
# Git and GitHub Training



simplilearn

# Git Plugin with IDE

# Learning Objectives

By the end of this lesson, you will be able to:

- Work with Git workflow on Eclipse IDE
- Add Eclipse project to GitHub repository
- Use Git in IntelliJ IDE

# EGit and Eclipse IDE

# What Is EGit?

EGit is an Eclipse plug-in which allows you to use the distributed version control system “Git” directly within the Eclipse IDE.


**NOTE**

EGit is based on the JGit library. JGit is a library which implements the Git functionality in Java.

Image source: https://www.eclipse.org/egit/

# Installation of EGit support into Eclipse IDE

**Problem Statement:** If Git tooling is not available, install it using the Eclipse installation manager.

**Steps to Perform:**

- Go to: http://download.eclipse.org/egit/updates to get the plugin repository location
- Go to **Help** and navigate to **Install New Software**
- To add repository location, click **Add**. Enter repository name as **EGit Plugin**. Location will be the URL copied from the above step. Now click **Ok** to add repository location
- Wait for a while and it will display the list of available products to be installed. Expend **Eclipse Git Team Provider** and select **Eclipse Git Team Provider**. Now click **Next**.

ASSISTED PRACTICE

# Working with Git on Eclipse IDE

# Create a Git Repository using Eclipse

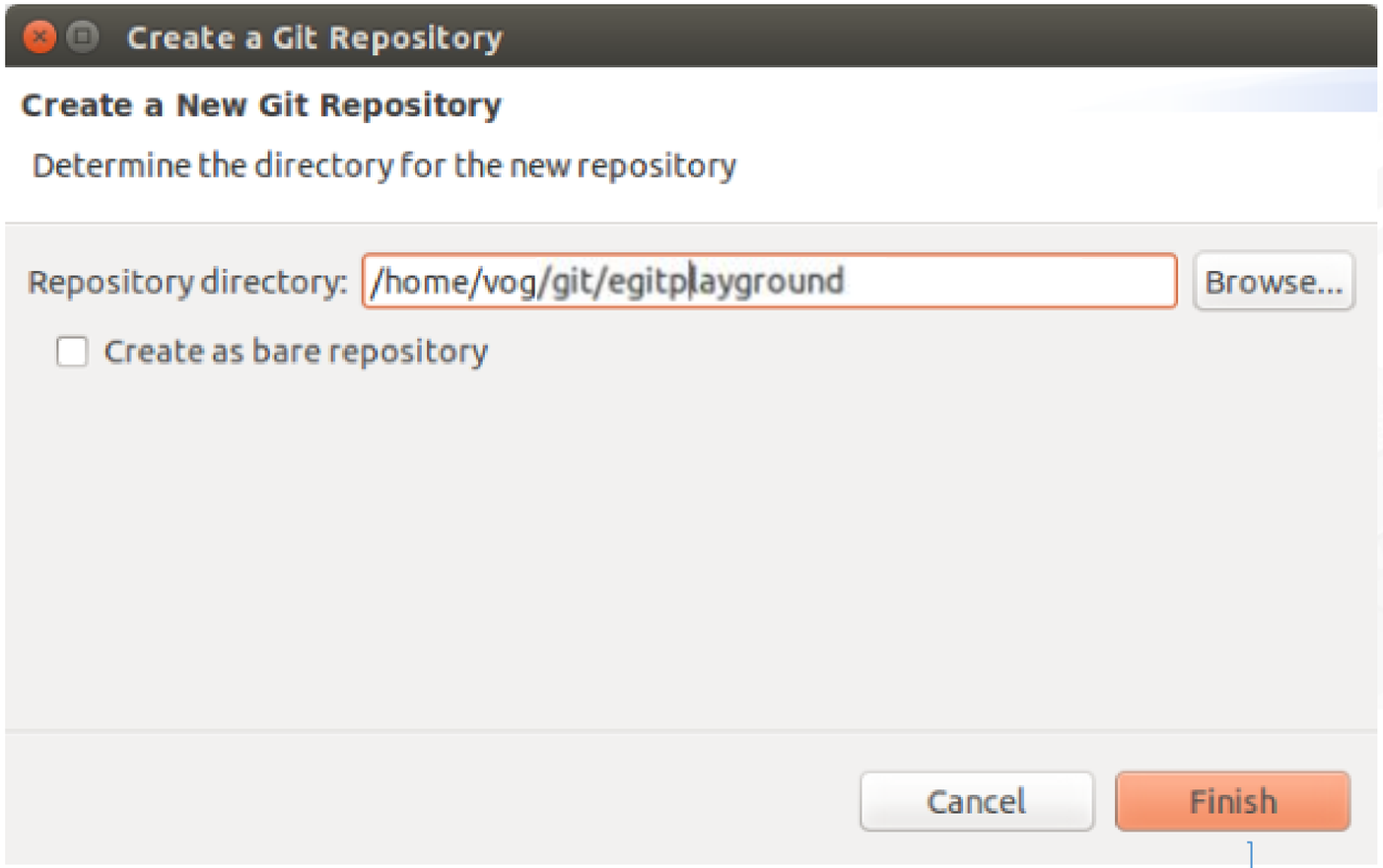
- Create a Git Repository
- Create a ".gitignore" file
- Create an Eclipse project
- Share Project
- Use Git Staging
- Commit Changes
- Review Commit History

Open the Git Repositories view by navigating to the **Window>Show>View>Other>Git>Git Repositories** menu entry. From the toolbar, select the **Create a new Git Repository** and add it to this view entry.

# Create a Git Repository using Eclipse

- Create a Git Repository
- Create a ".gitignore" file
- Create an Eclipse project
- Share Project
- Use Git Staging
- Commit Changes
- Review Commit History

A dialog box appears which allows you to specify the directory for the new Git repository. Select a new directory outside of your workspace.




Press the Finish button. Now the Git repository is created and a reference to it is added to the **Git Repositories** view.

# Create a .gitignore file

- Create a Git Repository
- Create a ".gitignore" file
- Create an Eclipse project
- Share Project
- Use Git Staging
- Commit Changes
- Review Commit History

All files and directories which apply to the pattern described in a top-level ".gitignore" file are ignored by Git.

**NOTE**

Eclipse Git does not allow to create a file directly in the top-level folder of your repository. You must do this outside of the Eclipse IDE using the command line.

Create a Git Repository

Create a .gitignore file

Create an Eclipse project

Share Project

Use Git Staging

Commit Changes

Review Commit History

# Create an Eclipse Project

Create a new Java project called by navigating to the **File>New>Other>Java Project** menu entry

```
package com.sample.git.first;

public class GitTest {
  public static void main(String[] args) {
    System.out.println("Git is fun");
  }
}
```

Create a Git Repository

Create a .gitignore file

Create an Eclipse project

Share Project

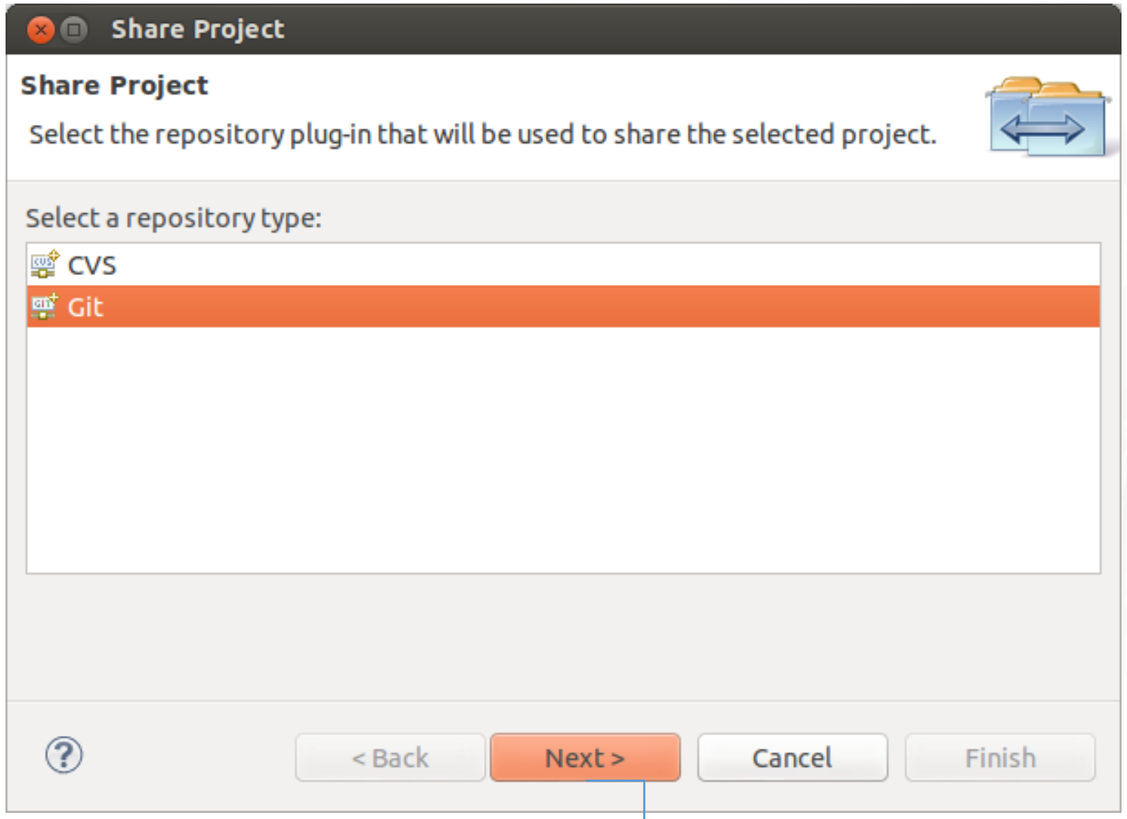
Use Git Staging

Commit Changes

Review Commit History

# Share Project

Share your new project under version control with Git by right clicking on your project and selecting **Team>Share>Git.**

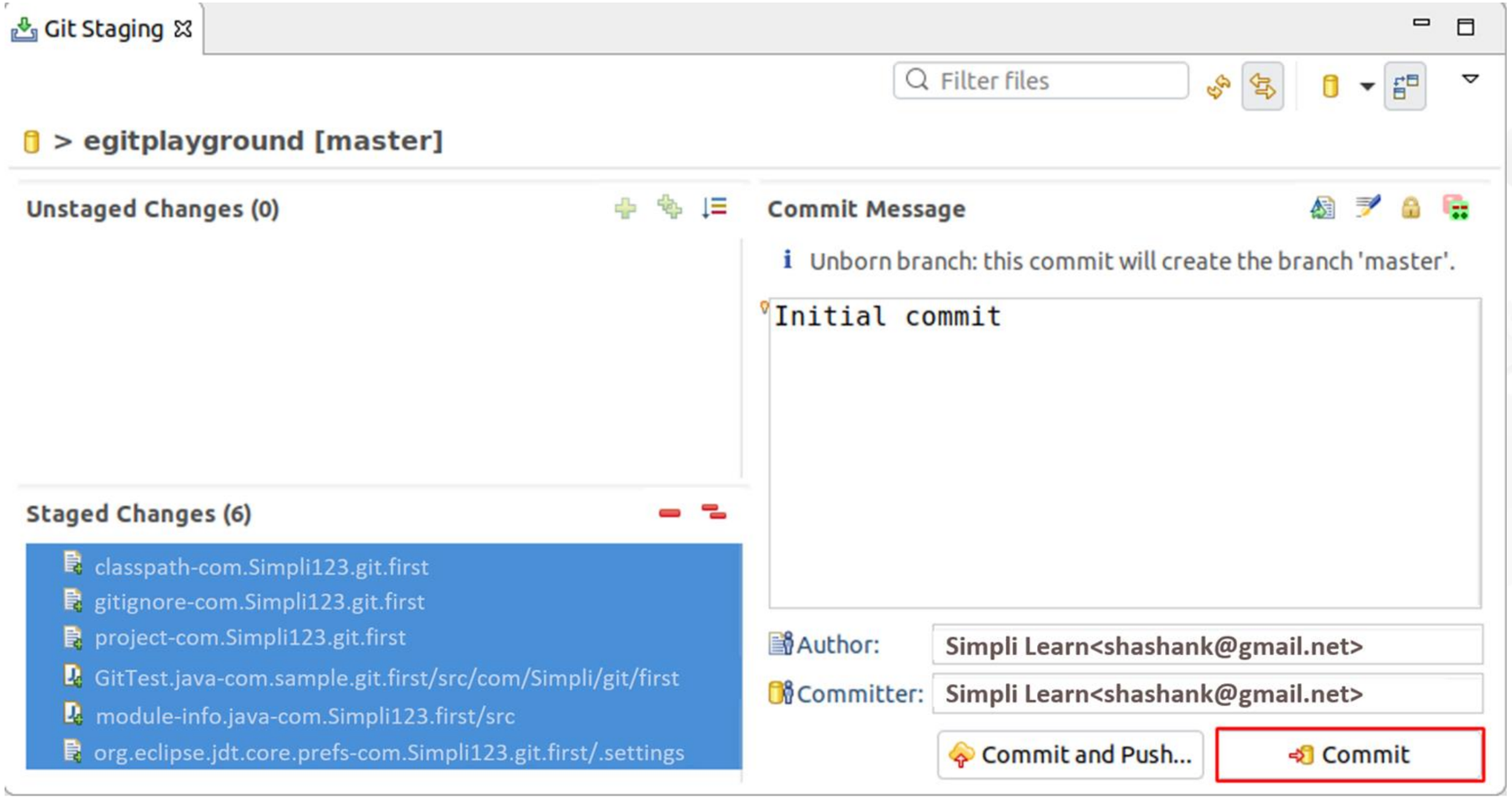



Click **Next** and select your existing Git repository from the drop-down list.

# Use Git Staging Area

- Create a Git Repository
- Create a .gitignore file
- Create an Eclipse project
- Share Project
- **Use Git Staging**
- Commit Changes
- Review Commit History

Open the Git Staging view, using **Window>Show>View>Other>Git>Git Staging**. In this view drag all files into the **Staged Changes** area, write a meaningful commit message, and press the commit button.



# Commit Changes

Create a Git Repository

Create a .gitignore file

Create an Eclipse project

Share Project

Use Git Staging

Commit Changes

Review Commit History

Change the System.out.println message in your GitTest class. Stage the file and commit the changes.

```
package com.sample.git.first;

public class GitTest {
  public static void main(String[] args) {
    System.out.println("Git is cool");
  }
}
```

Create a Git Repository

Create a .gitignore file

Create an Eclipse project

Share Project

Use Git Staging

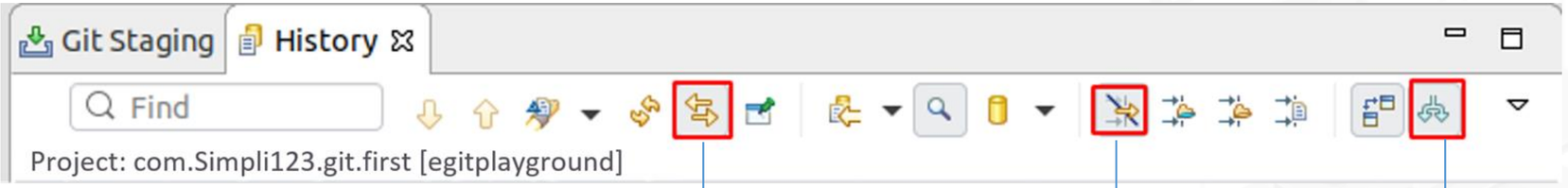
Commit Changes

Review Commit History

# Review Commit History

Open the **History view** by navigating to the **Window>Show>View>Other>Team>History** menu. Use it to review which files were included in your individual commits.

In the **History** view click the toggle buttons as shown in the screenshot.




