

Connecting ECLIPSE and GitHub



© Manish Jaya Devadiga
manishjayadevadiga@gmail.com

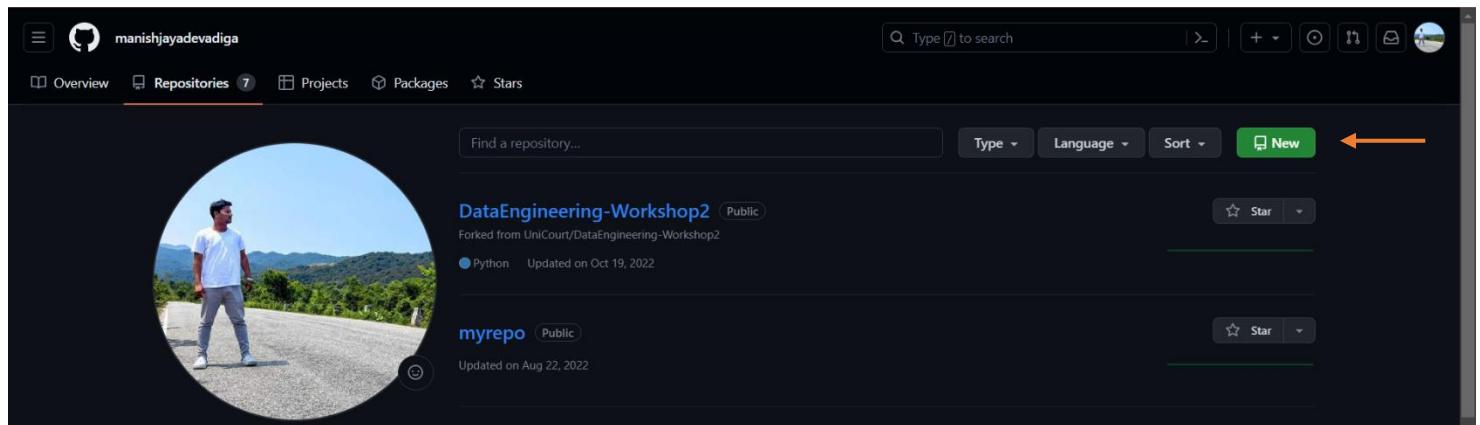
Note: Here we will see how to connect “Eclipse” and “GitHub” and upload java programs from eclipse.

A written connection pdf is also uploaded in the repository.

Now follow the below steps correctly for smooth and error free connection.

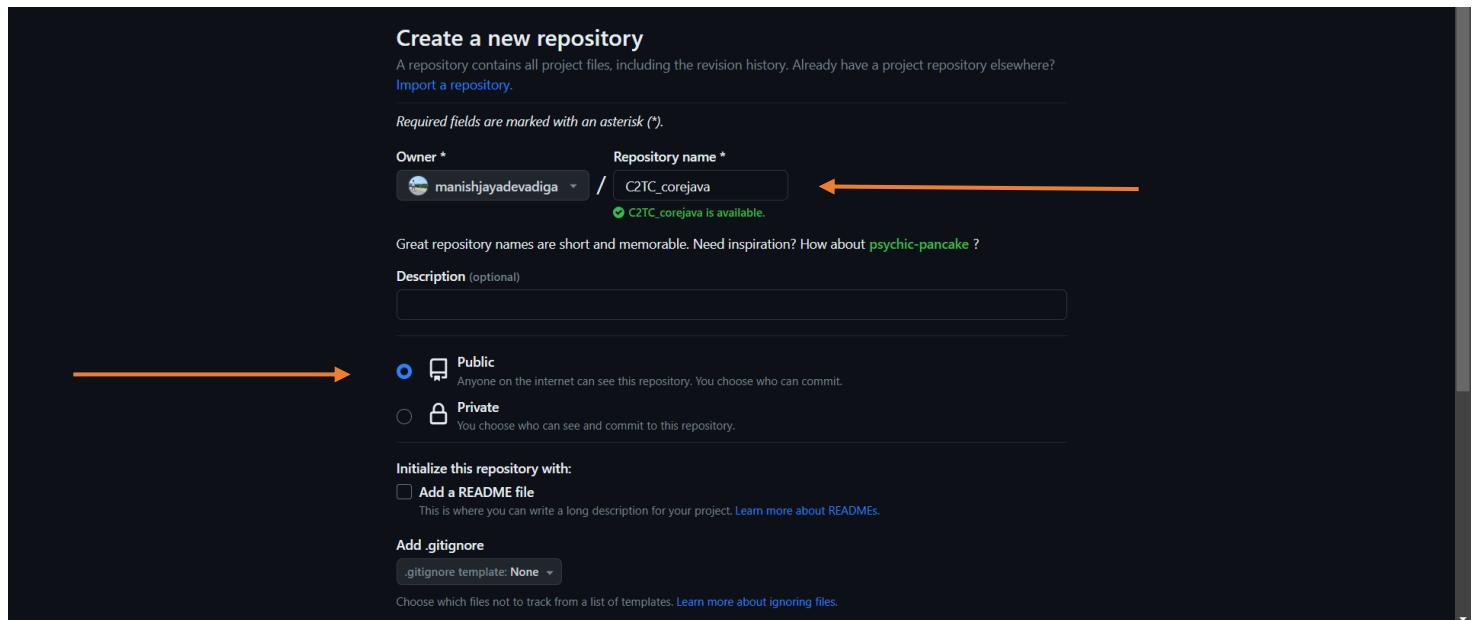
GitHub:

- Open GitHub and [create your account](#).
- After account creation, click on “[My repository](#)”.
- Click “[New Repository](#)”.



The screenshot shows the GitHub homepage with a dark theme. At the top, there's a navigation bar with links for Overview, Repositories (7), Projects, Packages, and Stars. A search bar is at the top right. Below the navigation, there's a section for finding repositories. Two repositories are listed: "DataEngineering-Workshop2" (Public) and "myrepo" (Public). An orange arrow points to the "New" button in the top right corner of the main content area.

- Now give name of repository as “[C2TC_corejava](#)” and set it as [public](#).

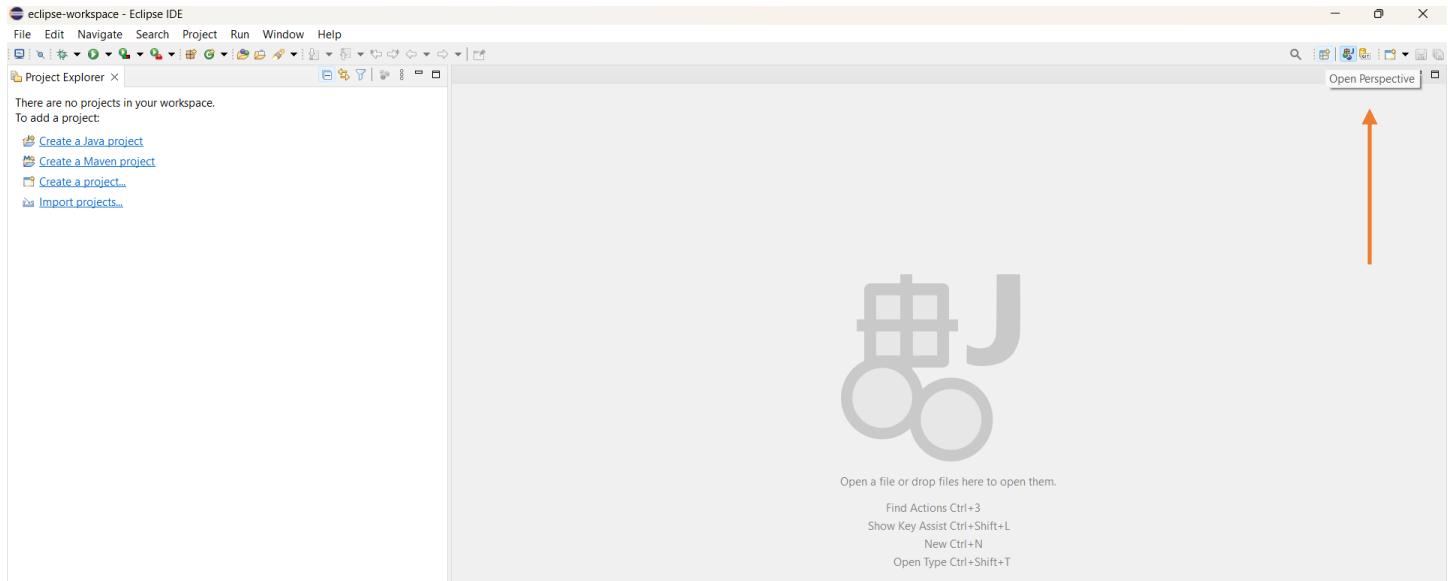


The screenshot shows the "Create a new repository" form on GitHub. It has a dark background. At the top, it says "Create a new repository". Below that, there's a note about repository contents and an "Import a repository" link. The "Required fields are marked with an asterisk (*)." section includes "Owner" (set to "manishjayadevadiga") and "Repository name" (set to "C2TC_corejava"). An orange arrow points to the "Repository name" field. Another orange arrow points to the "Public" radio button, which is selected. The "Description (optional)" field is empty. Under "Initialize this repository with:", there's a checkbox for "Add a README file" which is unchecked. The "Add .gitignore" section shows ".gitignore template: None".

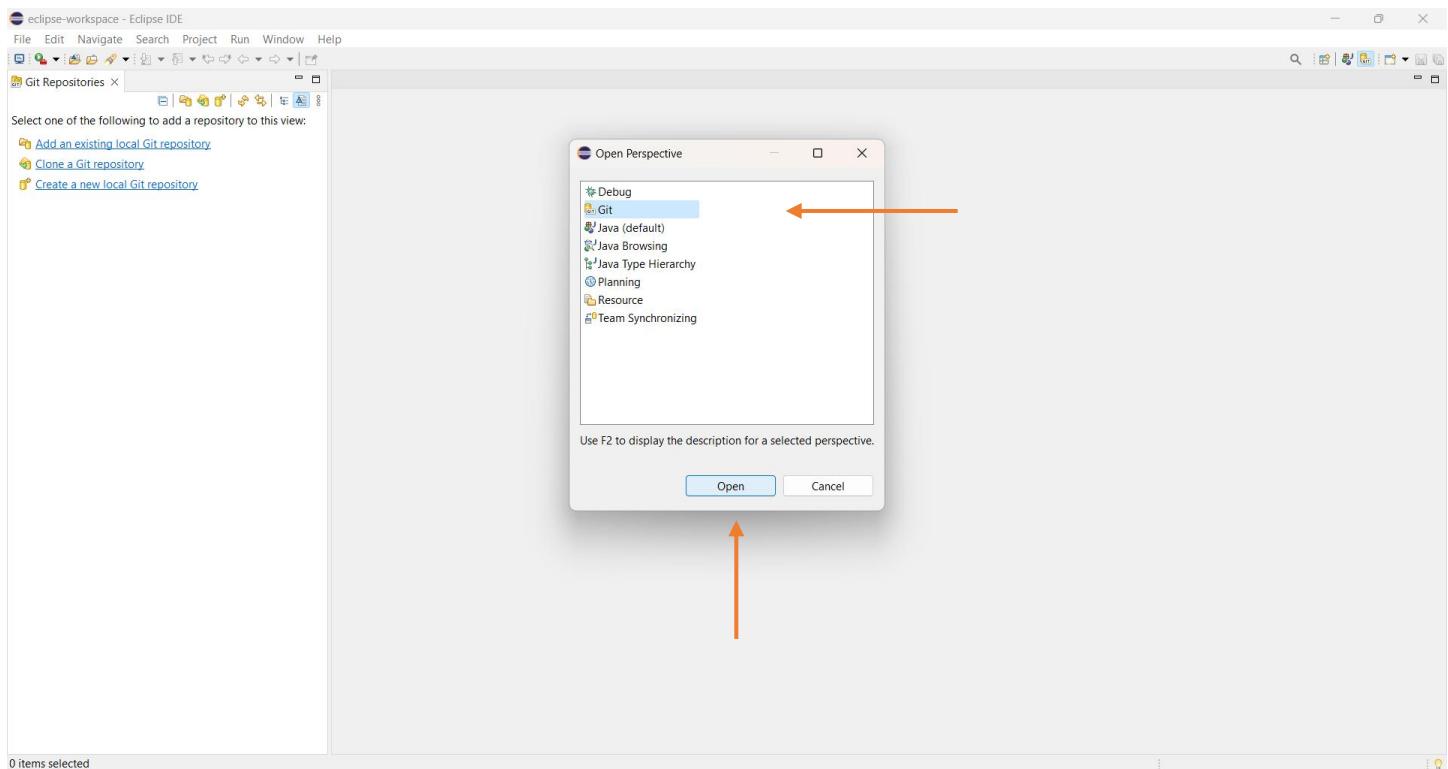
- Now click “**Create Repository**”.

Eclipse:

- Now open Eclipse.
- In the top right click “**Open Perspective**”



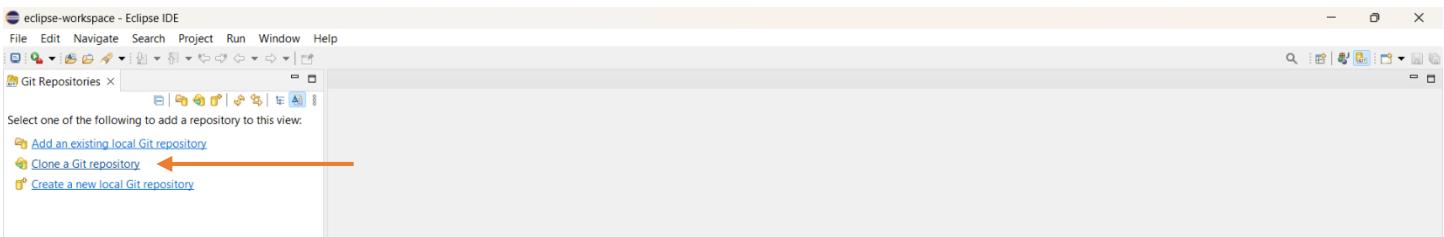
- Click on “**Git -> Open**”



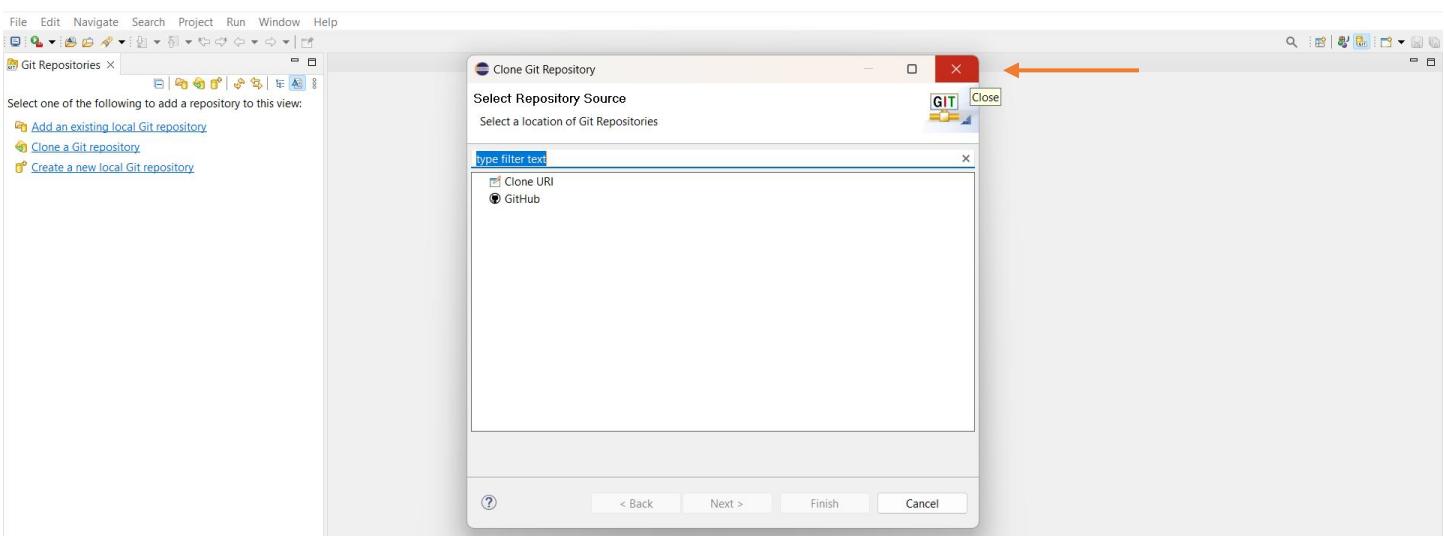
- Now click “**Windows -> Show View -> Git Repositories**”



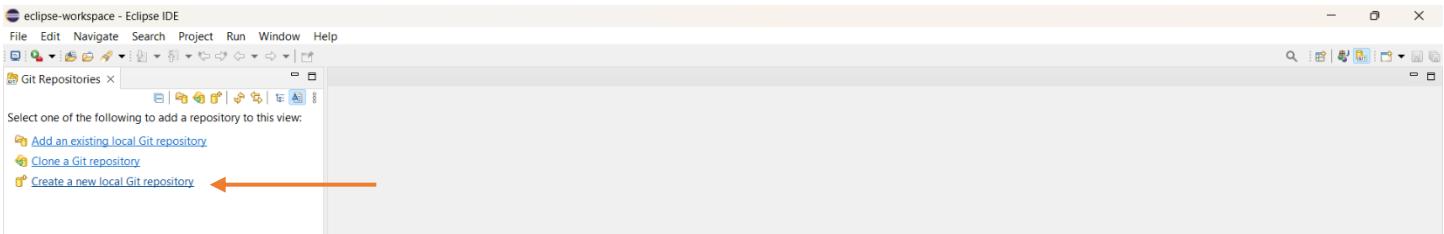
- Now in top-left corner click “**Clone a git repository**”



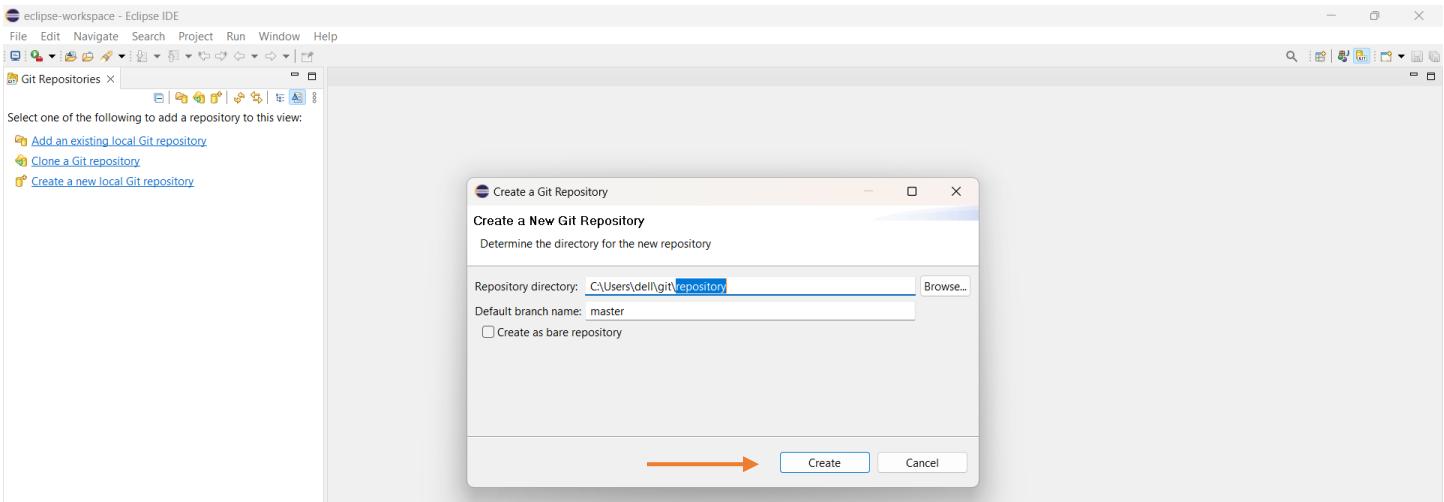
- Close** the dialog box.



