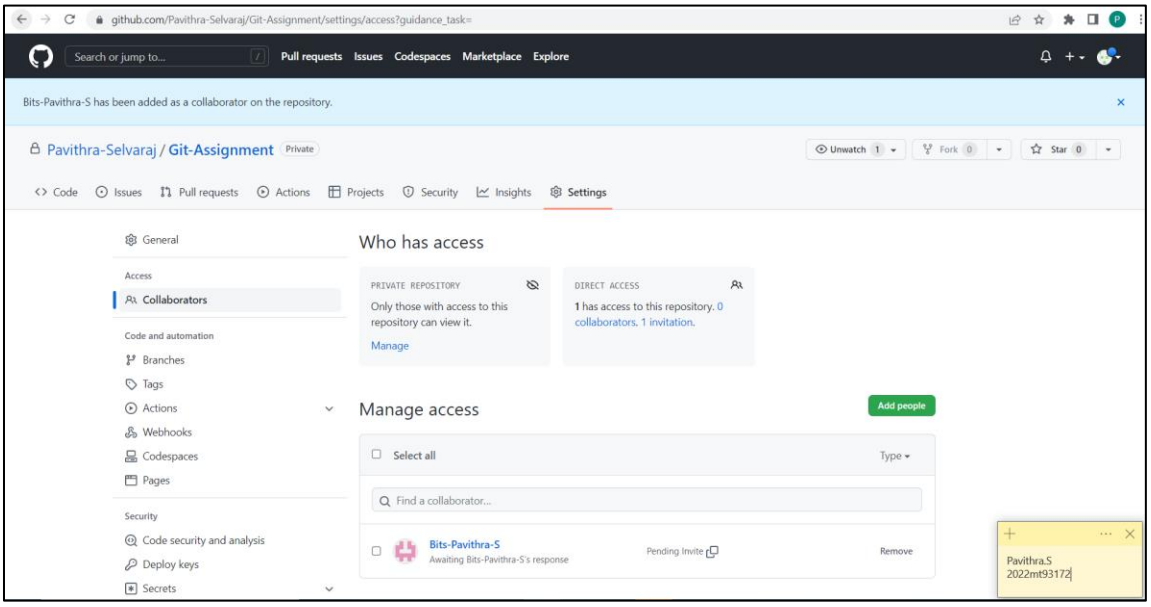


We are going to invite “Bits-Pavithra-S” (Dummy account for assignment purpose) into this repository.

Search the name and click on the Add option.



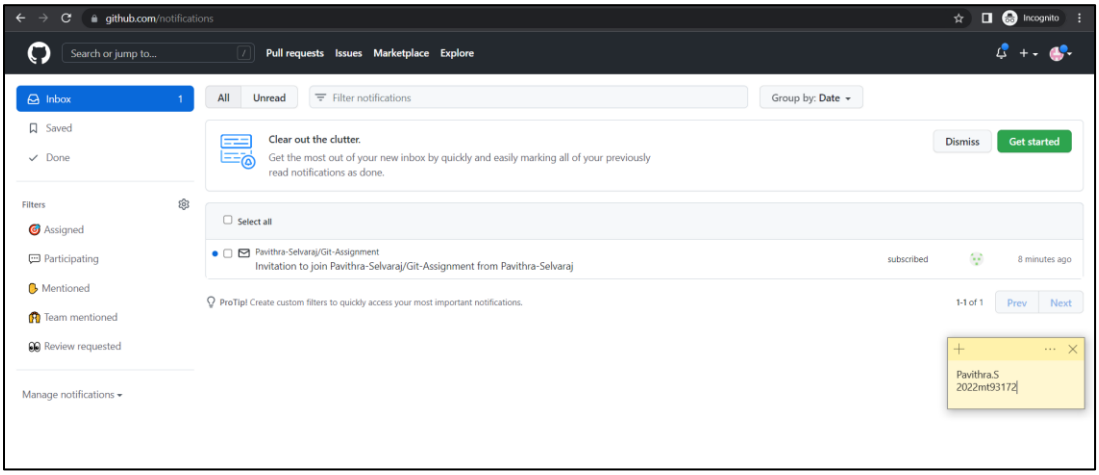
5. An invite is now sent to the collaborator.



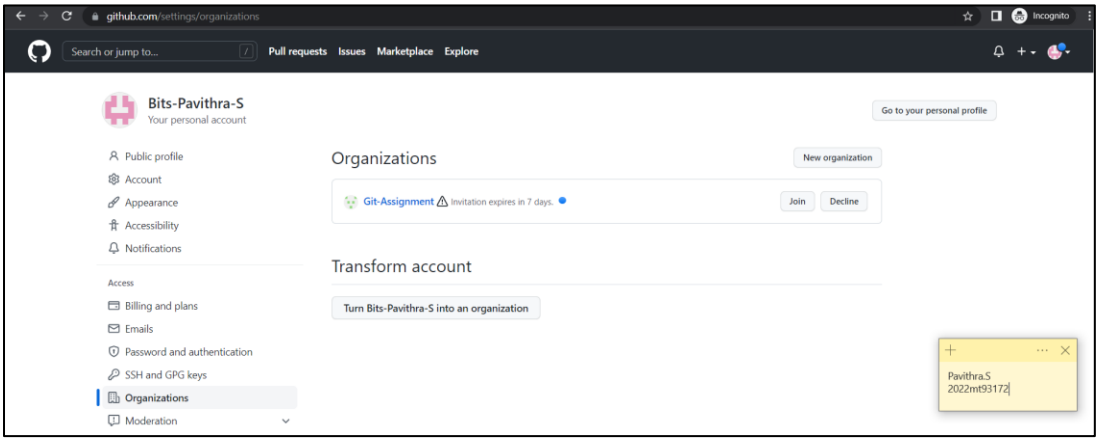
It now the collaborators work to accept the invite and be part of the repository.

Accepting the Invite

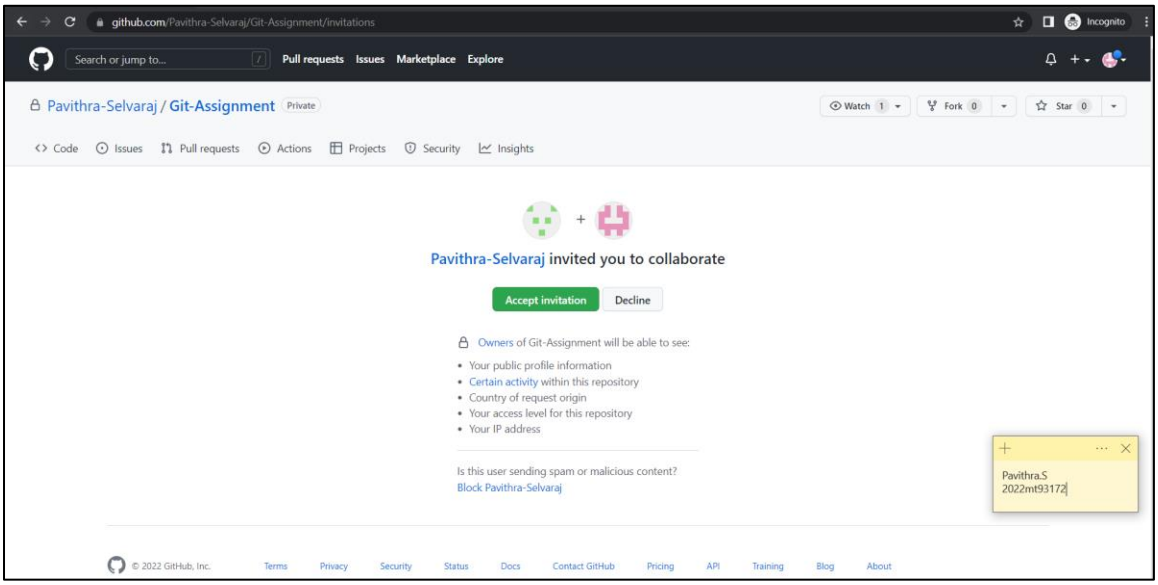
1. Once the invite is sent by the owner, a notification is received at the collaborator end.



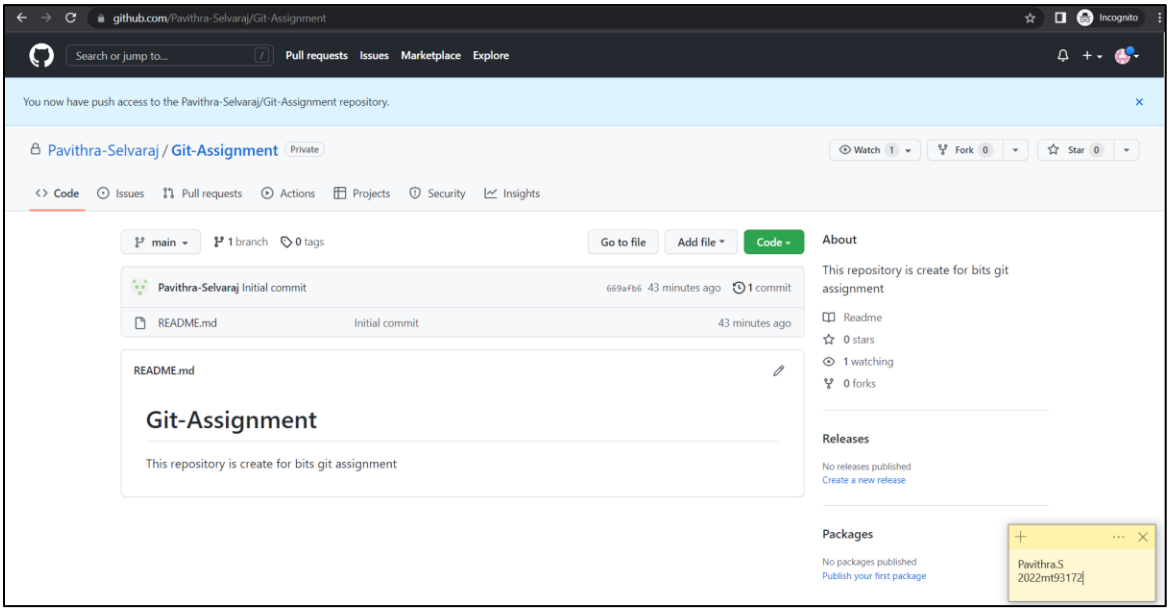
2. Open the notification to accept the request.



3. Click the “Join” button.



4. Click the “Accept Invitation” button.



The collaborator is now added to the repository.

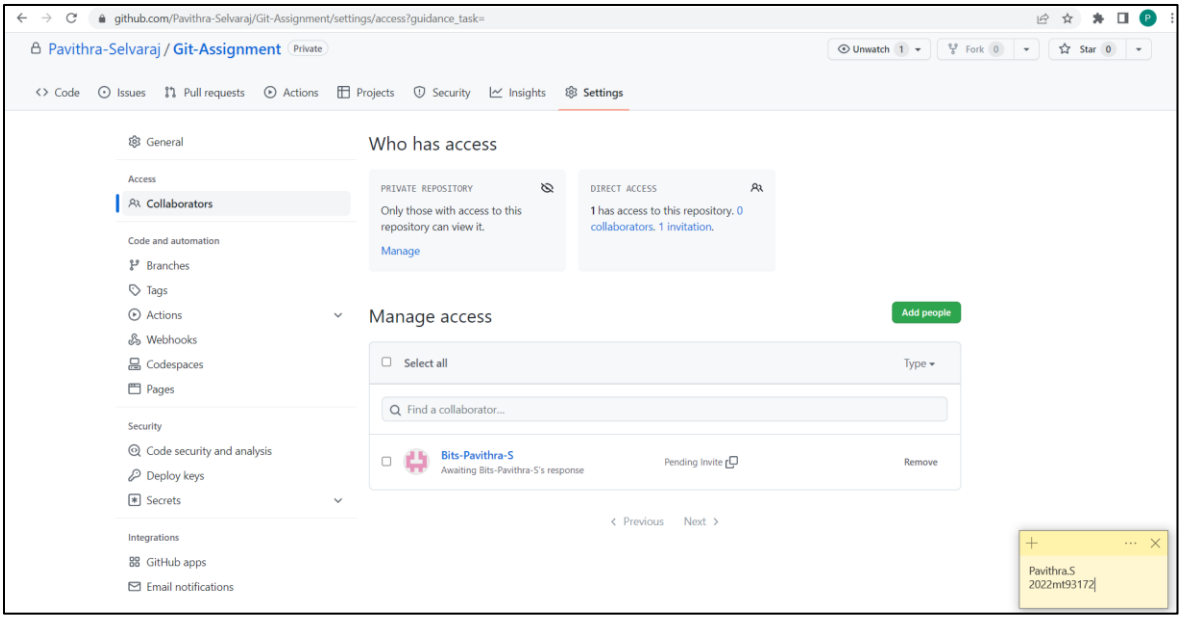
The permission levels for the collaborator and different branches can be setup in the github account directly.

Notification Setup

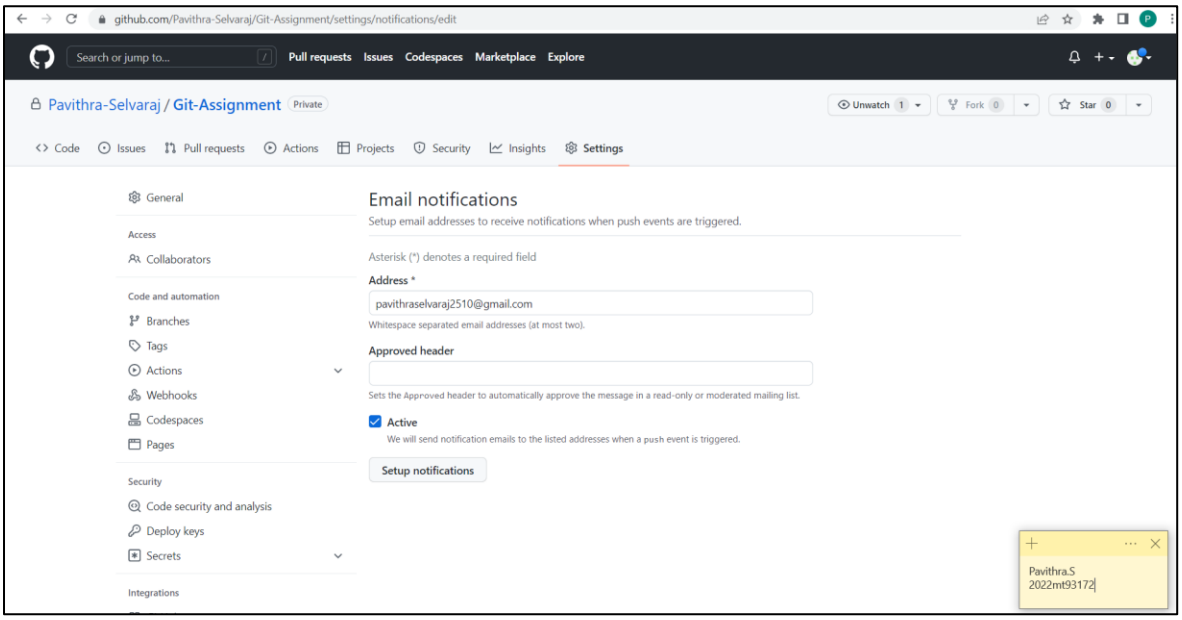
Every time a pull request is initiated, we can configure one or more email ids to send a email notification to.

The steps are,

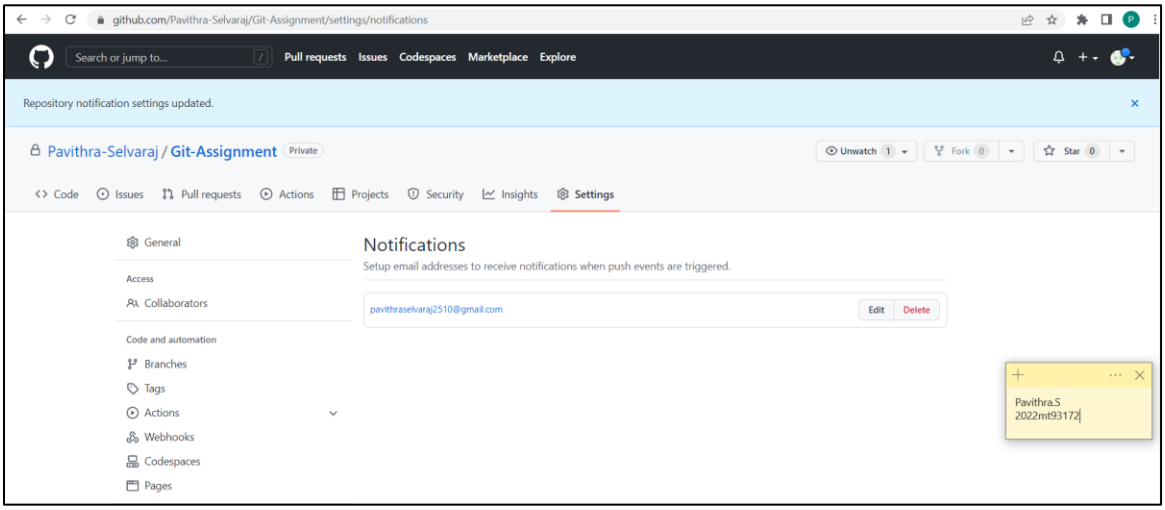
1. Open Settings.



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

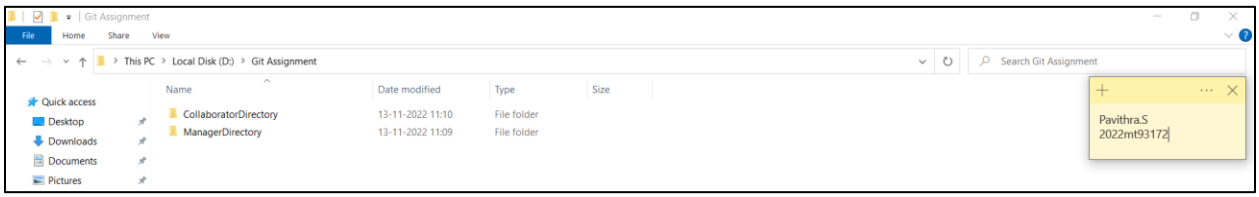
To copy an existing git repository to collaborate, we can clone that repository in out local machine and work.

The command for cloning is,

git clone <repository-link>

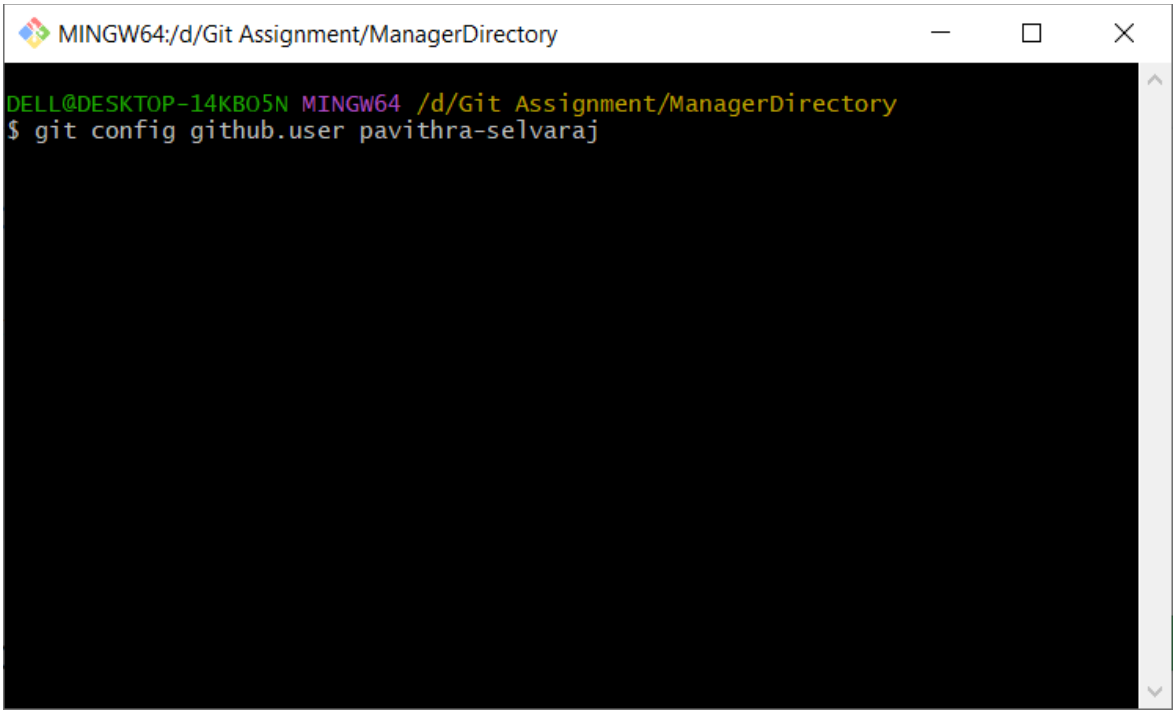
To clone our “Git-Assignment” repository, let us first create a folder in the required location.