Link with Editor and Selection

Show all changes in repository containing the selected resources

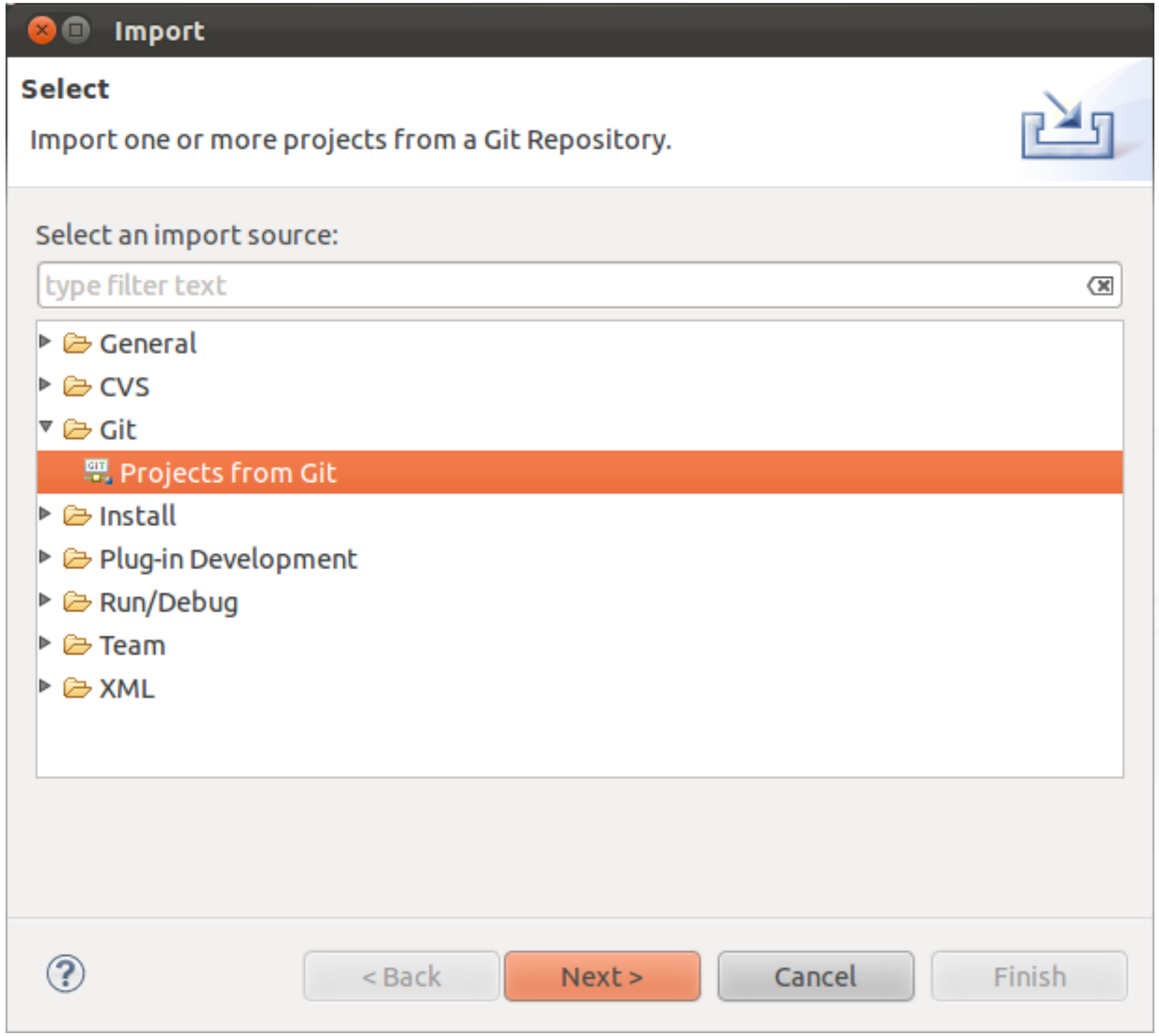
Show all Branches and Tags

# Clone a Repository in Eclipse

# Cloning an Existing Repository

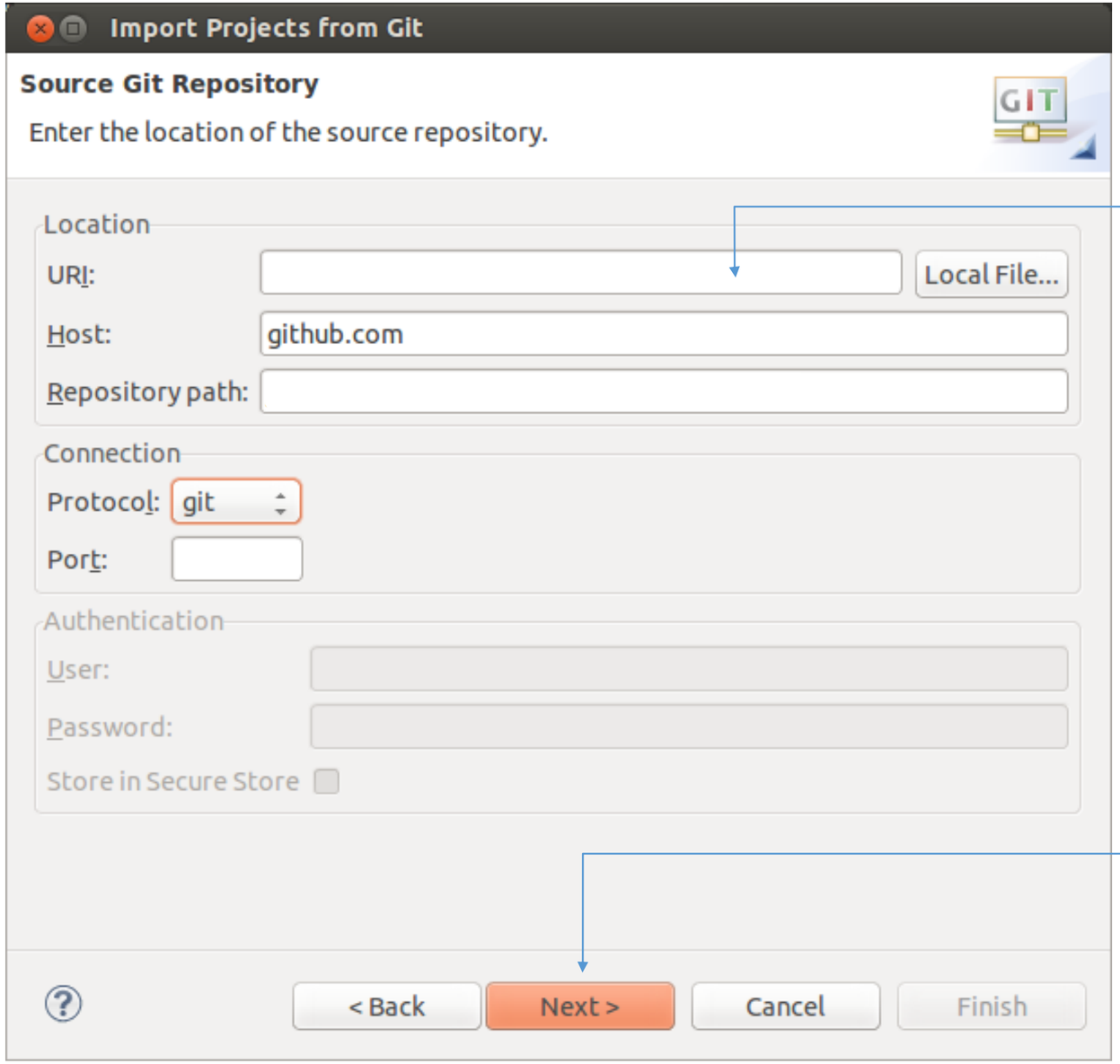
To clone a Git repository and import the existing projects from this repository into your workspace, follow these steps:

1. Select **File**
2. Navigate to **Import**
3. Open **Git** folder and Click **Projects from Git**
4. Select **Clone URL** in the next dialog box



# Cloning an Existing Repository

Enter the URL to your Git repository which you want to clone.

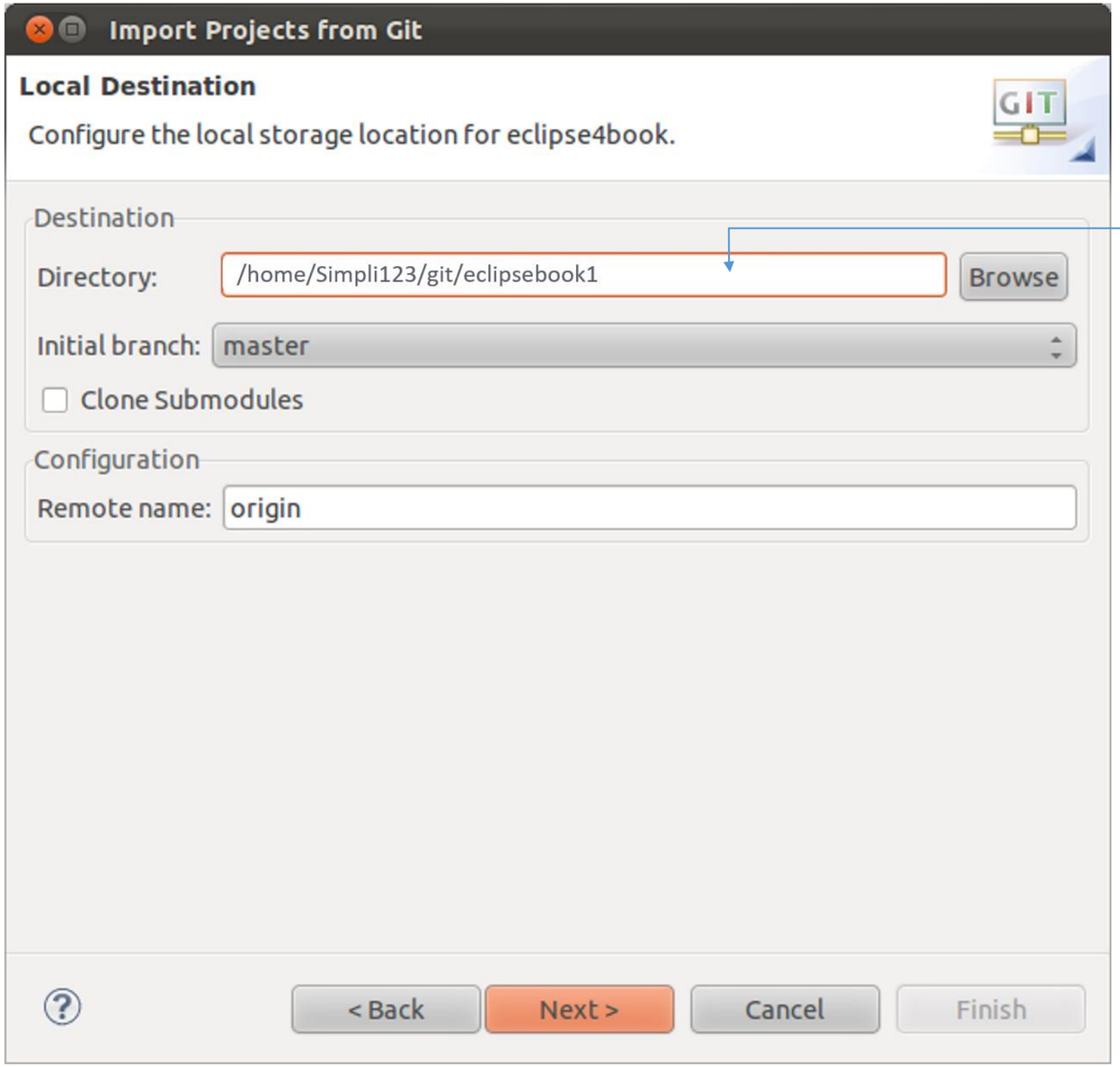



Enter the URL of your repository

After pressing **Next**, the system will allow you to import the existing branches. Select at least the master branch.

# Cloning an Existing Repository

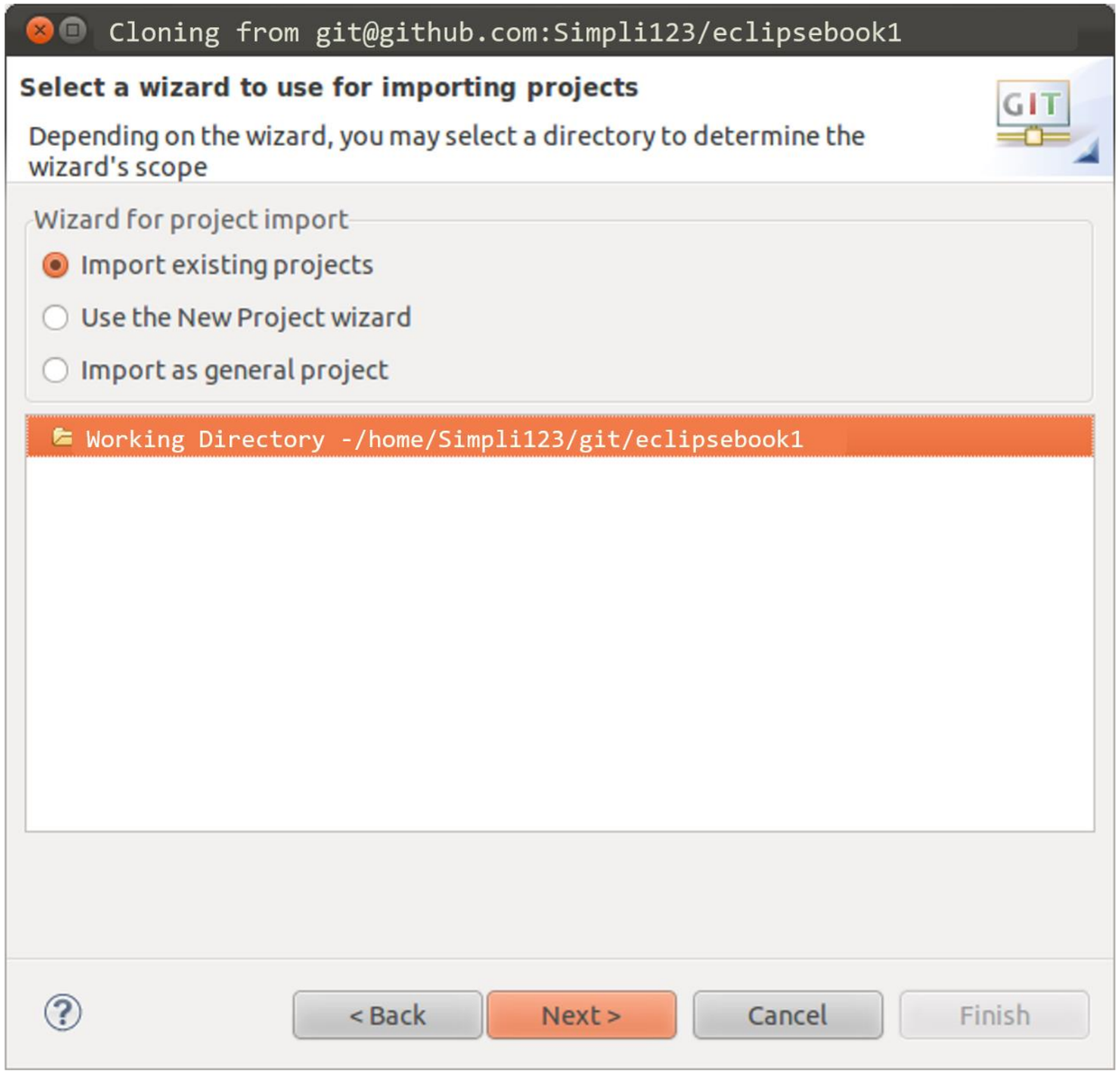
Specify where the repository should be copied to and which local branch should be created initially.