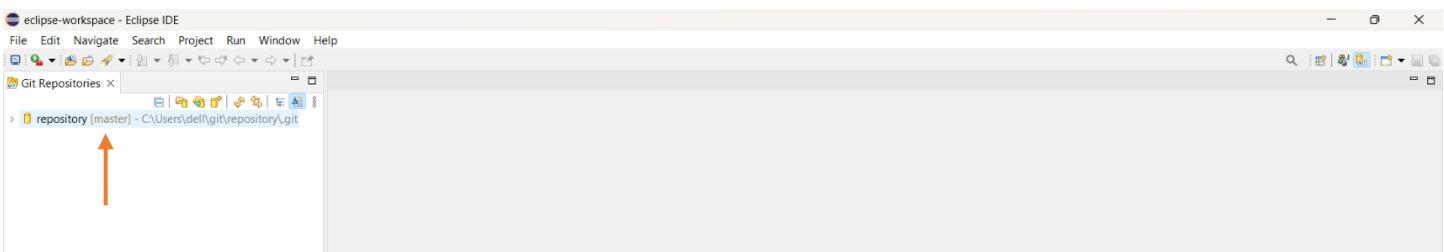
- Click on “Create a new local Git Repository”



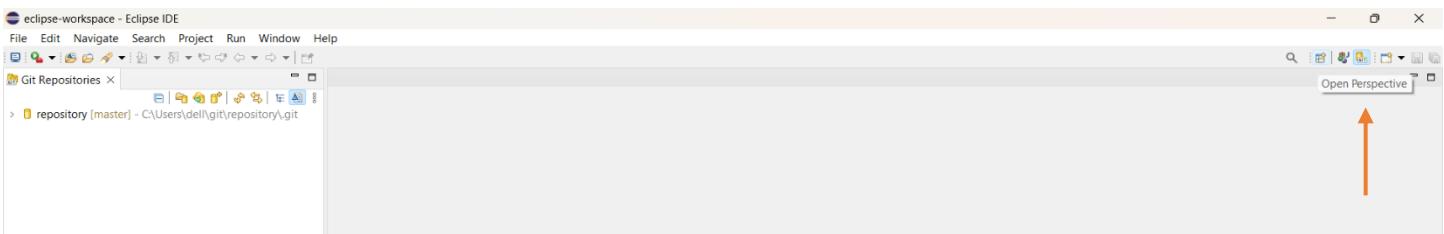
- Now leave everything as-it-is. Click “Create”.



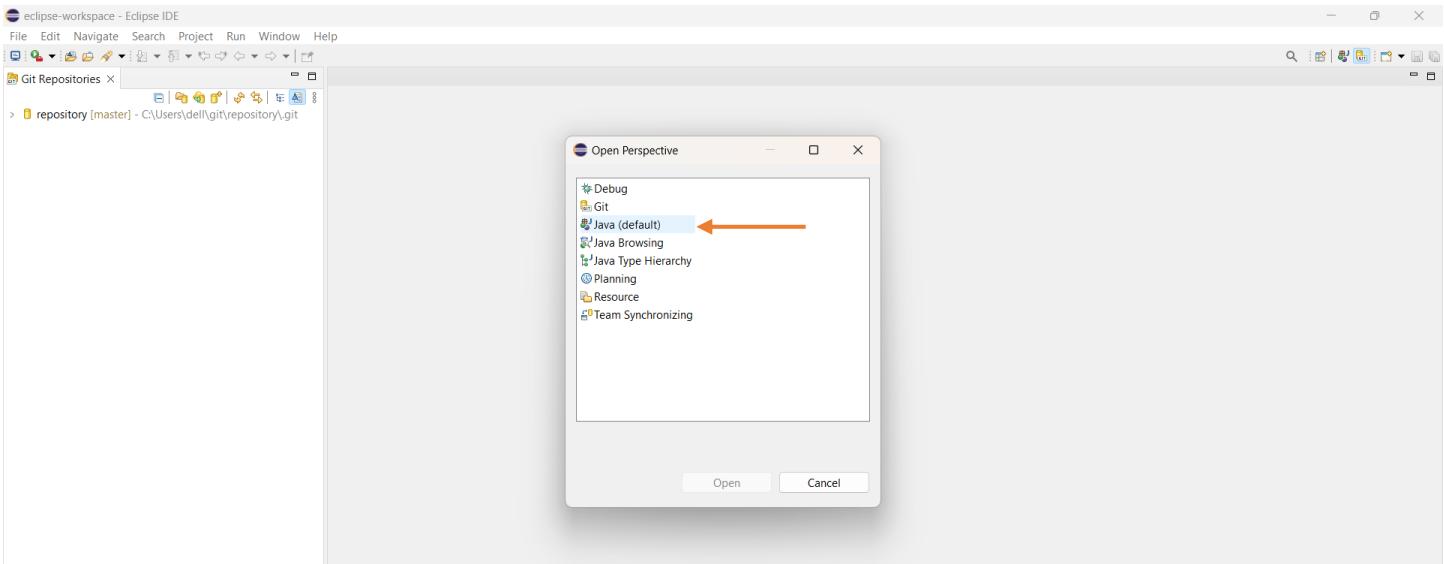
- Now in top left corner you can see “repository[master]”. Leave as it is.



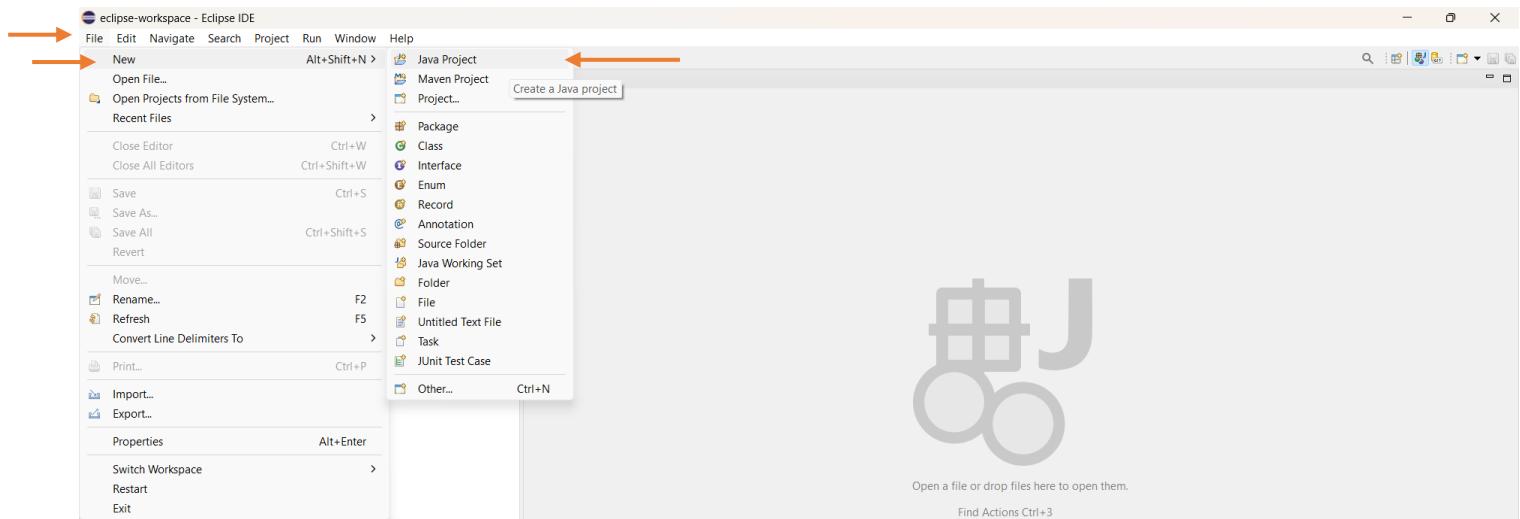
- Now click “Open Perspective”.



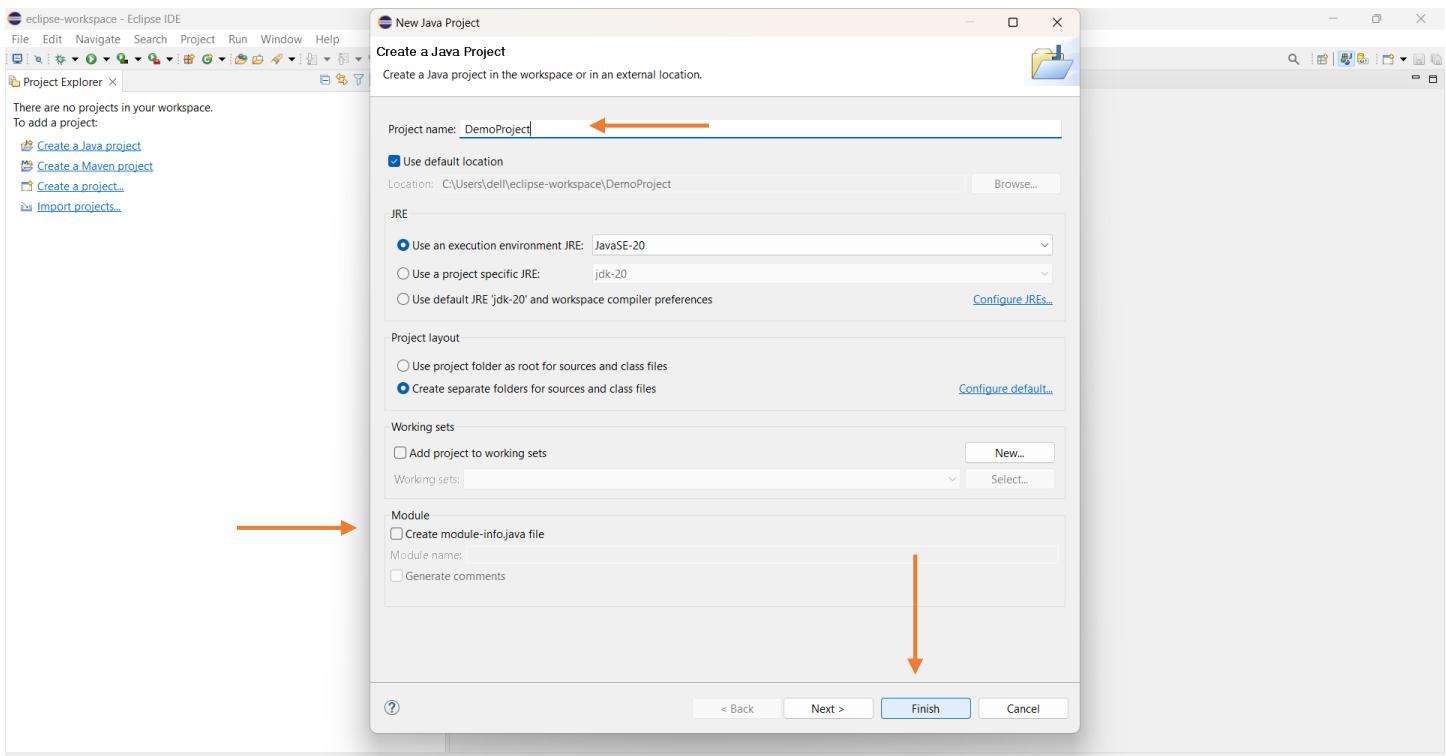
- Click “**Java(default)**” and “**Open**”.



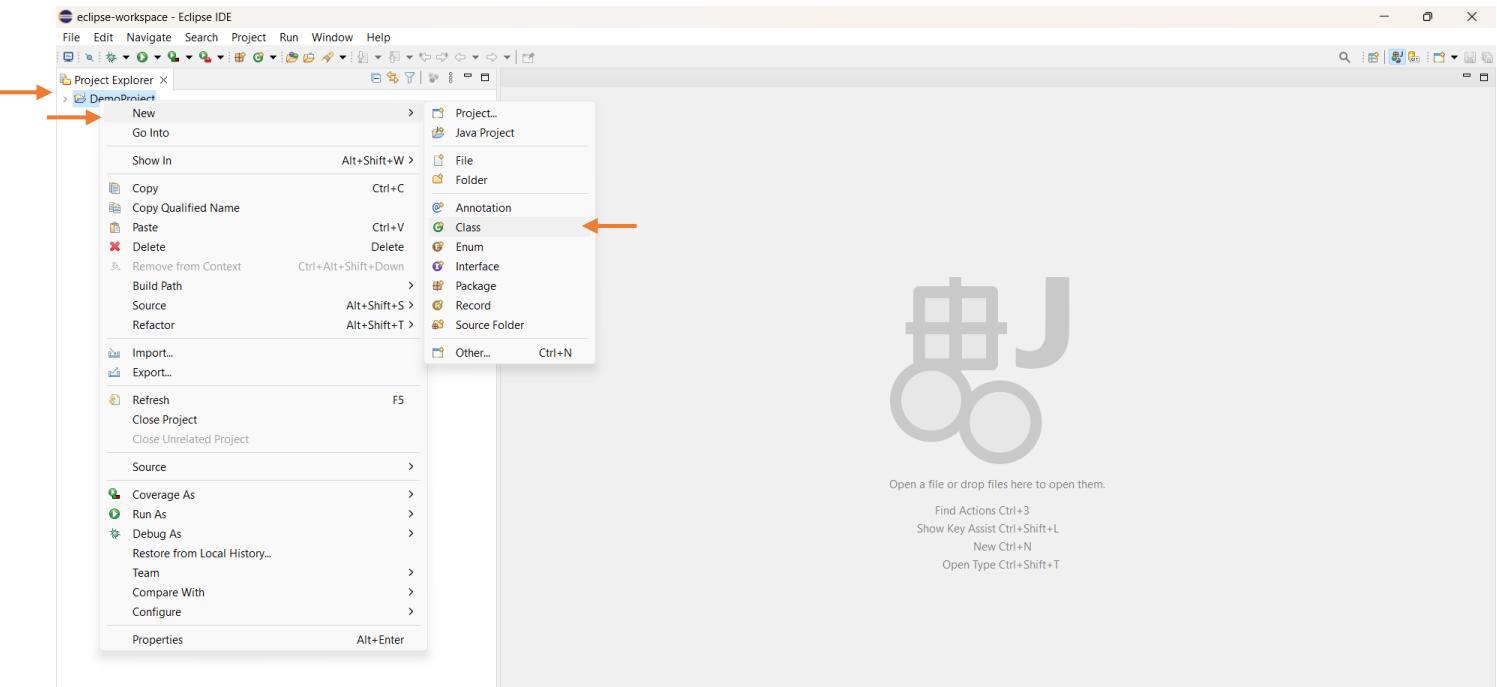
- Now we will **create Projects** to push to the GitHub Repository.
- Click on “**File -> New -> Java Project**”.



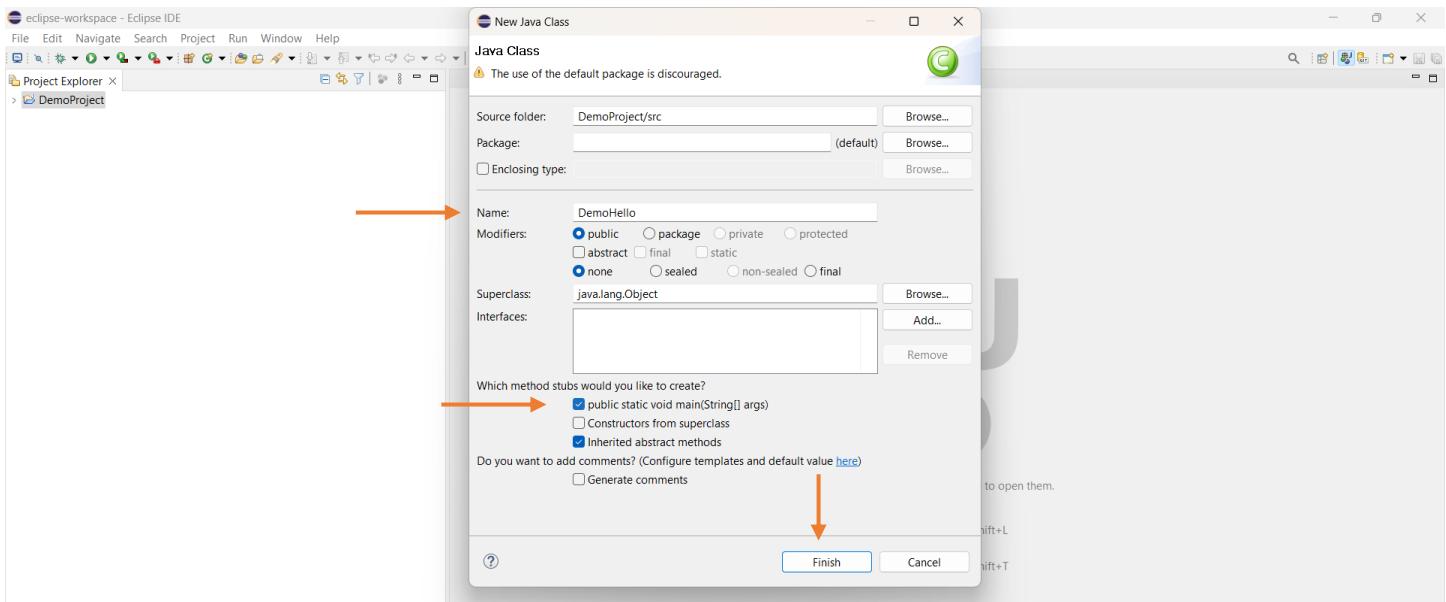
- Give the name of project as “**DemoProject**”.
- Uncheck “**Module-info.java**”
- Click “**Finish**”.



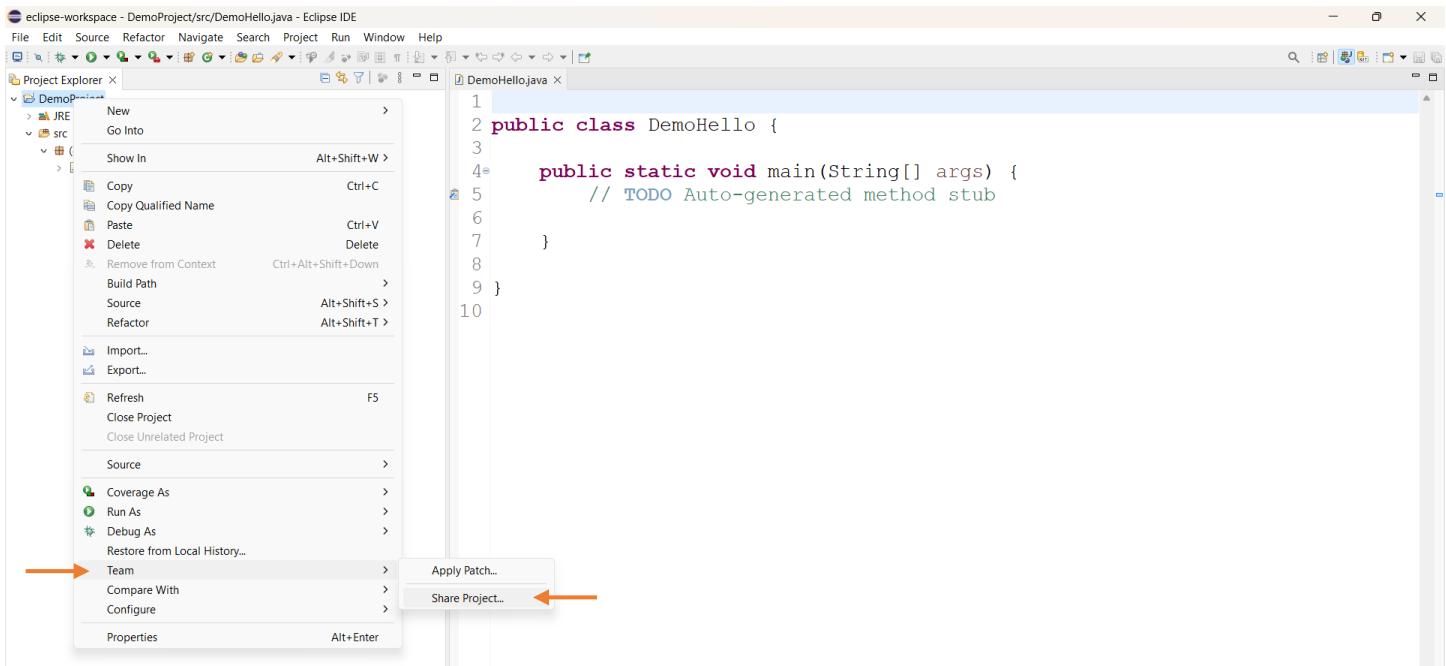
- Now let us create class inside this DemoProject.
- Right-click on “DemoProject -> New -> Class”.**



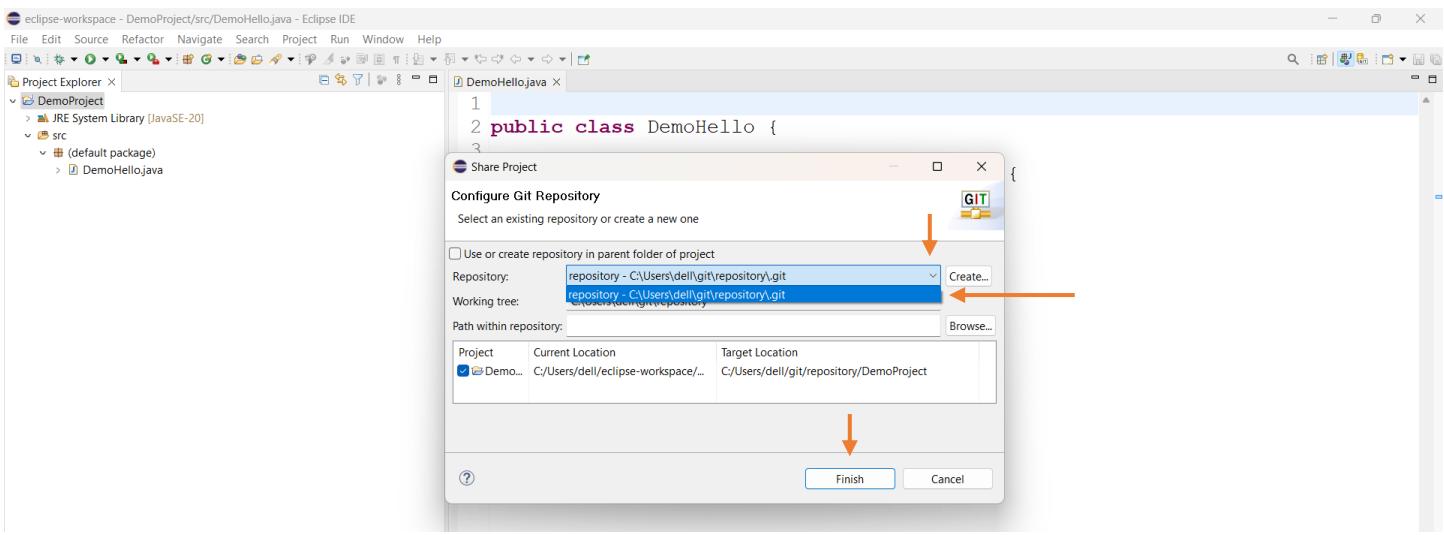
- Name the class as “**DemoWorld**”
- Check the “**public static void main (String args[])**”
- Click “**Finish**”.