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.

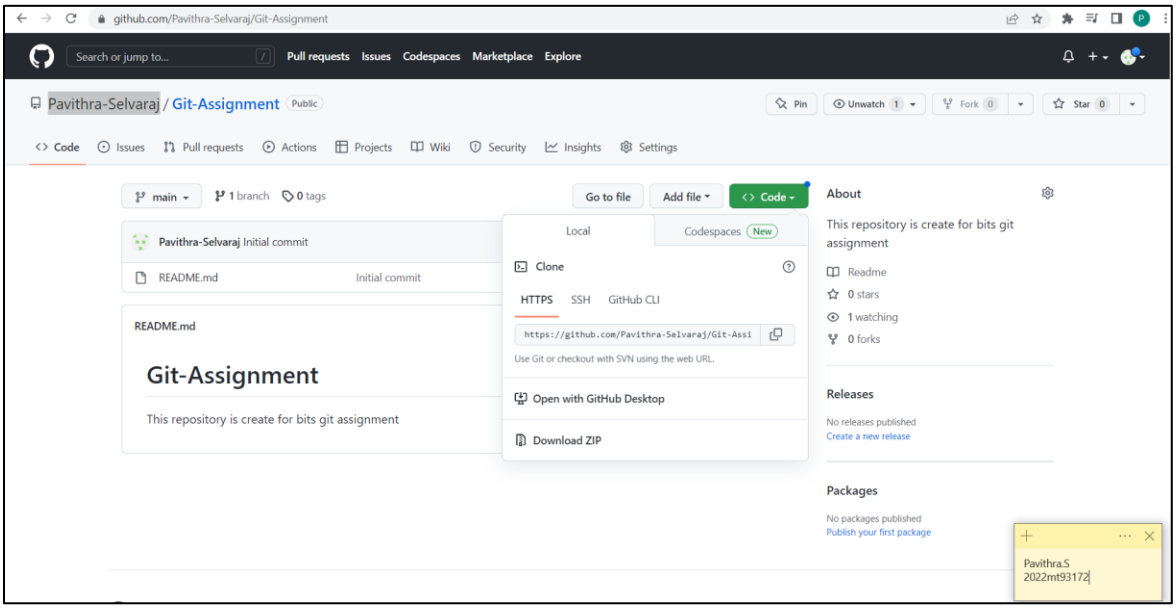


Steps to clone the repository are,

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

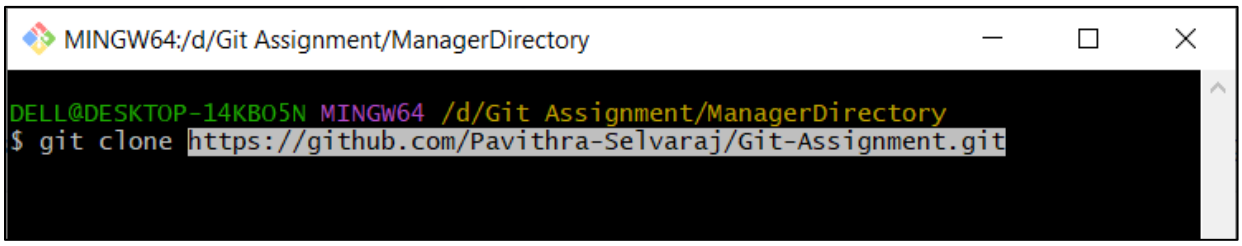


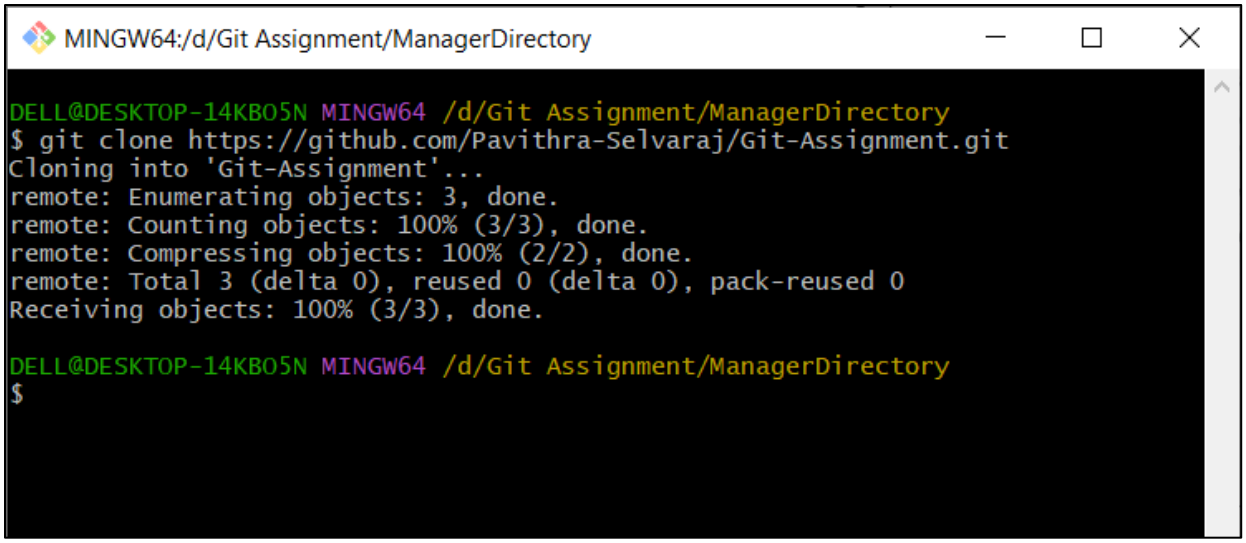
2. Copy the HTTPS link of the repository to clone from GitHub.



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

   git clone <https link>



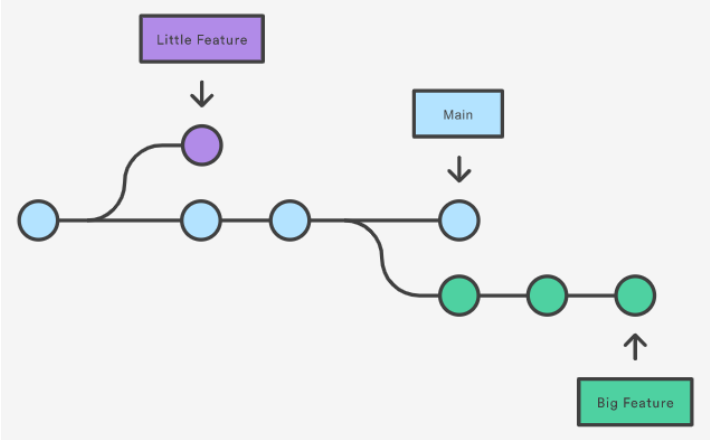


The repository is now cloned and the files are retrieved in the local folder.

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.



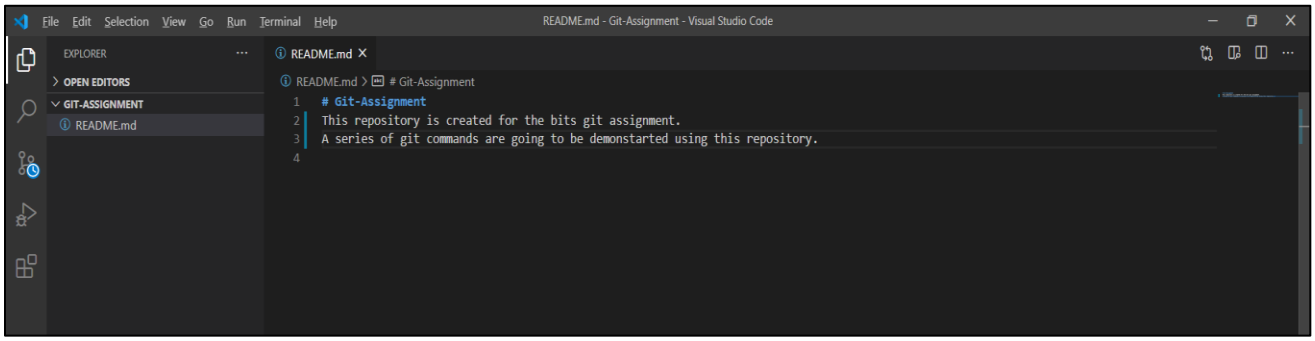
A new branch can be created by using the command,

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

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

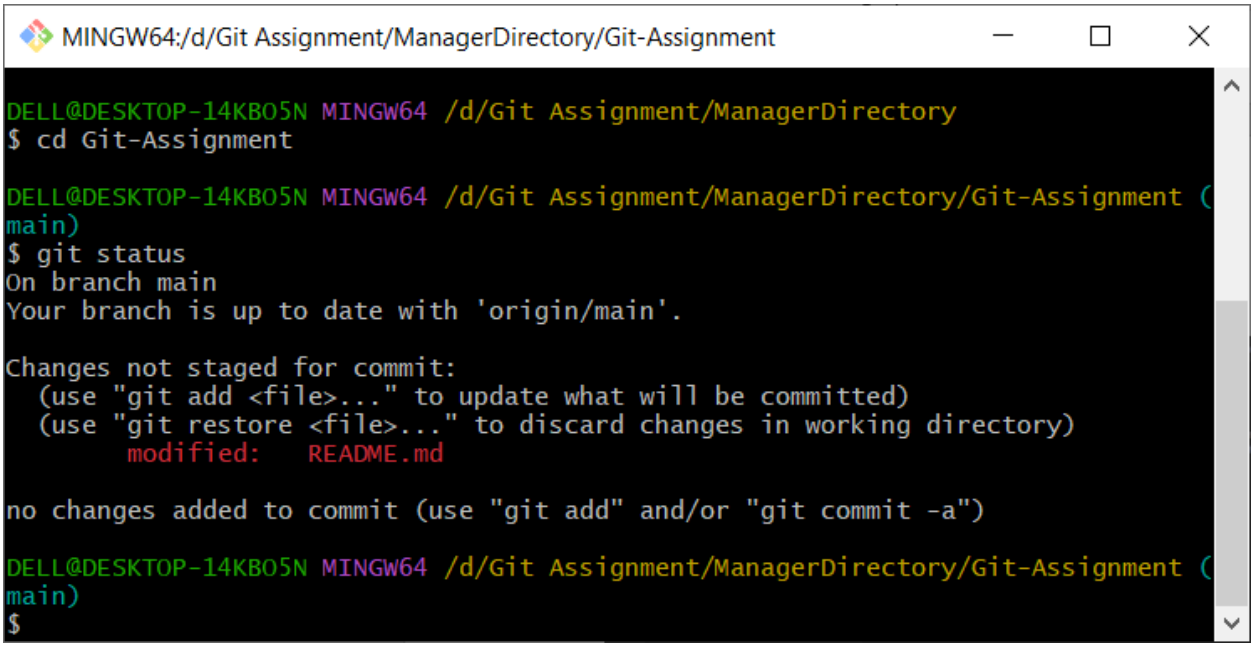
```
git checkout <branch-name>
```

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

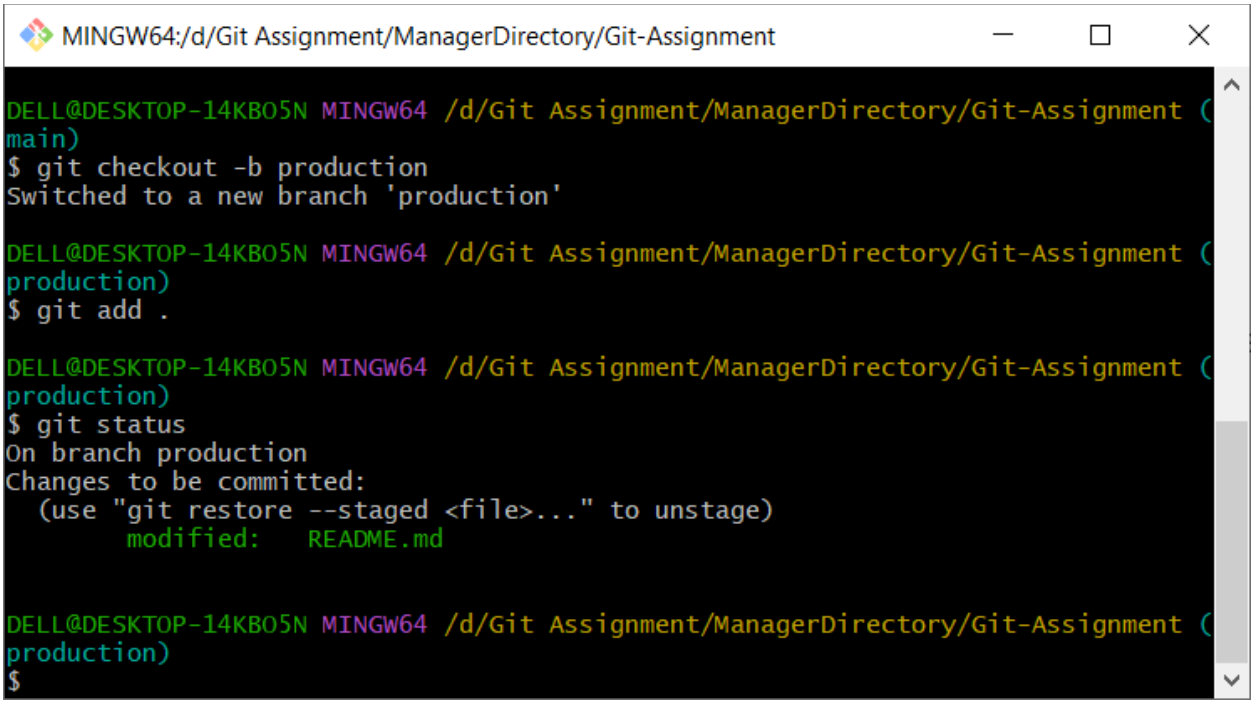


The git status command shows the state of the working directory and the staging area.

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



Let's create a new branch named "production" branch.



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.

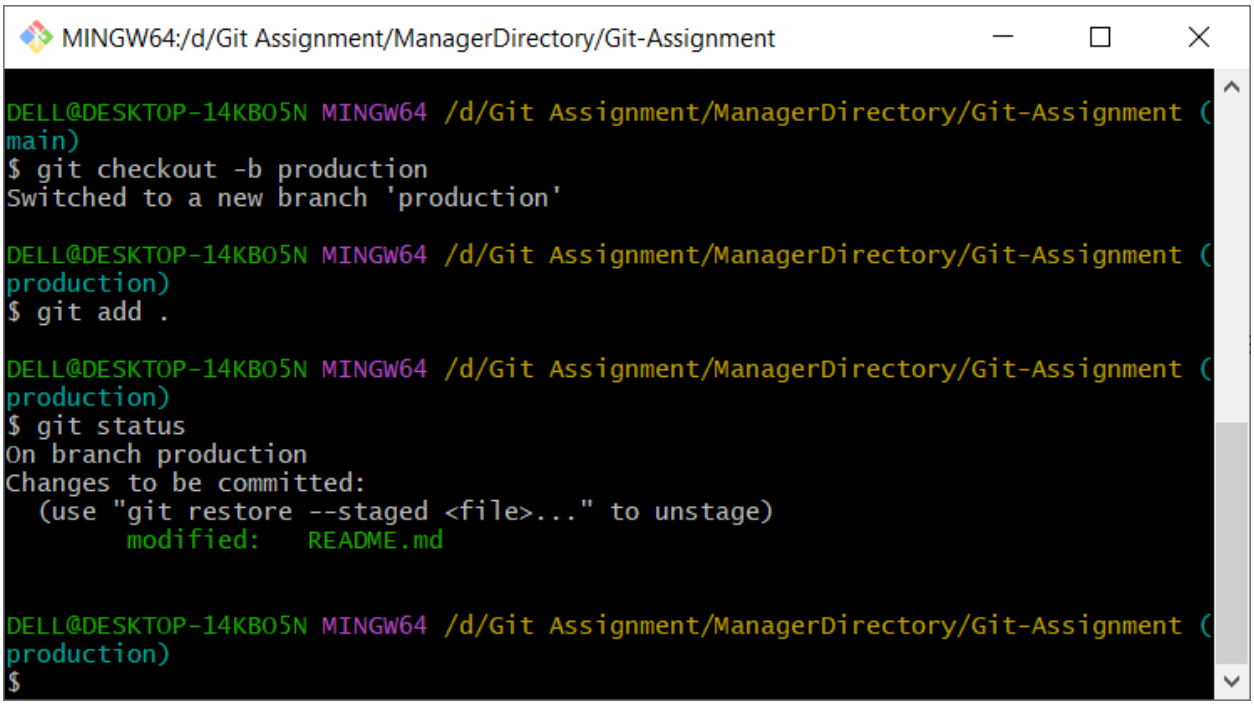
Or, use,

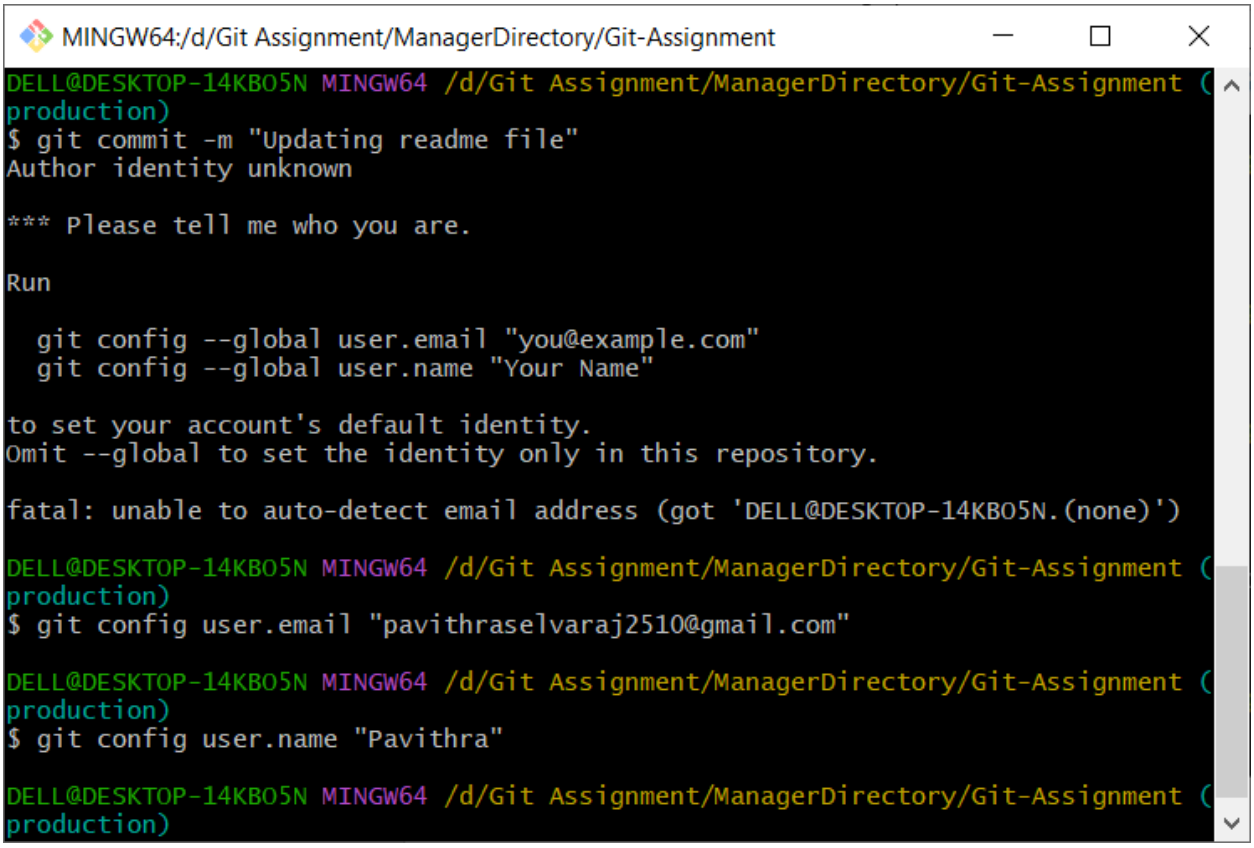
git add <file-name>

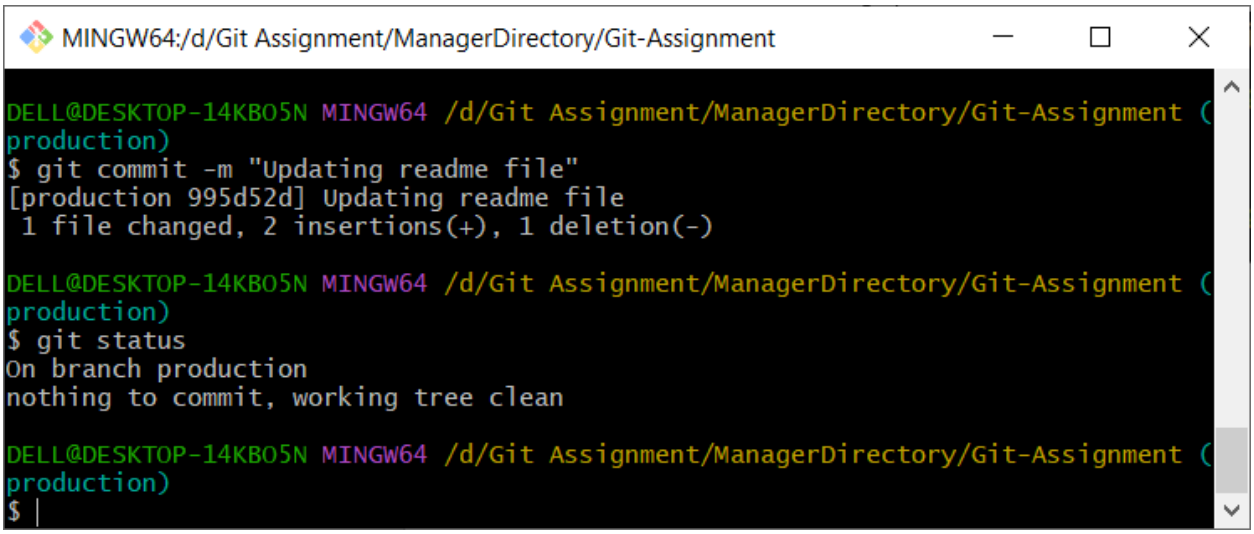
This can be used to add individual files.