Specify the destination directory

# Cloning an Existing Repository

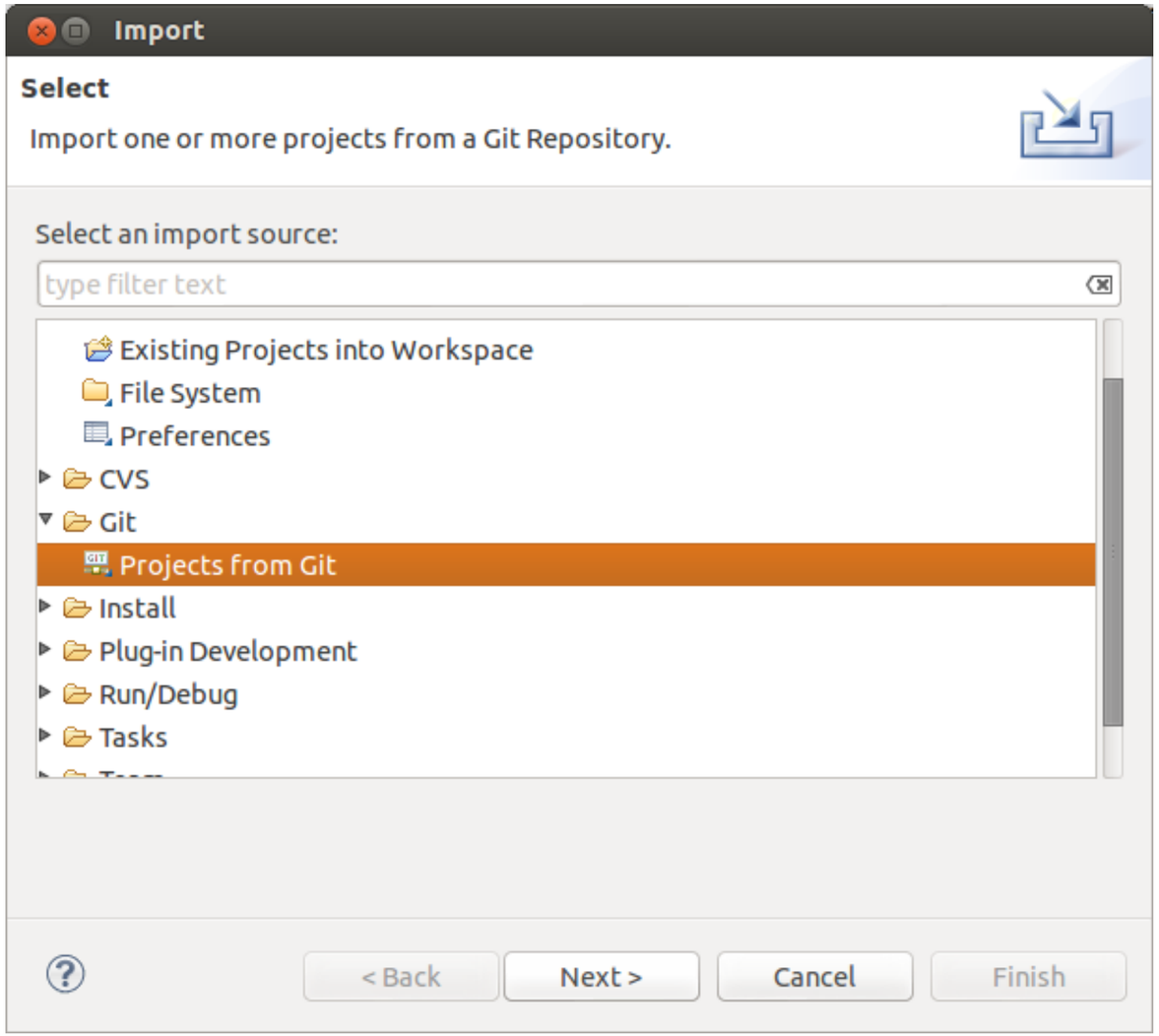
After the Git repository is cloned, import the existing projects.



# Import Projects from a Repository

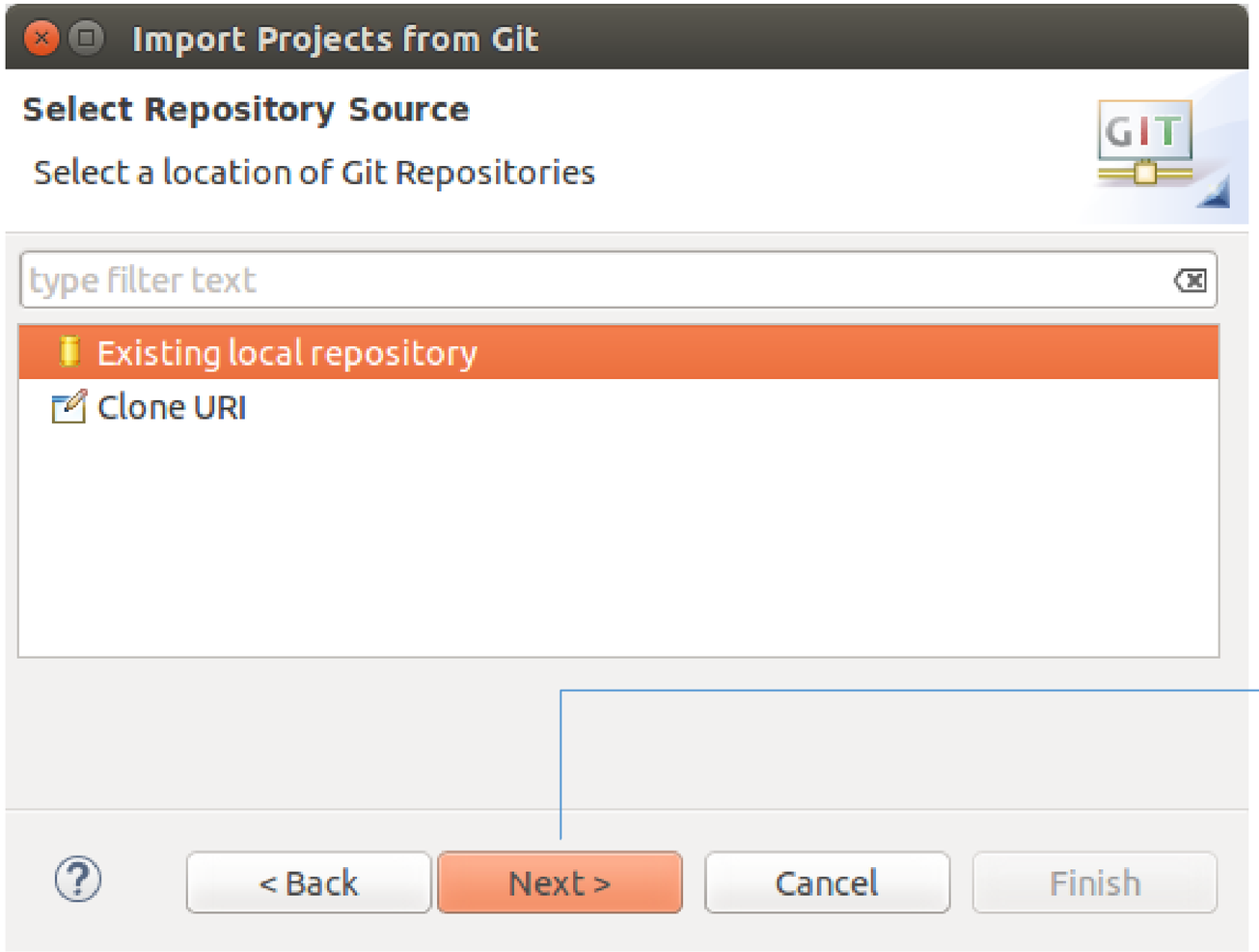
# Import Projects from a Git Repository

Import the projects into your workspace by navigating to the **File>Import>Git>Project** from Git menu entry.



# Import Projects from a Git Repository

Select **Local** to import from a local repository or **Clone URL** to clone the repository.

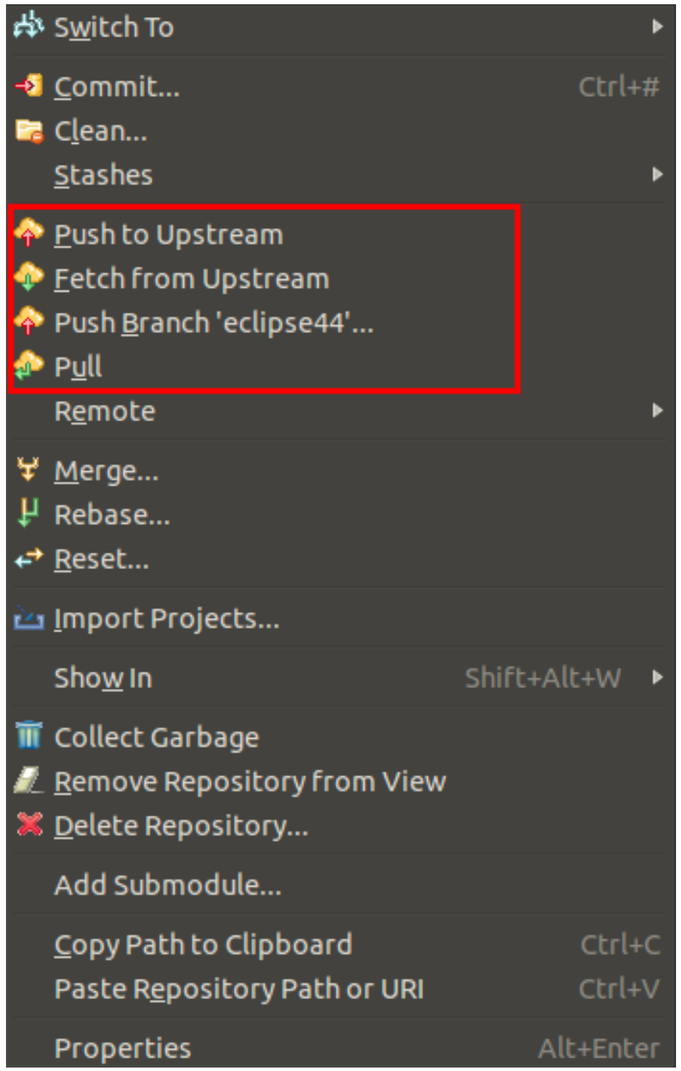



Click **Next** and select the repository to import the project stored in it.

# Git Operations in Eclipse

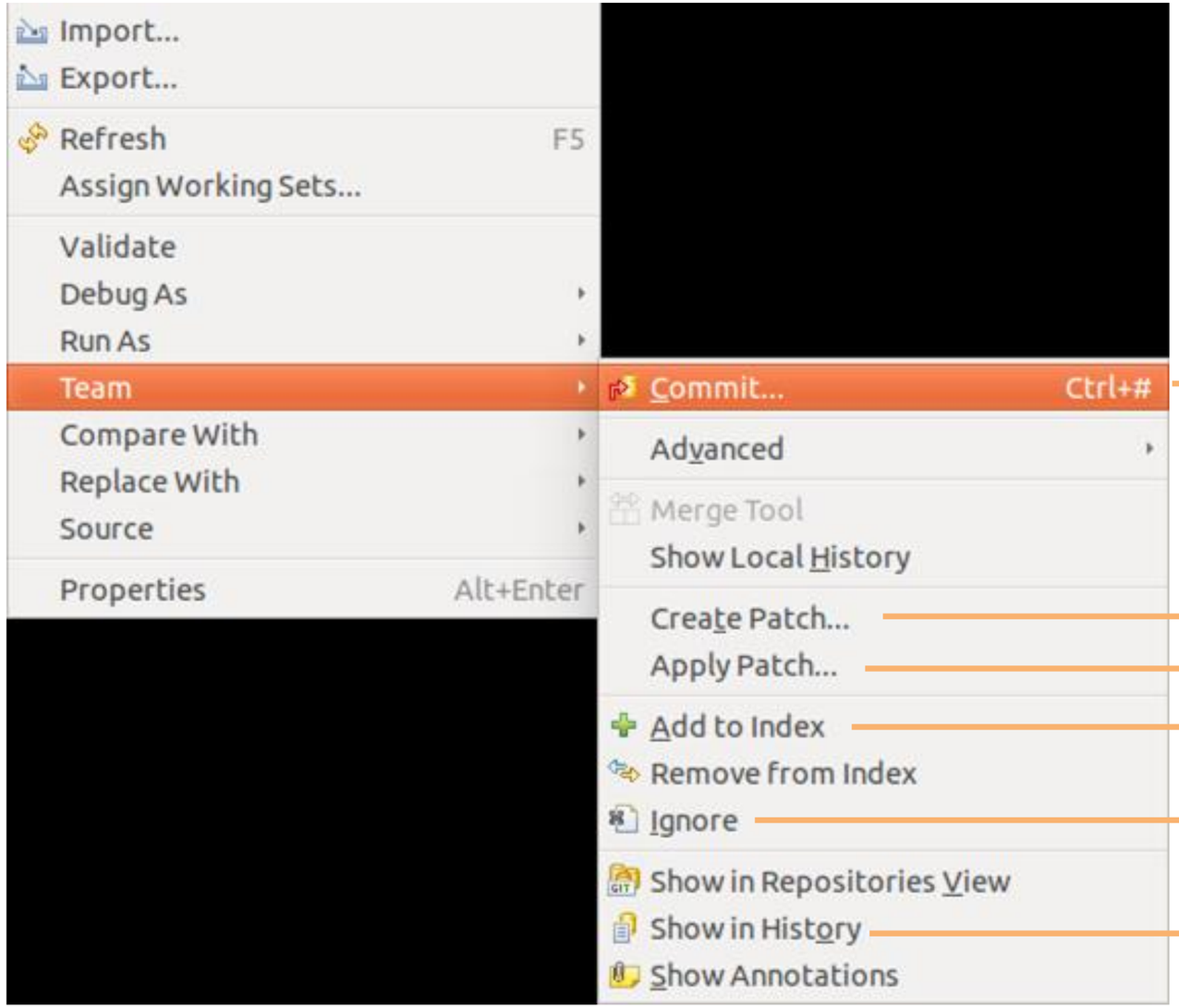
# Pull, Push, and Fetch

Use the **Git Repositories** view to pull, push, and fetch to remote repositories. Right-click on your repository and select the required operation.



# Basic Team Operations

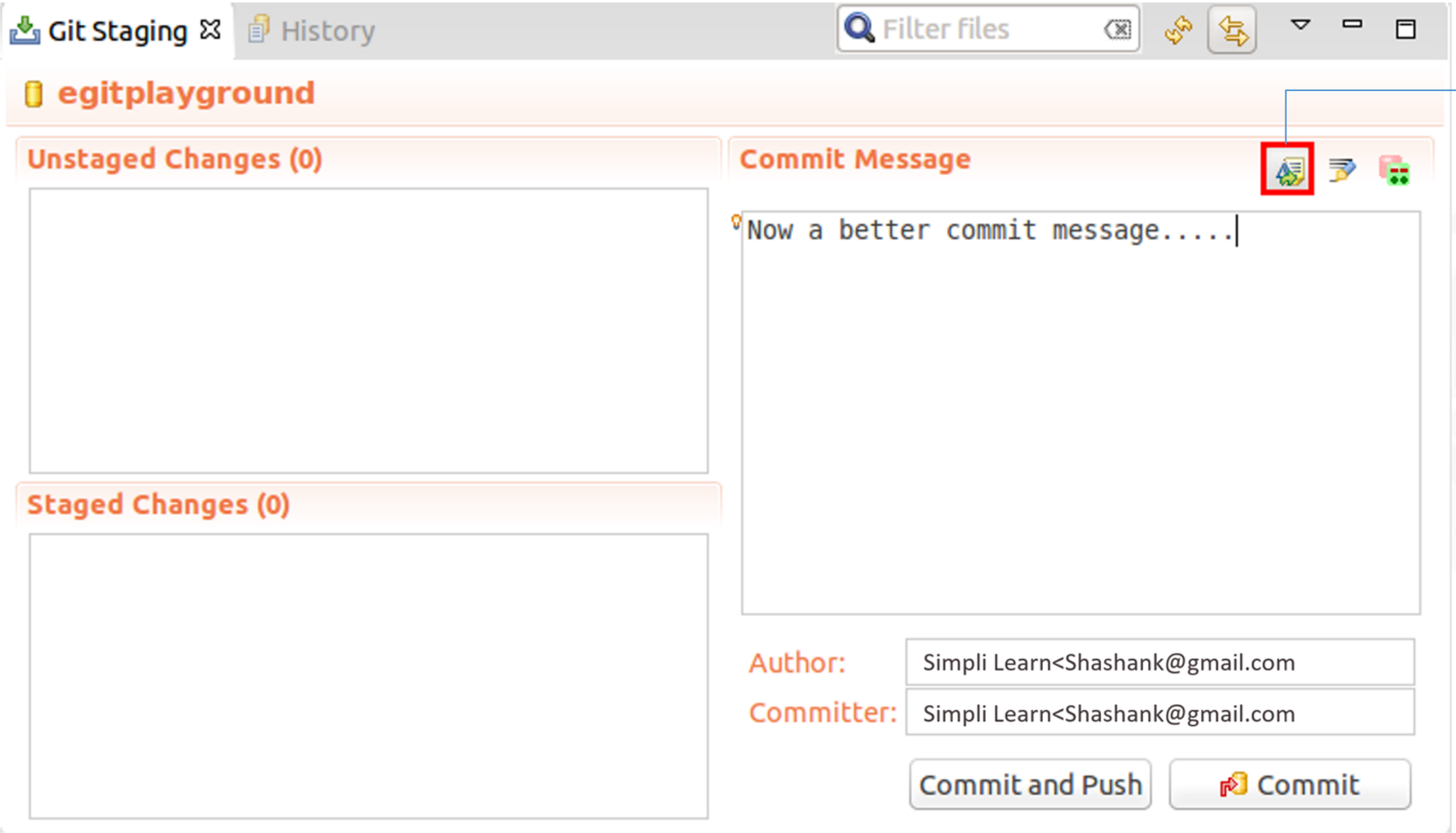
After placing a project under version control, you can start using team operations on your project.