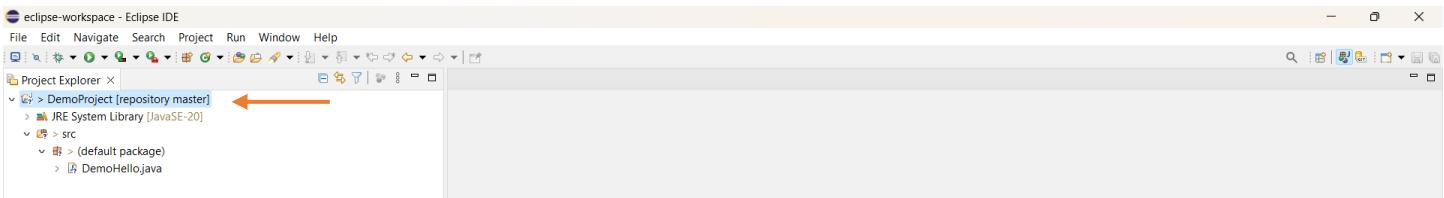
- Now write a small **Hello World** Program inside this.
- Now let's upload it to GitHub.
- Right click on “**DemoProject -> Team -> Share Project**”.



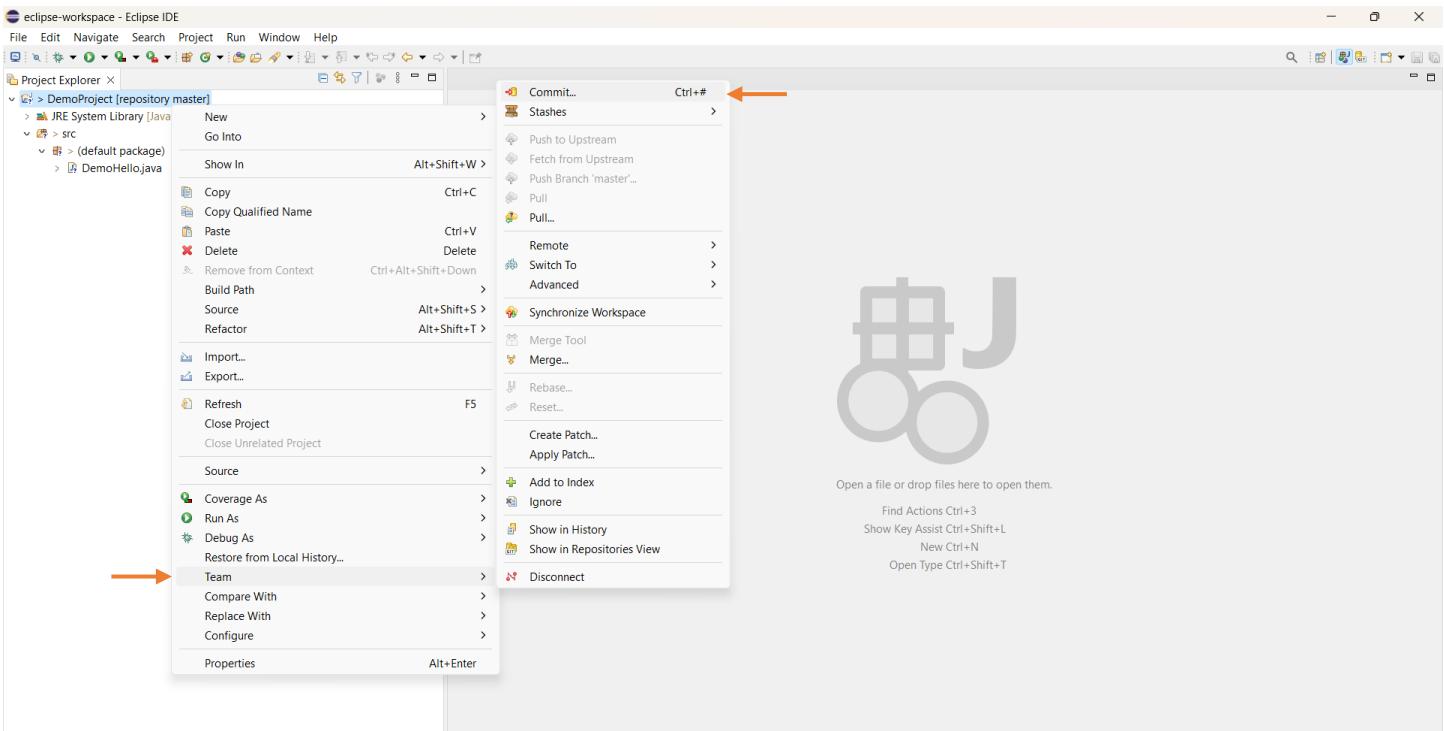
- Select **Repository** from dropdown and select the one which is available.
- Now Click “**Finish**”.



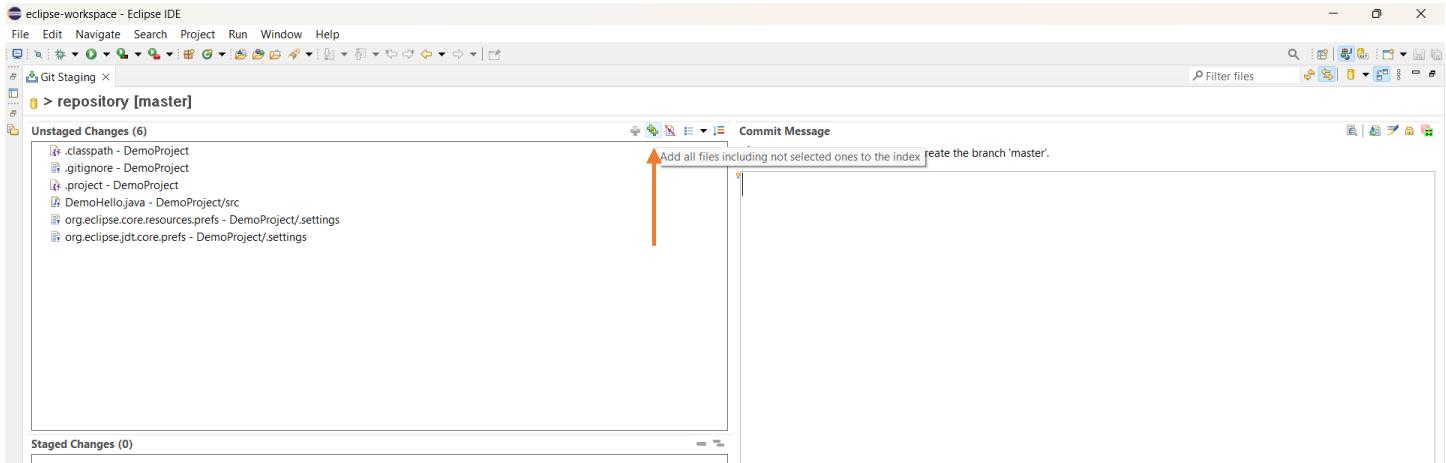
- Now in top-left corner you can see “**DemoProject[repository master]**”.



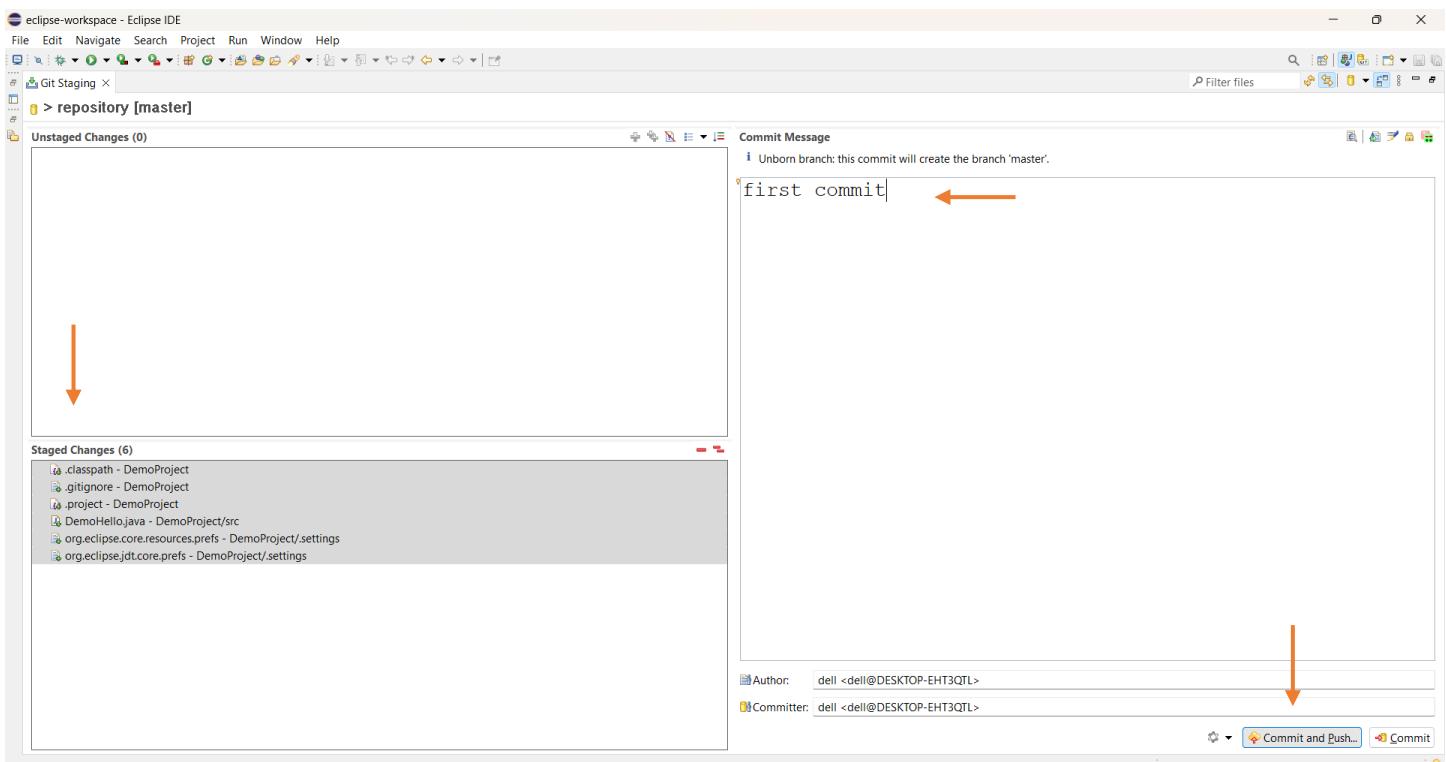
- Now Right-click on “**DemoProject -> Team -> Commit**”.



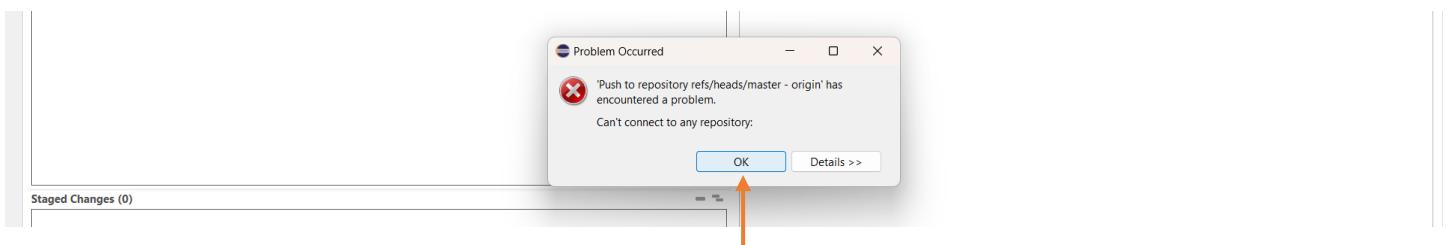
- Now **Git-Staging window** is opened.
- Press “**++**” (double-plus mark).



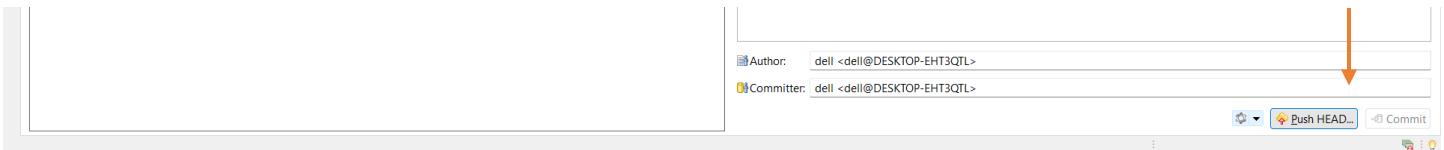
- Now you can see all the **Un-staged changes are moved to Staged Changes** below.
- Now in **Commit Message**, write “**first commit**” and click “**Commit and Push**”.



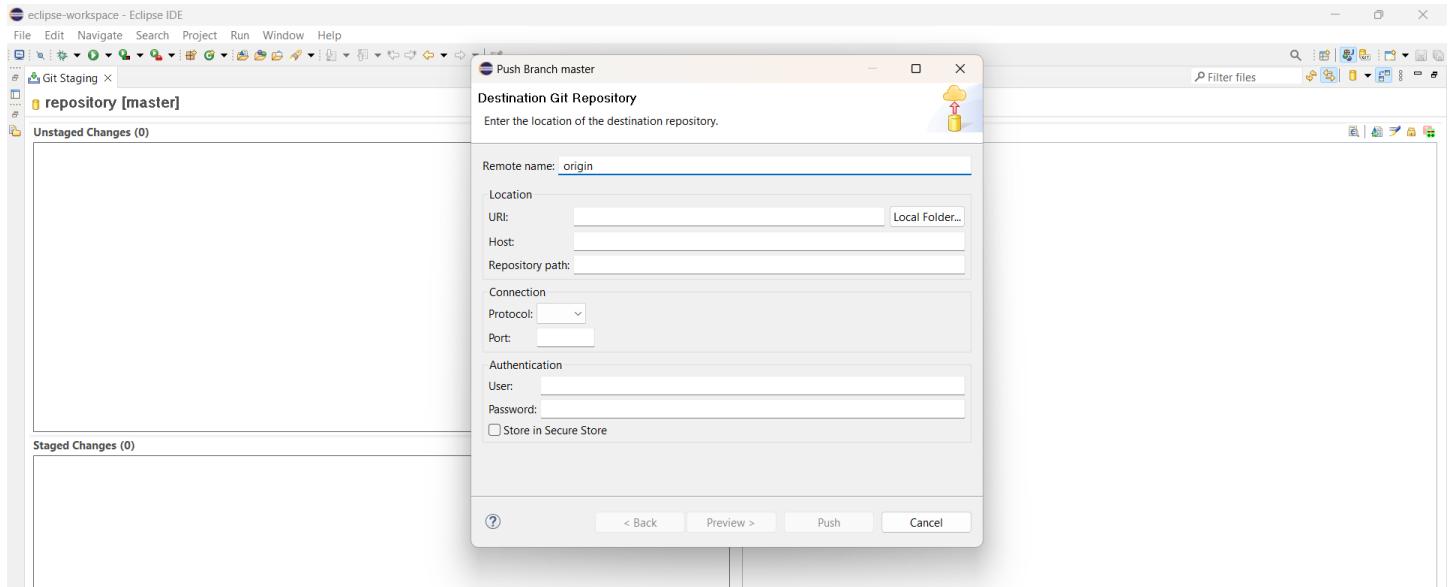
- If you get the below error, then you are going fine.
- Now Press “**OK**”.



- Now Click “**PUSH_HEAD**”.



- Now Destination **Git Repository box** is opened.



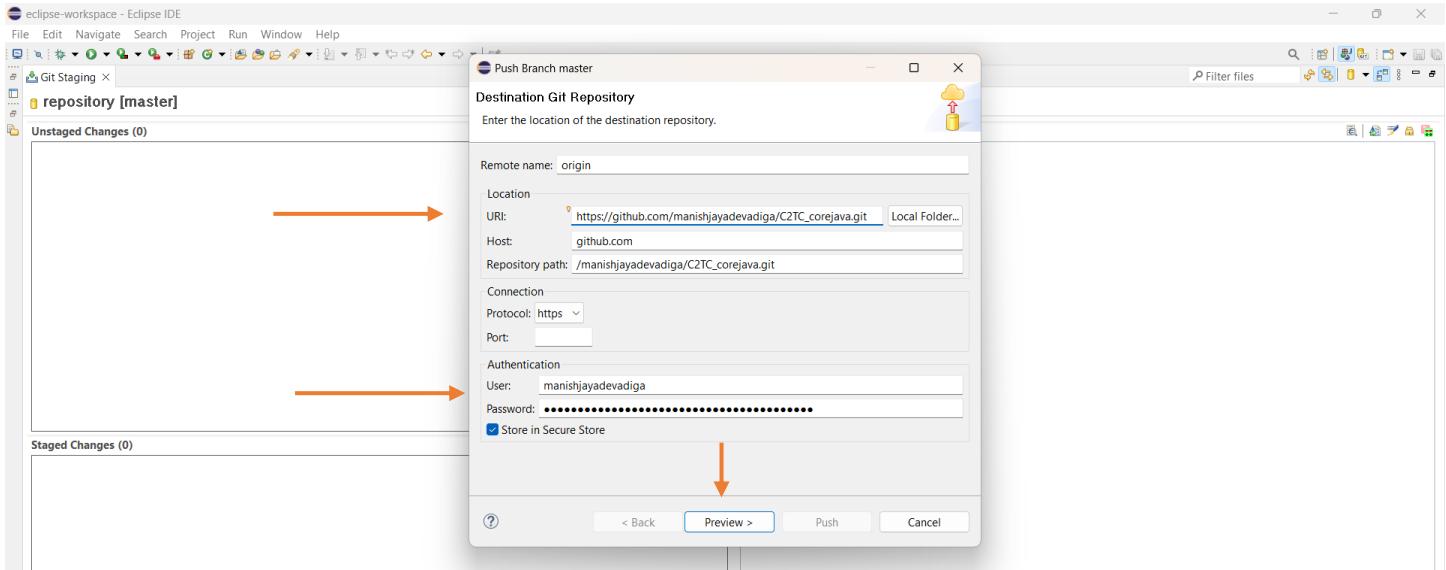
GitHub:

- Goto GitHub and “**Copy Repository Link**”.