Then commit using,

git commit -m <commit-message>




MINGW64:/d/Git Assignment/ManagerDirectory/Git-Assignment
DELL@DESKTOP-14KBO5N 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-14KBO5N.(none)')
DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$ git config user.email "pavithraselvaraj2510@gmail.com"
DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$ git config user.name "Pavithra"
DELL@DESKTOP-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)


MINGW64:/d/Git Assignment/ManagerDirectory/Git-Assignment
DELL@DESKTOP-14KBO5N 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-14KBO5N MINGW64 /d/Git Assignment/ManagerDirectory/Git-Assignment (production)
$ git status
On branch production
nothing to commit, working tree clean
DELL@DESKTOP-14KBO5N 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

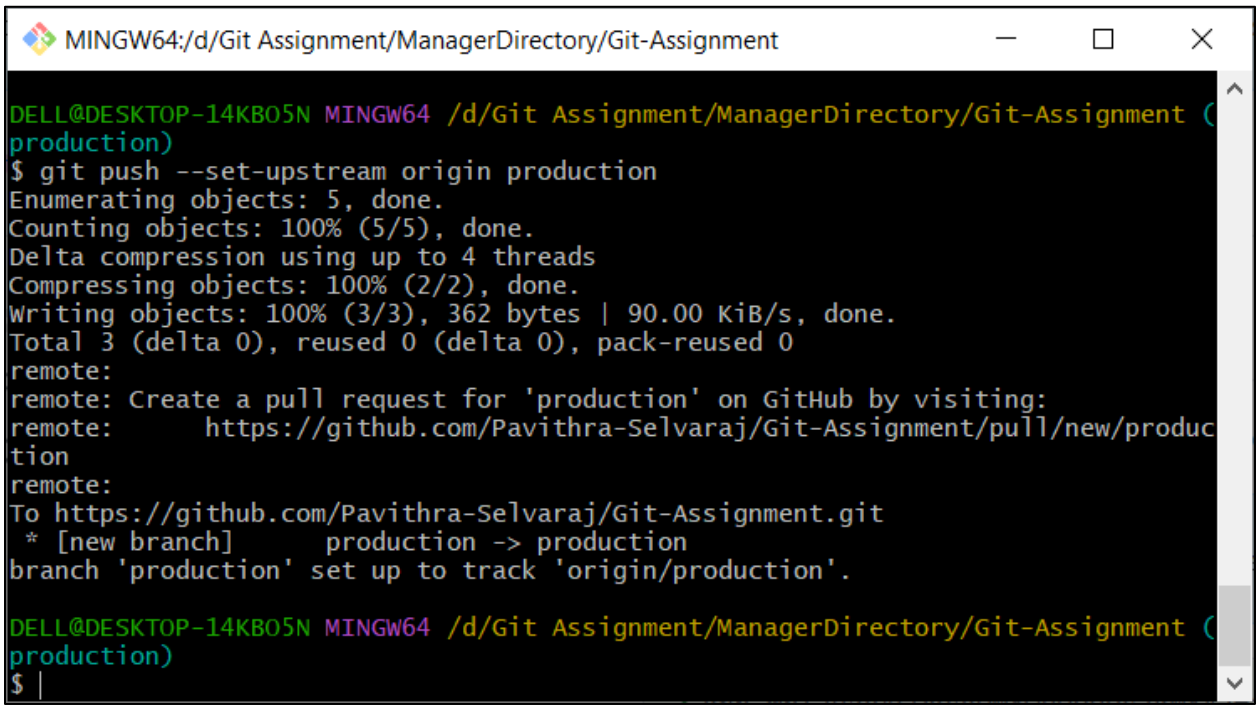
For example:

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

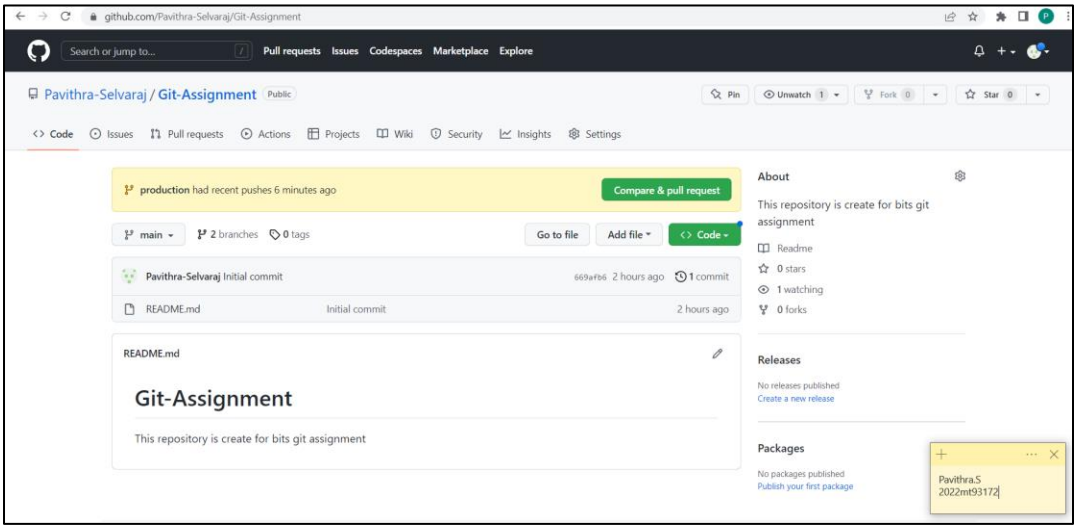
In our case we run,

git push --set-upstream origin production.

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



The production branch is now pushed to the origin.



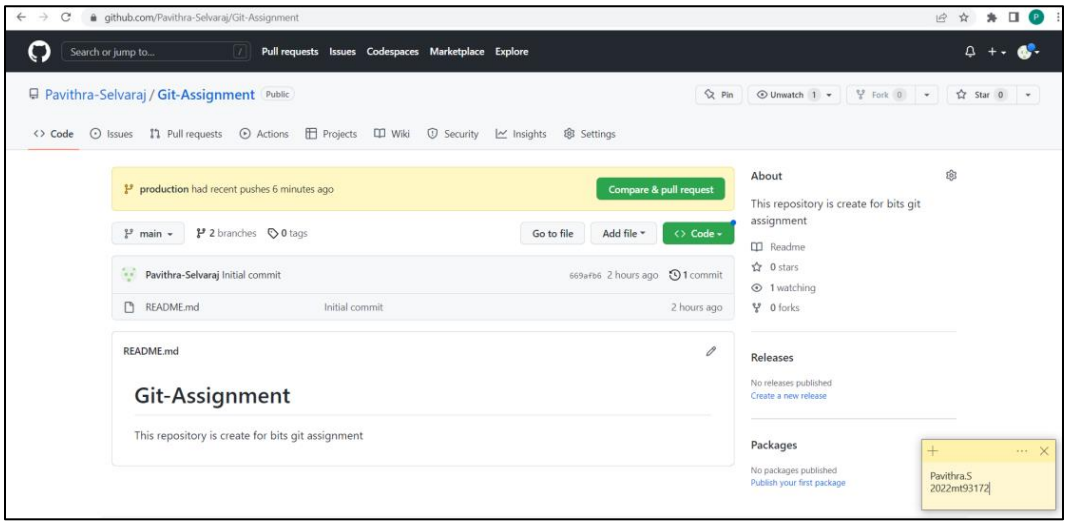
Pull Requests

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.

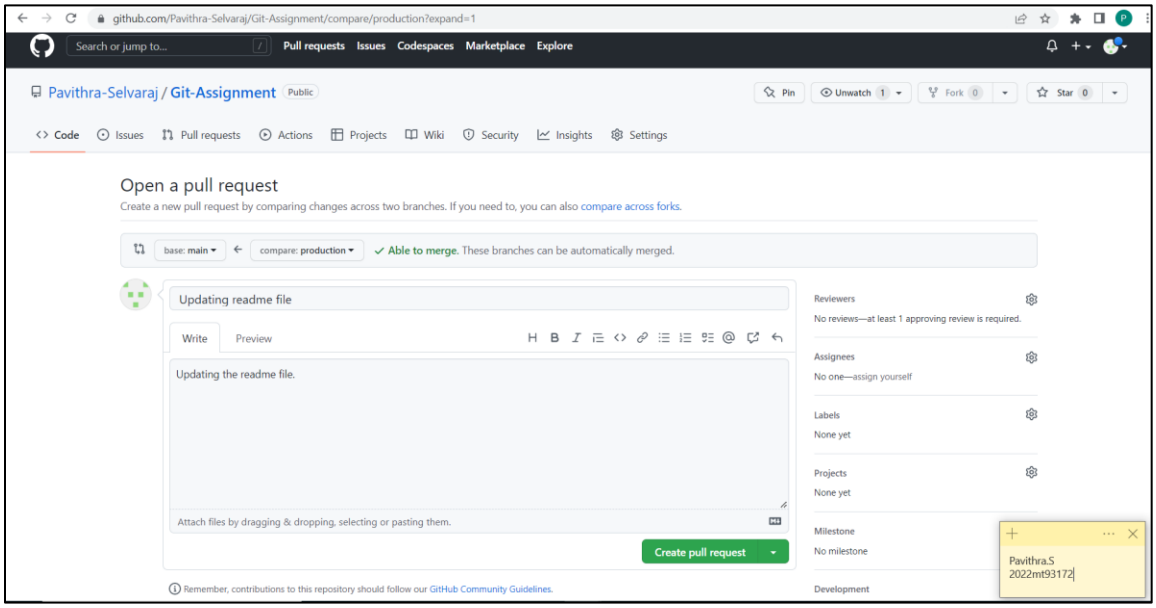
A pull request approval in Git will involve getting the project maintainer(s) to review the code part of the pull request. After code review, they will provide comments or approve the request. Approval will merge all the code changes directly into the requested repository.

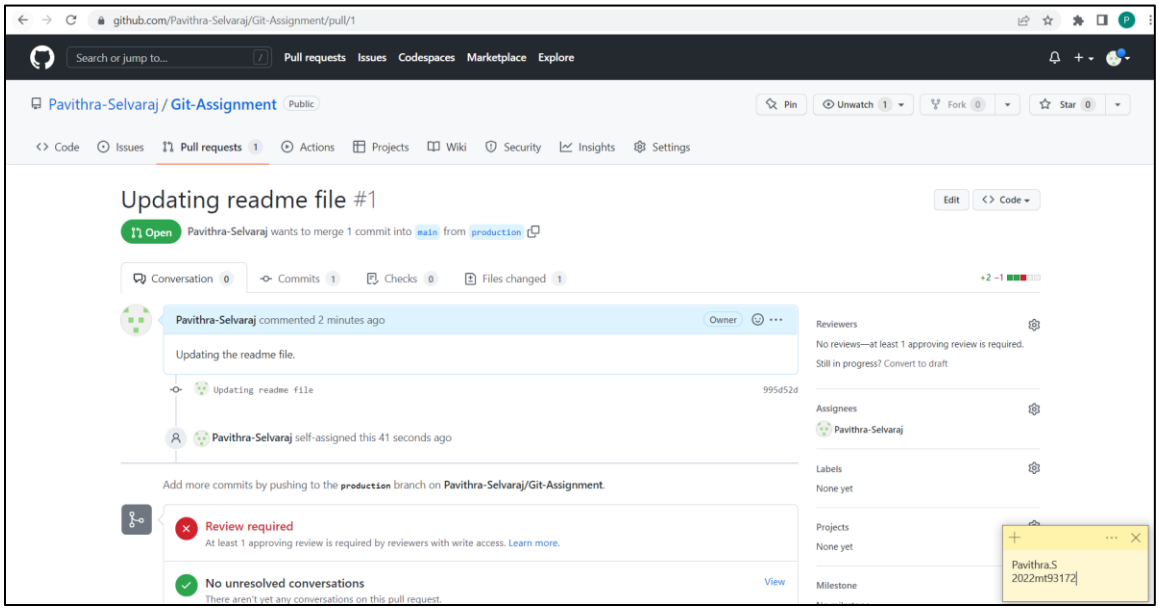
Let's initiate a pull request for the production branch we created and pushed to the origin to be merged into the main branch.

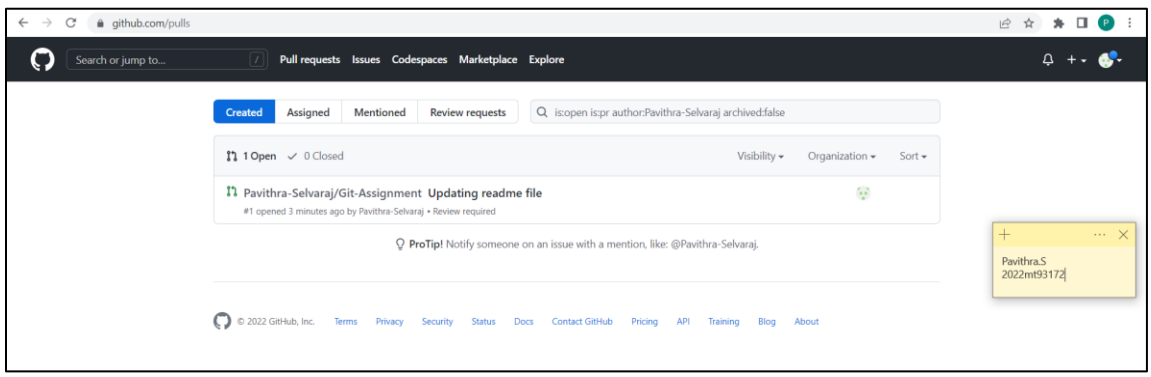
Click on "Compare & pull request". Or navigate to the pull request module to create one.



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



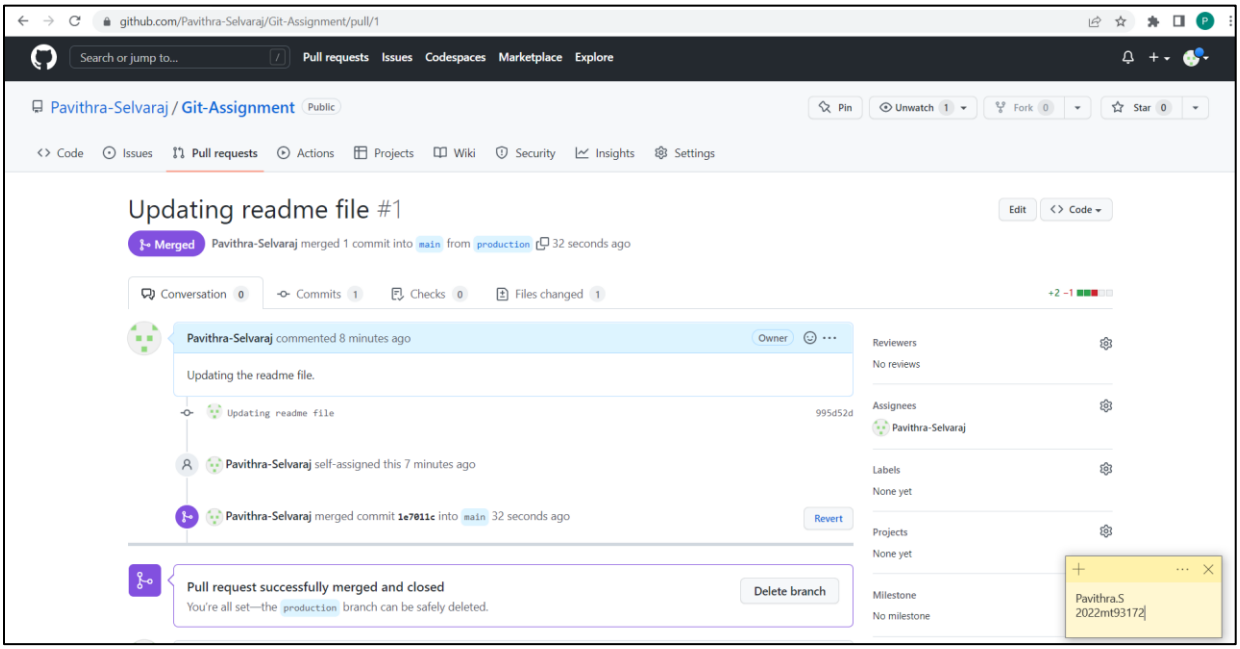




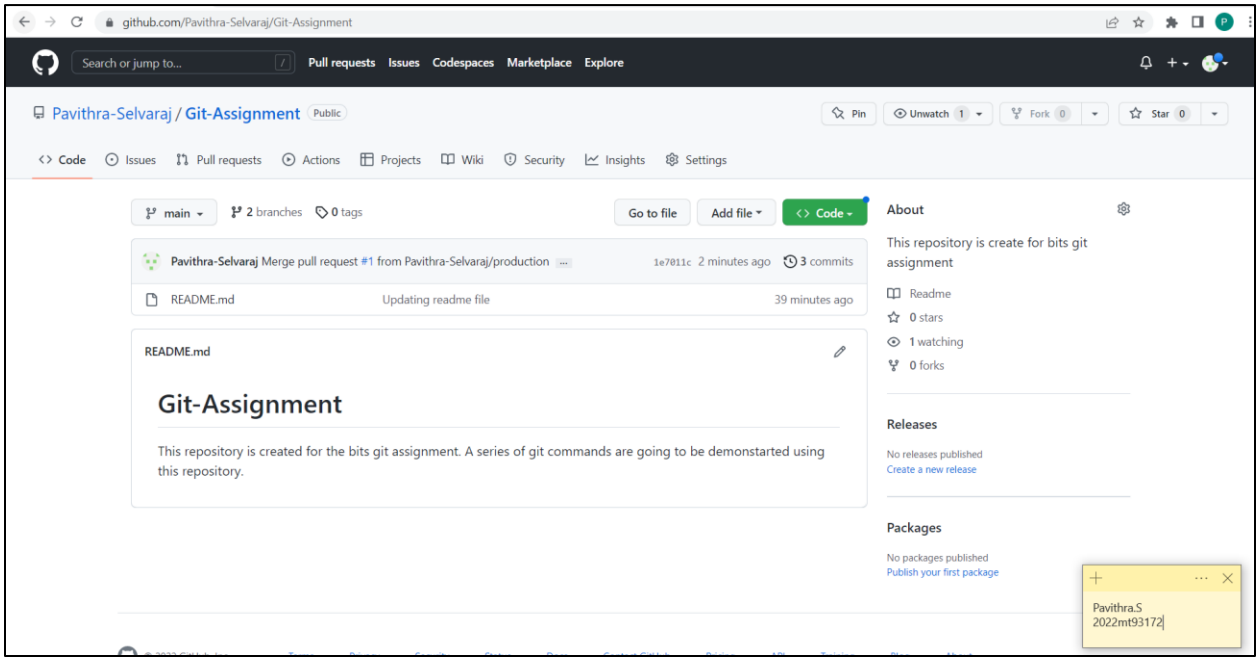
Since this is a manager account, we force accept the pull request.