- To open the commit dialog to create a new commit
- To create a patch
- To apply a patch
- To add the selected resources to the Git index
- To add a file to the .gitignore file
- To display the history of the selected resources

# Amending a Commit

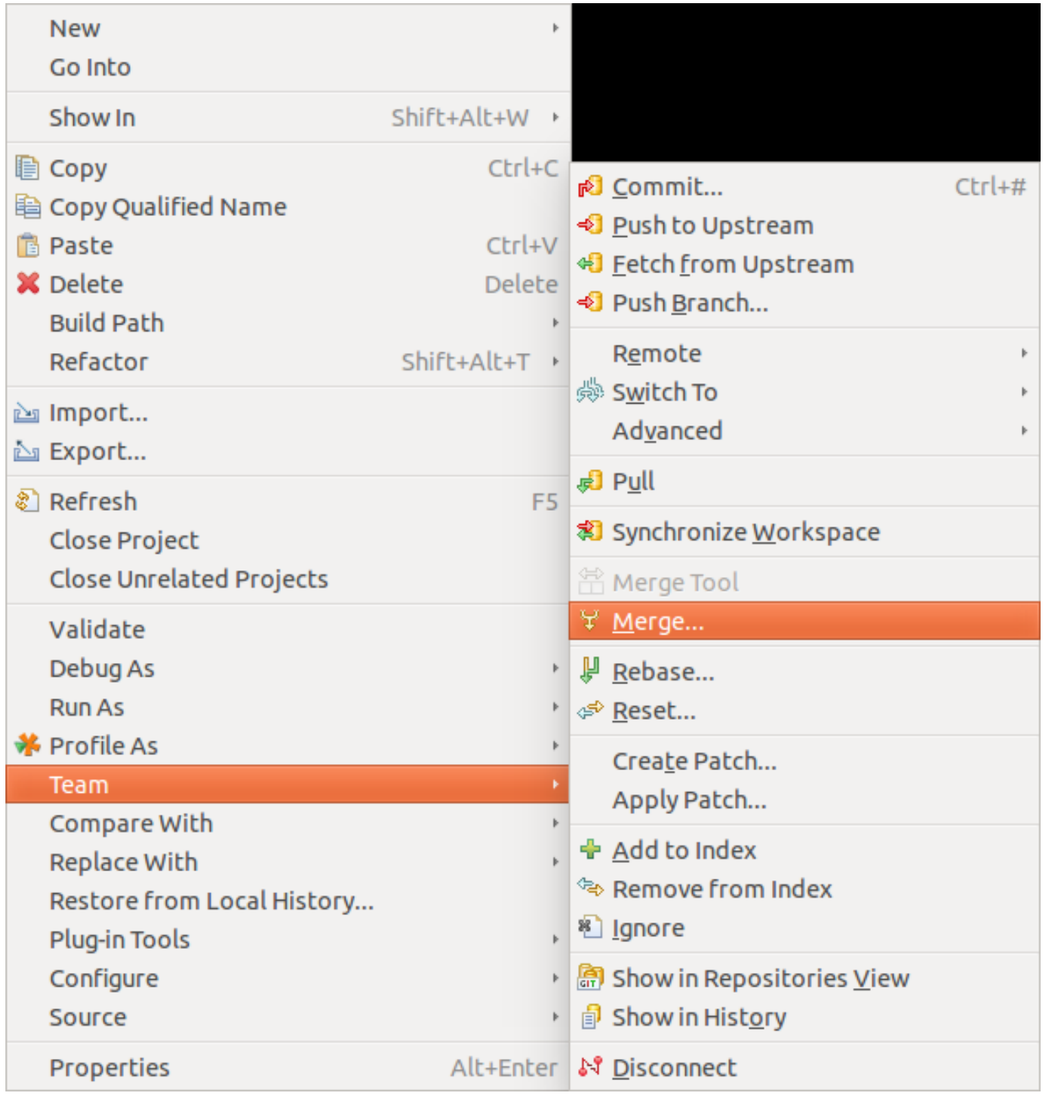
Git amend allows to alter the most recent commit. For example, you can change the commit message or add another modification.




Click here to perform the Git amend

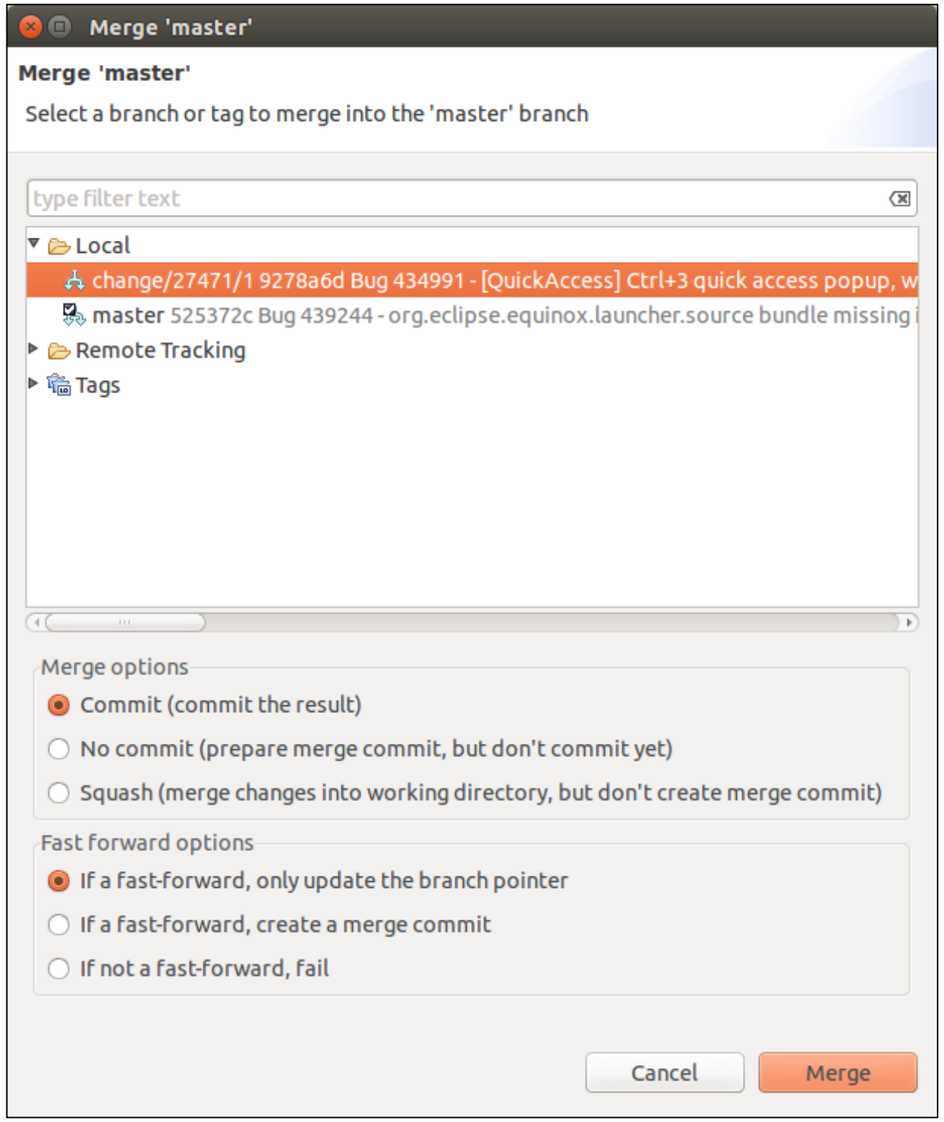
# Starting a Merge Operation in Eclipse

Choose the branch into which you want to merge the changes into, select your project, and click on **Team>Merge** to start the merge dialog.



# Starting a Merge Operation in Eclipse

Select a branch or tag to merge into the master branch.



# Solve the Conflicts Created by Merge

**Problem Statement:** During a Git operation, two changes are conflicting, you have to solve these conflicts manually before the final merge.

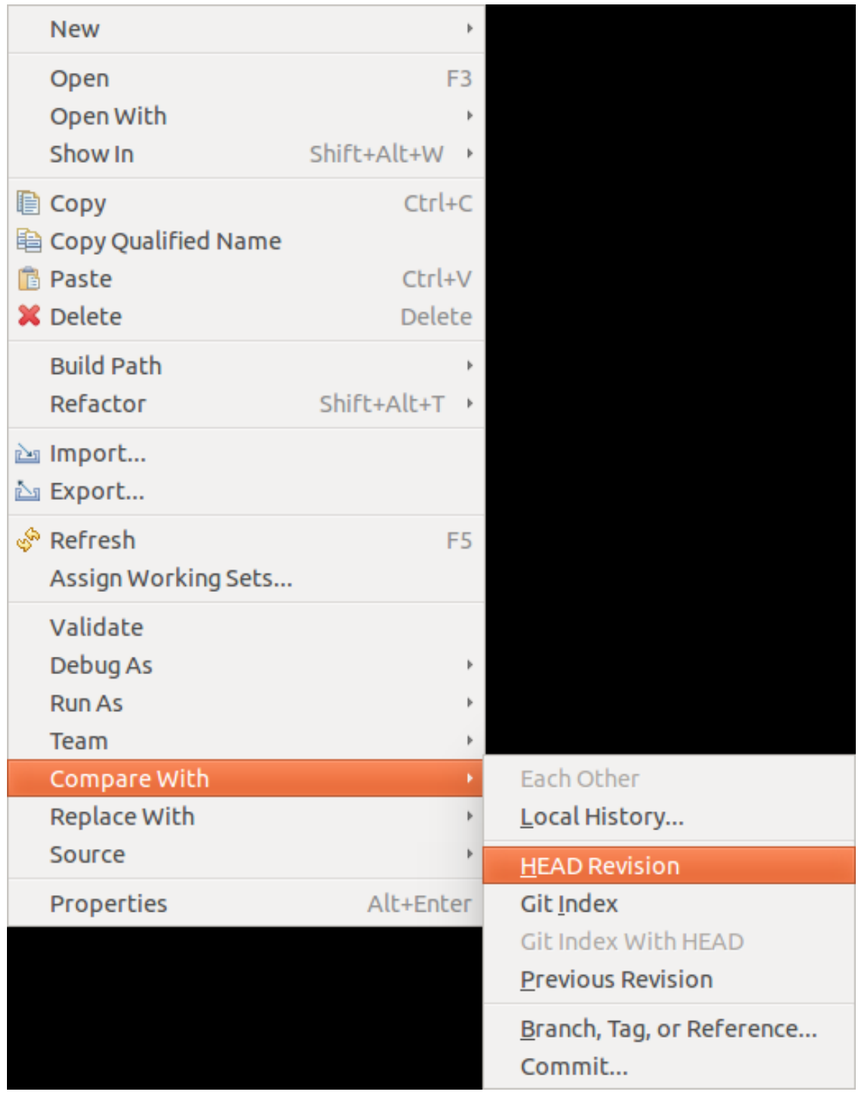
**Steps to Perform:**

- Right click on the file with merge conflicts and select **Team>Merge Tool.**
- Use the **Use HEAD (the last local version) of conflicting files** as merge mode from the dialog box.
- Use **Copy current from right to left** button to copy the changes from right to left.
- Select **Team>Add** from the context menu of the resource to mark the conflicts as resolved and commit the merge commit via **Team >Commit.**

ASSISTED PRACTICE

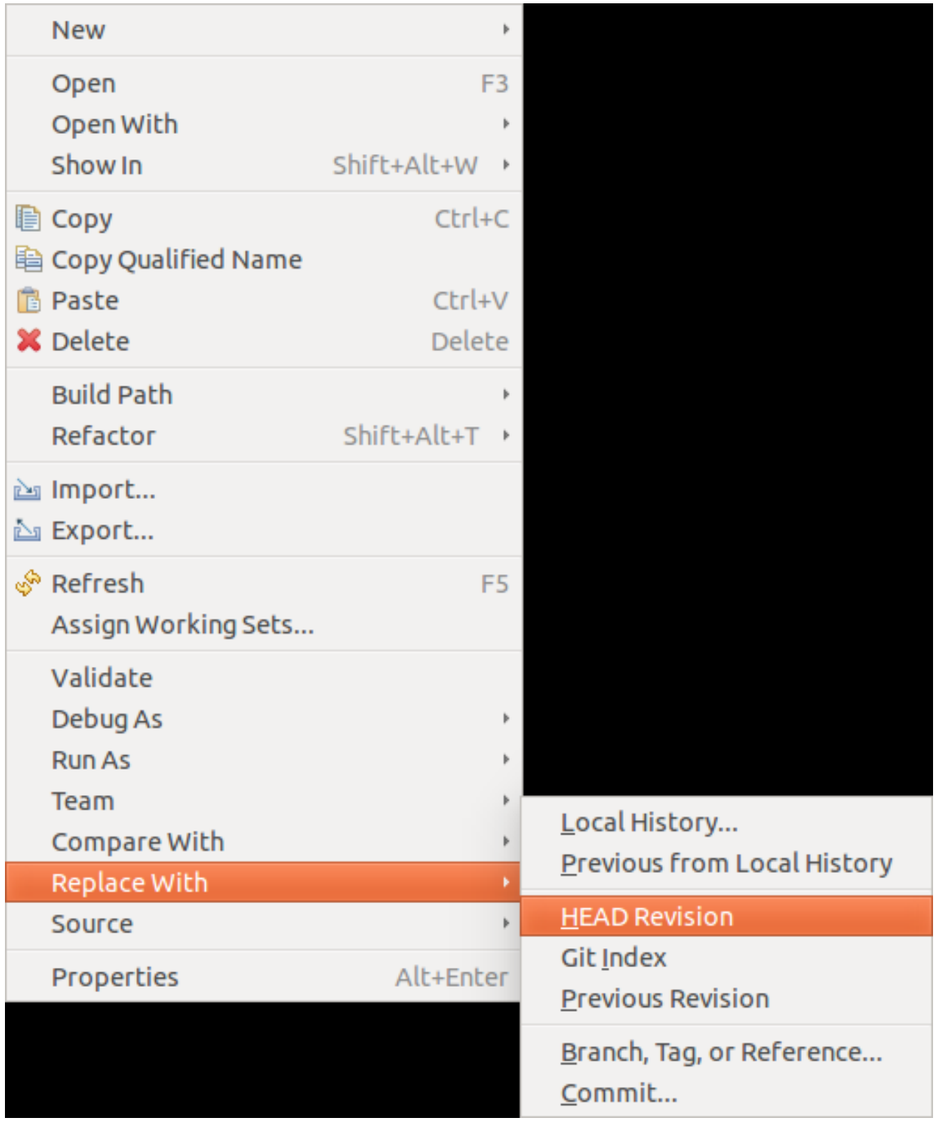
# Comparing Files Based on Git History

Use **Team>Compare With** to open the menu, and select what you want to compare with.