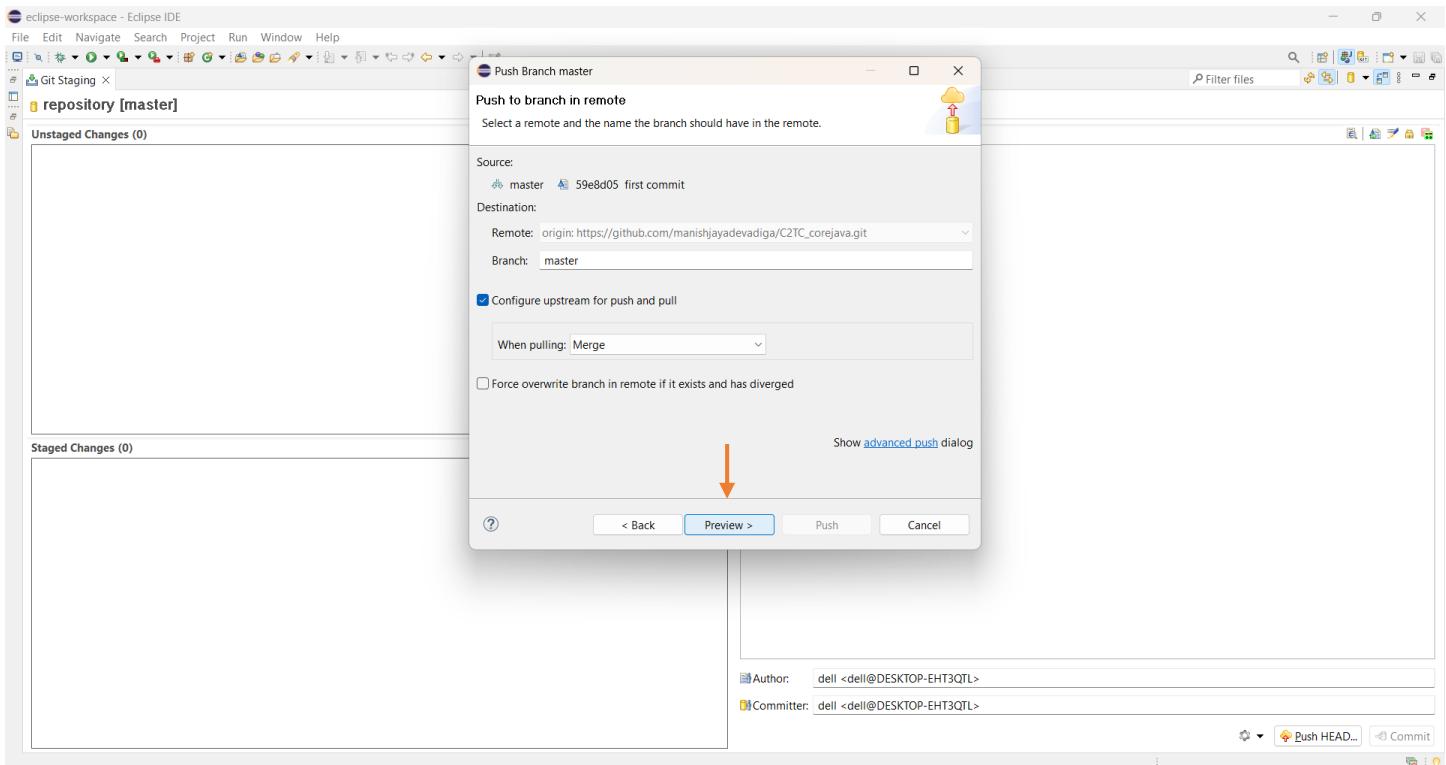
The screenshot shows a GitHub repository page for "C2TC_corejava". The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below the header, there are sections for "Set up GitHub Copilot" and "Add collaborators to this repository". A prominent dark blue banner at the bottom provides quick setup instructions, including a "Copy" icon (represented by a square with a right-pointing arrow) which is highlighted with an orange arrow. The banner also lists "HTTPS" and "SSH" options and a URL: https://github.com/manishjayadevadiga/C2TC_corejava.git.

Eclipse:

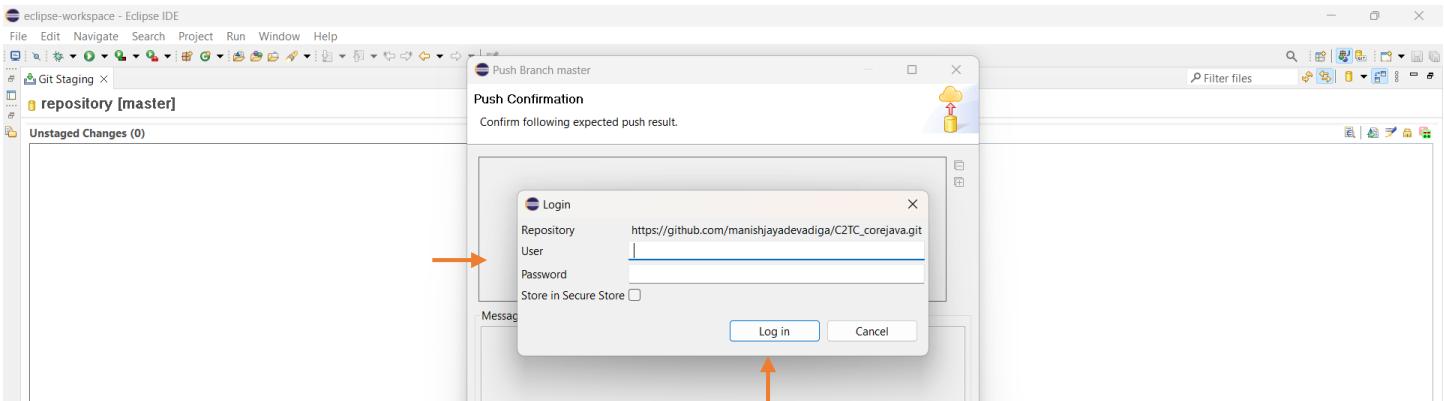
- Paste the copied link in “URL section” of the dialog box.
- Now in Authentication, “Give your GitHub Username and GitHub Password”.
- Check “Store in Secure Store” box and click “Preview”.



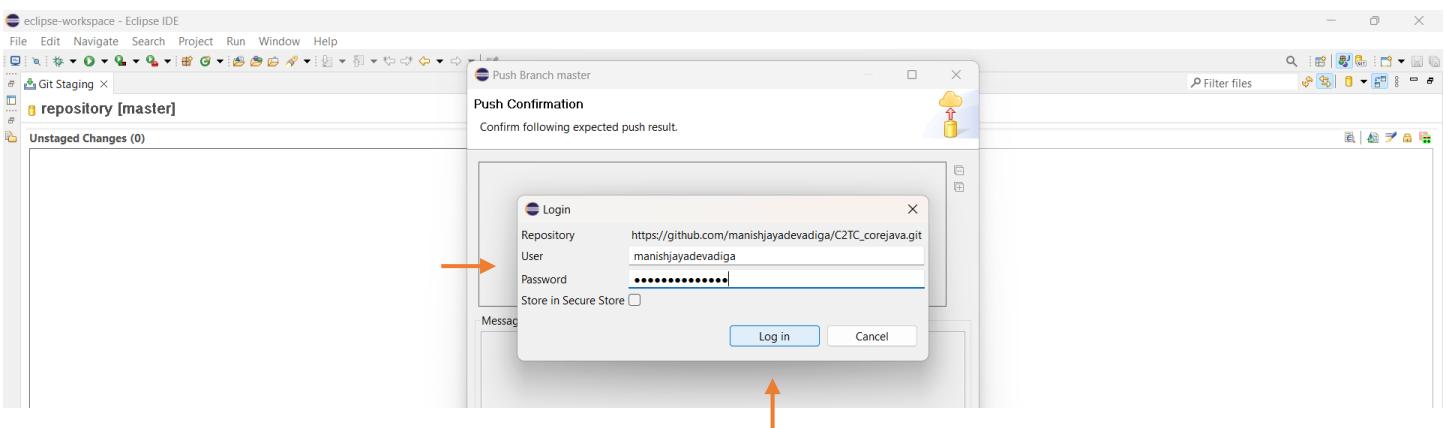
- Now click “Preview” again.



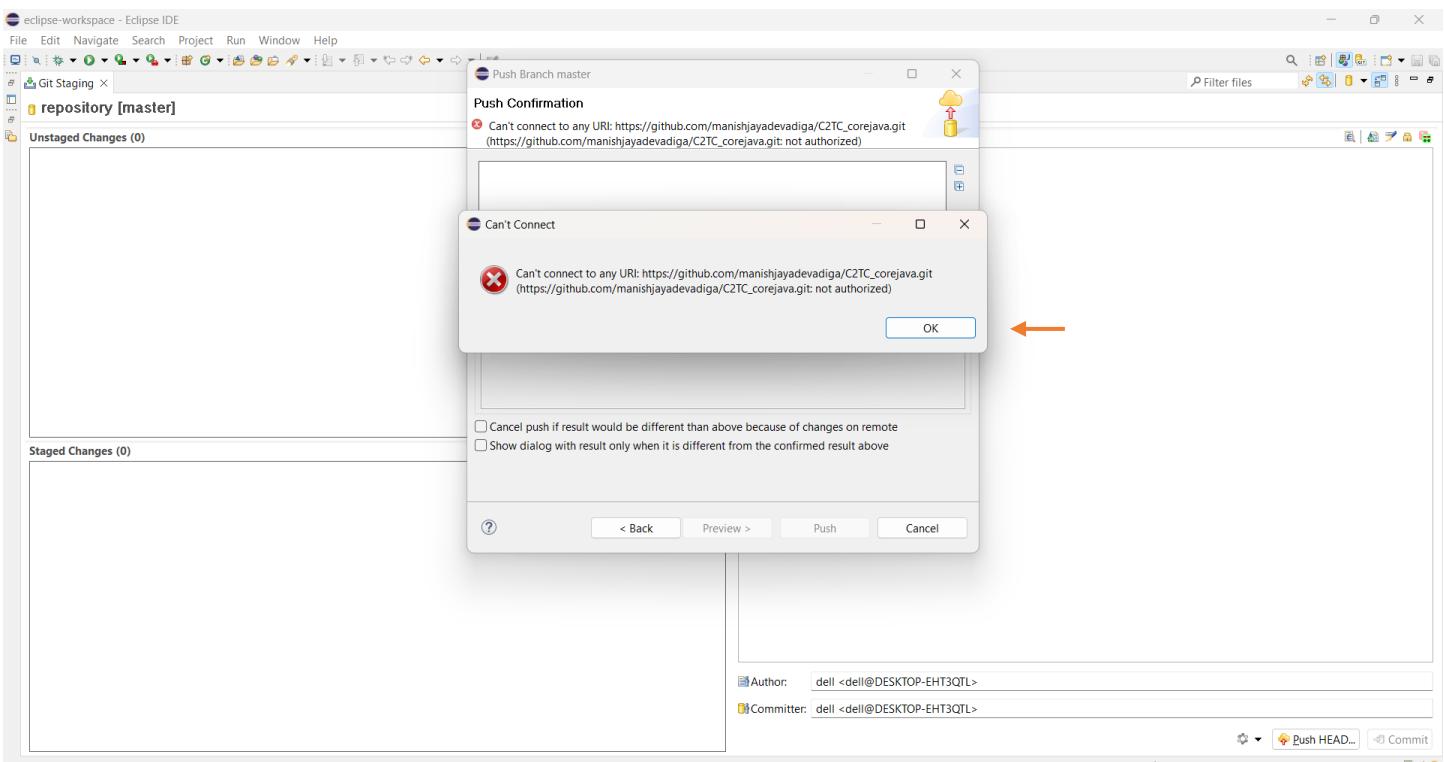
- A Login dialog box is opened.
- Give **GitHub username** and **GitHub password** and click “**Login**”.



- Do the above step again.

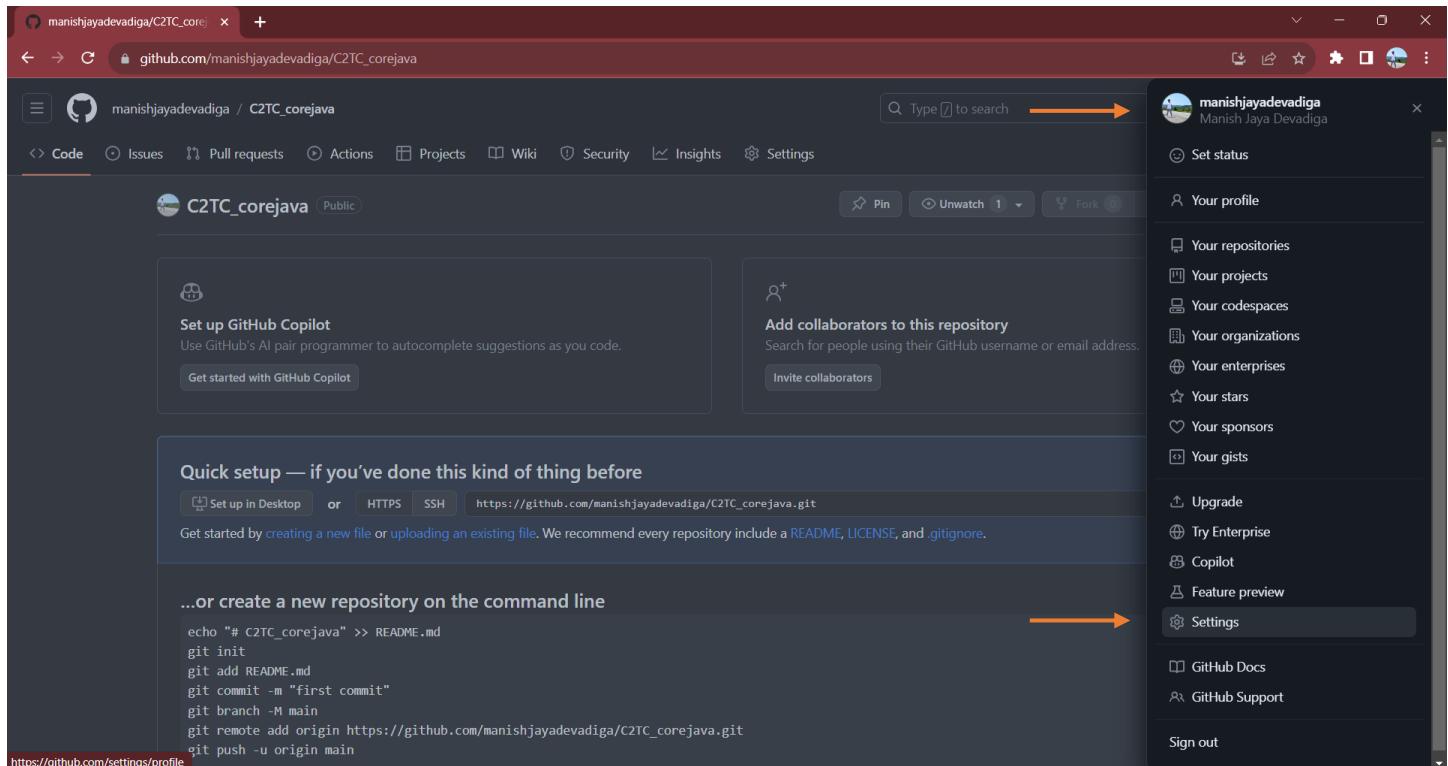


- Now **Can't Connect** box is displayed, Click “**OK**”.



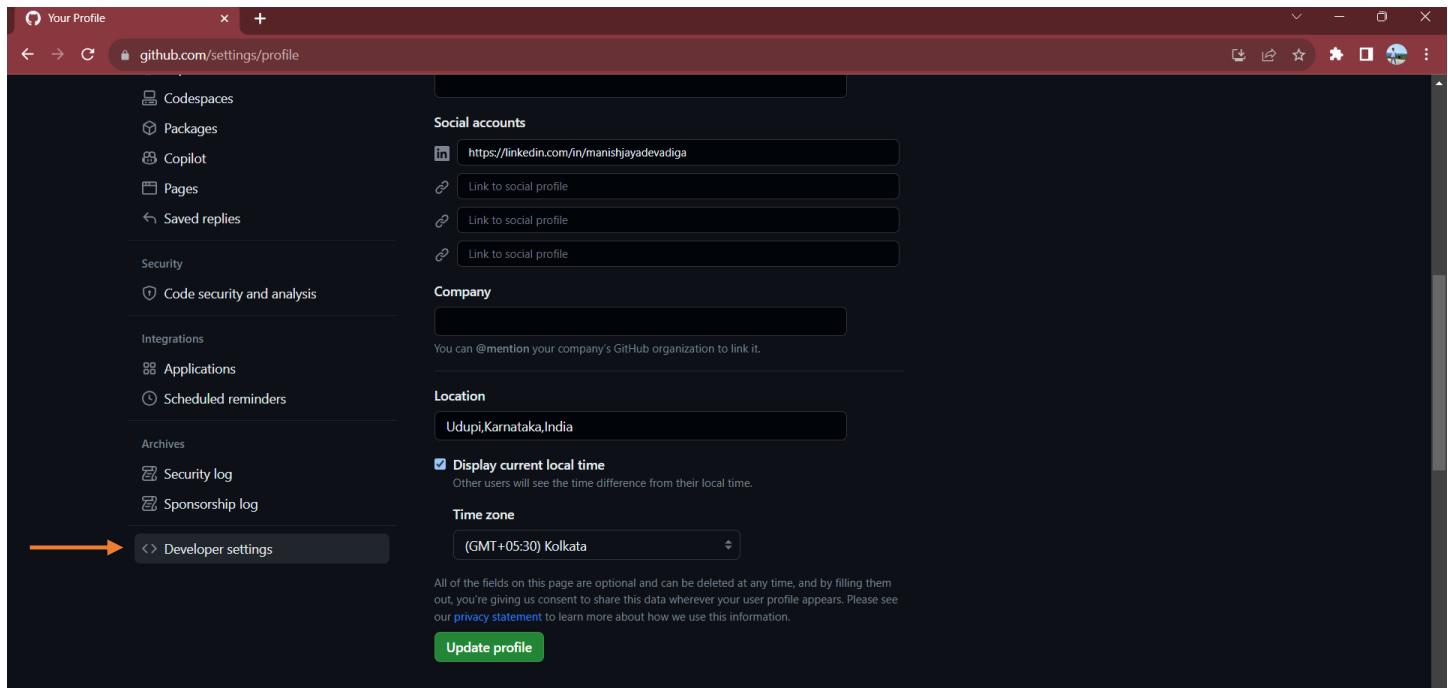
GitHub:

- Goto GitHub, “**Account (Profile Icon) -> Settings**”.



A screenshot of a GitHub repository settings page. The repository is named "C2TC_corejava". The left sidebar shows options like Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The main area has sections for GitHub Copilot setup and adding collaborators. A "Quick setup" section provides instructions for setting up the repository on desktop or command line. On the right, a sidebar lists various GitHub features and links, with "Settings" highlighted. An orange arrow points from the "Settings" link in the sidebar to the "Developer settings" link in the bottom navigation bar.

- Now Scroll Down to “**Developer Settings**”.



A screenshot of the GitHub developer settings page. The left sidebar includes sections for Codespaces, Packages, Copilot, Pages, Saved replies, Security, Code security and analysis, Integrations, Applications, Scheduled reminders, Archives, Security log, and Sponsorship log. The main content area contains fields for social accounts (LinkedIn profile), company information, location (Udupi, Karnataka, India), and time zone (GMT+05:30 Kolkata). A note at the bottom states that all fields are optional and can be deleted. An orange arrow points from the "Developer settings" link in the bottom navigation bar to the "Developer settings" link in the bottom navigation bar of the screenshot.

- Click “**Developer Settings**”.

- Now click “Personal Access tokens ->Tokens(classic)”.

Personal access tokens (classic)

Tokens you have generated that can be used to access the GitHub API.

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

[Generate new token](#) [Revoke all](#)

- Now click on “Generate new token”, and select “Generate new token (classic)”.

Personal access tokens (classic)

Personal access tokens (classic)

For general use

[Generate new token](#) [Revoke all](#)

- Now give Note as “C2TC Project” and set the Expiration to “No-expiration”.