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 changes are now merged and updated.



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 form production branch to main branch. So, now the production branch has redundant code only and can be removed.

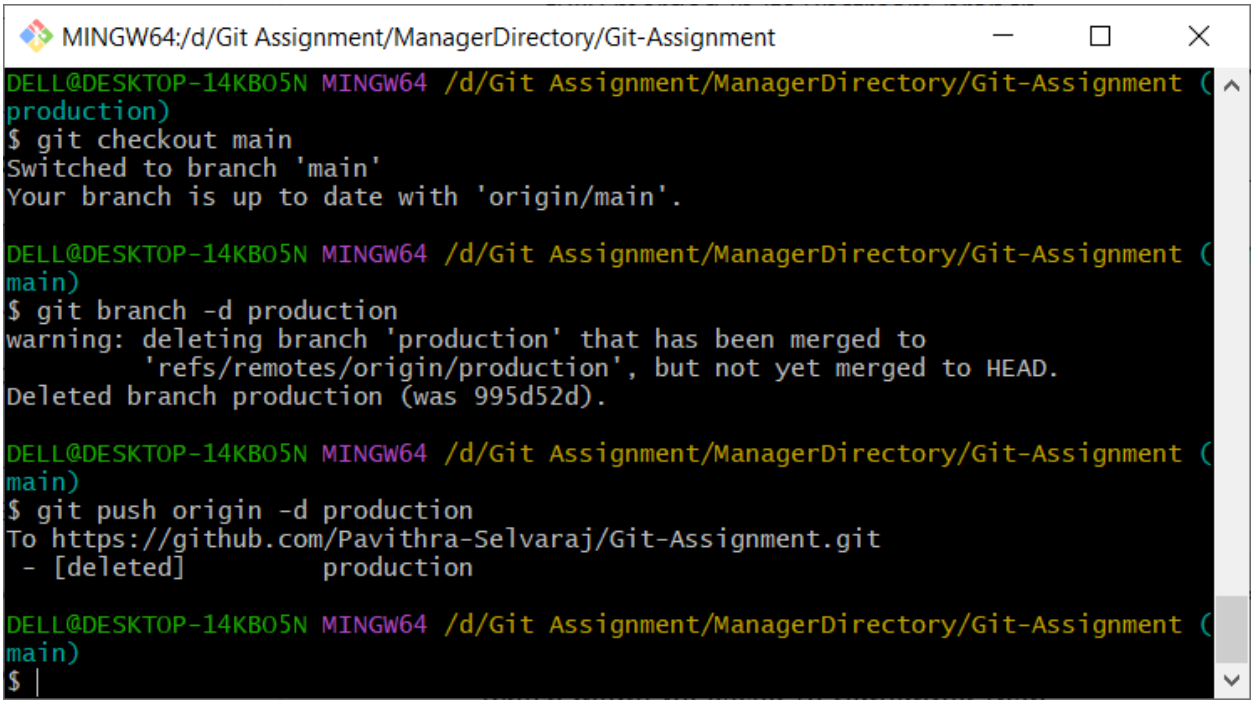
We cannot delete the branch in which we are currently in. So, move to another branch, here it's main and then delete the required branch.

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>



Merge Conflict

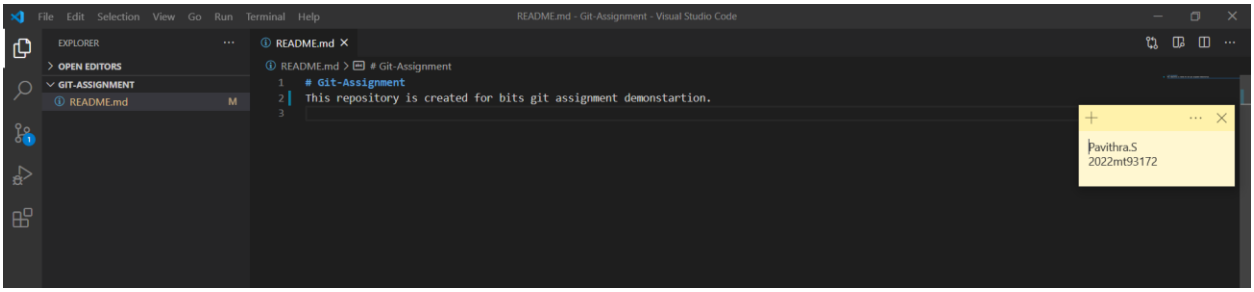
Conflicts arise when two people have changed the same lines in a file, or if one developer deleted a file while another developer was modifying it.

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.

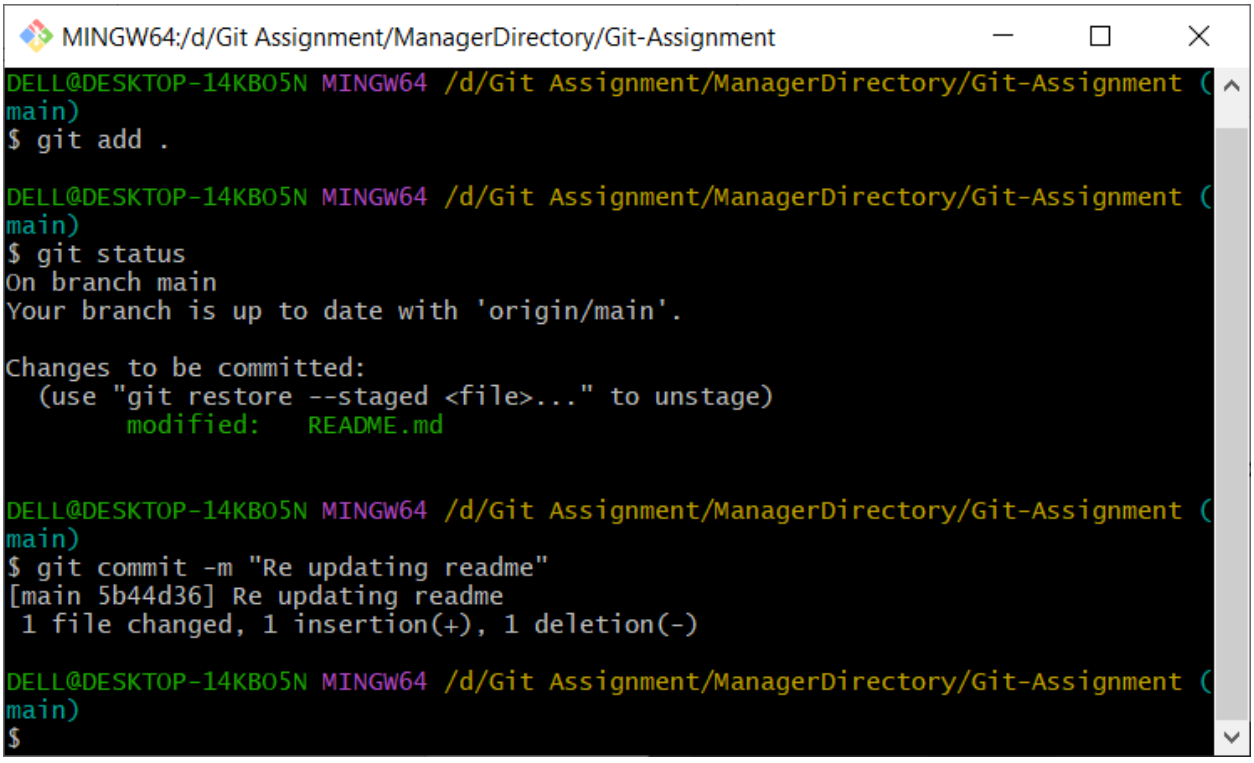
In our example, the main branch is now merged with production branch and the latest code is updated the origin.

But the repository in the local is not updated yet.

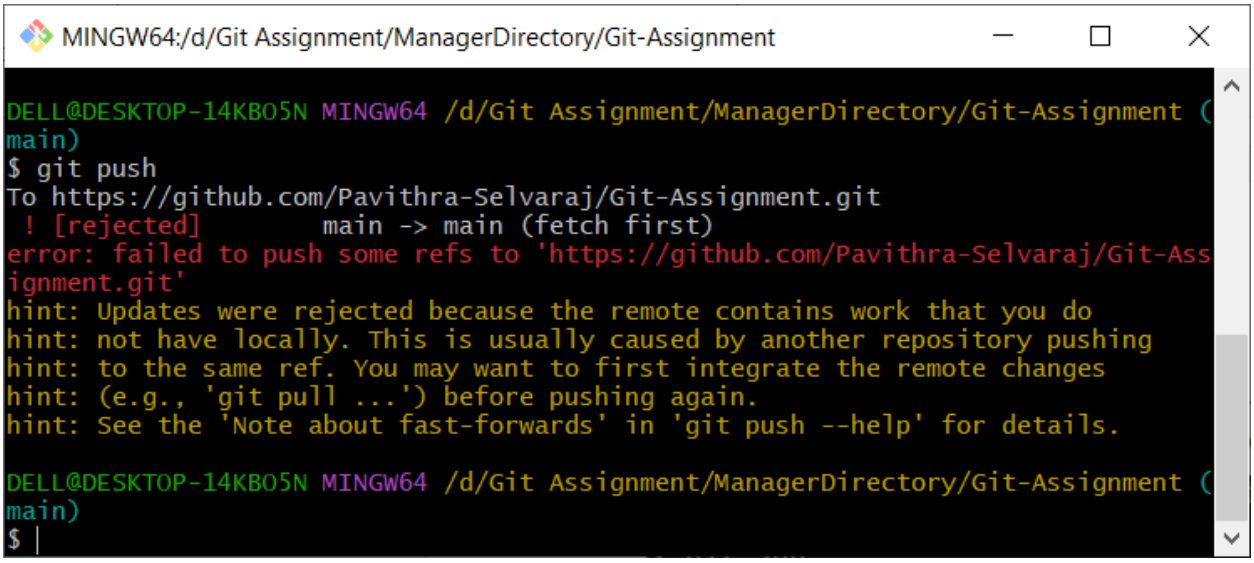
Let’s now edit this file again and try to push the changes to the origin.



Now, commit the changes.

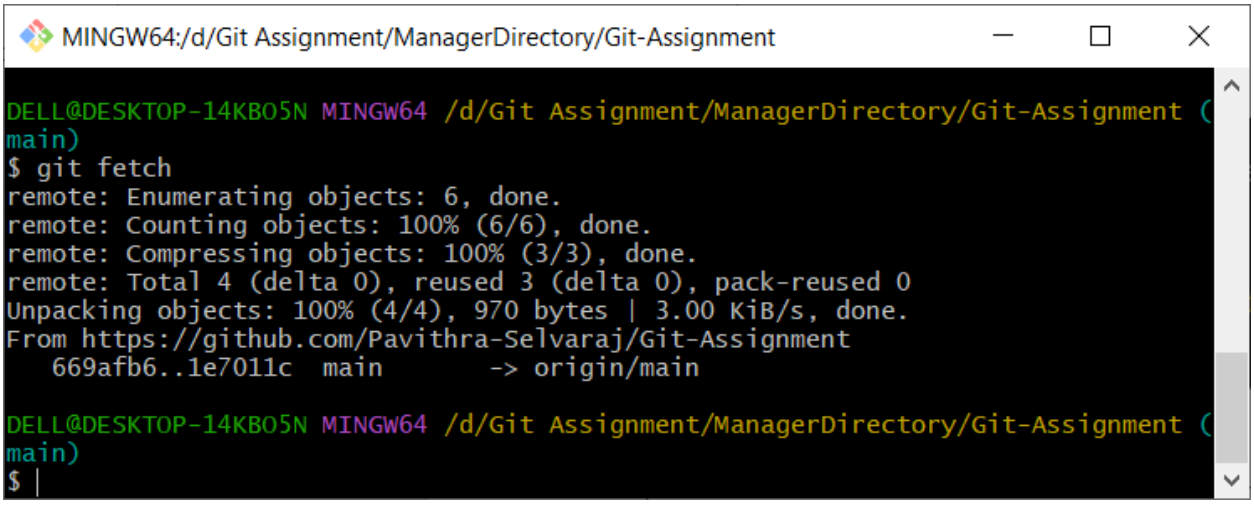


Push the changes to the origin.



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

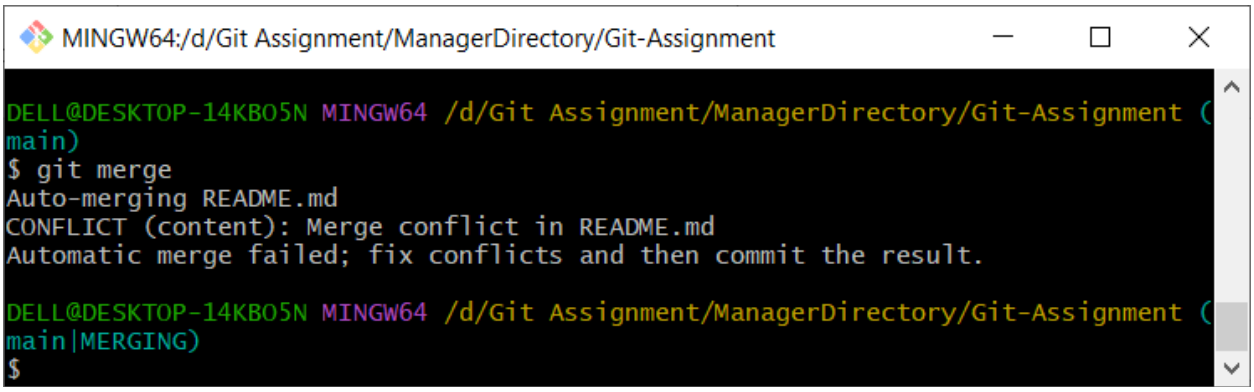
Let's fetch the changes.



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.

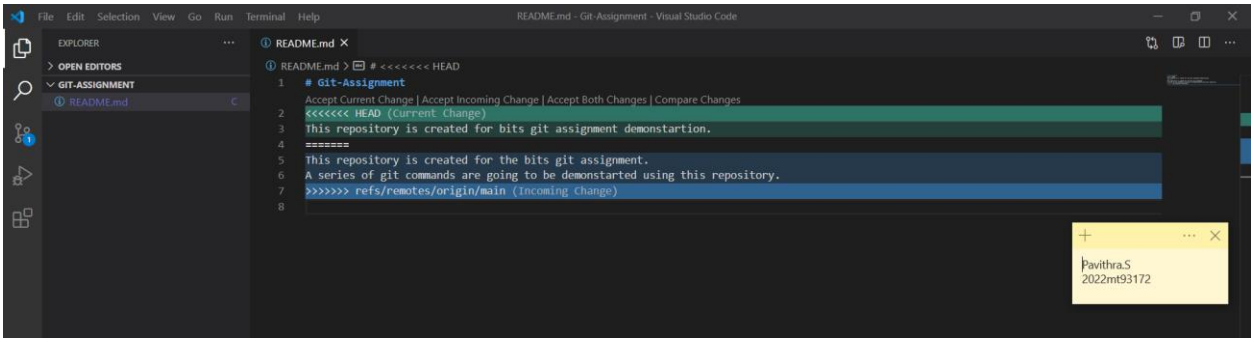
git fetch and git merge used together is similar to using git pull.



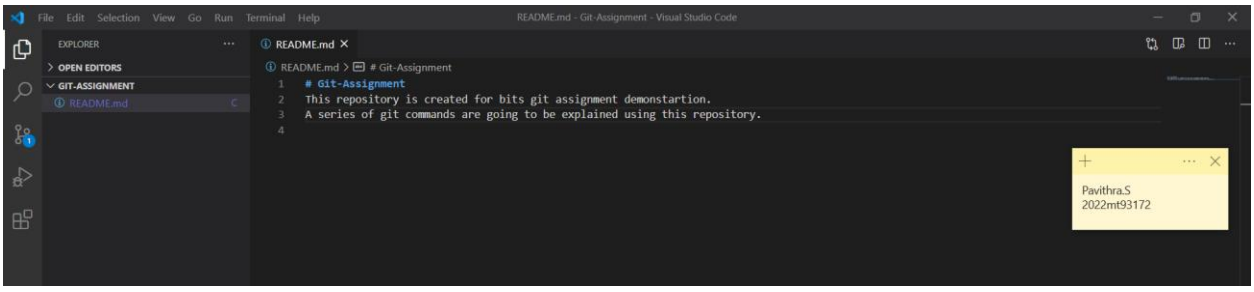
The merge is not successful. A merge conflict had occurred.

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.

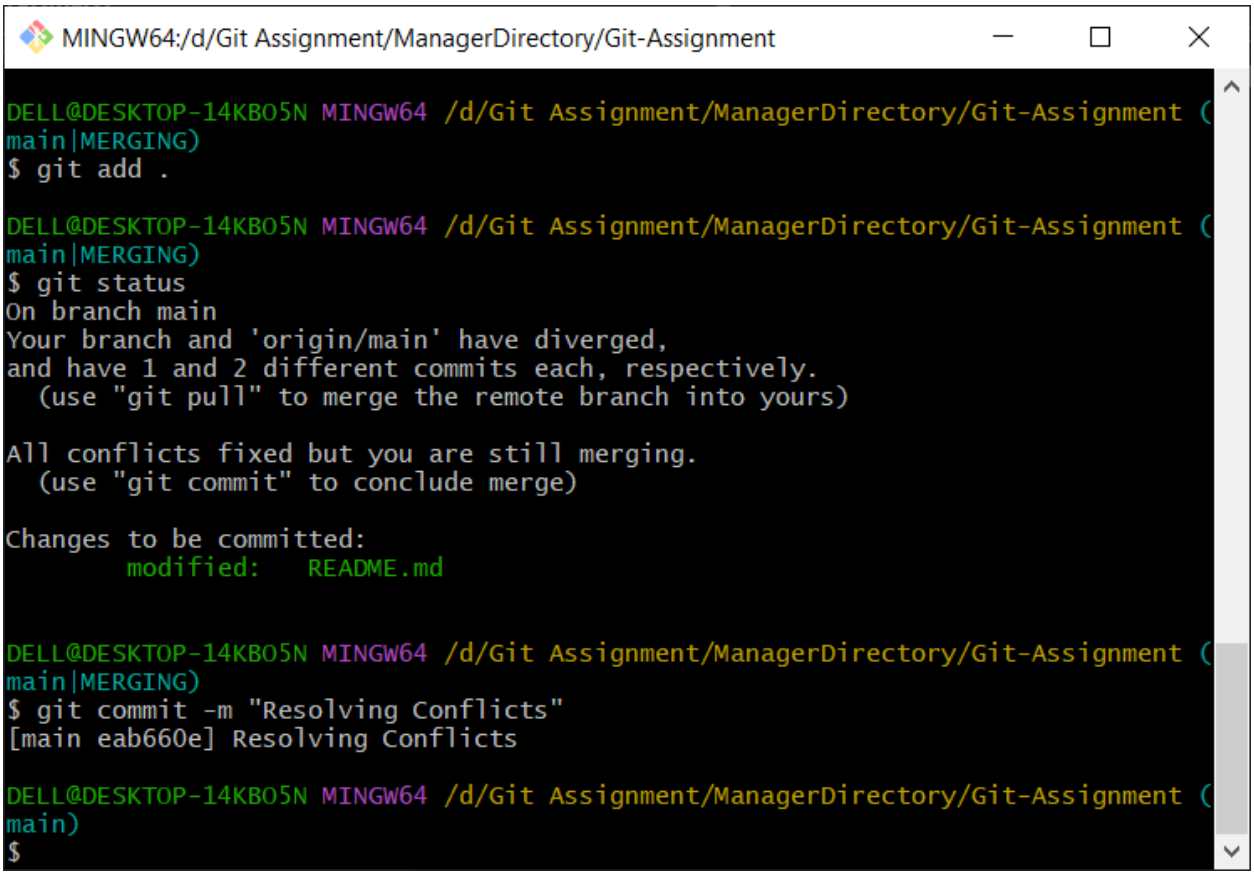


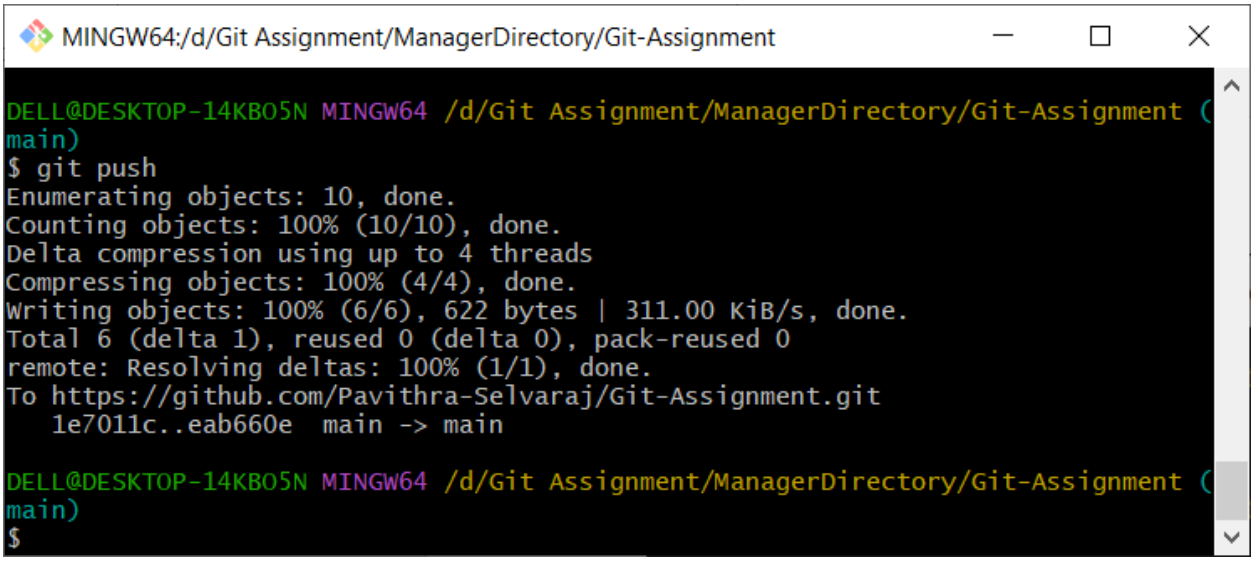
Merge conflicts can be resolved from our ide and then we commit the changes again.