# Replacing Files Based on Git History

The **Team>Replace With** menu entry allows you to replace the current selection with the version contained in the selected commit or the Git Index.

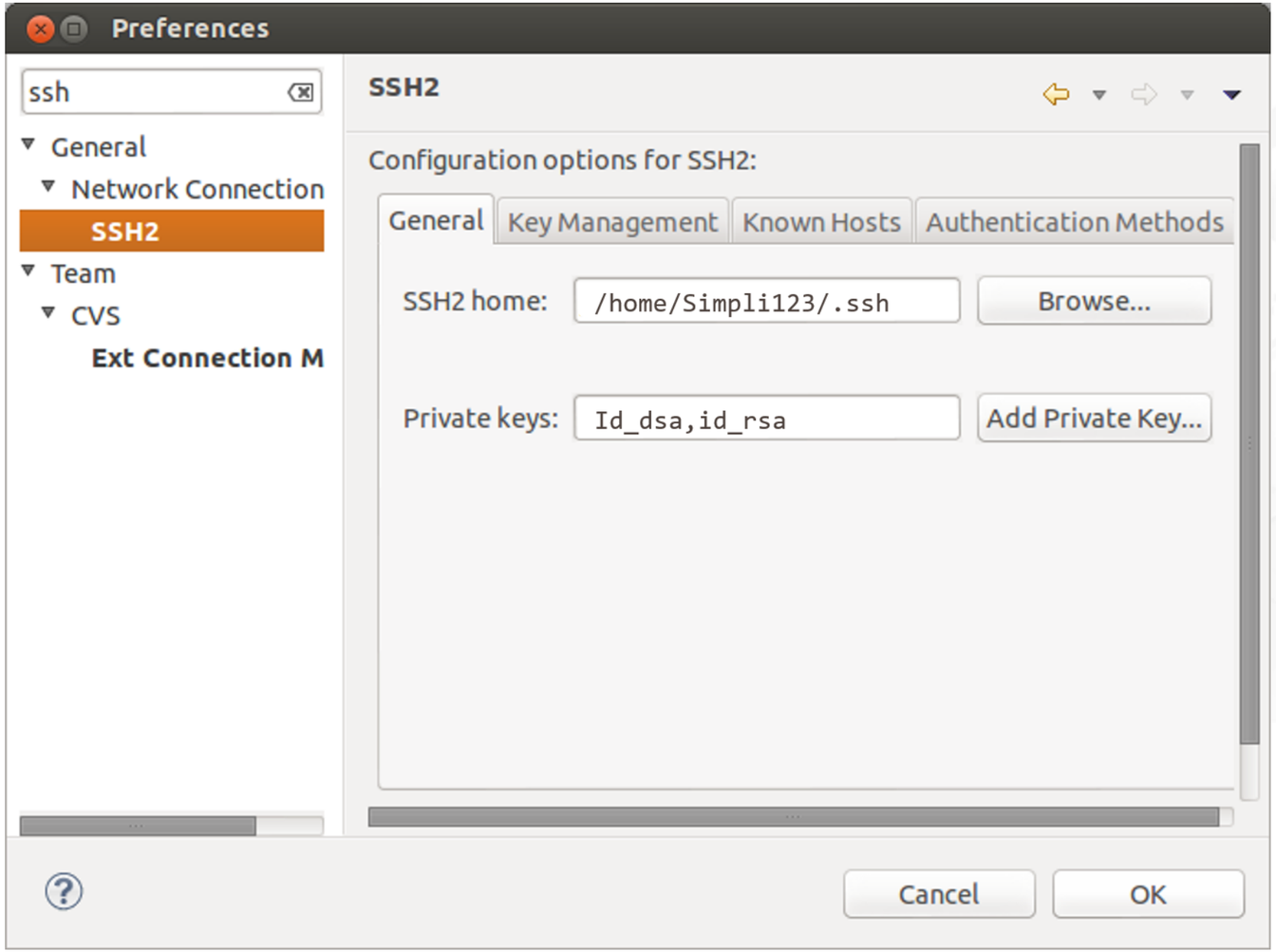


# Eclipse Git with GitHub

# Cloning a Project

Copy the URL from GitHub and navigate to **File>Import>Git>Projects from Git.** Eclipse fills out most of the fields based on the URL in the clipboard. Enter your username and password to push the changes to GitHub.

Alternatively, you can also use an SSH key. You can configure Eclipse to know your SSH by navigating to **Window>Preferences>General>Network Connection>SSH2** preference setting.



# Pushing Changes

After you made changes and committed them to your local repository. This requires write access to the GitHub repository.

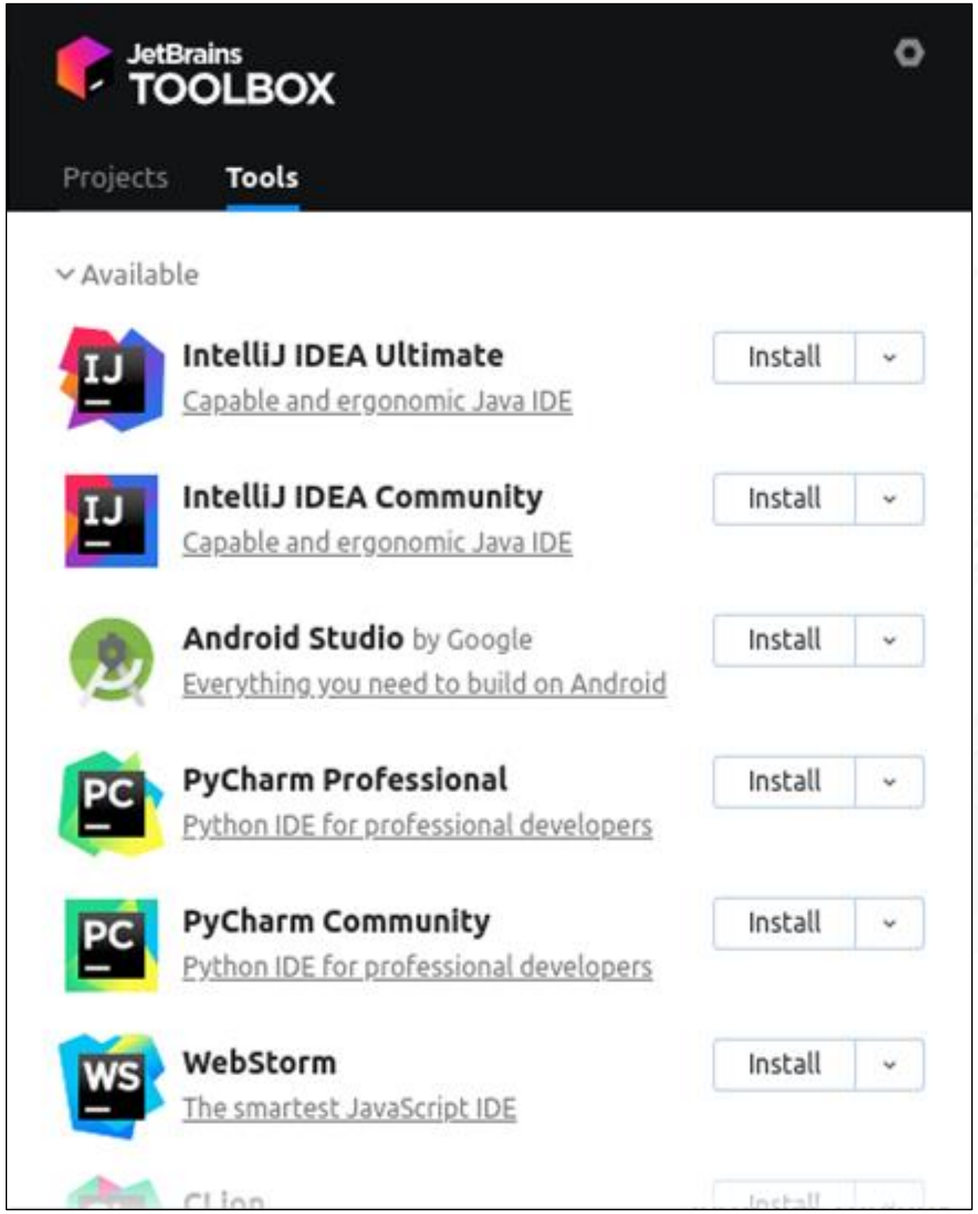
1. Create the Java project in Eclipse.
2. Right-click on the project and select **Team>Push** to push your changes to your GitHub repository.

# Git with IntelliJ

# Installing IntelliJ IDE

## Install the Toolbox App

1. Visit https://www.jetbrains.com/toolbox-app/ to download the tarball .tar.gz.
2. Extract the tarball to a directory that supports file execution.
3. Execute the **jetbrains-toolbox** binary from the extracted directory to run the Toolbox app, and select the product and version you want to install.
4. Log in to your JetBrains account from the Toolbox app and it will automatically activate the available licenses for any IDE that you install.



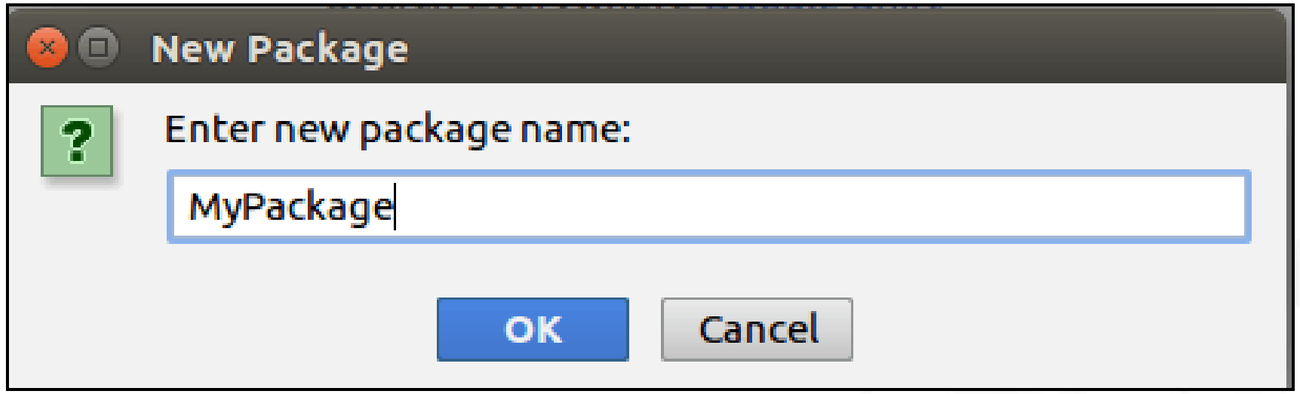
# Create a Project

1. Start IntelliJ IDE
2. Click **Create New Project**
3. Select the **Java Project** and **Project SDK,** and click **Next**
4. Type the **Project name** and select **Project location**. Click **Finish.**

# Create a Package and a Java Class

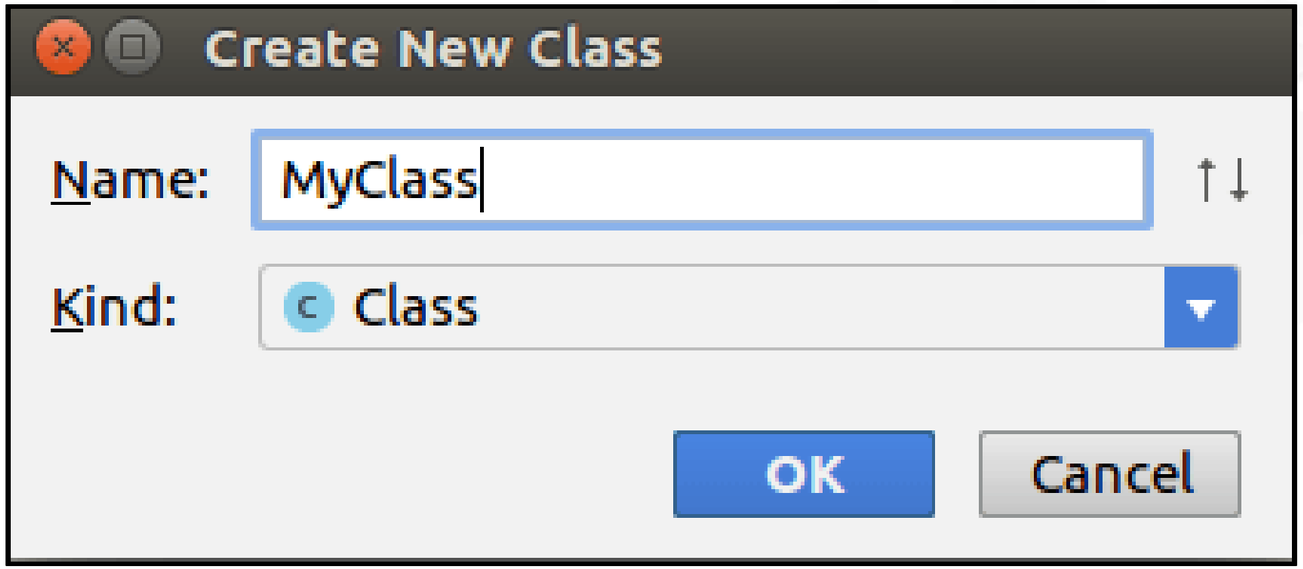
**Create a Package:**

1. Go to **Project Structure**.
2. Right-click and navigate to **src>New> Package**.
3. Enter the package name, and click **OK**.




**Create a Java Class:**

1. Go to **Project Structure**.
2. Right-click and navigate to **src>New> Java Class**.
3. Create a new class name, and click **OK**.