New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Note

C2TC Project

Expiration *

30 days

No expiration

No expiration	Full control of private repositories
<input type="checkbox"/> repos:public	Access commit status
<input type="checkbox"/> repos:fork	Access deployment status
<input type="checkbox"/> repos:admin	Access public repositories
<input type="checkbox"/> repos:invite	Access repository invitations
<input type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> workflow	Update GitHub Action workflows
<input type="checkbox"/> write:packages	Upload packages to GitHub Package Registry
<input type="checkbox"/> read:packages	Download packages from GitHub Package Registry
<input type="checkbox"/> delete:packages	Delete packages from GitHub Package Registry

- Check “All” the below boxes and click “Generate Token”.

The screenshot shows the GitHub settings interface for generating a new personal access token. A list of scopes is displayed, each with a checkbox. Most checkboxes are checked, indicating the selection of all available permissions. At the bottom, there is a green 'Generate token' button and a 'Cancel' button. An orange arrow points to the 'Generate token' button.

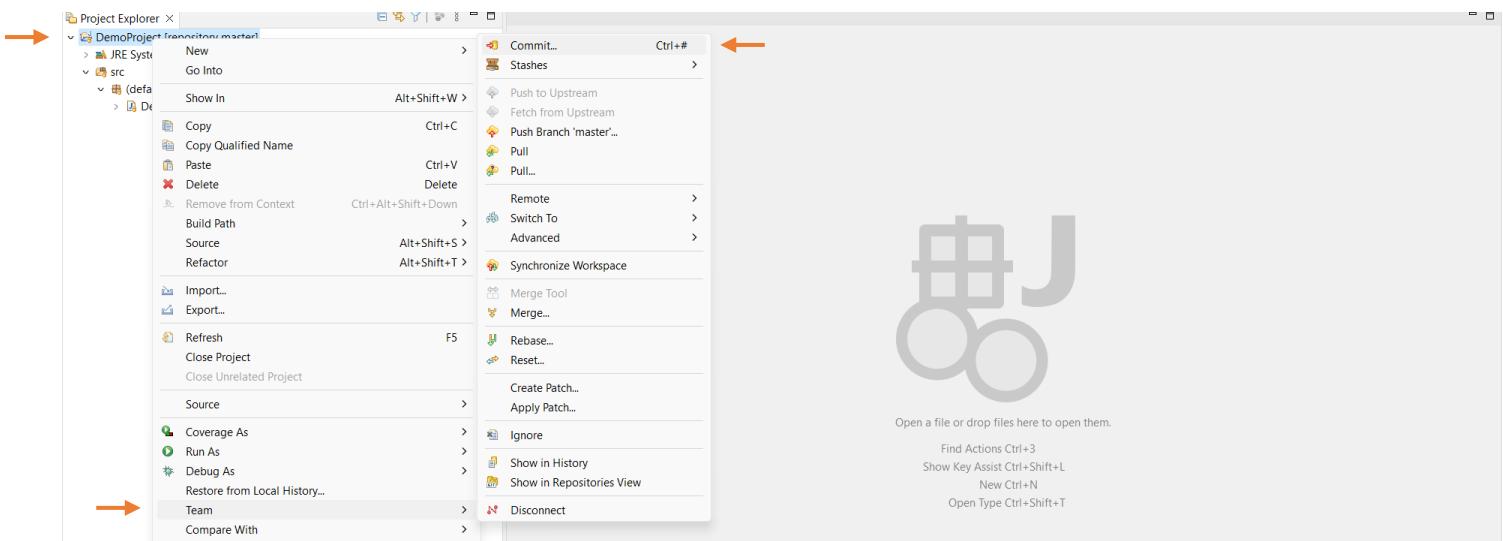
Scope	Description
<input checked="" type="checkbox"/> manage_runners:enterprise	Manage enterprise runners and runner groups
<input checked="" type="checkbox"/> manage_billing:enterprise	Read and write enterprise billing data
<input checked="" type="checkbox"/> readenterprise	Read enterprise profile data
<input checked="" type="checkbox"/> audit_log	Full control of audit log
<input checked="" type="checkbox"/> read:audit_log	Read access of audit log
<input checked="" type="checkbox"/> codespace	Full control of codespaces
<input checked="" type="checkbox"/> codespace:secrets	Ability to create, read, update, and delete codespace secrets
<input checked="" type="checkbox"/> copilot	Full control of GitHub Copilot settings and seat assignments
<input checked="" type="checkbox"/> manage_billing:copilot	View and edit Copilot for Business seat assignments
<input checked="" type="checkbox"/> project	Full control of projects
<input checked="" type="checkbox"/> read:project	Read access of projects
<input checked="" type="checkbox"/> admin:gpg_key	Full control of public user GPG keys
<input checked="" type="checkbox"/> write:gpg_key	Write public user GPG keys
<input checked="" type="checkbox"/> read:gpg_key	Read public user GPG keys
<input checked="" type="checkbox"/> admin:ssh_signing_key	Full control of public user SSH signing keys
<input checked="" type="checkbox"/> write:ssh_signing_key	Write public user SSH signing keys
<input checked="" type="checkbox"/> read:ssh_signing_key	Read public user SSH signing keys

- Now you can see that the token is generated.
- **Copy this token and keep it safe.** We will not get this link(code) again and we have to use this in future steps.

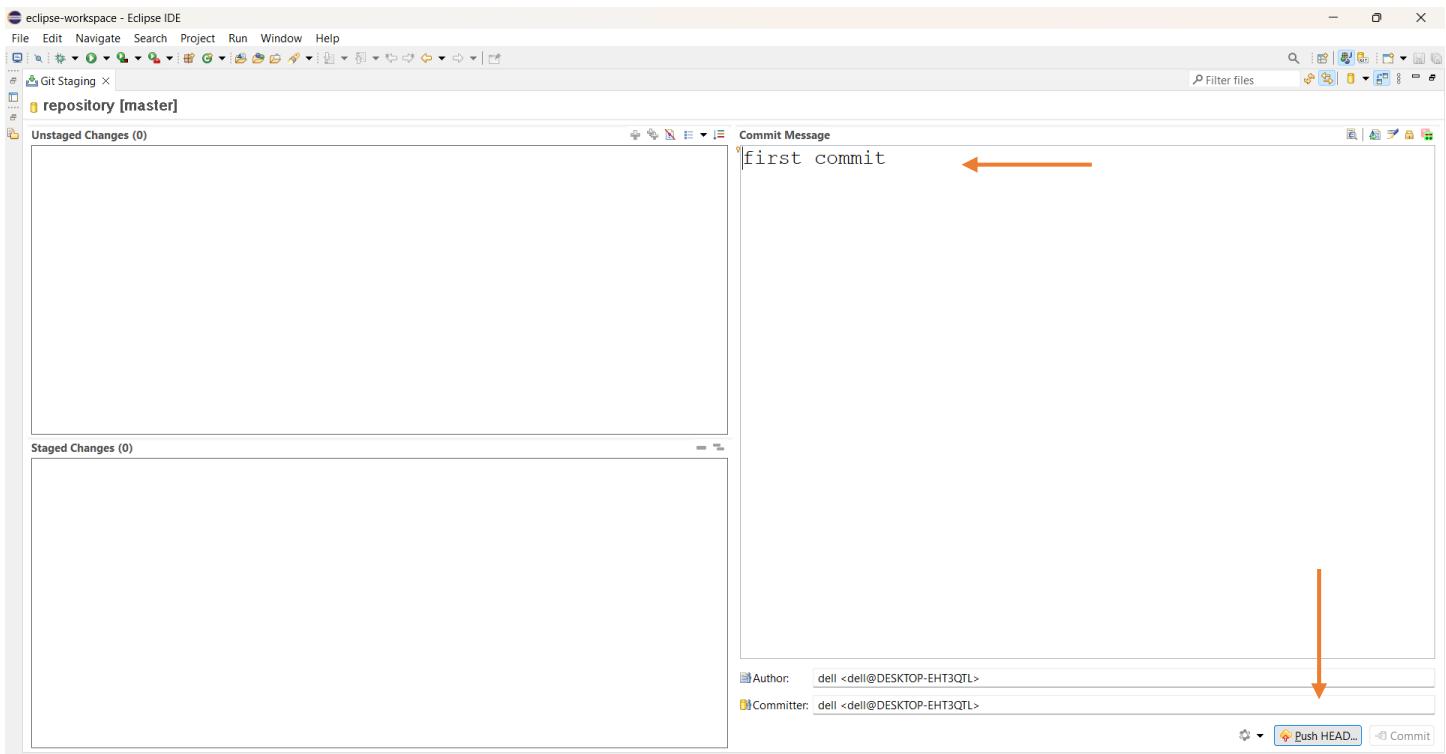
The screenshot shows the GitHub settings interface for managing personal access tokens. On the left, a sidebar lists 'GitHub Apps', 'OAuth Apps', and 'Personal access tokens'. Under 'Personal access tokens', 'Tokens (classic)' is selected. In the main area, a section titled 'Personal access tokens (classic)' shows a generated token. A message at the top of this section says, 'Make sure to copy your personal access token now. You won't be able to see it again!'. Below the message is a token card with a green checkmark icon, a copy icon, and a 'Delete' button. An orange arrow points to the token card.

Eclipse:

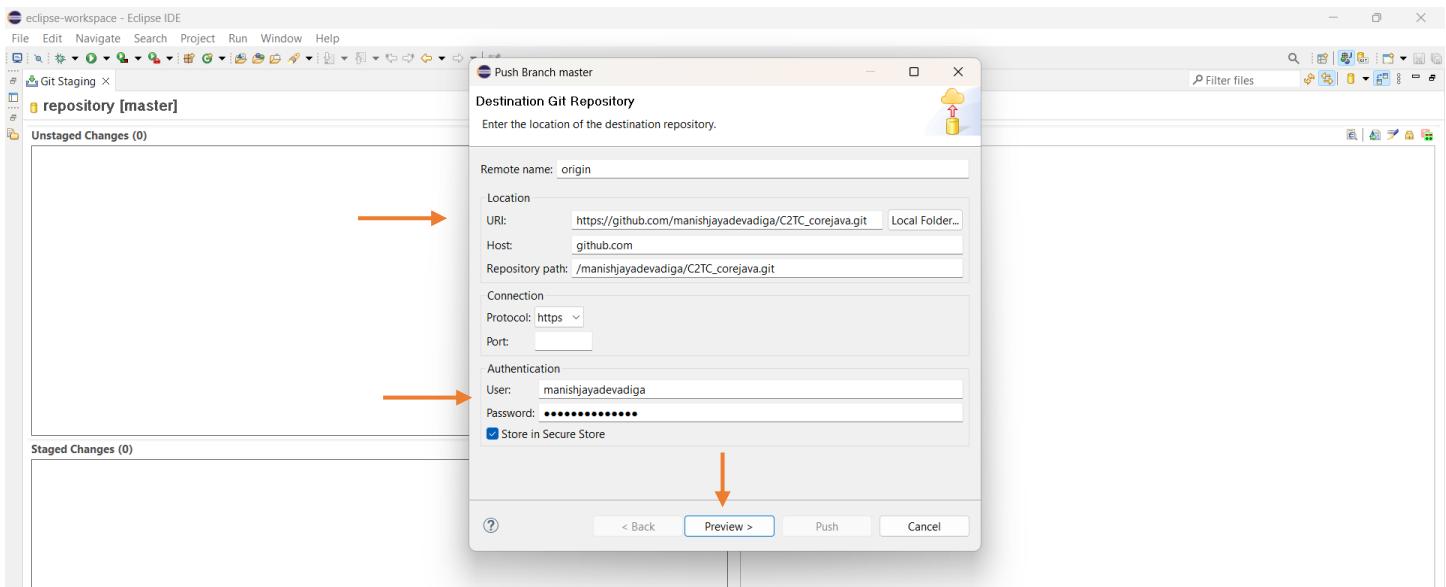
- Do the same steps which was done before.
- **Right click on project “DemoProject -> Team -> Commit”.**



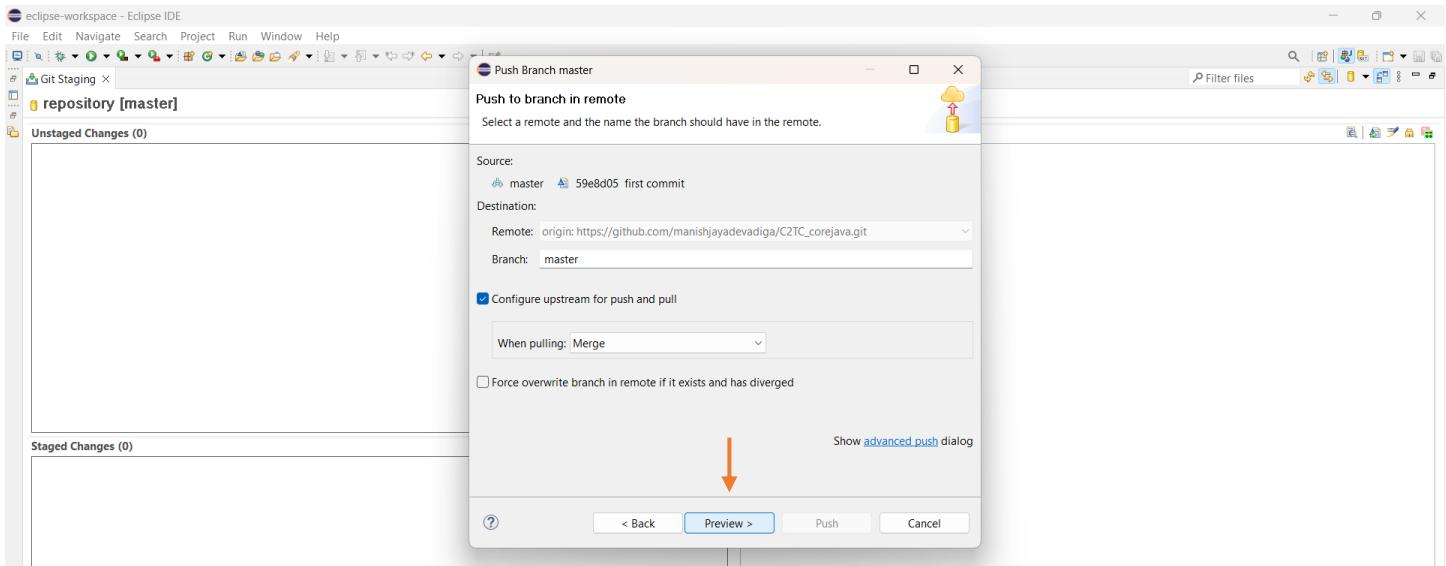
- Now write “**first commit**” and click “**PUSH HEAD**”.



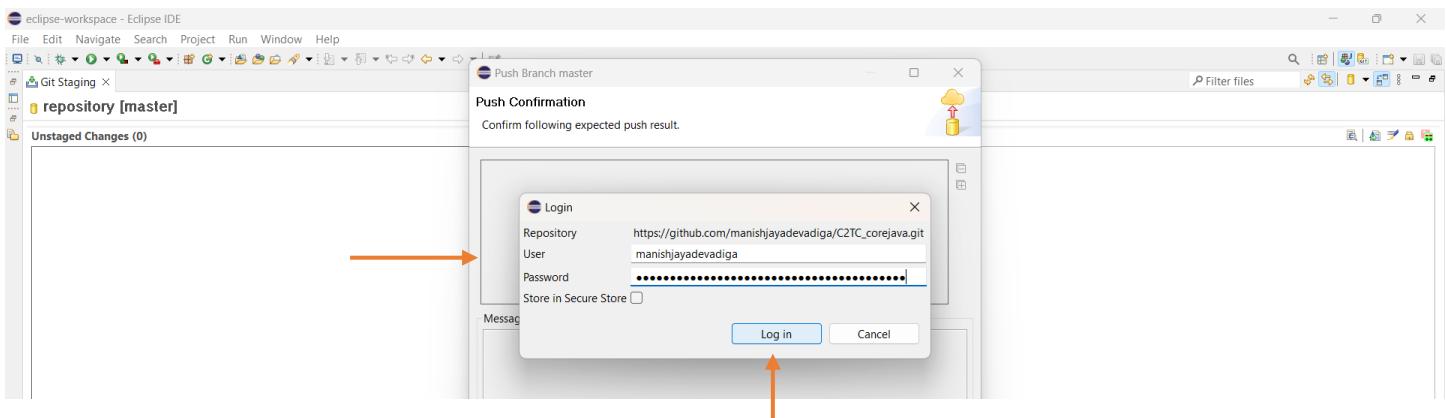
- Now Destination Git Repository box is opened.
- Give “**Repository URL, GitHub Username and GitHub Password**”.
- Check “**Store in Secure Store**” box.
- Now click “**Preview**”.



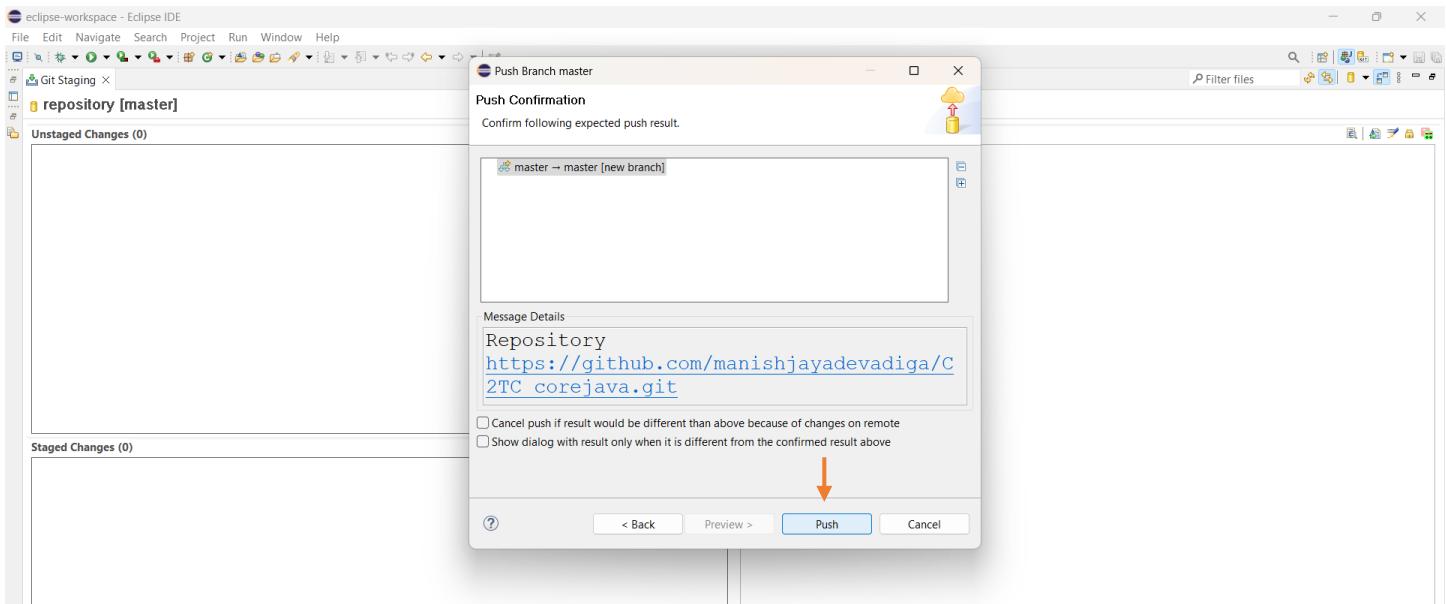
- Press “Preview” again.