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.



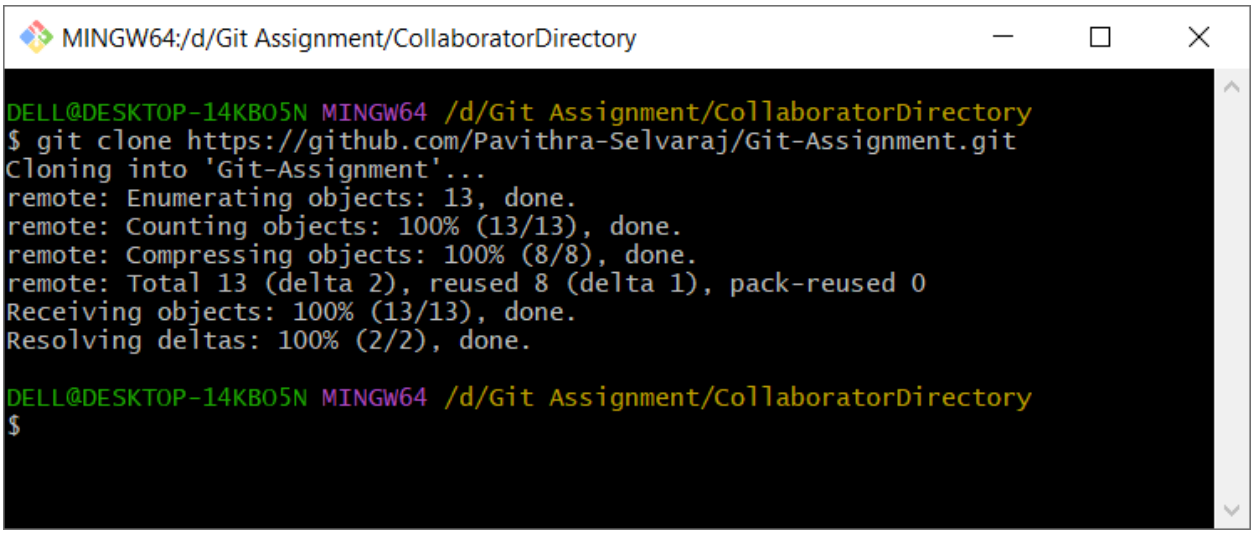


The conflicts are now resolved and changes are successfully pushed to the origin.

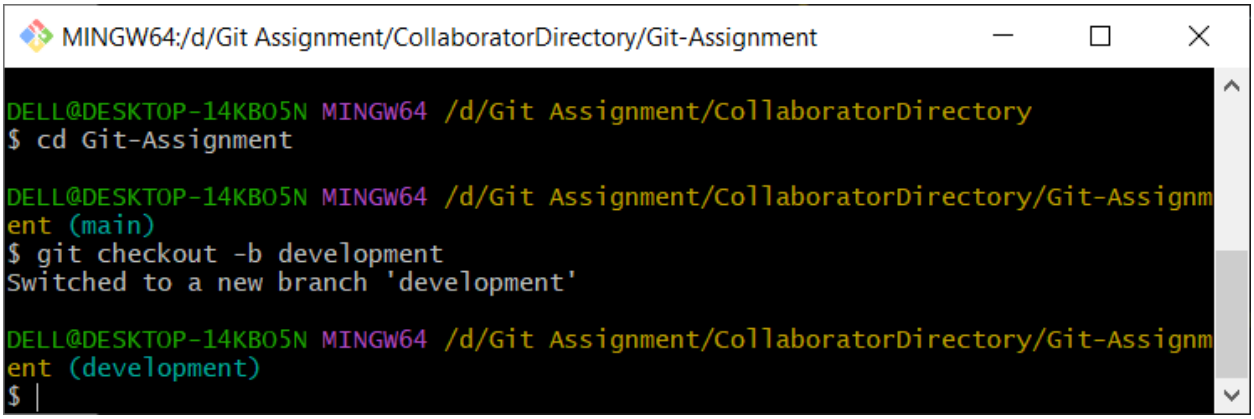
ReadMe and Gitignore Files

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.

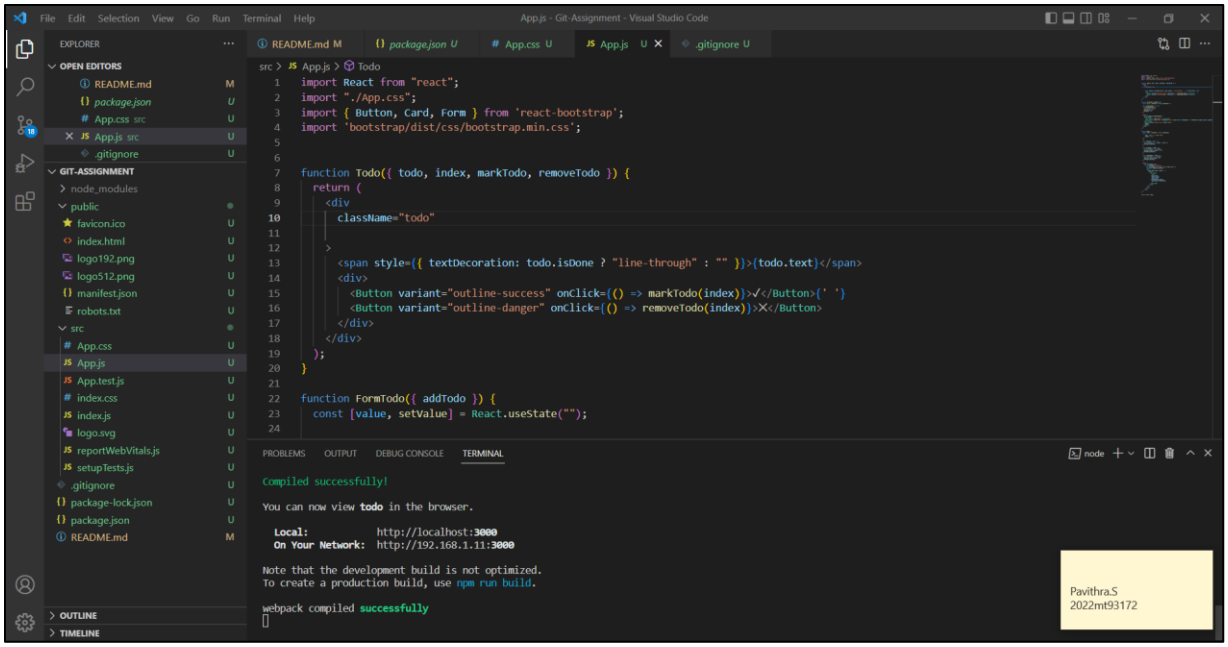


Then, we create a branch named “development”.



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

Then, commit and push to the origin as earlier.



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

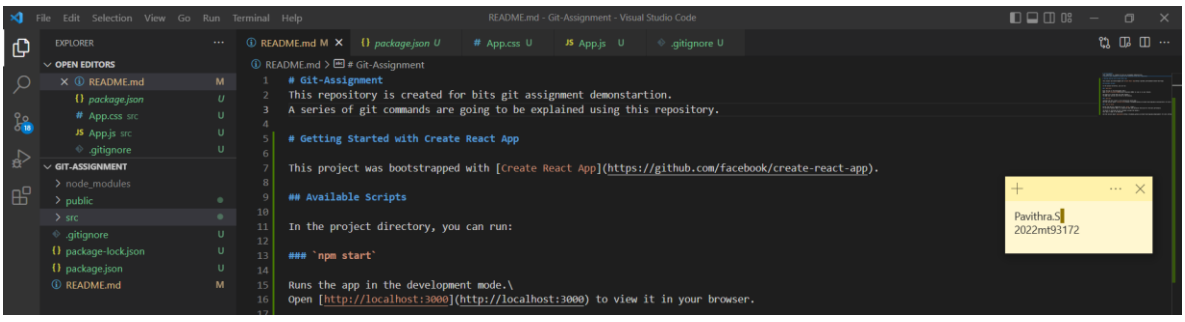
Readme.md

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



.gitignore

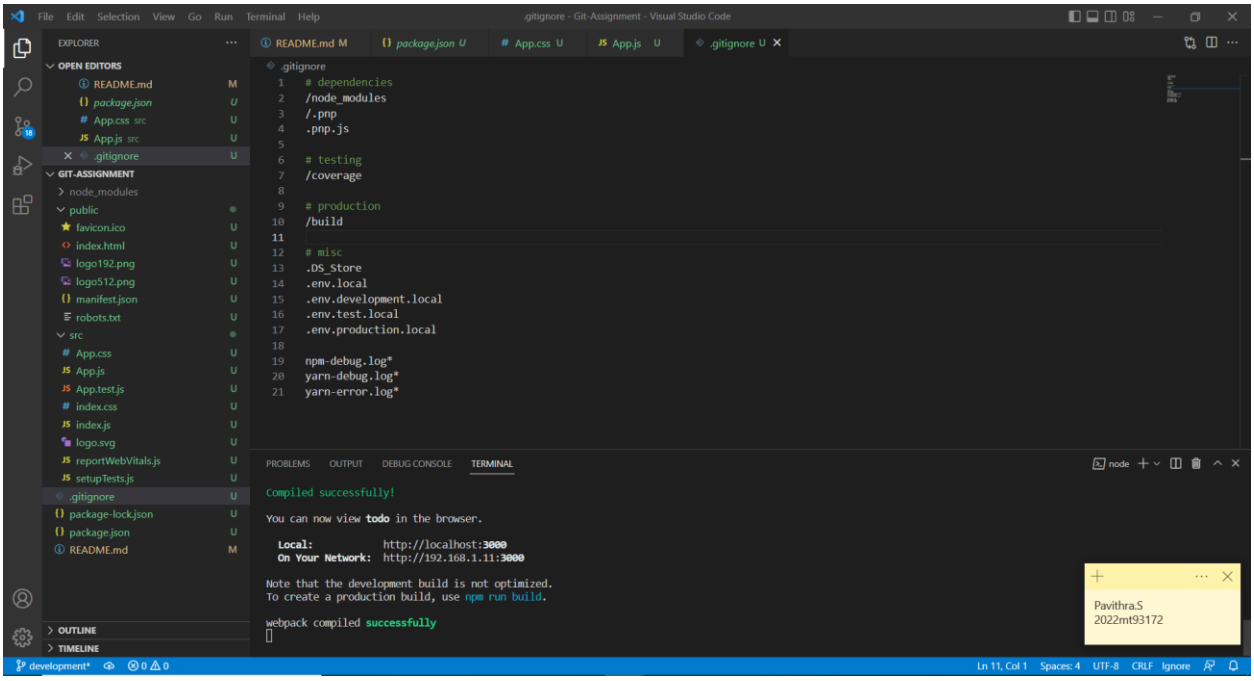
It’s a file created inside the project folder that ignores/prevents files from getting committed to the local and remote repositories.

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.

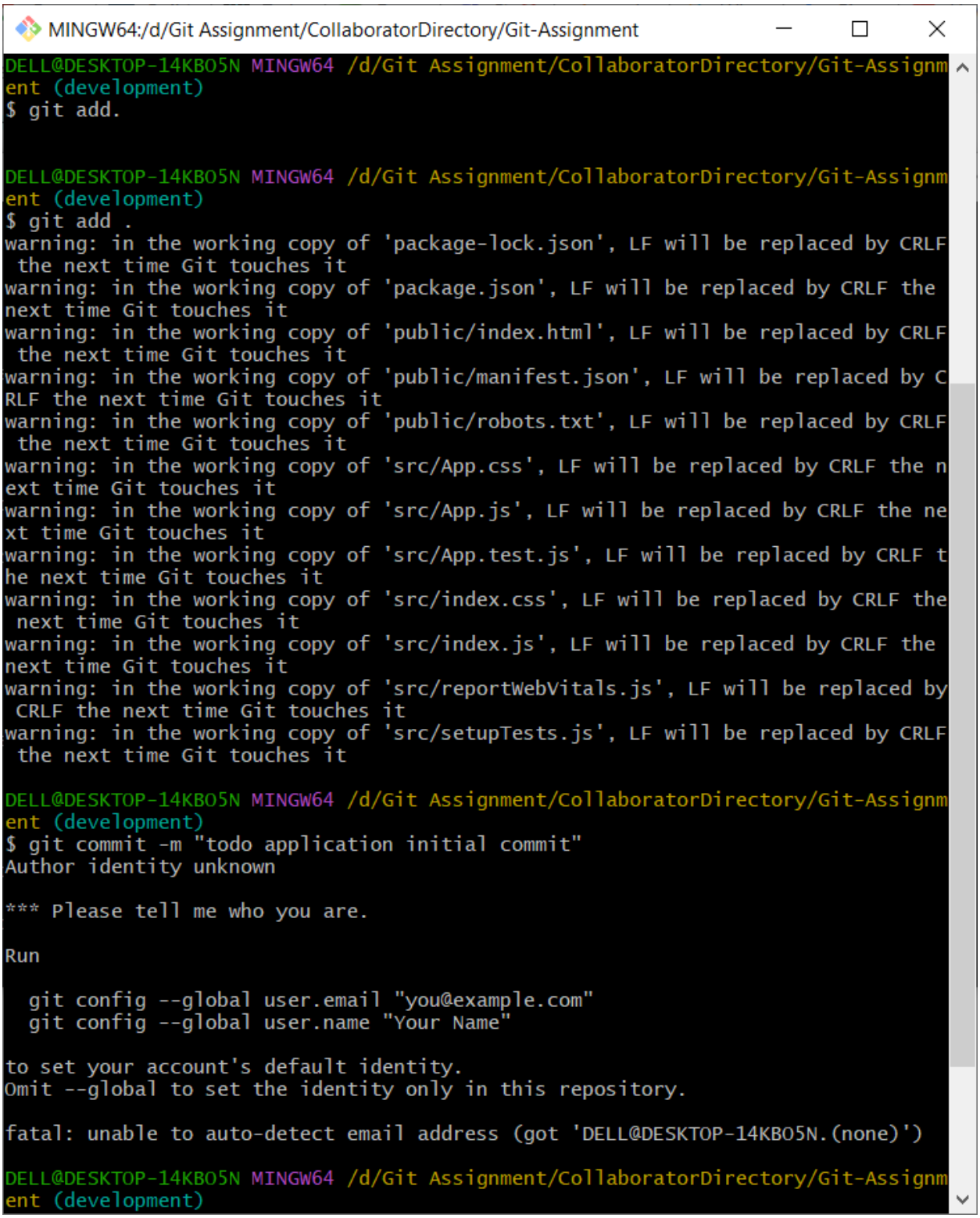
What Kind of Files Should You Ignore?

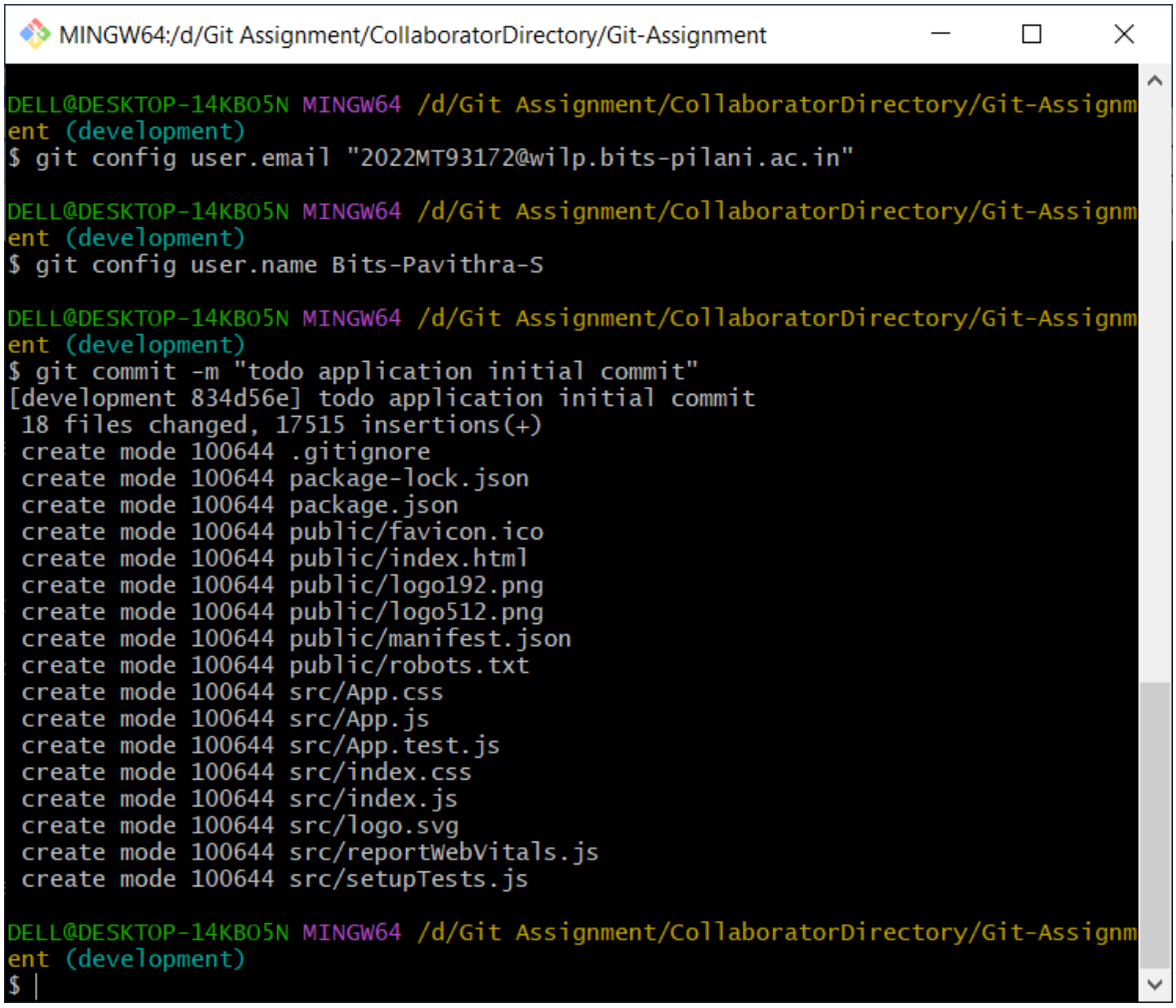
- Log files
- Files with API keys/secrets, credentials, or sensitive information
- Useless system files like .DS_Store on macOS
- Generated files like dist folders
- Dependencies which can be downloaded from a package manager. Eg: node_modules.
- And there might be other reasons (maybe you make little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



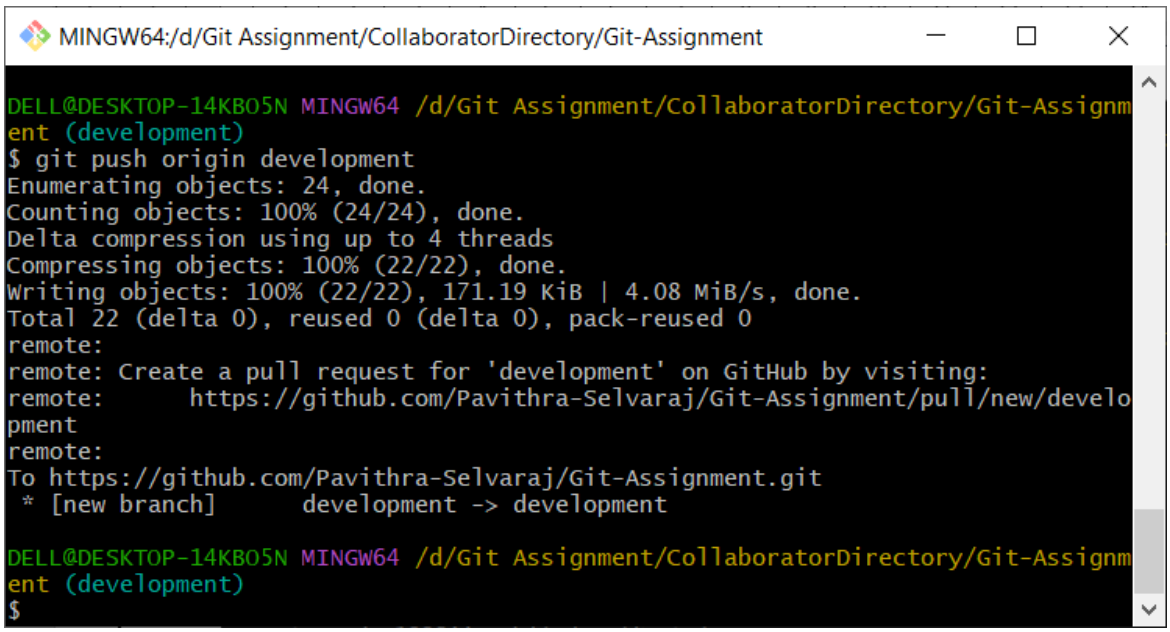
Gitignore files are something that will be part of almost every project. It’s important to ignore the correct files.