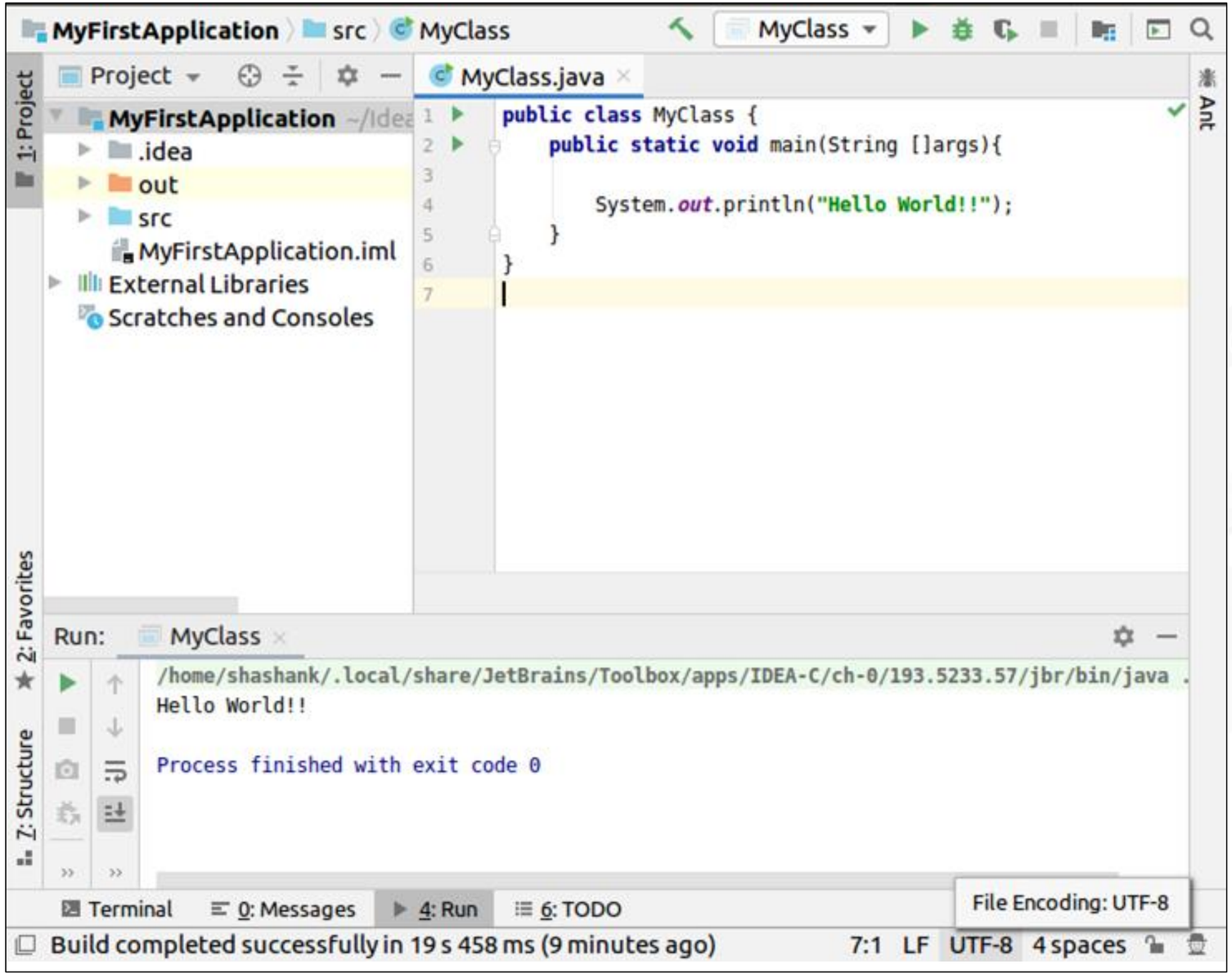


# Run a Java Application

Steps to run a Java application:

1. Write the code in the editor window.
2. Select the class name and click **Run.**

   The output will appear on the **console.**



# Git Integration

Git integration can be performed by the following methods:

Git Integration

01 Sign up on GitHub and create a new repository.

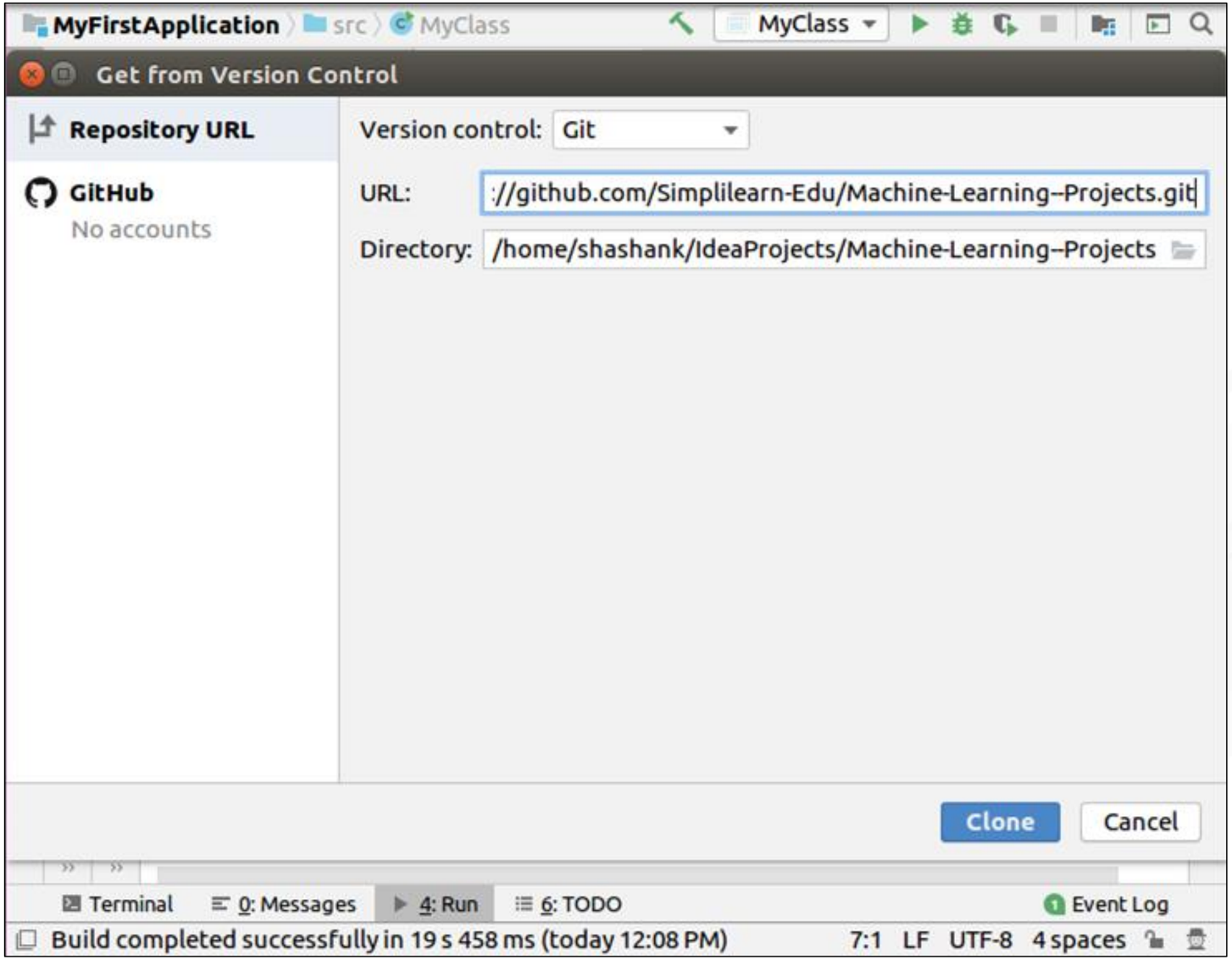
02 Open IntelliJ IDE. Go to **VCS>Enable Version Control Integration**.

03 Select version control system to associate with the project root and select Git in drop down menu.

# Git Clone

Steps to clone a Git repository:

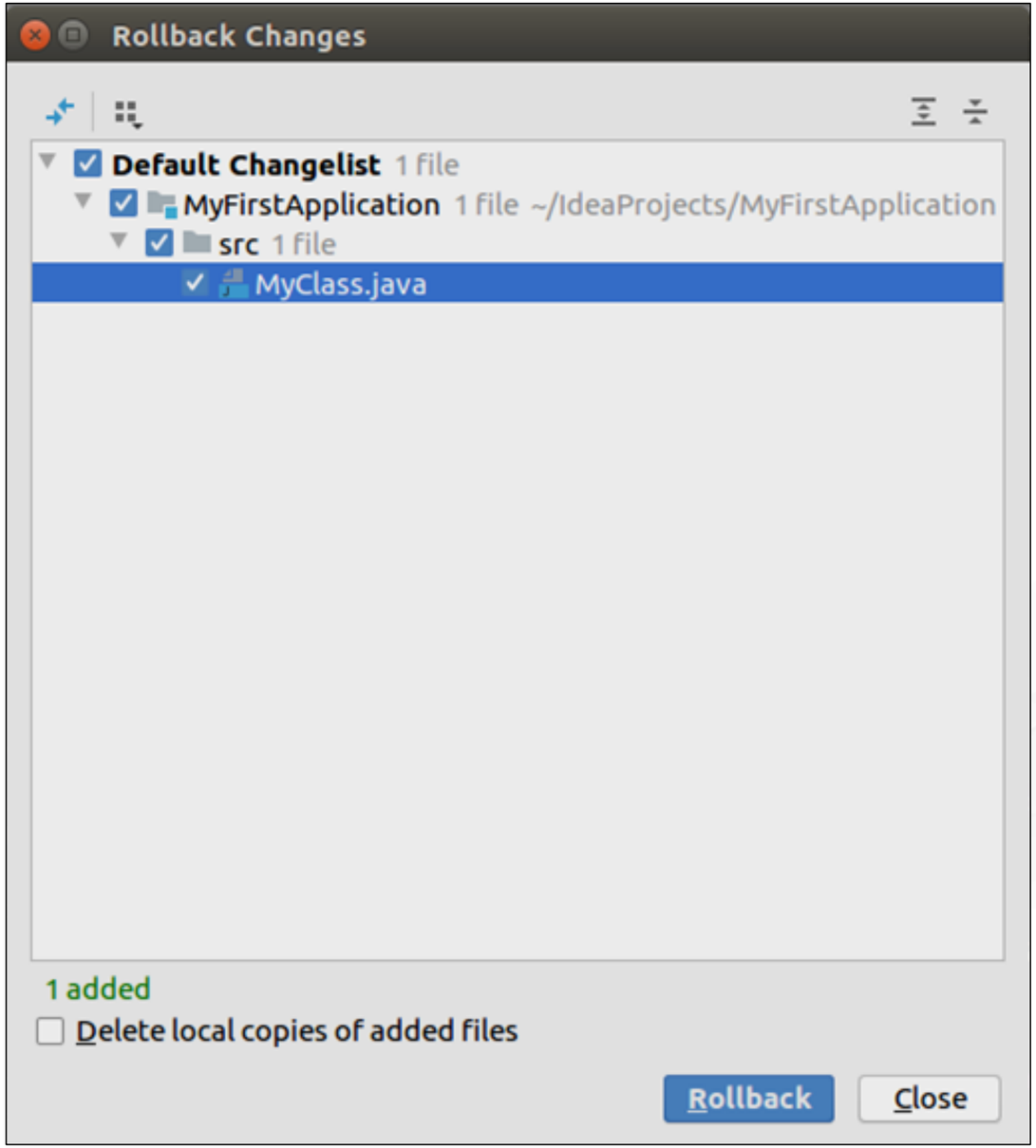
1. Go to **File>New>Project from Version Control>Git.**
2. Enter the repository URL, directory name, and click **Clone.**



# Rollback Local Changes

Steps to rollback local changes:

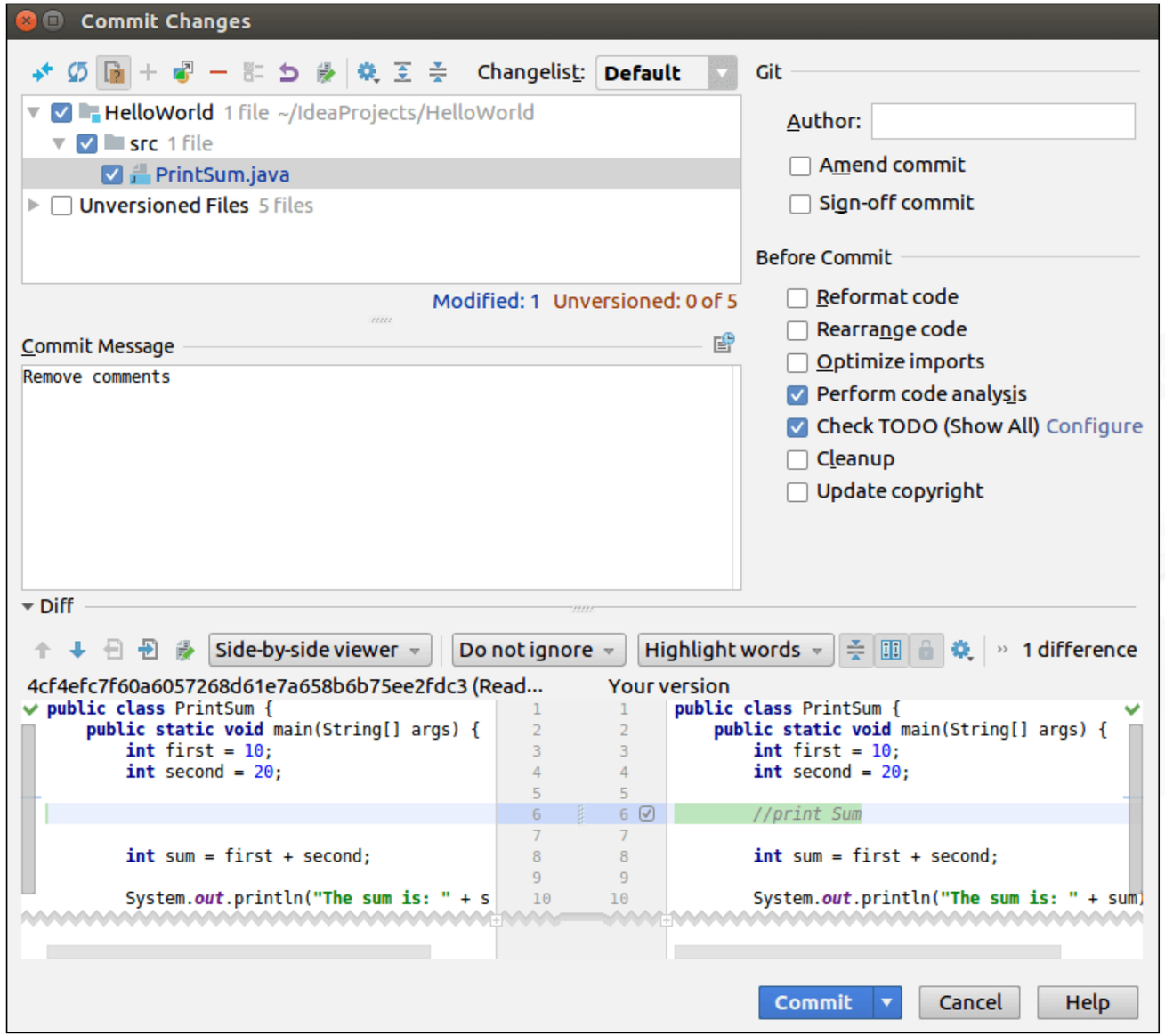
1. Go to **Build>VCS>Git>Revert.**
2. Click **Rollback** in the dialog box.



# Commit Changes Locally

Steps to commit changes locally:

1. Select the modified file under Git version control.
2. Go to **VCS->commit changes or VCS>Git>commit file.**
3. Select the check box of the files to be committed in the dialog box.
4. Enter the **commit message** and click **Commit.**



# Commit Part of a File

**Problem Statement:**. You need to select the modified code chunks related to a specific task, that you want to include in a commit in the **Commit Changes** dialog box.