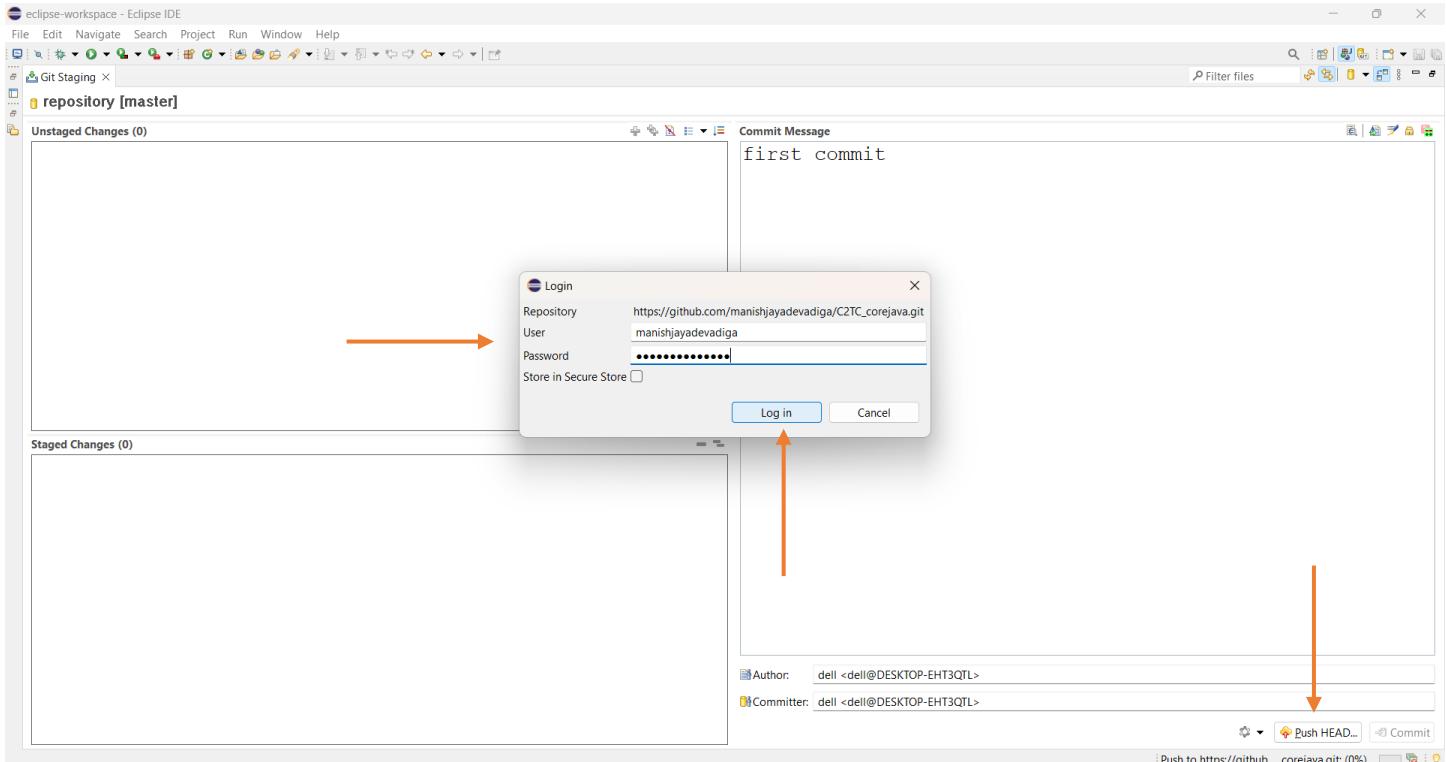
- New **Login Box** is opened.
- Give “**GitHub User Name**”.
- In the Password section “**Paste the token generated previously**”, **Uncheck “Secure Store”** and click “**Login**”.



- Now **Push Confirmation Box** is opened.
- You can observe “**master -> master [new branch]**”.
- Don’t do anything, just click “**Push**”.

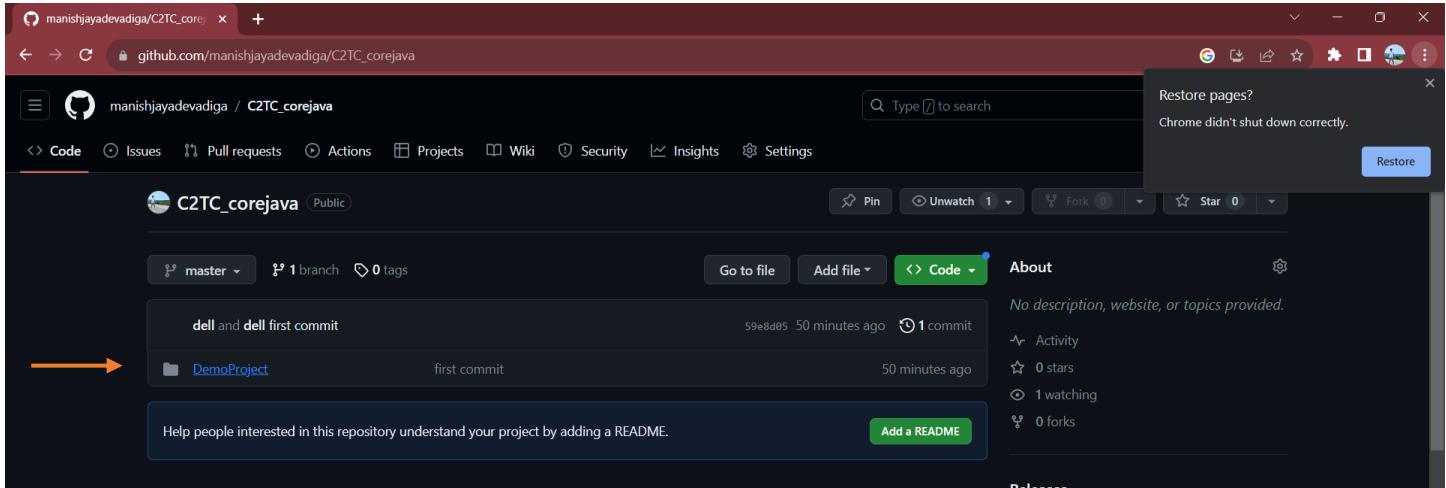


- New **Login Box** will appear.
- Give **GitHub Username and GitHub password** and click “**Login**”.
- Now press “**PUSH_HEAD**”.



GitHub:

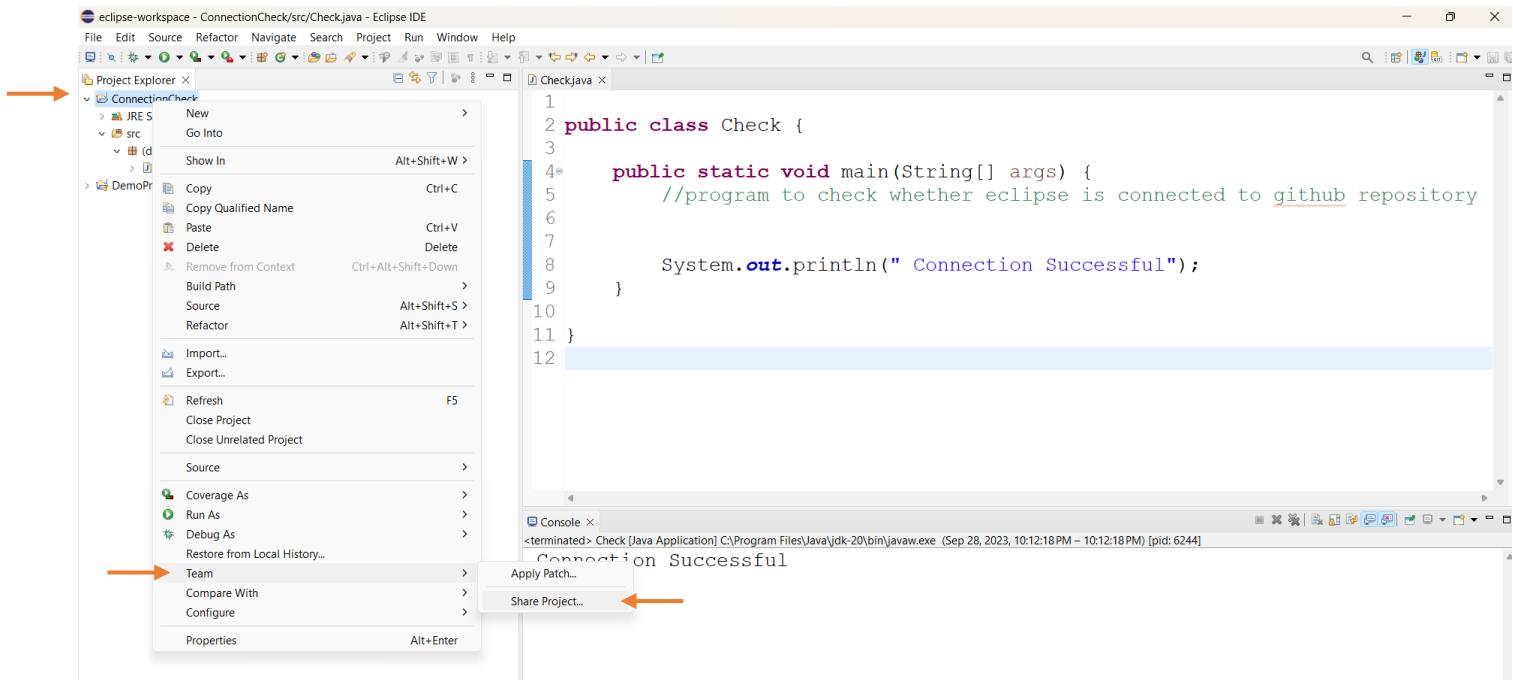
- Now open GitHub.
- Here **you can see the files (DemoProject)** which is pushed from eclipse to GitHub repository C2TC_corejava.
- Click it, you will find programs in **src** folder.



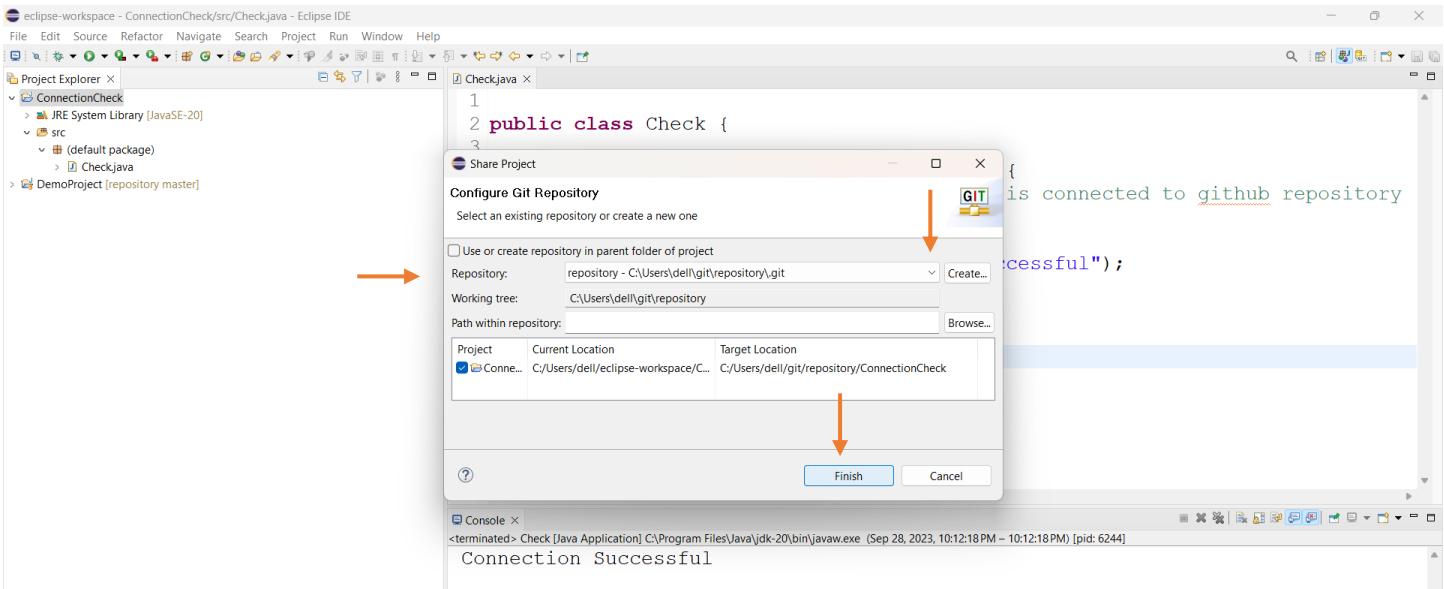
- **We are now done** with connecting Eclipse and GitHub.
- Now let us see how we can **push future files to GitHub**.

Eclipse:

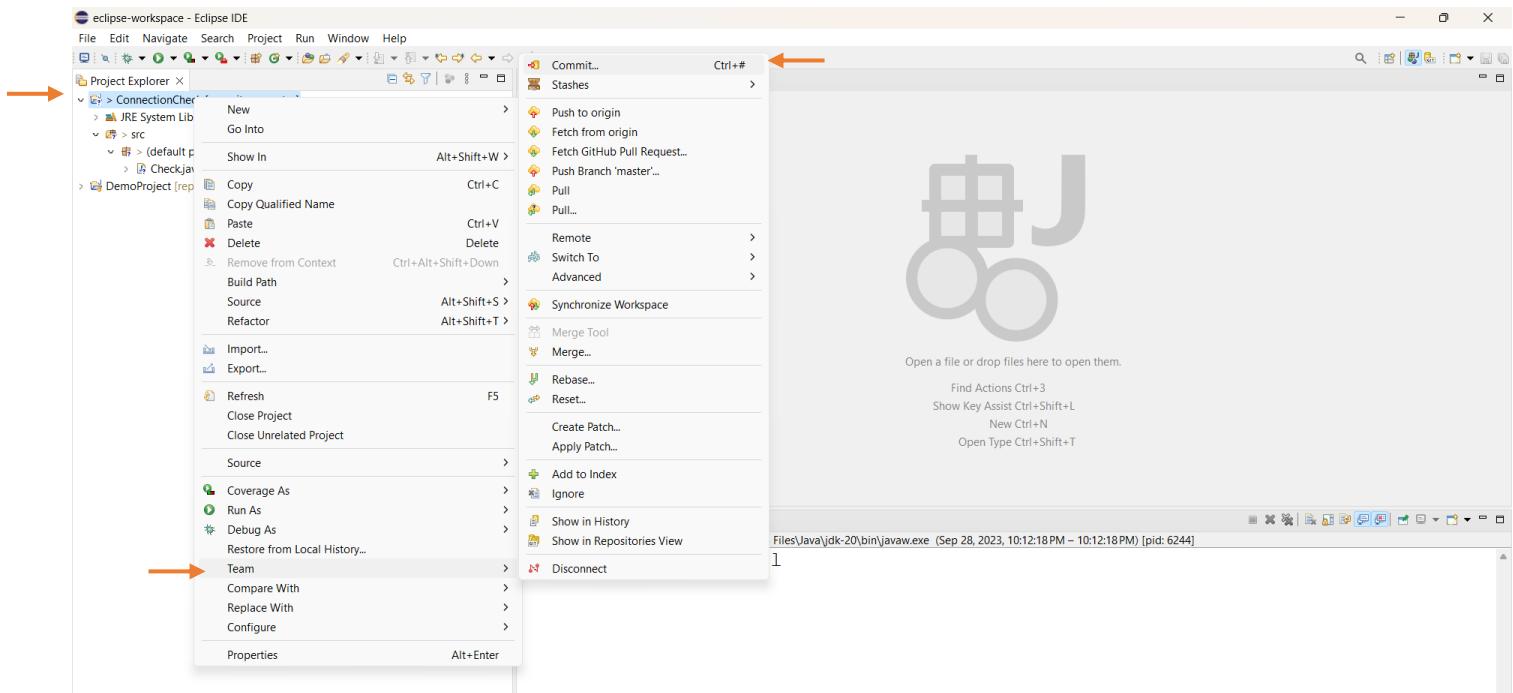
- Create a new java project “**ConnectionCheck**”.
- Right click on “**ConnectionCheck -> Team -> Share Project**”.



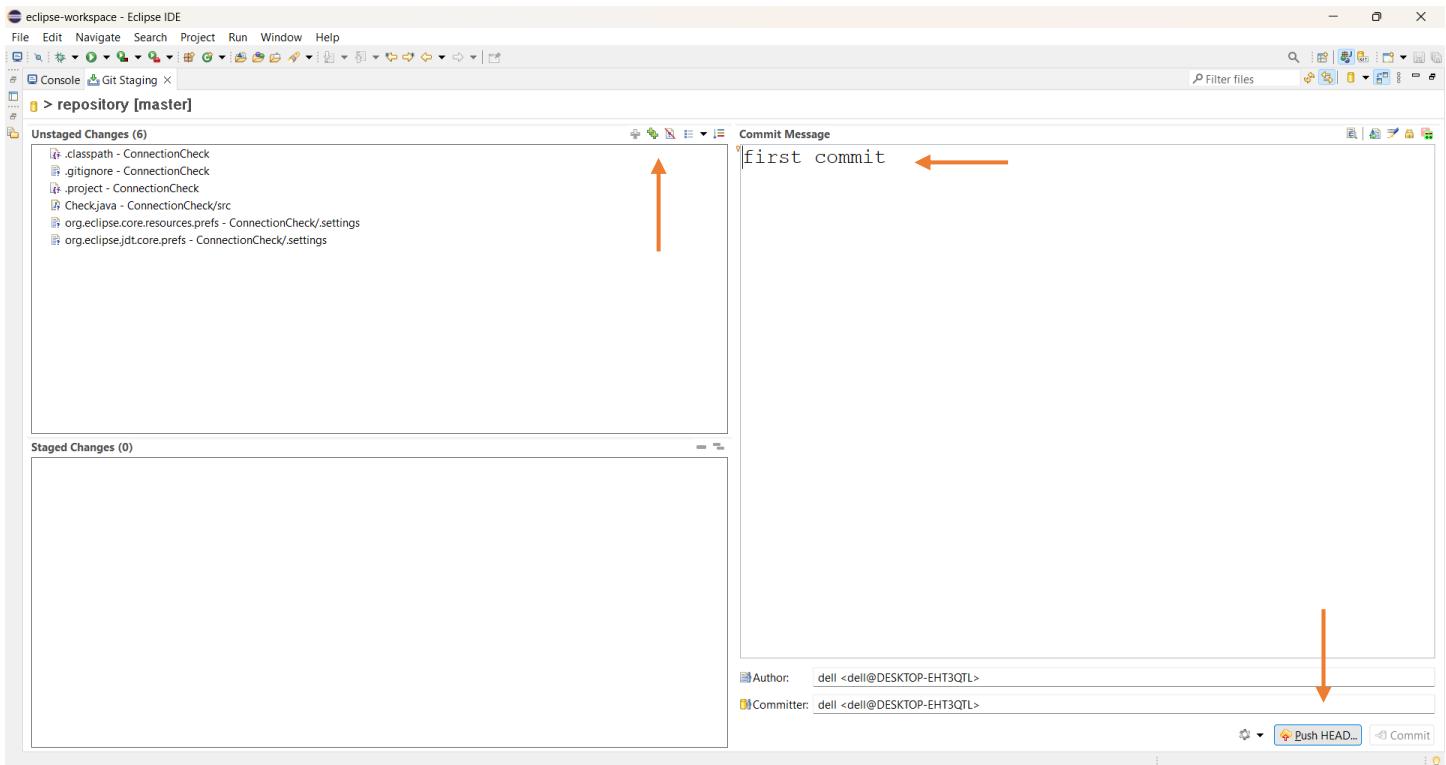
- Select repository from drop down and click “Finish”.



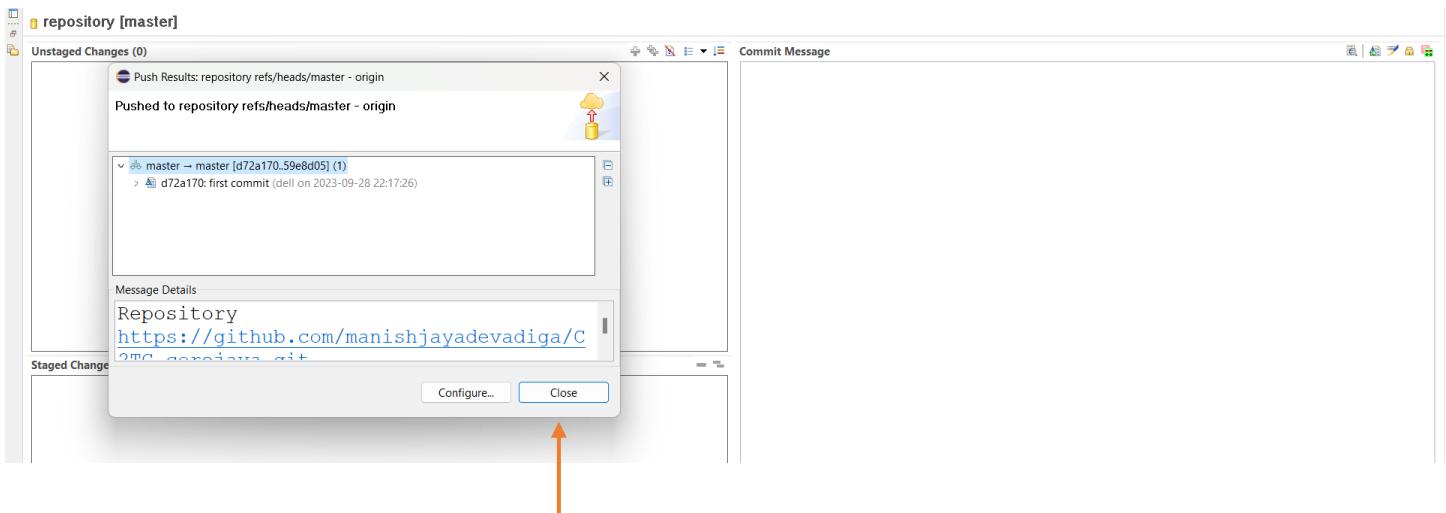
- Right click “ConnectionCheck -> Team -> Commit”.