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





Now, push the code to origin development branch.



Force Push and Reset Commit

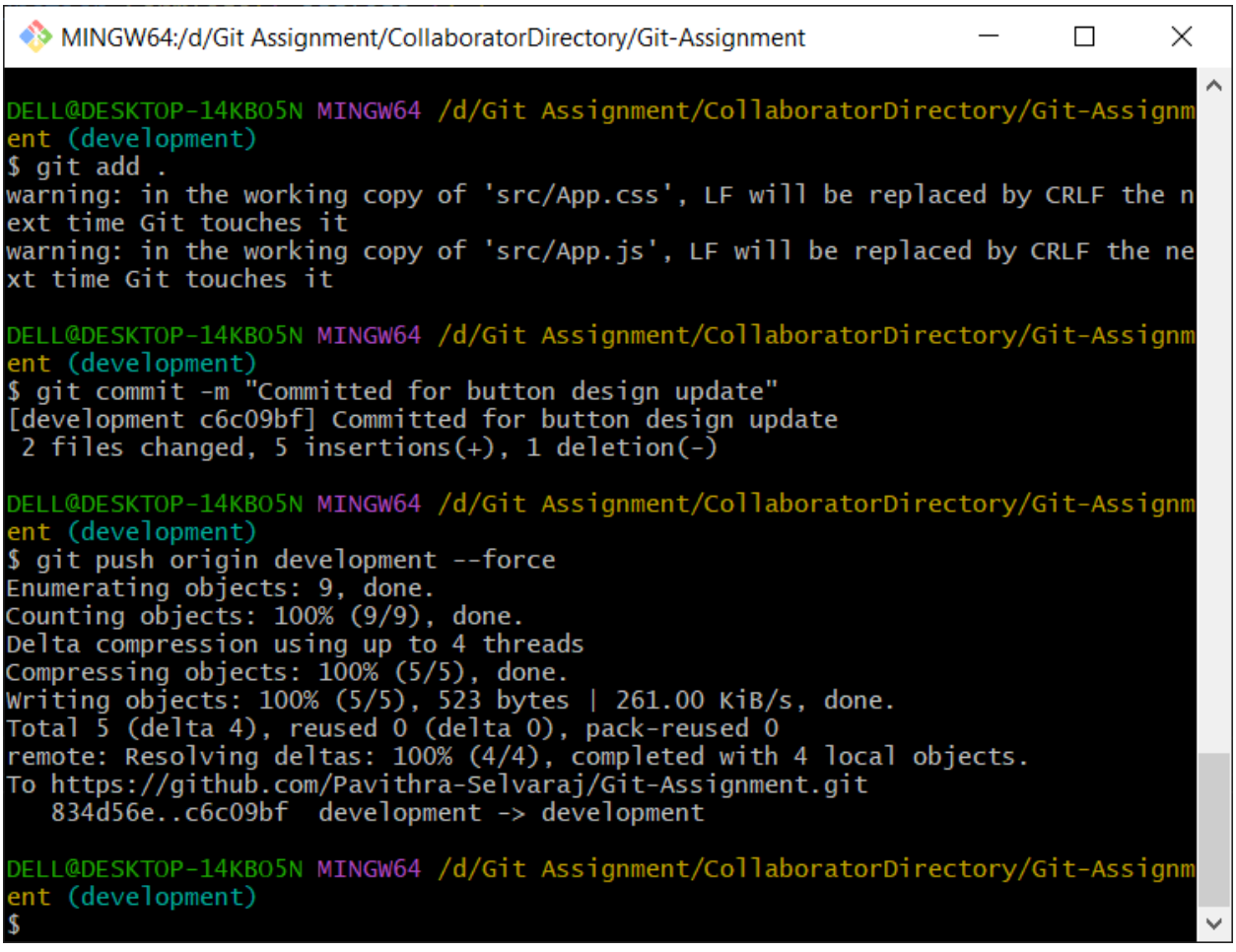
Git will normally only allow us to push the changes if we have previously updated the local branch with the latest commits from its remote counterpart. Only when we are up-to-date, we will be able to push new commits to the remote.

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.

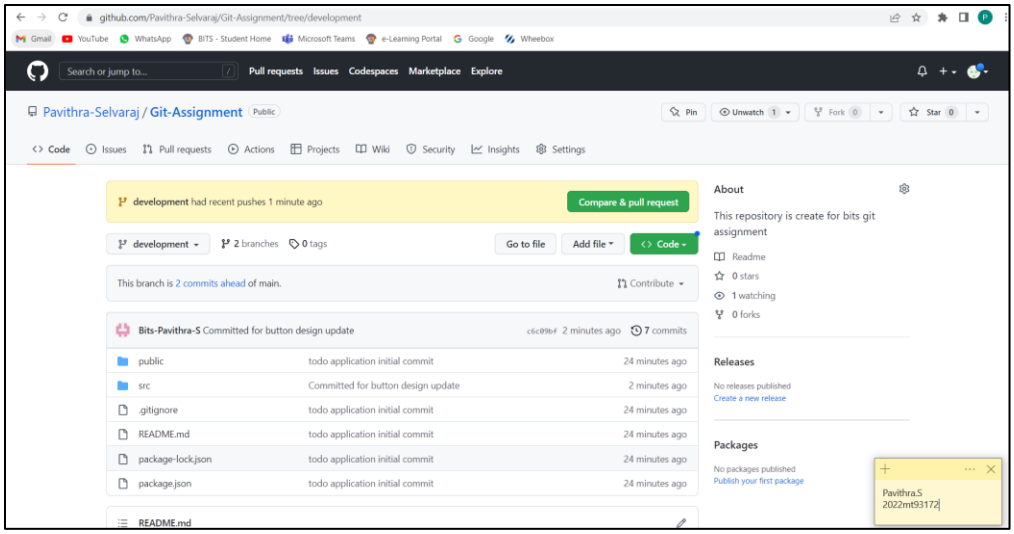
To demonstrate this, lets add a new commit to our code and force push it.

The command will be,

git push origin <Branch-Name> --force



The 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.

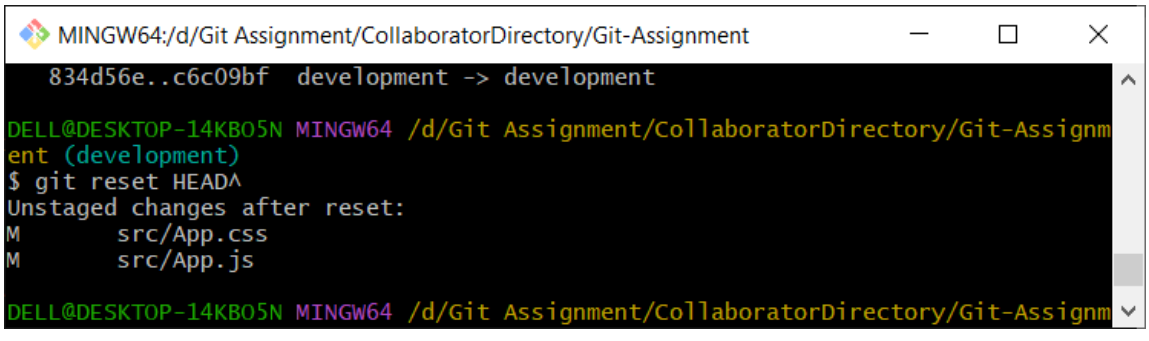
To undo the last commit locally, we can use,

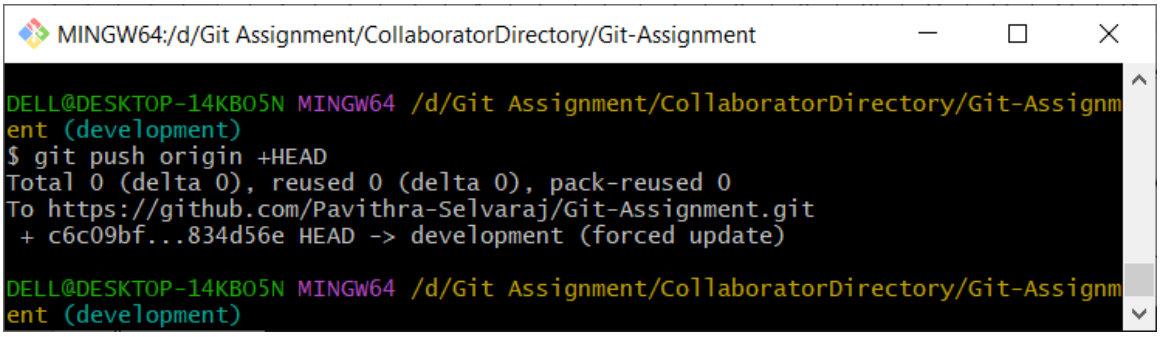
git reset HEAD^

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

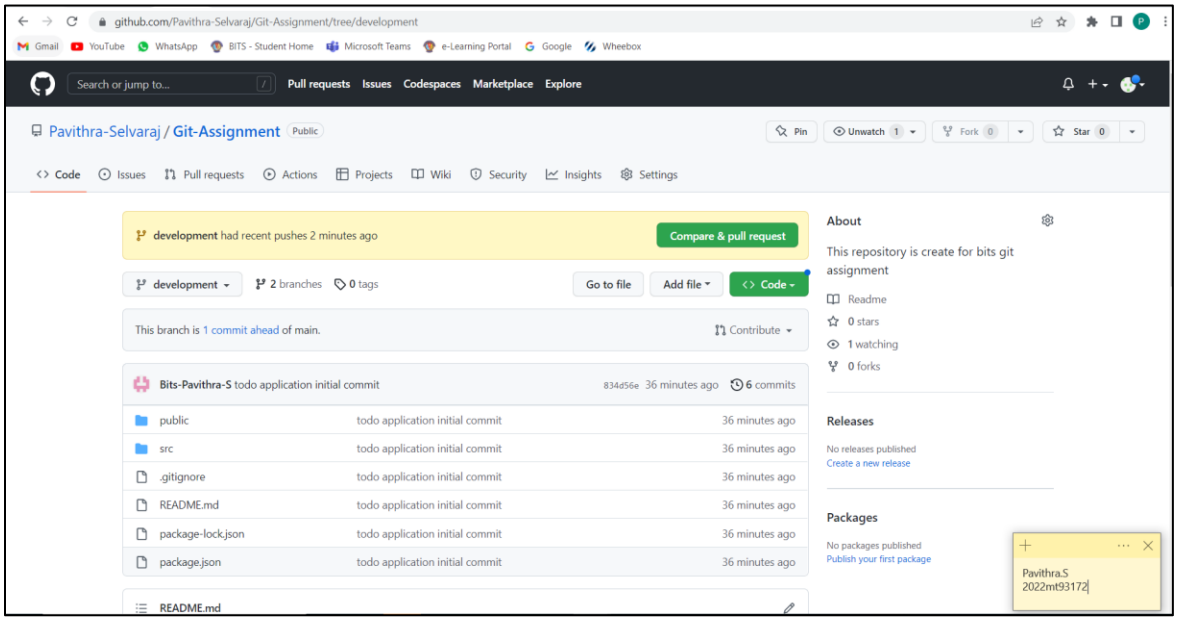
git push origin +HEAD

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

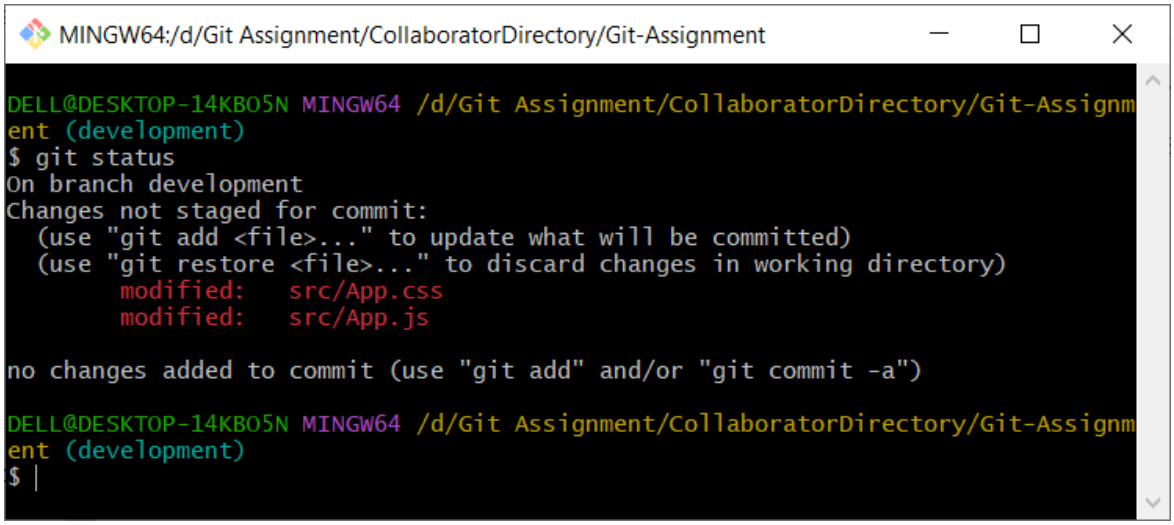




The last commit is now removed.



The changes are now revered to our working directory.



Creating Tags

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

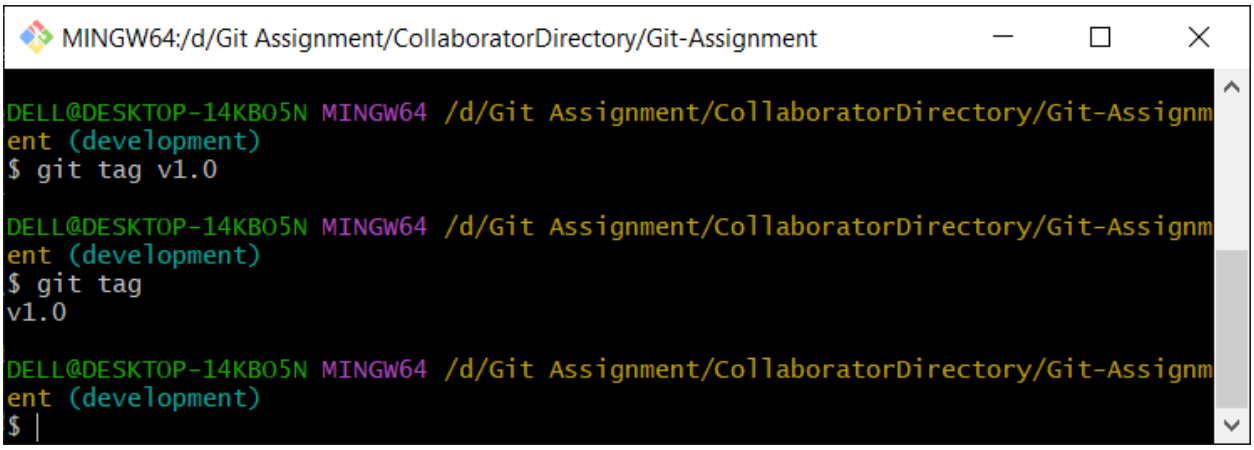
We create new Git tags in order to have a reference to a given release of our software.

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>

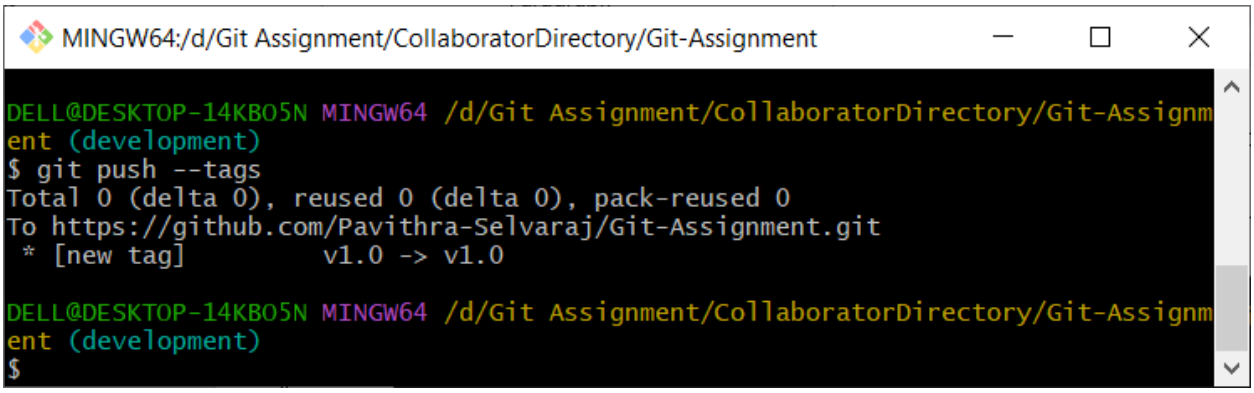
To check if the tag is created successfully, we can use the "git tag" which will list all available tags.



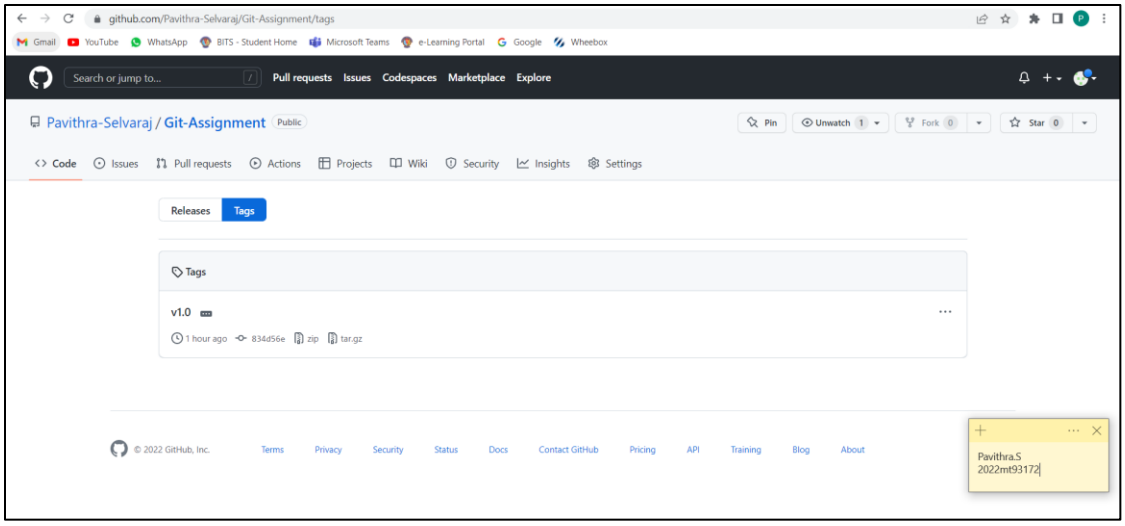
The "git push" command does not automatically push our tags to the remote repository.