**Steps to Perform:**

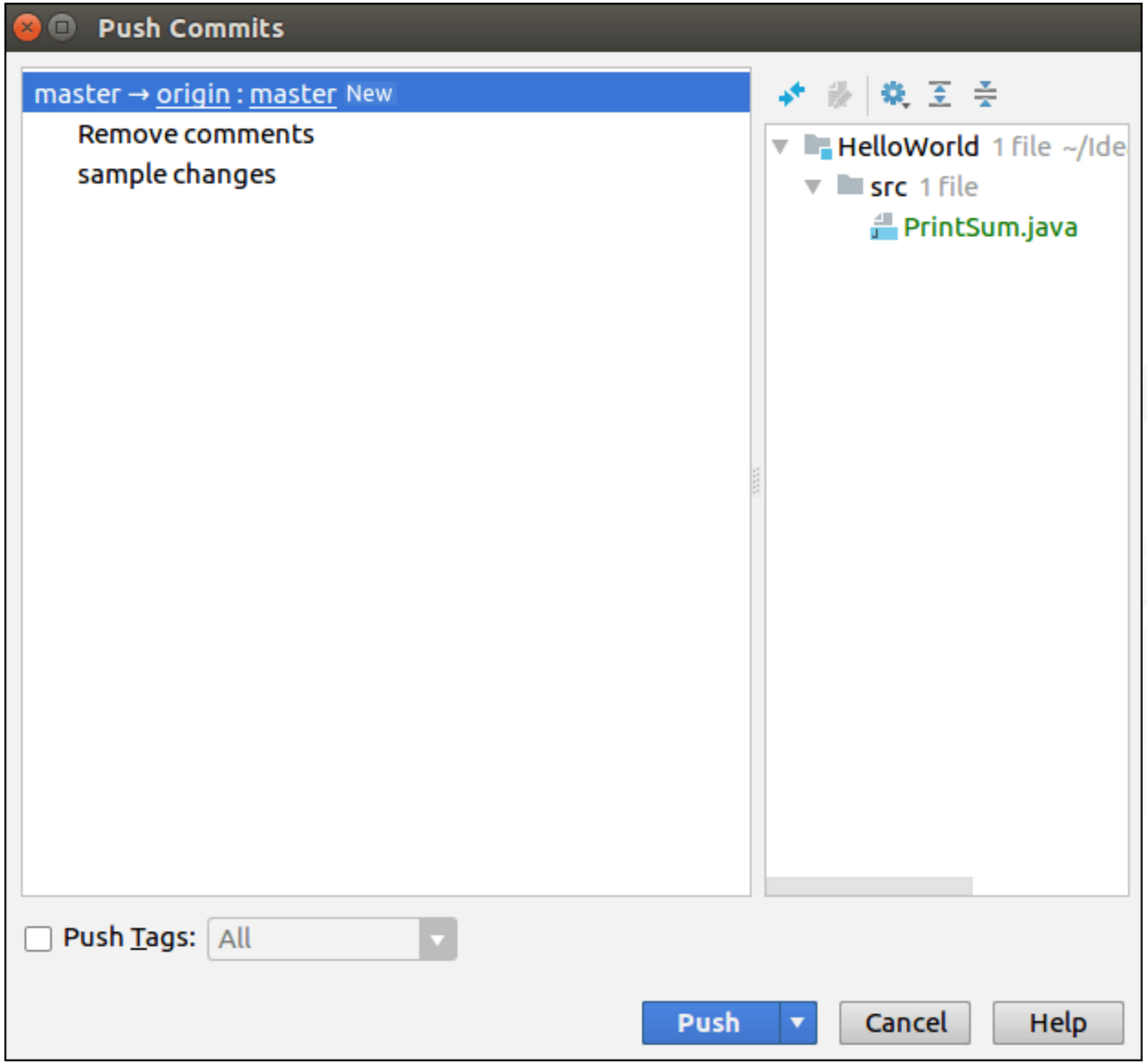
- Invoke the **Commit Changes** dialog box.
- Click **Diff** to display the differences between the repository version and the local version of the selected file.
- Select the checkbox next to each chunk of modified or newly added code that you want to commit.
- Click **Commit.**

ASSISTED PRACTICE

# Push Changes to Remote Repository

Steps to push changes to remote repository:

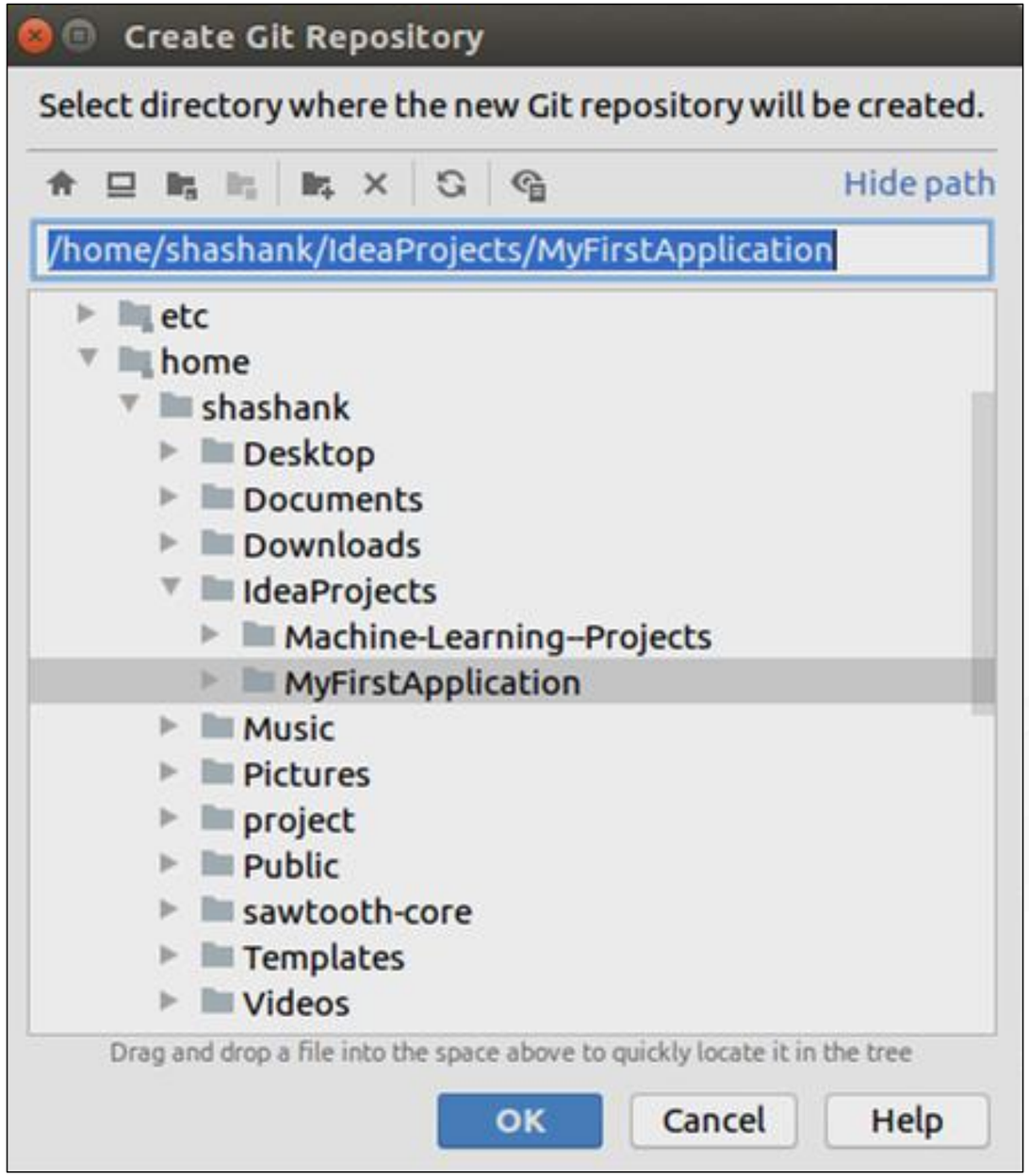
1. Go to **VCS>Git>Push.**
2. Select the commit and click **Push.**



# Import Existing Project to Git

Steps to import an existing project to Git:

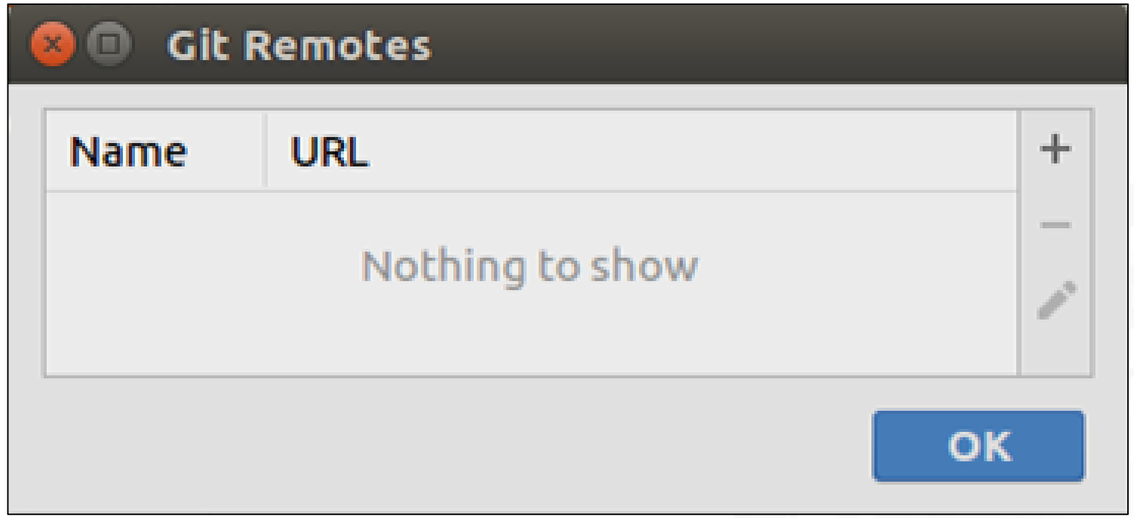
1. Go to **VCS>Import into Version Control>Create Git Repository.**
2. Select the project and click **OK.**

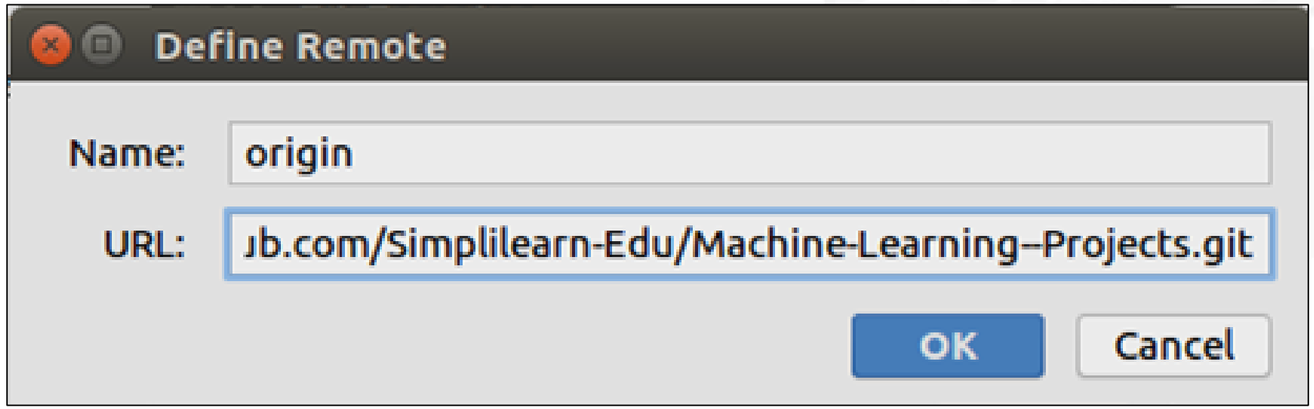


# Define Remote

Steps to define remote:

1. Go to **Build>VCS>Git >Remotes.**
2. Click **+** icon to add URL.
3. Paste the URL and click **OK.**

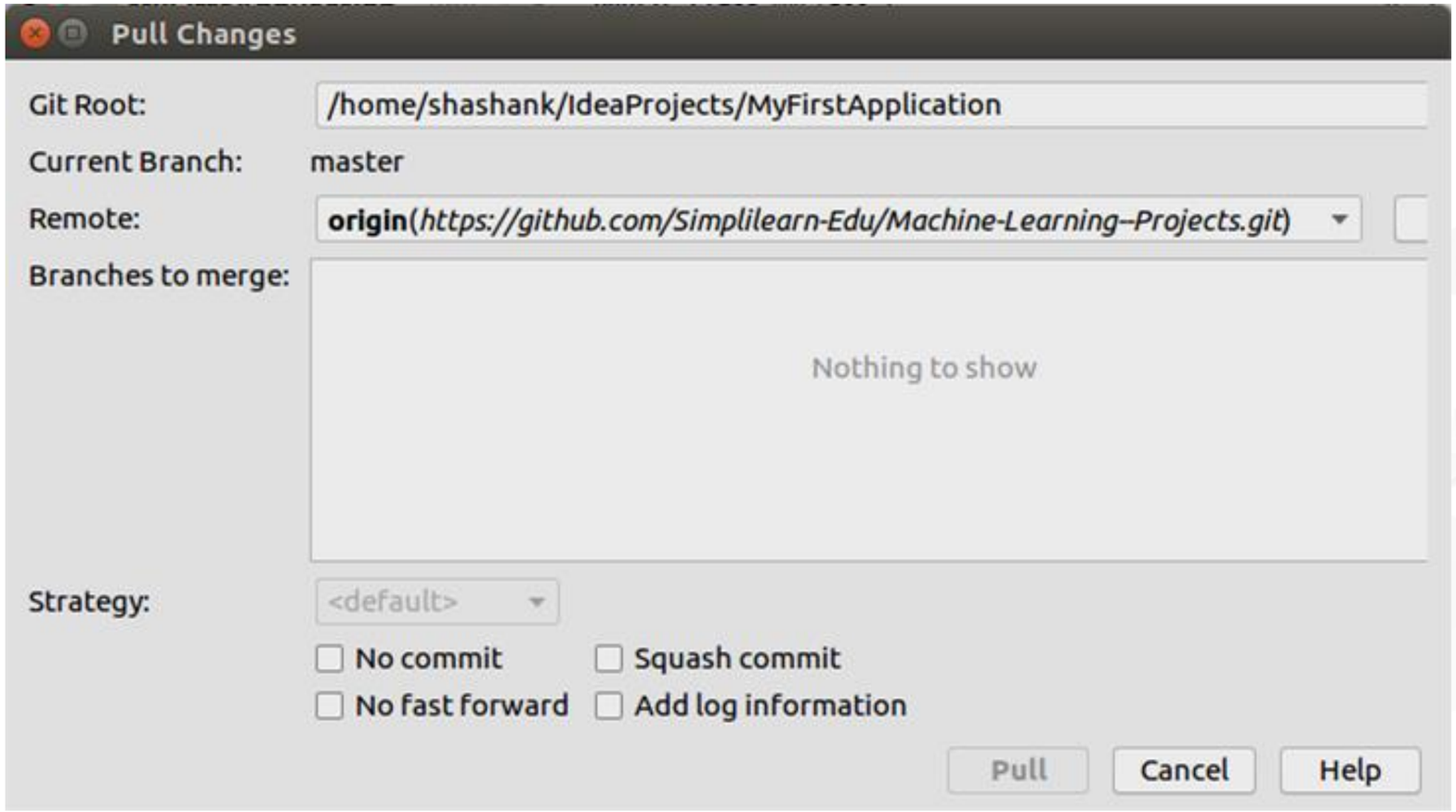


# Get Updates

Steps to receive updates from the repository:

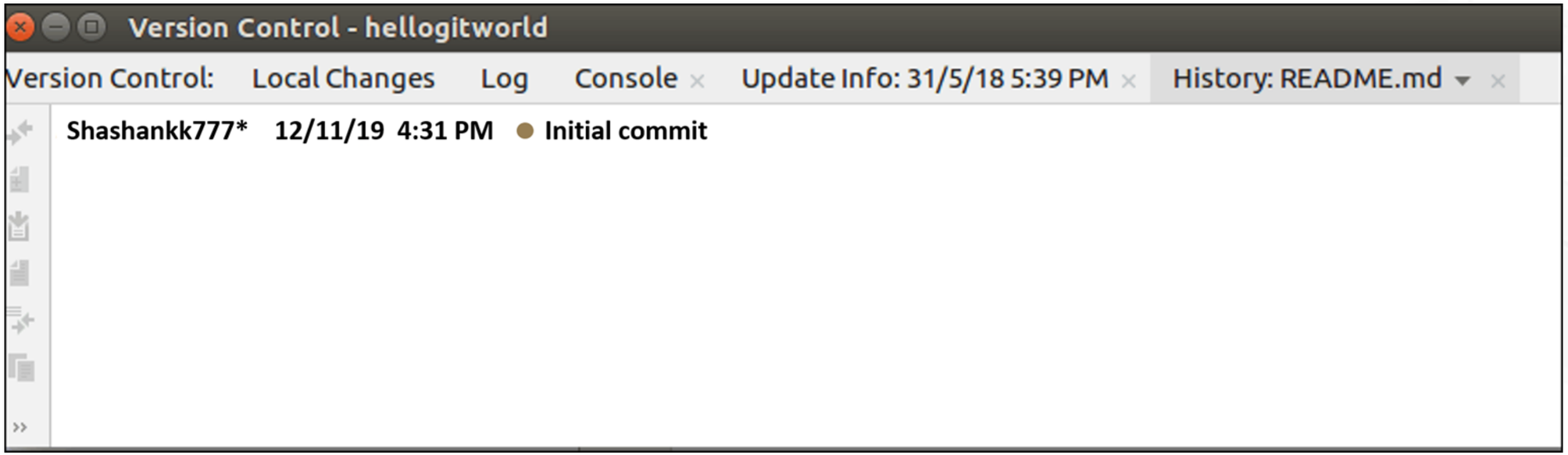
1. Go to **Build>VCS>Git >Pull.**
2. Select the required option.
3. Click **Pull.**



# Show History or Log

Steps to show history or log::

- Go to Build>VCS>Git >History.