- Press “**++ ->first commit**”.
- Click on “**Commit and Push**”.



- You must see the below box. [if not, I have discussed the possible error in the next pages]
- Click “**Close**”.



GitHub:

- Now open GitHub.
- Here **you can see the files (ConnectionCheck) which is pushed from eclipse to GitHub repository C2TC_corejava.**
- Click it, you will find programs in **src** folder.

manishjayadevadiga/C2TC_corejava

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

C2TC_corejava Public

master 1 branch 0 tags

Go to file Add file Code

dell and dell first commit d72a170 3 minutes ago 2 commits

first commit 3 minutes ago

DemoProject first commit 1 hour ago

Add a README

No description, website, or topics provided.

Activity 0 stars 1 watching 0 forks

Releases No releases published Create a new release

Eclipse:

- If you get **ERROR:master ->master [rejected – non-fast-forward]**.

eclipse-workspace - DemoChecking/src/checking.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer X

Push Results: repository - origin

Pushed to repository - origin

master -> master [rejected - non-fast-forward]

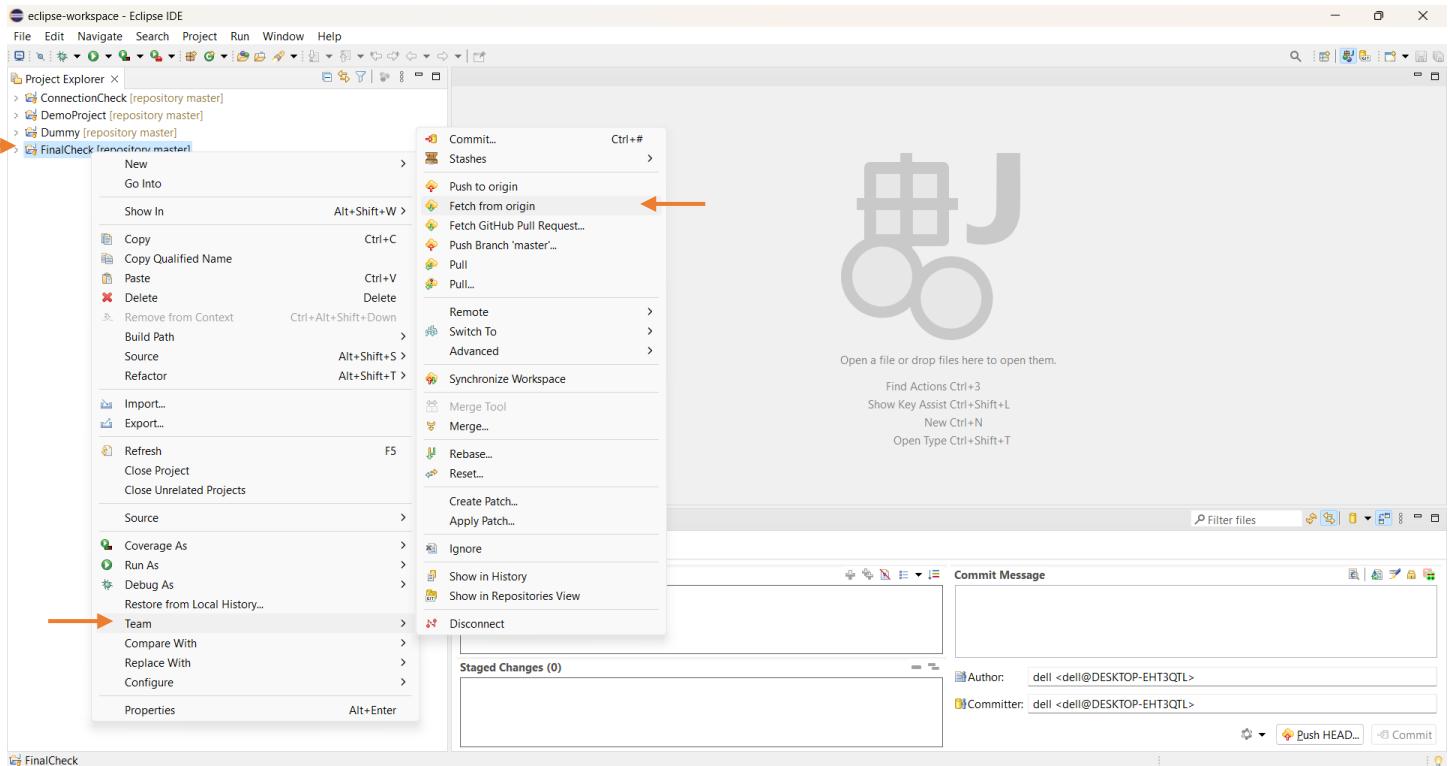
Repository https://github.com/manishjayadevadiga/C2TC_corejava.git

Configure... Close

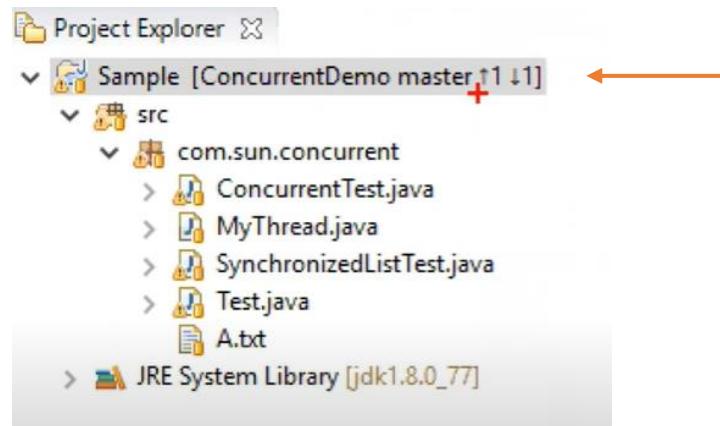
- Do the following steps carefully.
- Close** the above dialog box.

Let's Solve This:

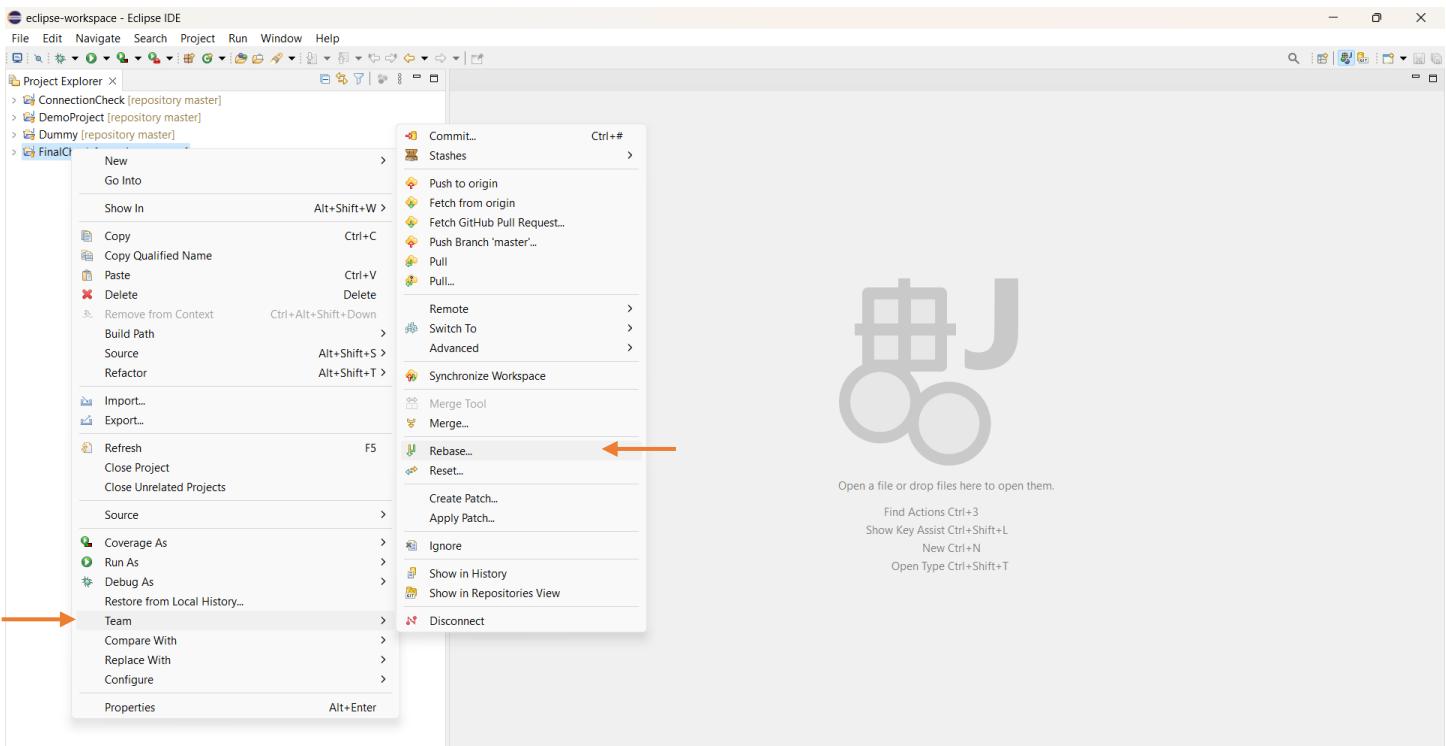
- Right click “Project -> Team -> Fetch from Origin”.



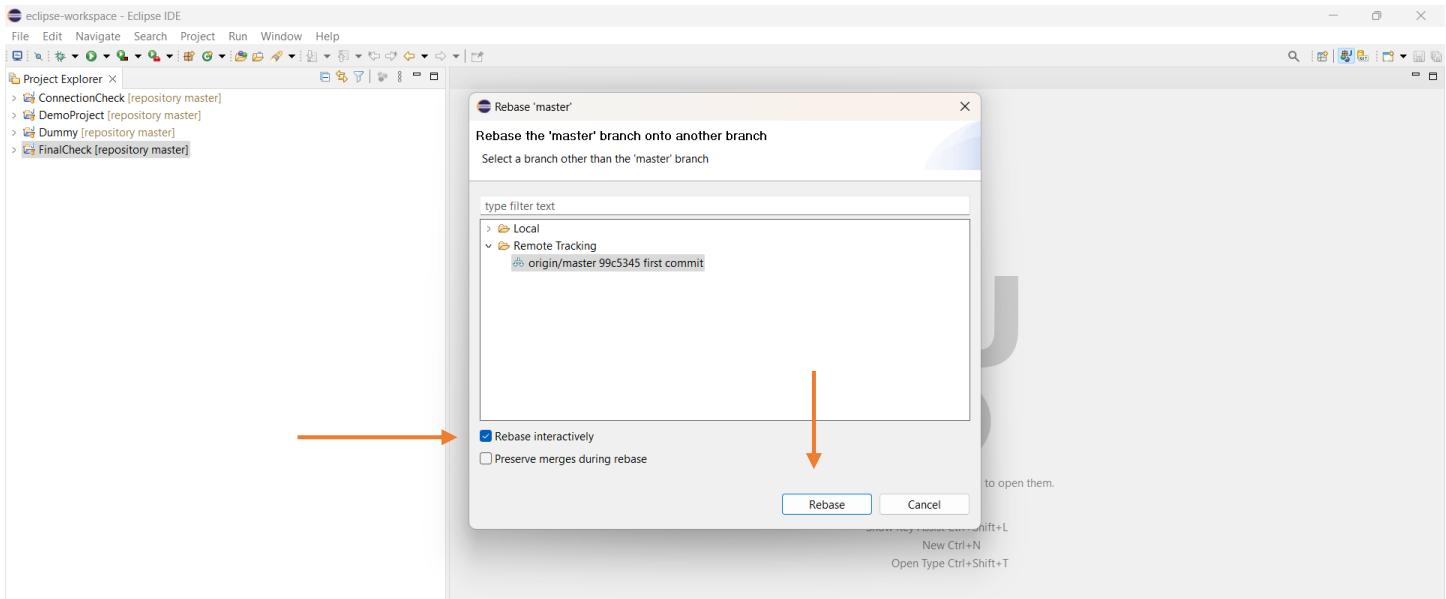
- Now in top left we will see one up arrow, one down arrow.



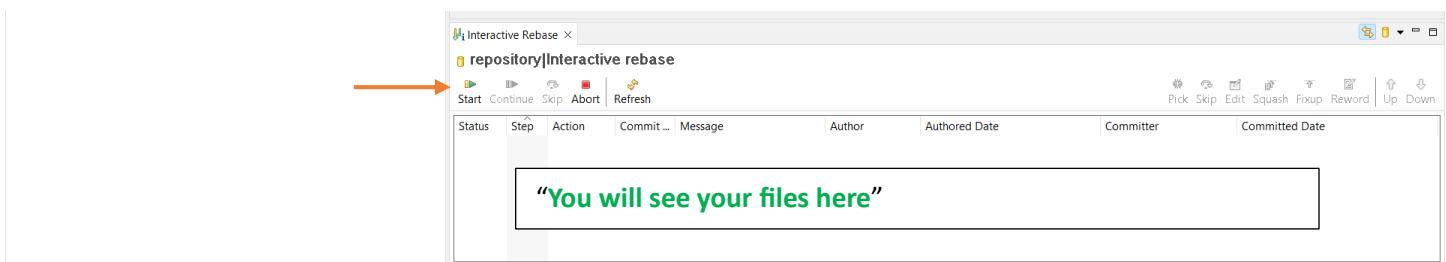
- Before pushing we need to “**perform rebase**”.
- Right click on “**Project -> Team -> Rebase**”.



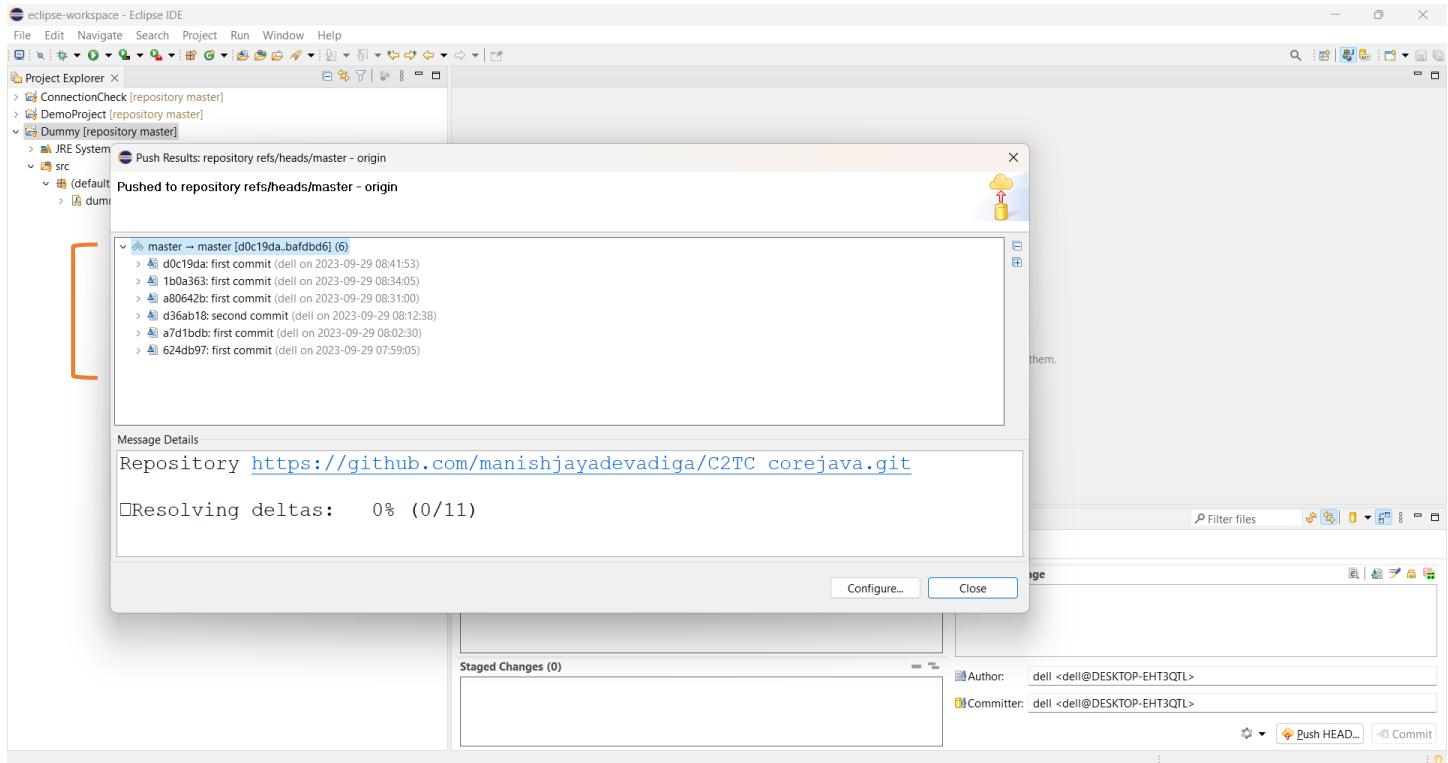
- Select “**Rebase interactively**” to avoid conflict issues.
- Click “**Rebase**”.



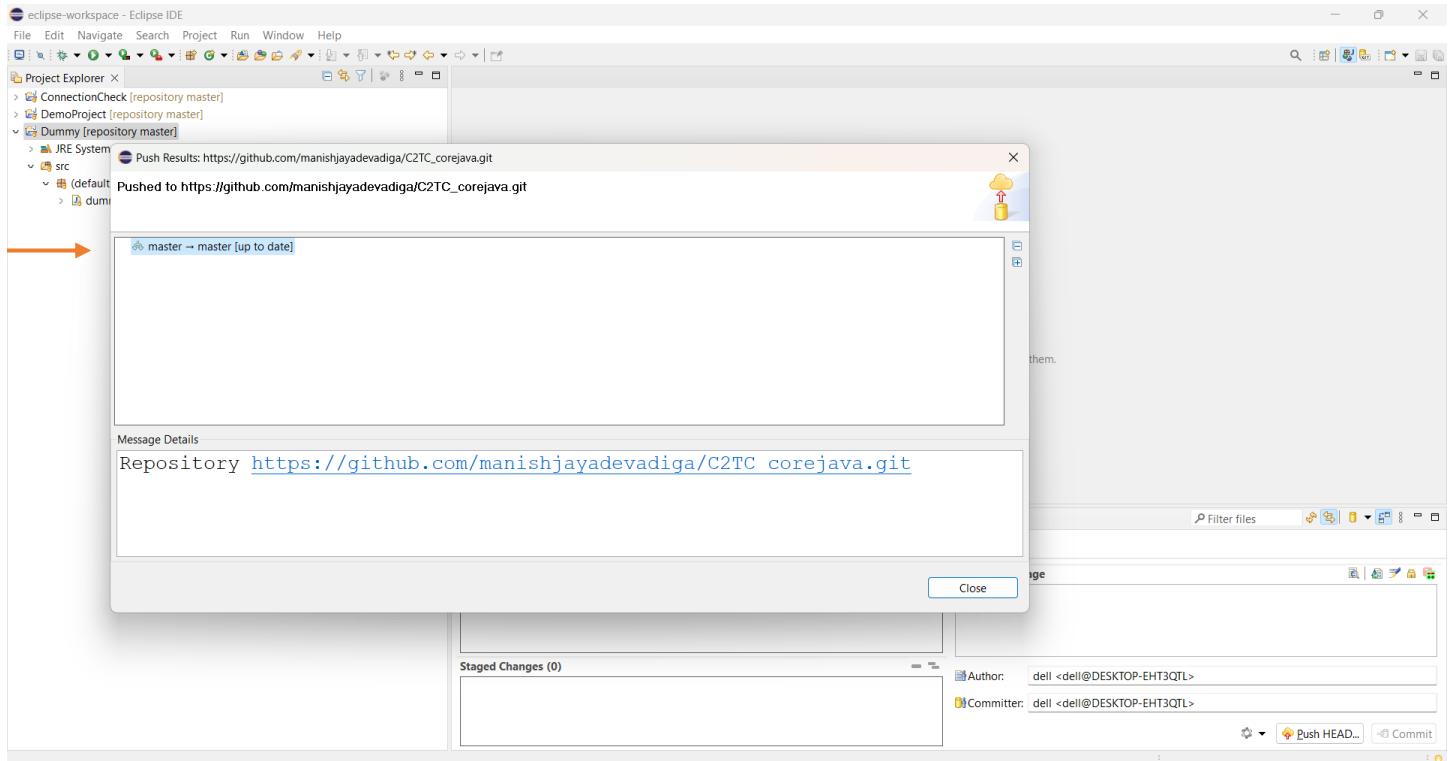
- Now **you will see your files here** (mine is empty because I have solved my error). Now press “**Start Icon**”.