We need to explicitly mention as follows,

git push--tags



The tag is now pushed to the origin.



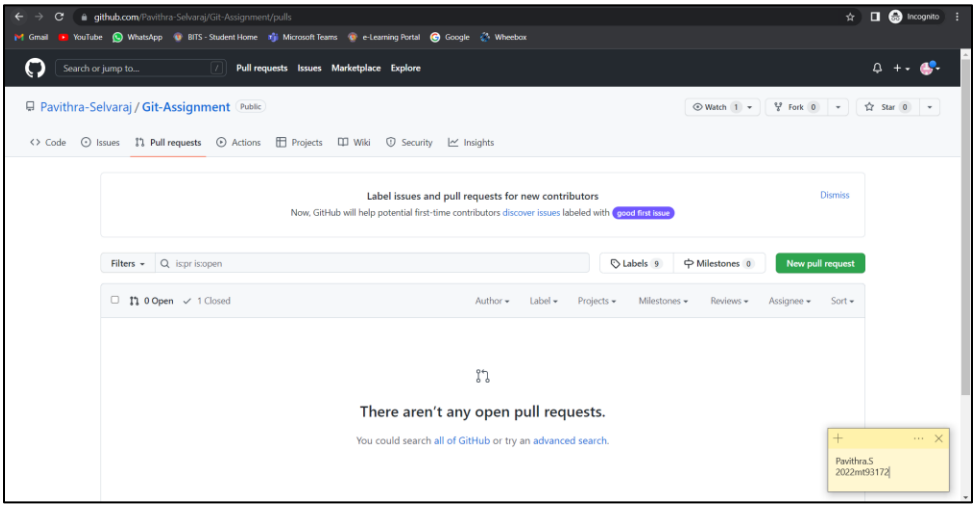
Releasing Features

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.

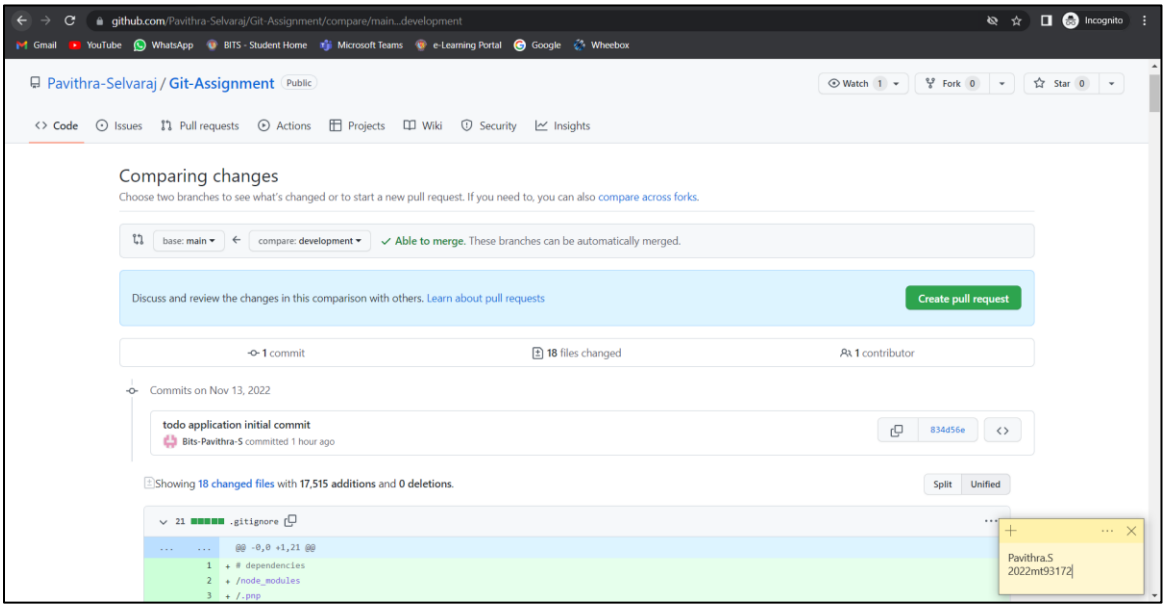
In our example here, lets consider that we are going to merge the changes from development branch to the main branch (which is the master branch in our case).

Pull Request:

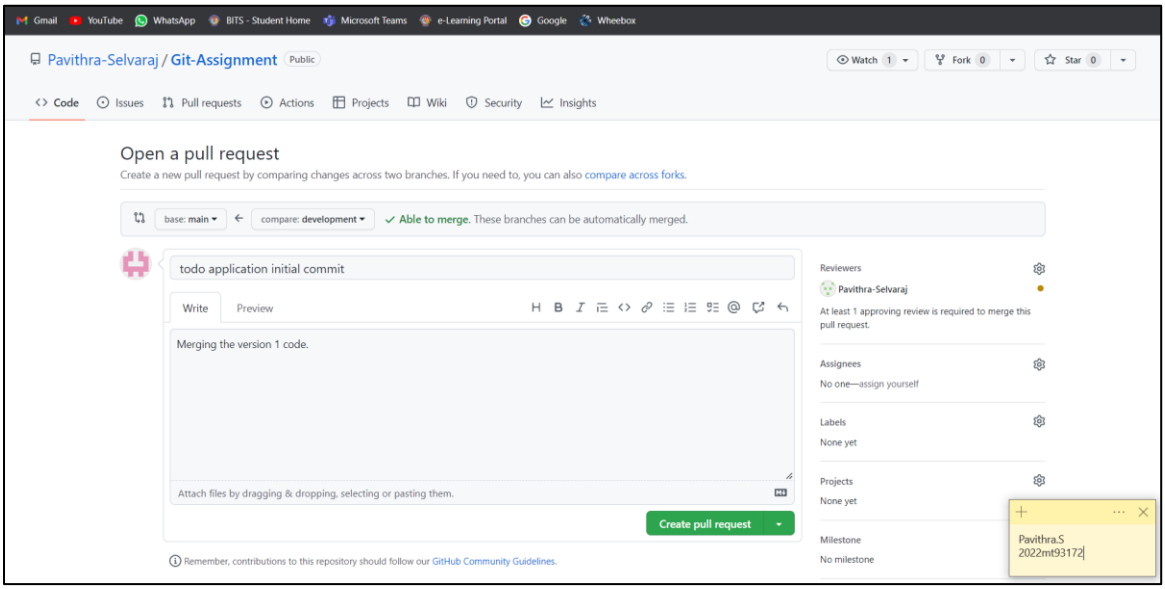
1. Click on “Pull Requests” tab in the navigation.



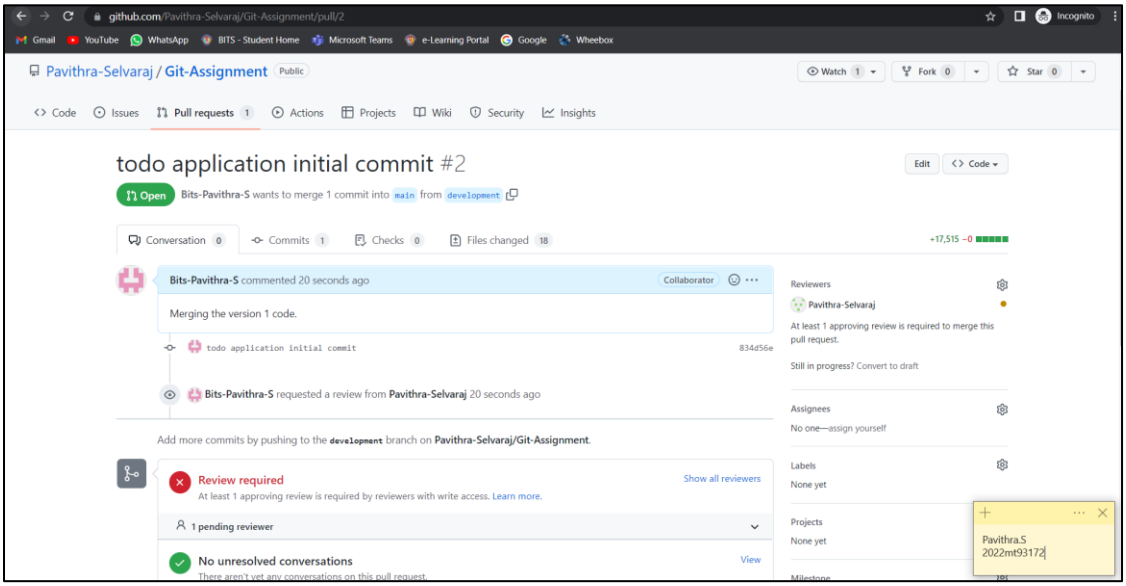
2. Click “New pull request” button. Select the branches to compare. Check the commits and click “Create pull request”.



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



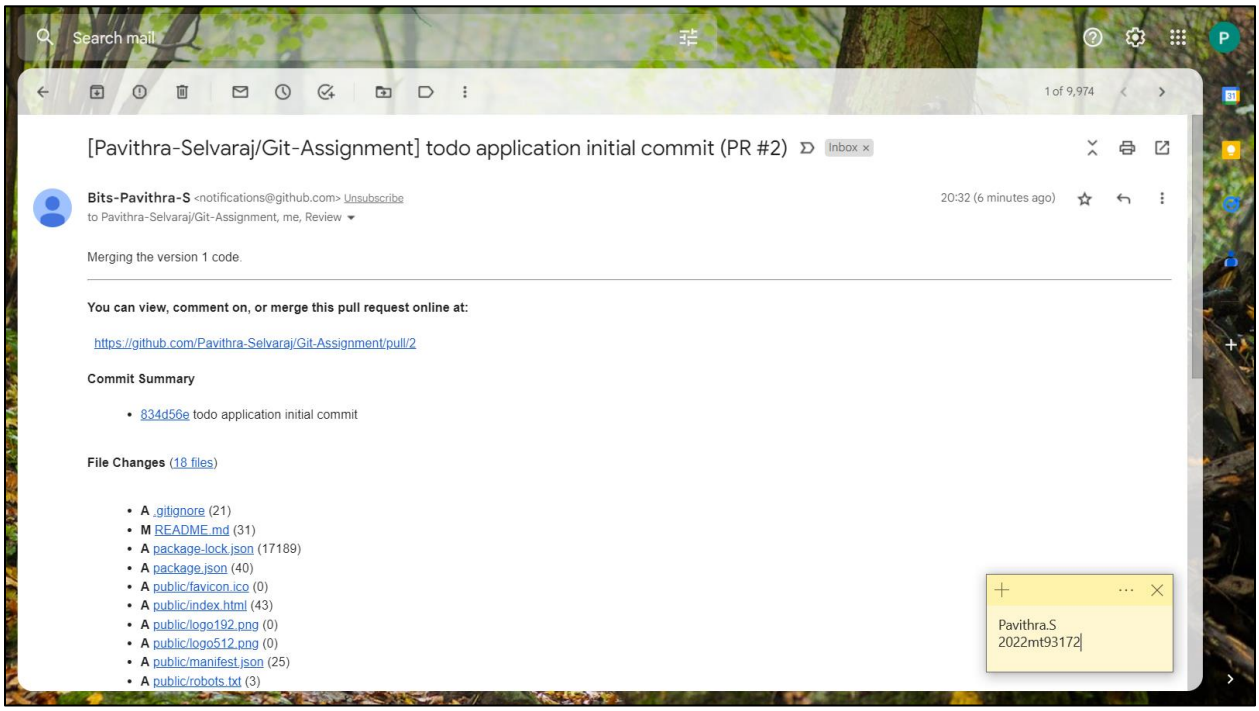
4. The pull request is created.

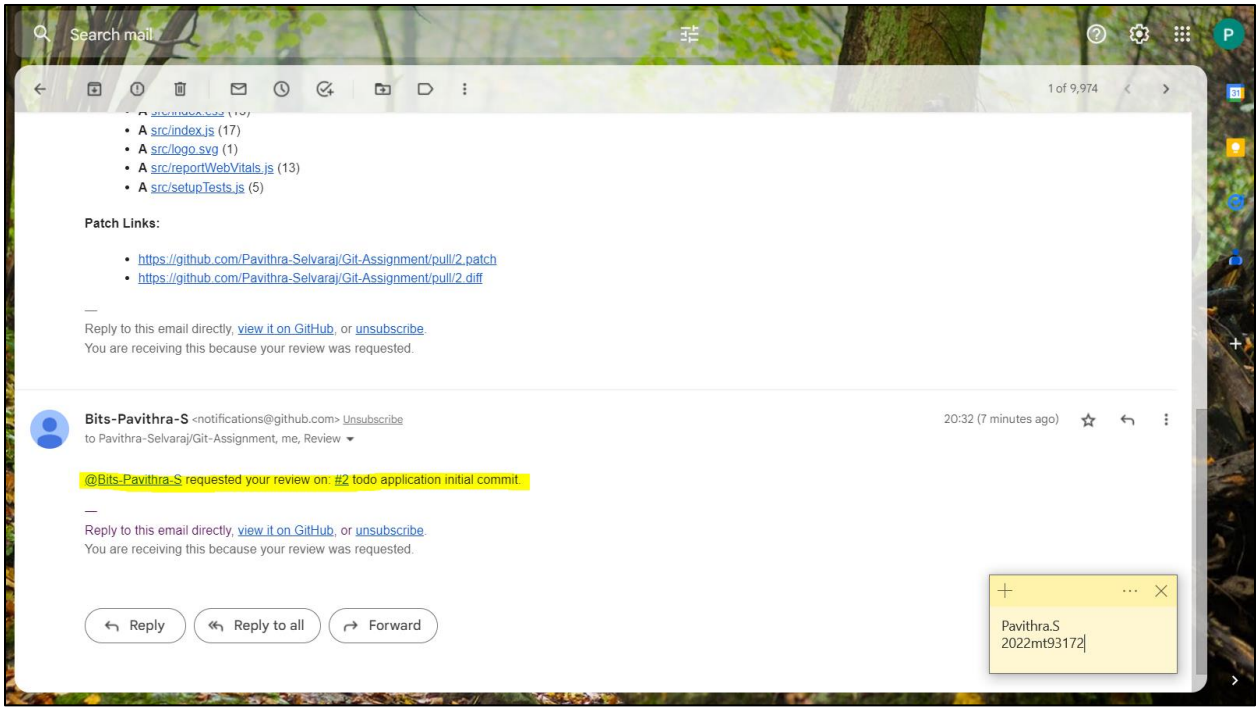


The review request is now sent to the manager (the person chosen as reviewer).

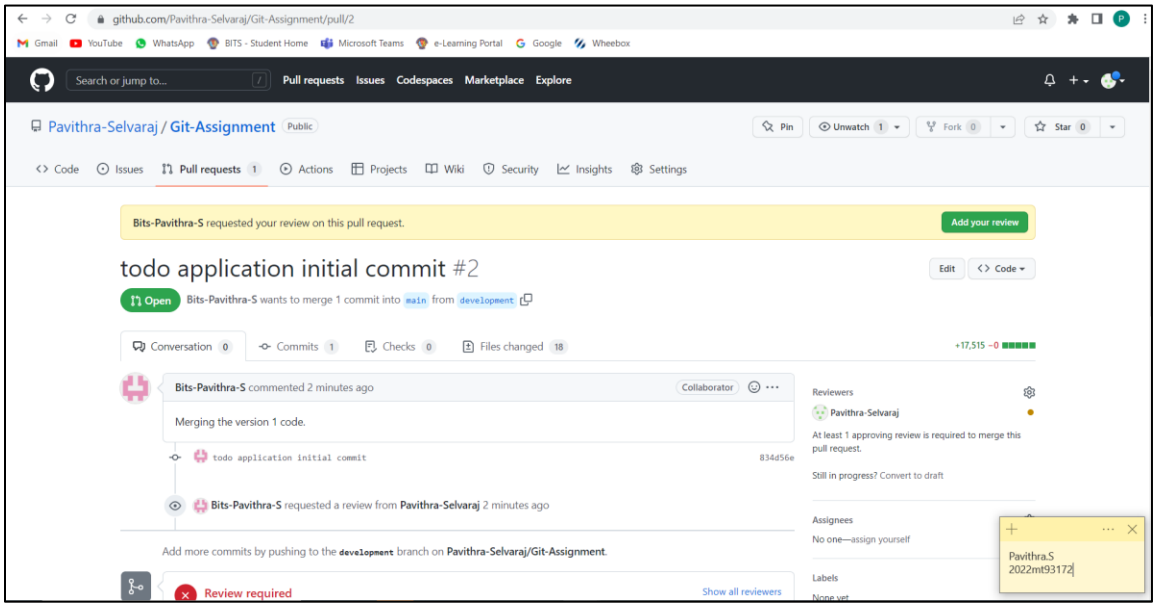
Request Approval

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





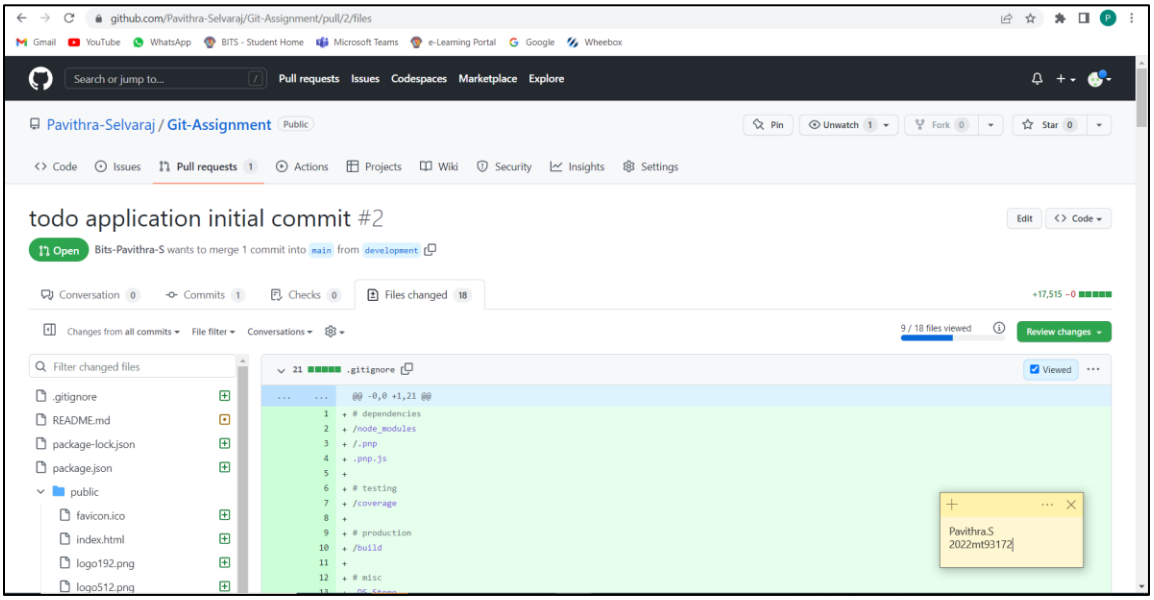
The reviewer can also see the request in the pull request tab.



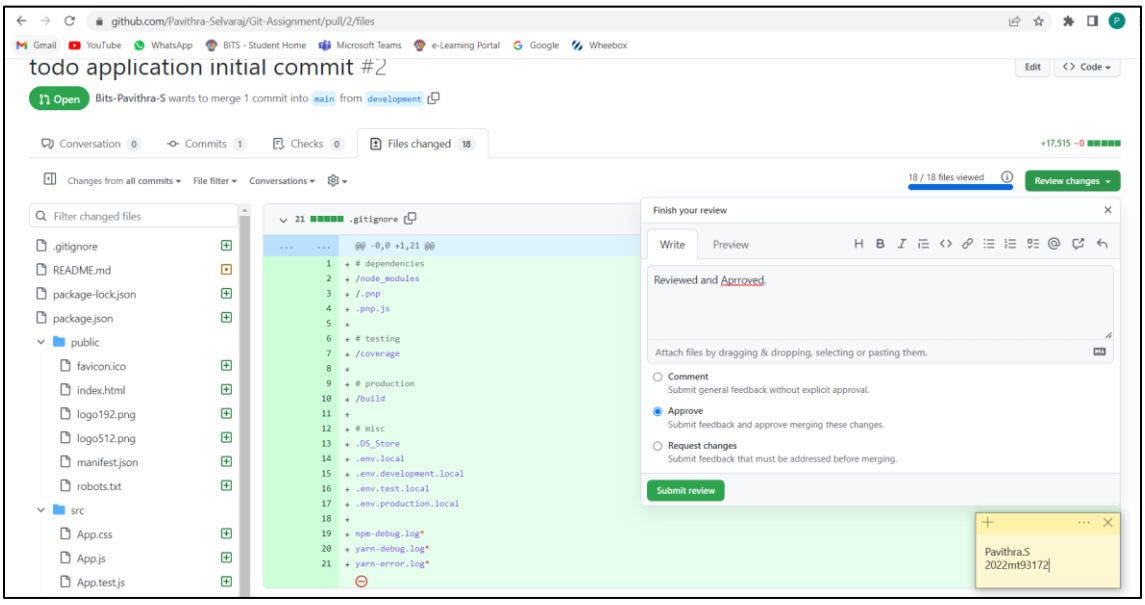
The reviewer can add review and approve the code once reviewed. If there are changes required, comments can be added and the code can be merged after the comments are resolved.