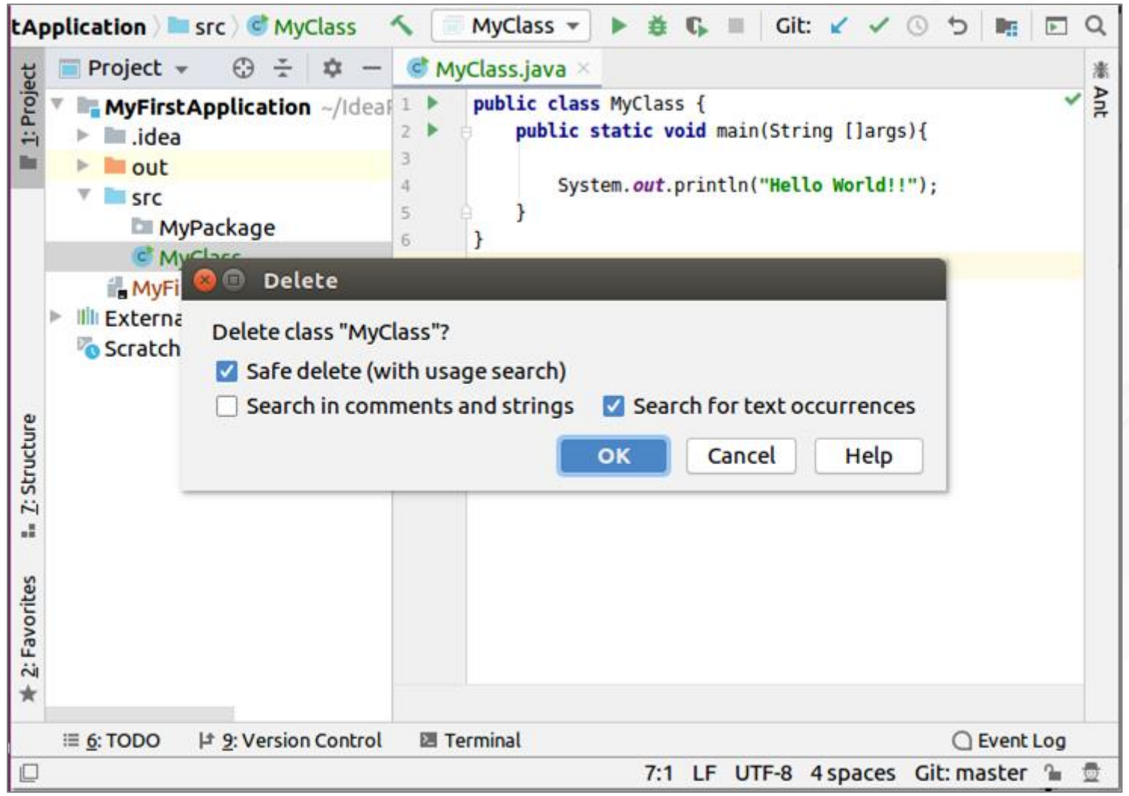


# Delete Files from the Repository

A file deleted under version control still exists in the repository until the changes are committed. The deleted file is placed to the active changelist and is highlighted in grey.

Steps to delete files from the repository :

1. Select a file in the **Project** tool window and press Delete, or choose **Delete** from the context menu.
2. In the dialog that opens, choose whether you want to delete this file without searching for usages, or to perform *safe delete* to ensure that you are deleting an unused file by checking the **Safe delete** option.
3. Commit the changes to the repository.



# Comparing Nodes and File Versions

**Comparing a modified file with its repository version:**

Select a file in the Local Changes tab of the Version Control tool window and click on the toolbar, or press Ctrl+D.

**Comparing the current revision of a file with another branch:**

1. Select a file in the Project tool window and choose <your_VCS>. Select **Compare With Branch** from the context menu.
2. Choose the branch you want to compare the current file version with from the dialog box.

**Comparing the current revision of a file with a selected revision in the same branch:**

1. Select a file in the Project tool window and choose <your_VCS>. Select **Compare With** from the context menu.
2. Choose a revision you want to compare the current file version with from the dialog box.

# Get a Previous Version of a File

**Problem Statement**: You need to return to a particular version of the file after performing a commit that includes changes to several files without discarding the whole document.

**Steps to Perform:**

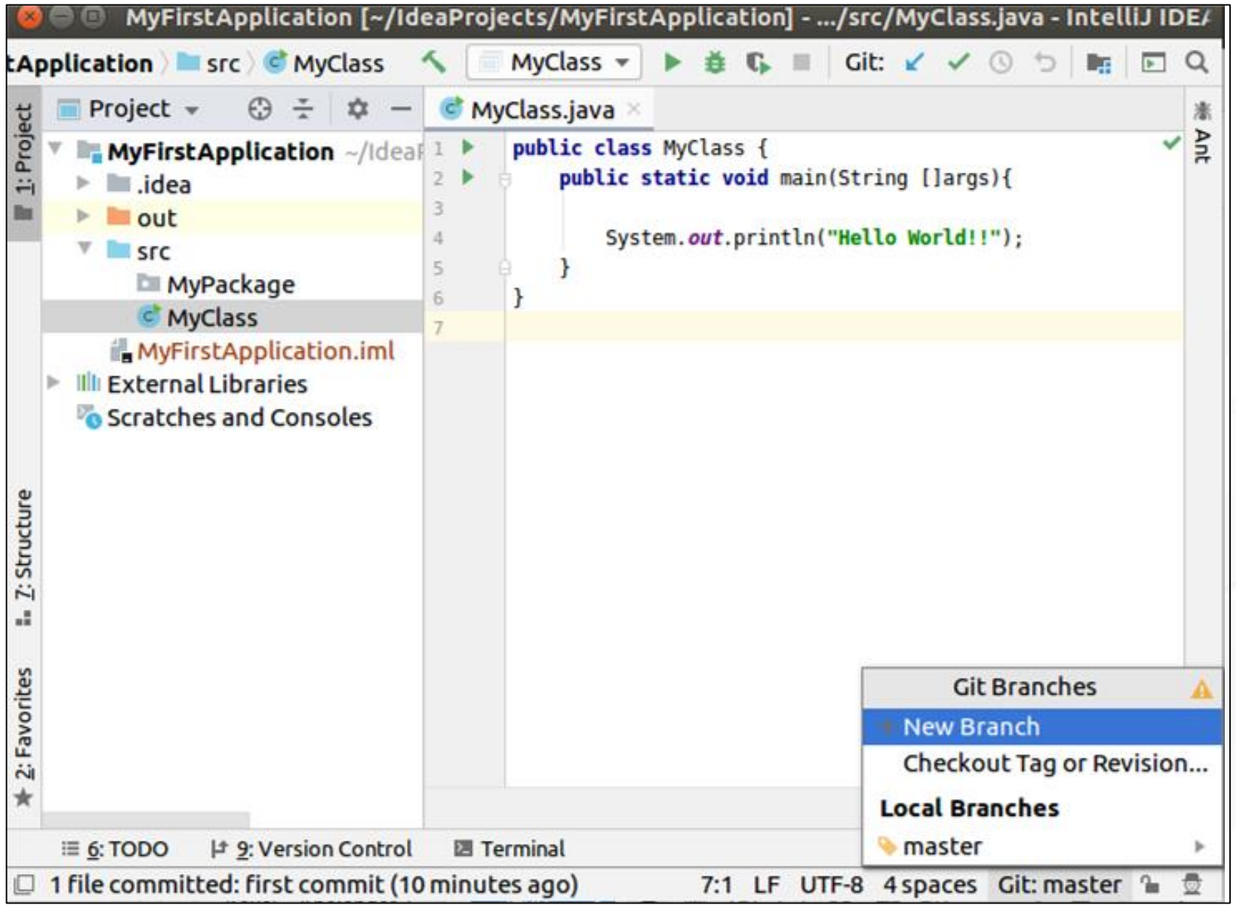
- Select the required file.
- Select **Git>Show History** from the main **VCS** menu or from the context menu of the selection.
- Select the revision you want to roll back to and choose **Get** from the context menu.

ASSISTED PRACTICE

# Git Branches

Steps to invoke Git branches popup:

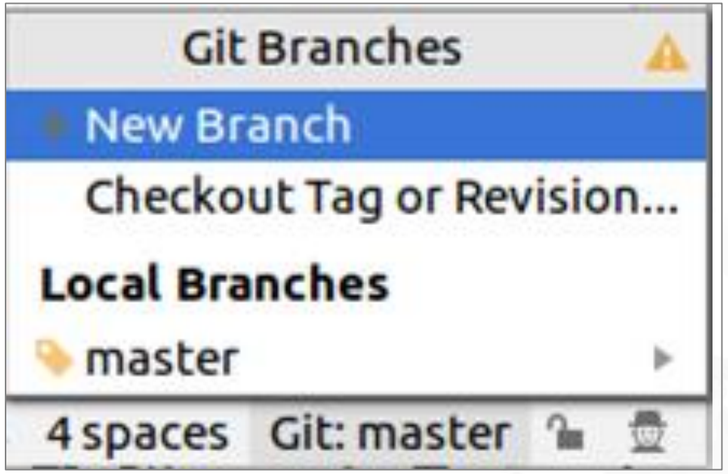
1. Click the Git widget in the Status bar and select **Branches.**
2. Toggle the **Show Only Favorites** and the **Show More** commands at the bottom of the **Git Branches** popup.

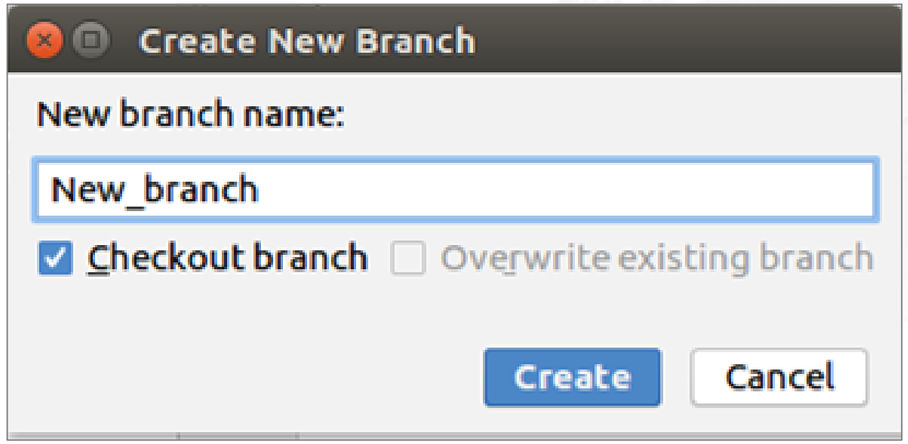


# Create a New Branch

Steps to create a new branch:

1. In the **Git Branches popup**, choose **New Branch.**
2. Specify the branch name, and make sure the **Checkout branch** option is selected if you want to switch to that branch.



# Checkout a Branch as a New Local Branch

**Problem Statement:** To work in a branch created by your co-worker, you need to check it out to create a local copy of that branch. Perform an update or a fetch operation before checking a branch out to make sure you have the full list of remote branches.

**Steps to Perform:**

- In the **Branches** popup, select a branch that you want to check out locally from **Remote Branches** or **Common Remote Branches** .
- Choose **New Branch from Selected** from the list of available operations.
- Enter the name of the new branch and click **Checkout.**

ASSISTED PRACTICE

# Merge Branches

**Problem Statement:** You need to merge your issue into the master branch as the work on a new branch is complete and ready to be merged into the master branch.

**Steps to Perform:**

1. In the **Git Branches** popup, select the target branch that you want to integrate the changes to and choose **Checkout** from the popup menu to switch to that branch.
2. Click the **Branches** popup at the bottom of the IntelliJ IDEA window, select the branch that you want to merge into the target branch, and choose **Merge into Current** from the submenu.

ASSISTED PRACTICE

# Undo Changes

Steps to revert uncommitted changes:

1. Open the **Version Control** tool window and switch to the **Local Changes** tab.
2. Select one or more files that you want to revert and select **Revert** from the context menu.

Steps to revert a pushed commit:

1. Locate the commit you want to revert in the **Log** view, right-click it and select **Revert** from the context menu.
2. Click **Commit.**

# Key Takeaways

- EGit is an Eclipse plugin which allows you to use the distributed version control system “Git” directly within the Eclipse IDE.
- Git pull updates remote tracking branches, merges changes into the working branch of the local repository, and reports conflicts.
- Conflicts must be added to the Git Index before the conflicted files can be committed.

# Knowledge Check

| Knowledge Check 1 | When reviewing a branch prior to promoting, what is the preferred method to assure they are all looking at the same changes? |
|---|---|

a. Pull the branch and look at the changes

b. Share the desktop of the user that made the changes

c. Fetch to update EGit Remote Tracking and use **Synchronize With each Other** to see the changes in the branch to origin/master

d. Push the branch

| Knowledge Check 1 | When reviewing a branch prior to promoting, what is the preferred method to assure they are all looking at the same changes? |
|---|---|

a. Pull the branch and look at the changes

b. Share the desktop of the user that made the changes

c. Fetch to update EGit Remote Tracking and use **Synchronize With each Other** to see the changes in the branch to origin/master

d. Push the branch

The correct answer is **c**

Fetch to update EGit Remote Tracking and use Synchronize With each Other to see the changes in the branch to origin/master.

| Knowledge Check 2 | What tool is used to resolve model merge conflicts? |
| --- | --- |

a. Merge tool

b. Synchronize view

c. Git Staging view

d. Repository view

| Knowledge Check 2 | What tool is used to resolve model merge conflicts? |
|---|---|

a. Merge tool

b. Synchronize view

c. Git Staging view

d. Repository view

The correct answer is **a**

Merge tool is used to resolve model merge conflicts.

| Knowledge Check 3 | After resolving conflicts what action must be taken before the conflicted files can be committed? |
|---|---|

a. Conflicts must be committed to the local repository

b. Conflicts must be added to the Git Index

c. Conflicts must be pushed upstream

d. Conflicts must be committed to the Eclipse workspace

| Knowledge Check 3 | After resolving conflicts what action must be taken before the conflicted files can be committed? |
|---|---|

a. Conflicts must be committed to the local repository

b. Conflicts must be added to the Git Index

c. Conflicts must be pushed upstream

d. Conflicts must be committed to the Eclipse workspace

The correct answer is **b**

**Conflicts must be added to the Git Index before the conflicted files can be committed.**

| Knowledge Check 4 | What does Git pull do? |
|---|---|

a. Synchronizes local changes with upstream changes

b. Shares local changes with others

c. Updates your remote-tracking branches under refs/remotes//

d. Updates remote tracking branches, merges changes into the working branch of the local repository, and reports conflicts

| Knowledge Check 4 | What does Git pull do? |
|---|---|

a. Synchronizes local changes with upstream changes

b. Shares local changes with others

c. Updates your remote-tracking branches under refs/remotes//

d. Updates remote tracking branches, merges changes into the working branch of the local repository, and reports conflicts

The correct answer is **d**

**Git pull updates remote tracking branches, merges changes into the working branch of the local repository, and reports conflicts.**

| Knowledge Check 5 | When should you commit changes from a branch to a product's master branch? |
|---|---|

a. Whenever an error is found, and a fix is made

b. After reviewing and testing

c. Never, all work is done in task-based branches

d. Before reviewing and testing

| Knowledge Check 5 | When should you commit changes from a branch to a product's master branch? |
|---|---|

a. Whenever an error is found, and a fix is made

b. After reviewing and testing

c. Never, all work is done in task-based branches

d. Before reviewing and testing

The correct answer is **b**

Changes must be committed after reviewing and testing the changes in the branch.

# Push an IntelliJ Project into GitHub

**Problem statement:** You have created a Java project in your local directory that needs to be shared with the test engineering team for verification. Push the project into a newly created GitHub repository for sharing the code and tracking changes.

You must use the following:

- Git: To create and work with a local Git repository
- IntelliJ: To create a Java project
- GitHub: To create a remote repository

LESSON-END PROJECT