- Now Rebase has been finished “Successfully”.
- Now perform the same Push operations which we have been doing to push to git hub.
- You will see the below dialog box after Push and Commit.



- Now “Close” this.
- Click “PUSH_HEAD”. Here we can see the branch is up to date.
- We have successfully solved the error. Now we can easily push our codes to GitHub.



GitHub:

- Here we can see that all the files have been pushed to GitHub repository.

The screenshot shows a GitHub repository page for 'C2TC_corejava'. The repository has 1 branch and 0 tags. A commit from 'dell' titled 'dell and dell first commit' is shown, dated 1 minute ago with 9 commits. The commit details show five files: ConnectionCheck, DemoChecking, DemoProject, Dummy, and Eclipse Github Connection (Written).... The last commit was an upload via Eclipse. There is a callout bracket pointing to the commit list. On the right side, there are sections for 'About', 'Releases', 'Packages', and 'Languages'.

- Now “Close Eclipse”.
- Try to push file to GitHub (To see if everything is working fine).

Final Check:

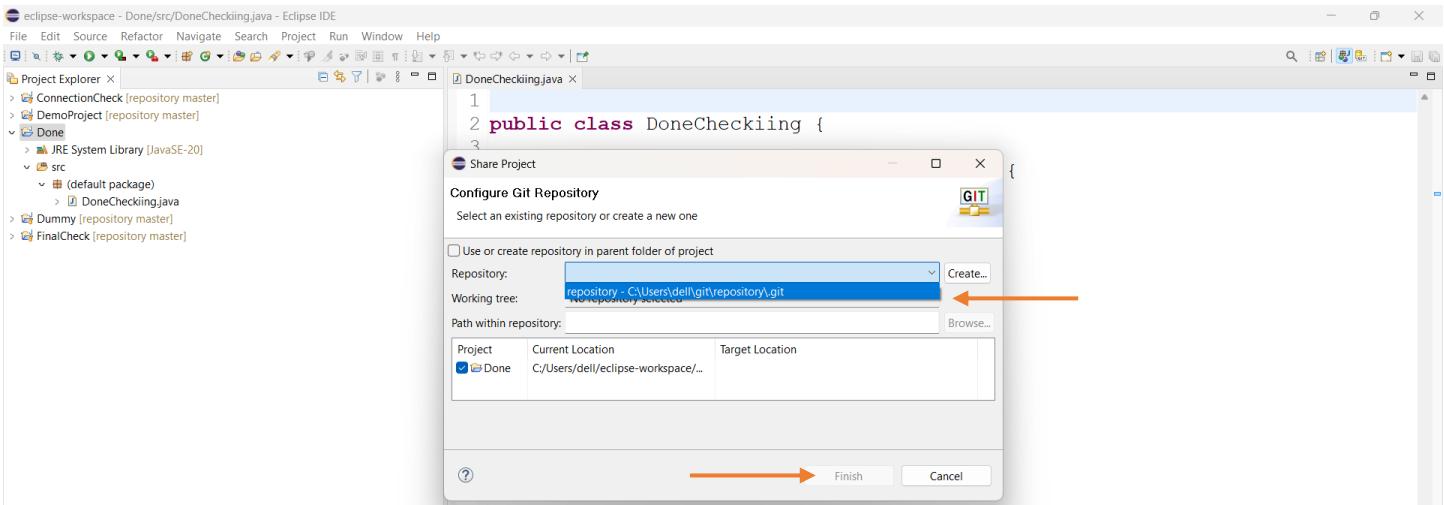
Eclipse:

- Create a java project named “Done”. Right click “Done -> Team -> Share Project”.

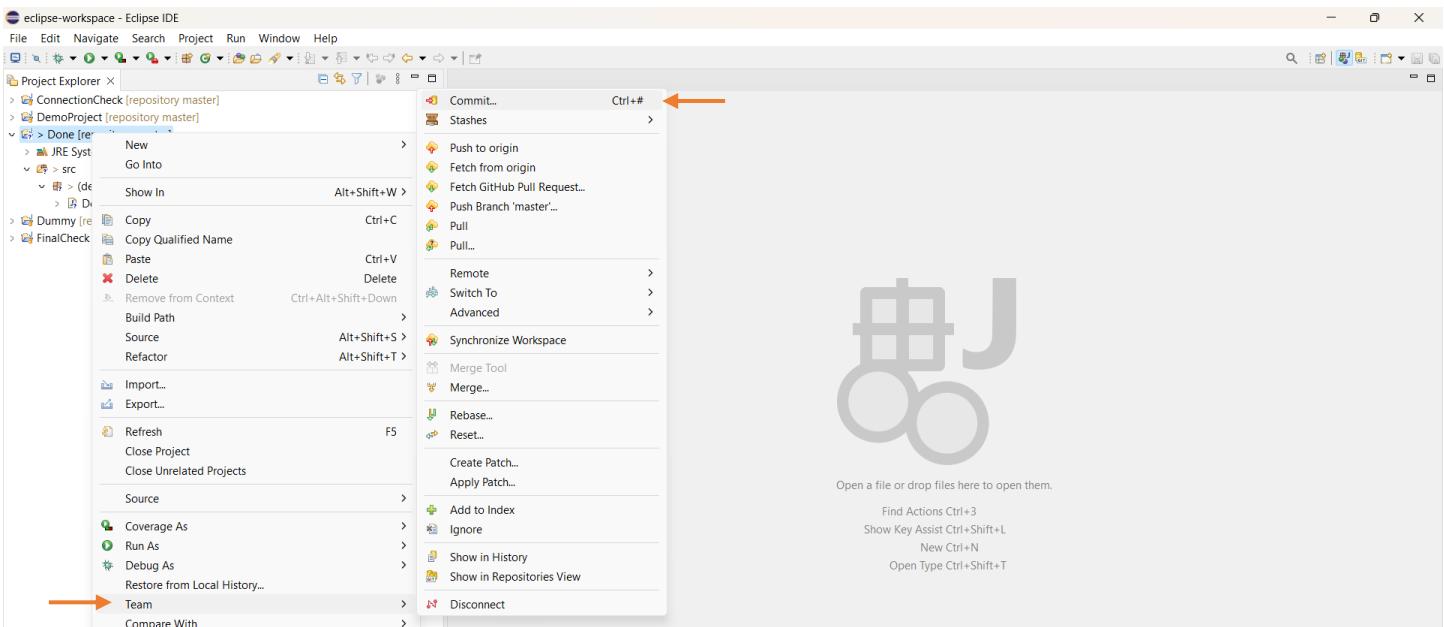
The screenshot shows the Eclipse IDE's Project Explorer. A context menu is open over a project named 'Done'. The menu path 'Team' is highlighted with a red arrow, and the option 'Share Project...' is also highlighted with a red arrow. The code editor on the right shows a Java file 'DoneChecking.java' with the following code:

```
1
2 public class DoneChecking {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7     }
8
9 }
```

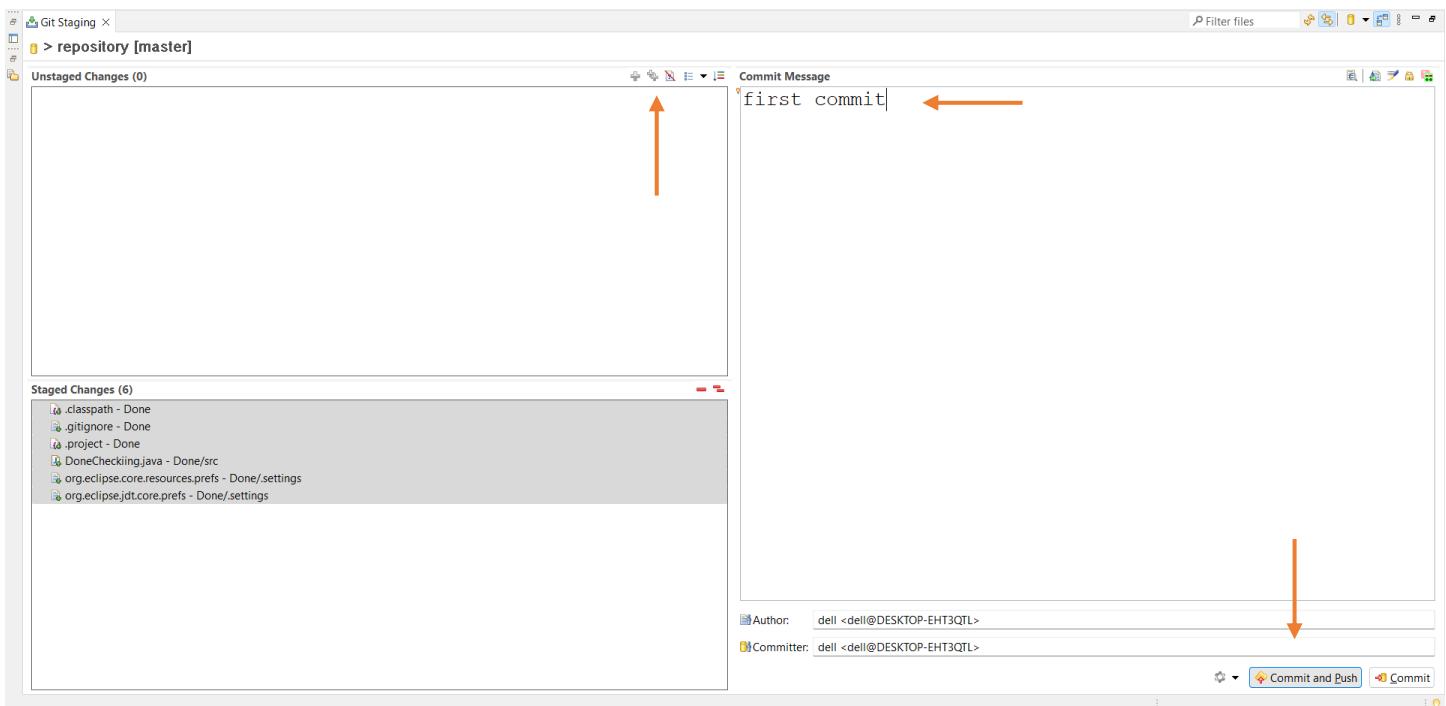
- Select repository from dropdown and click "Finish".



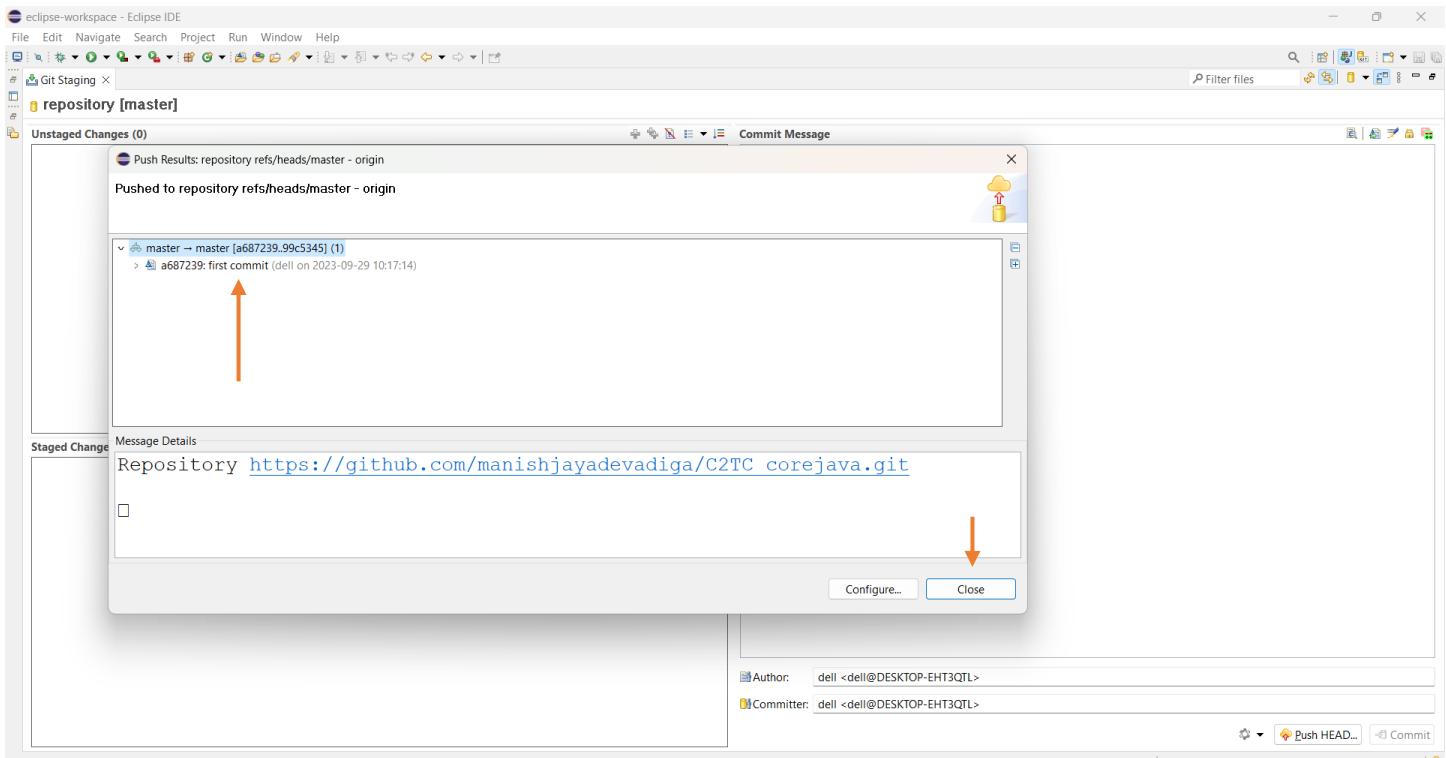
- Right click "Done -> Team -> Commit ".



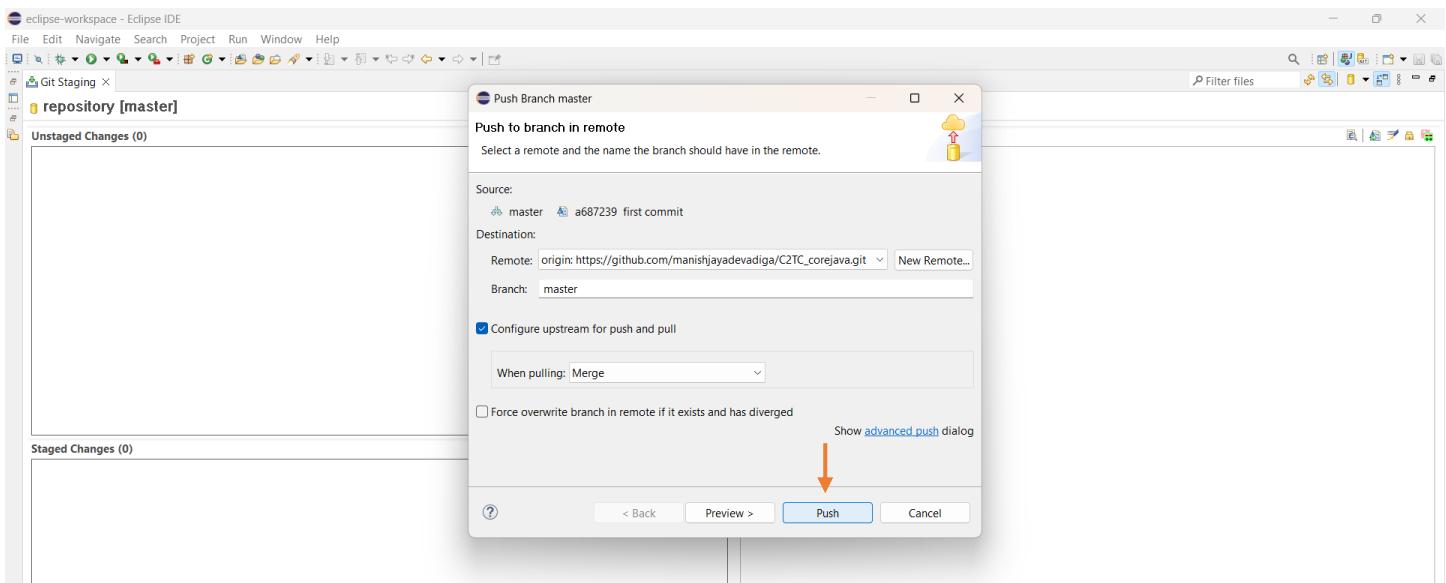
- Now Git-Staging window will be opened.
- Press “++ ->first commit”.
- Click on “Commit and Push”.



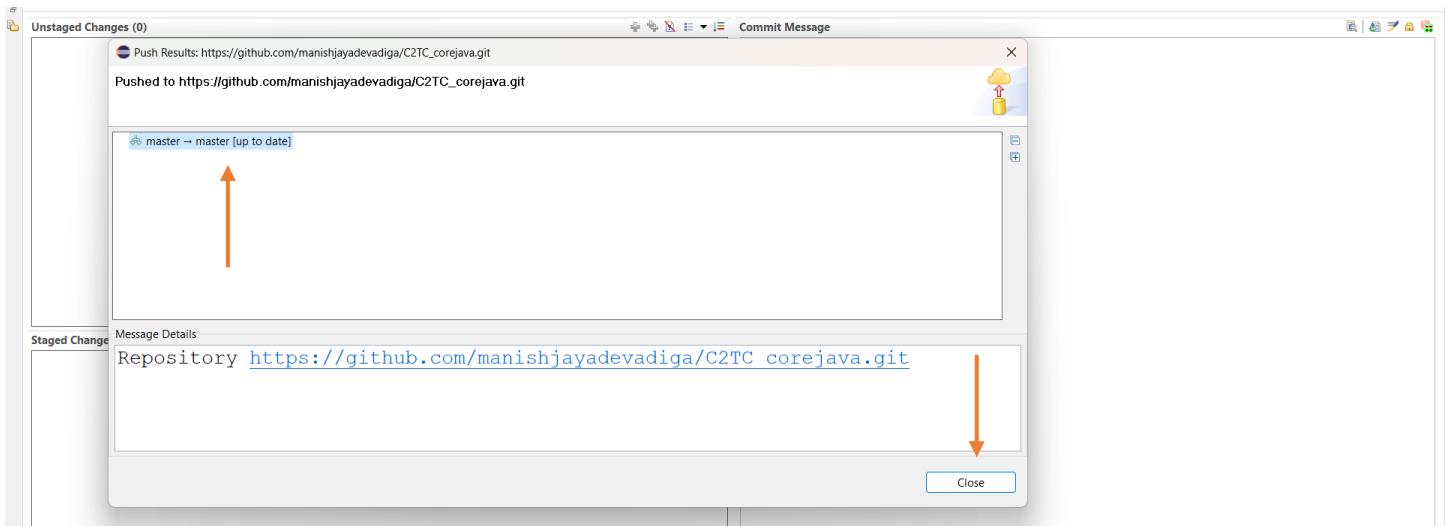
- We can see “Pushed to repository”.
- We can see “No error like: non-fast-forward”.
- Now “Close” it. And Click “PUSH_HEAD”.



- Now click “Push”.



- The **push is successful** and branch is up to date.
- Click “**Close**”.



GitHub:

- Here we can see that Project “**Done**” have been pushed to GitHub repository successfully.

C2TC_corejava Public

master 1 branch 0 tags

dell and dell first commit	a687239	1 minute ago	11 commits
ConnectionCheck		12 hours ago	
DemoChecking		1 hour ago	
DemoProject		13 hours ago	
Done		1 minute ago	
Dummy		1 hour ago	
FinalCheck		1 hour ago	
Eclipse Github Connection (Written)...	Add files via upload	11 hours ago	

About

No description, website, or topics provided.

Activity

1 star

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Help people interested in this repository understand your project by adding a README.

Add a README