Follow the steps below to review and merge code.

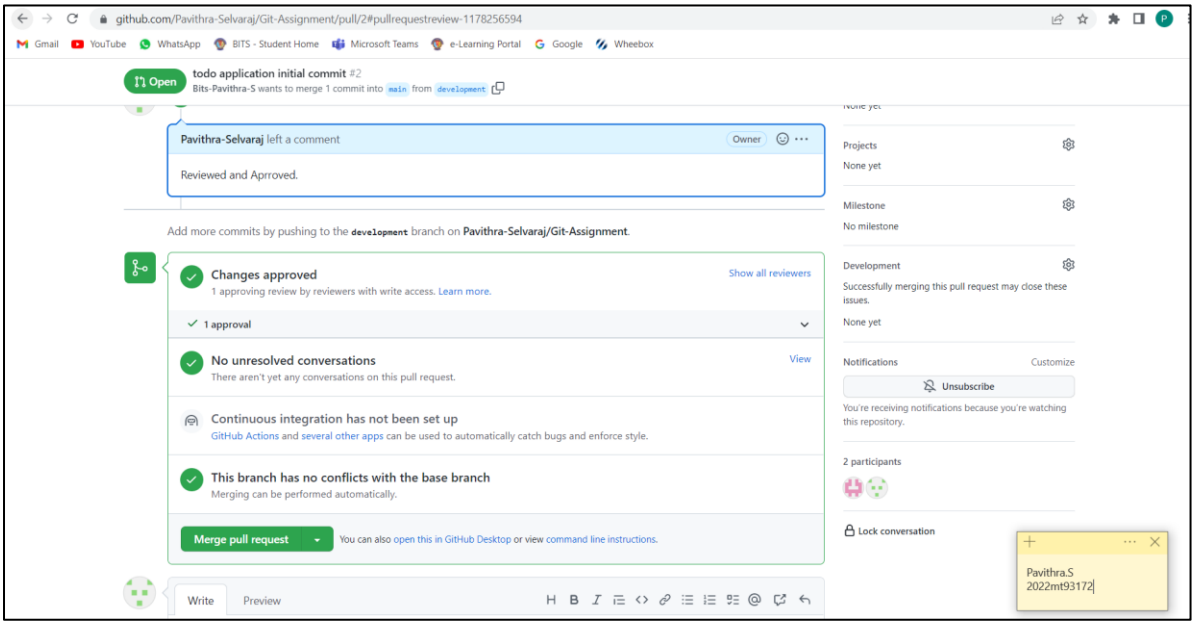
1. Click on “Add your review”.



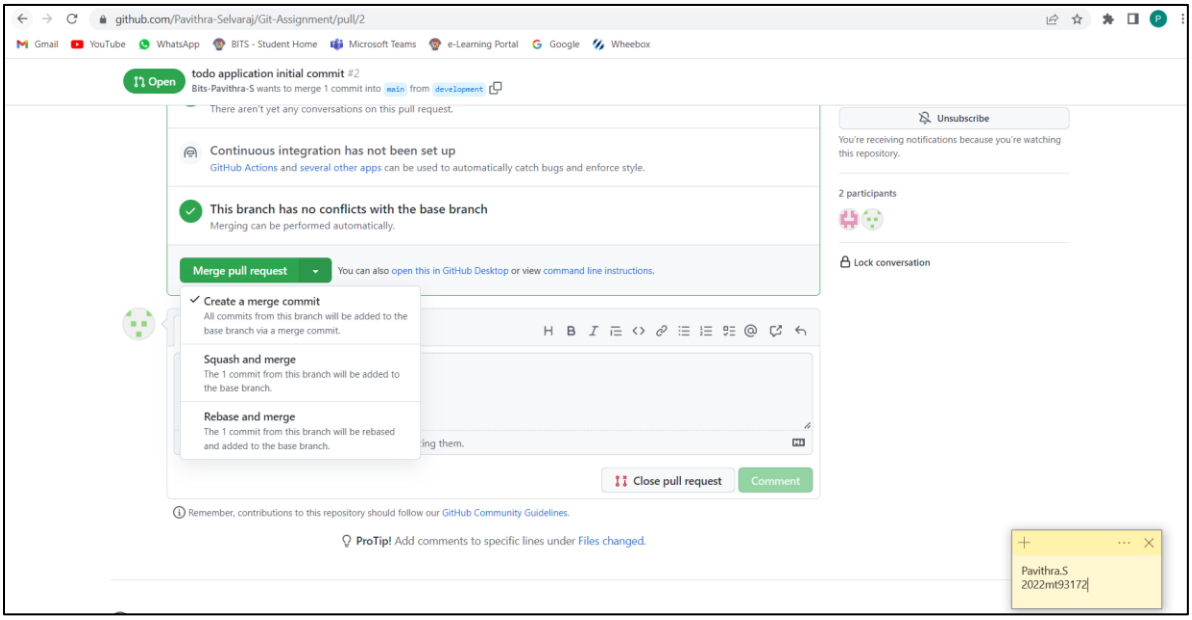
2. Review the changes and add comments if any. Else check all and click “Review changes”.



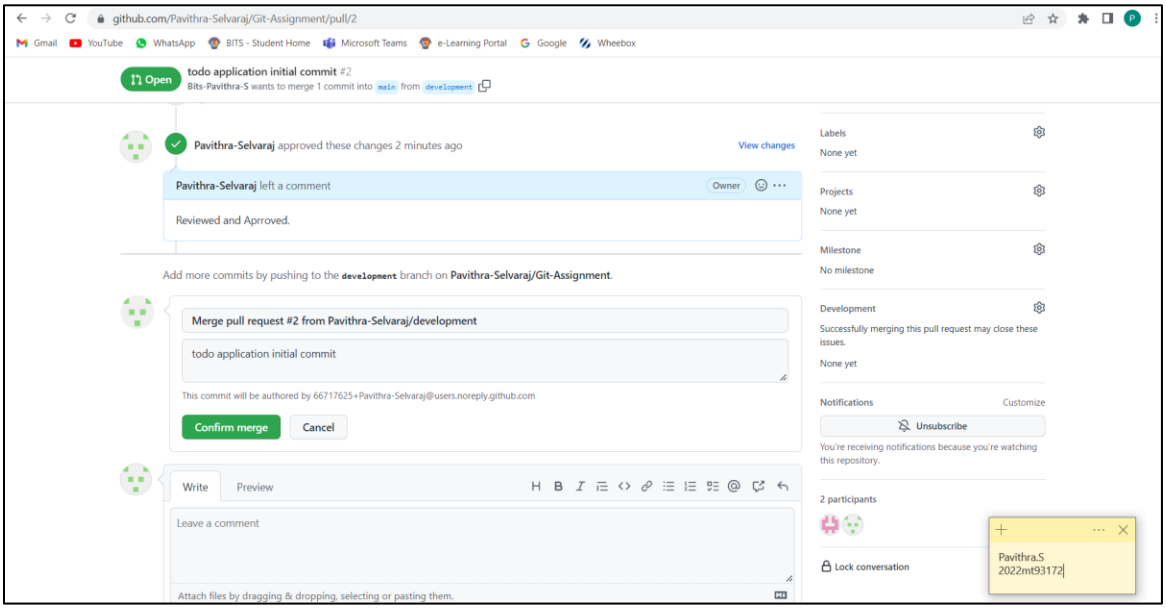
3. Click “Submit Review” with the option that suits. Here lets directly select approve and click the button.



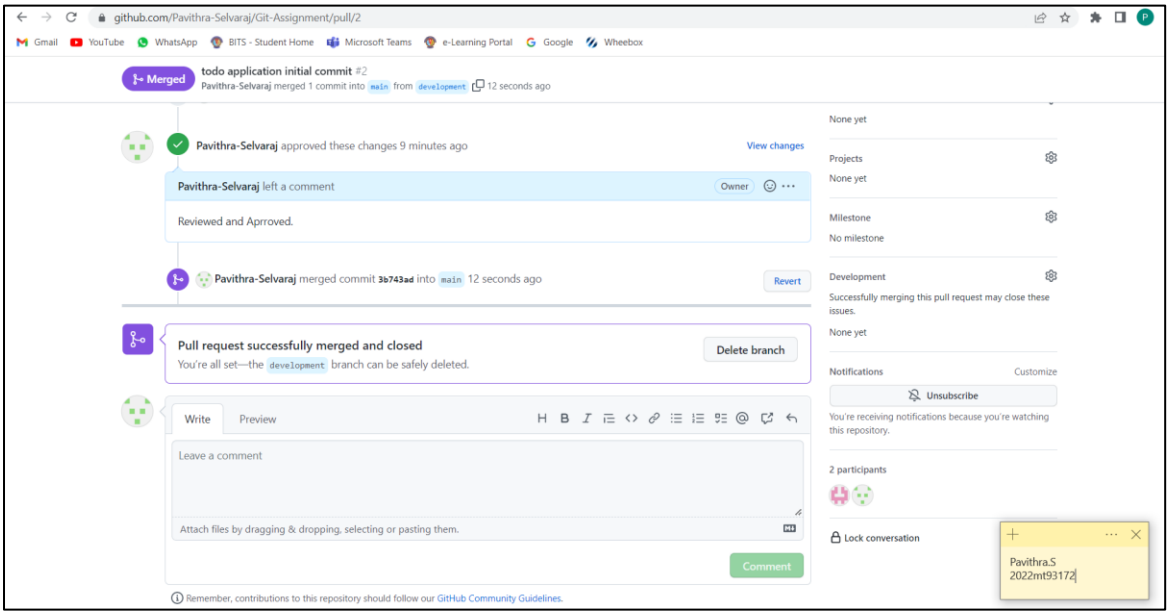
4. The code can be now merged in any one of the ways below. Select an option and click on “Merge pull request” button.



5. Click “Confirm merge”.



6. The code is successfully merged.

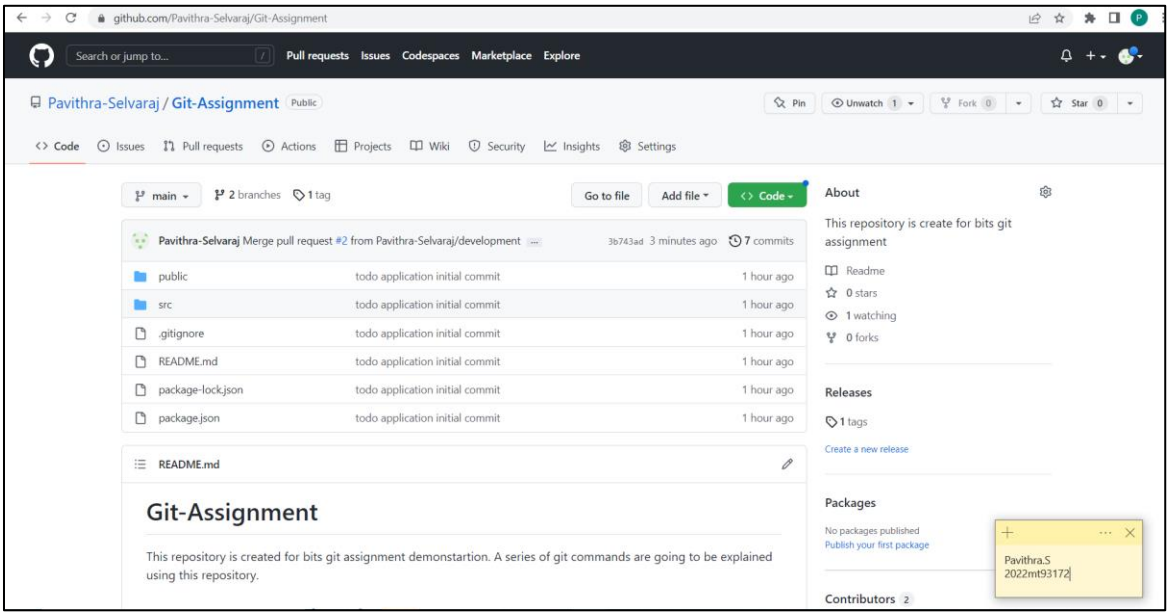


Creating a Release

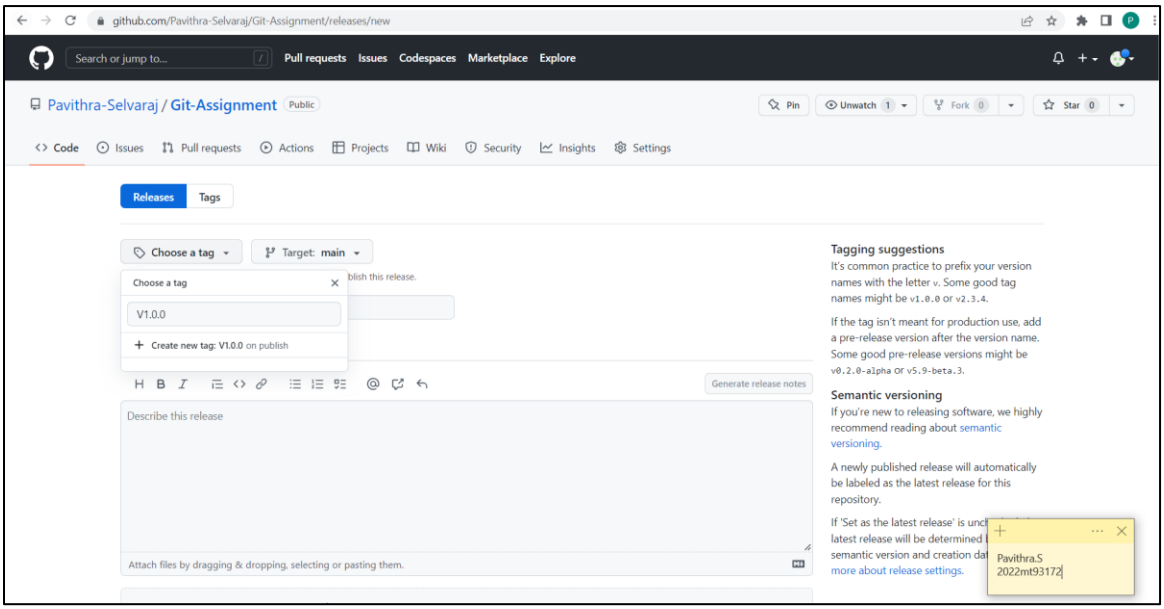
We can make a release using the release module in GitHub.

Follow the steps below,

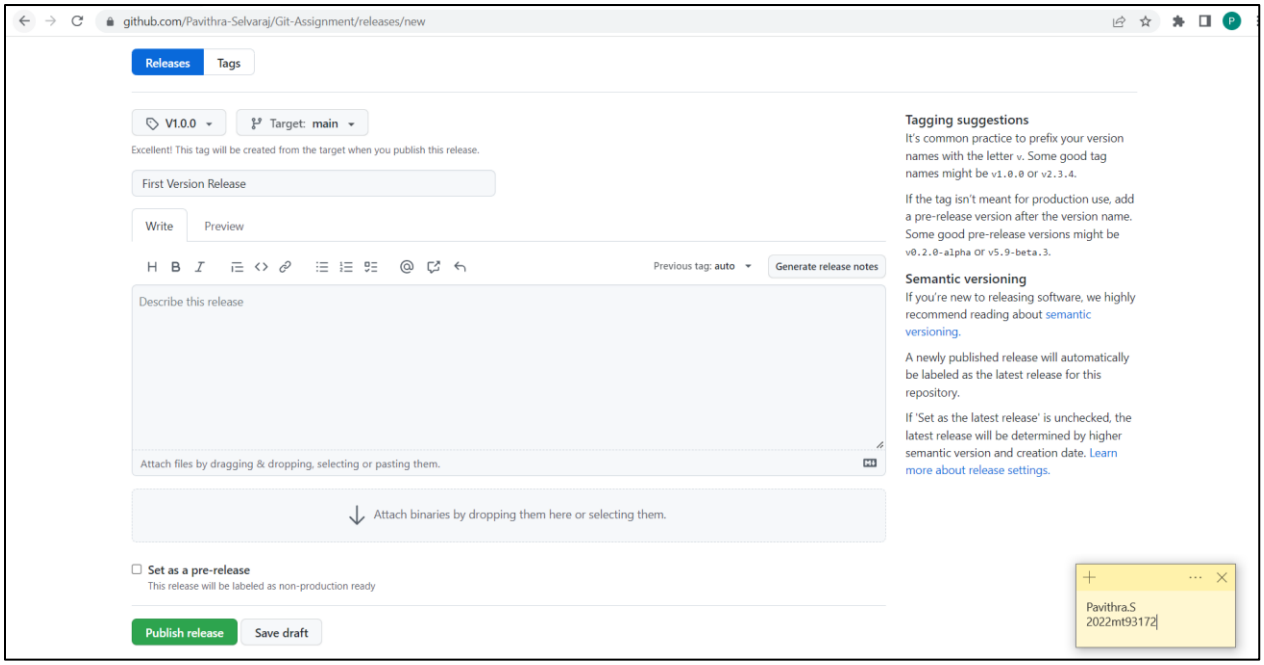
1. Click on “Create a new release” in the right option pane.



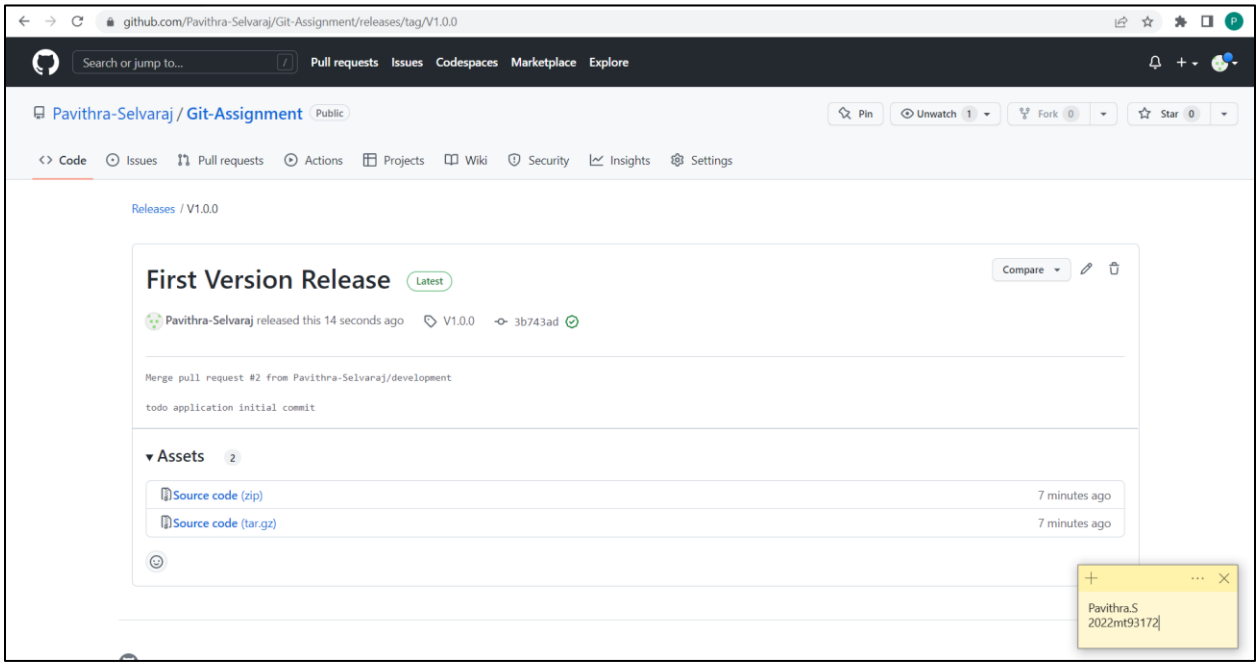
2. Click on choose a tag and select a new tag or an existing one.



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 then to origin.
- Releasing features with versions.