Tutorial 11 – Eclipse GitHub Workflow

Contents

[1 Introduction 1](#_Toc112502951)

[2 Person 1 – Establish Initial Codebase & Local Version Control 2](#_Toc112502952)

[3 Person 1 – Commit Locally 6](#_Toc112502953)

[4 Person 1 – Push to GitHub *master* Branch 7](#_Toc112502954)

[5 Person 1 – Establish Personal Branch 10](#_Toc112502955)

[6 Person 1 – Write Code & Push to Personal Branch 11](#_Toc112502956)

[7 Person 1 – Pull Request – Merge Personal Branch to *master* 14](#_Toc112502957)

[8 Person 2 – Create Personal Branch & Clone Remote Repo 15](#_Toc112502958)

[9 Person 2 – Write Code 19](#_Toc112502959)

[10 Person 2 – Push to Personal Branch 20](#_Toc112502960)

[11 Person 2 – Delete a File 22](#_Toc112502961)

[12 Person 2 – Pull Request – Merge Personal Branch to *master* 23](#_Toc112502962)

[13 Submission 23](#_Toc112502963)

# Introduction

**(Read, no action required)** This tutorial serves as a reference for how your team can use GitHub.

This tutorial will introduce you to working with Eclipse and GitHub to do Java development. The most recent versions of Eclipse come with a plugin, EGit which integrates Git into Eclipse.

One thing I know these tutorials do not address is how to roll back (revert) changes in Eclipse.

For this tutorial we refer to Person 1 and Person 2 as a simulation of a team. As you do this tutorial, you will play the role of both these persons. Person 1 is one person on your team who will establish the initial code repository on GitHub. This will be done once. Person 2 is any other people on your team.

This is a workflow you can use for your team project.

1. Person 1 creates a local code base, places it under source control locally (git), then makes the initial push to the *master* branch of your repo on GitHub.
2. Person 1 then makes a branch on GitHub, pull the branch to their local machine, codes, commits locally, pushes to their branch on GitHub, and then eventually opens a pull request to merge with the *master* branch.
3. Person 2 creates a branch on GitHub, the creates local workspace, clones their branch to their local workspace, codes, commits locally, pushes to their branch on GitHub, and then eventually opens a pull request to merge with the *master* branch.

This tutorial starts completely from scratch. You will create new local and remote repositories.

This tutorial is based on: <https://eclipsesource.com/blogs/tutorials/egit-tutorial/>

Finally, as we move towards the project, you (and your team) MUST have a workflow that employs version control. As the labs showed, you can do it a number of ways:

1. **Use Eclipse (or IntelliJ) connected directly with GitHub. This tutorial shows this approach.**
2. Use the command line to clone, pull, commit, push, etc while you work in your IDE of choice (Eclipse, IntelliJ).
3. Use GitHub desktop to clone, pull, etc while you work in your IDE of choice (Eclipse, IntelliJ). This is addressed in a later tutorial.
4. Last resort (and not the best choice) is to use no local version control. Simply download code, work in your favorite IDE, then drag/drop into GitHub. If you use this approach, you need to make sure and commit often (e.g. drag/drop).

You need to confirm that all members of your team have a workflow.

# Person 1 – Establish Initial Codebase & Local Version Control

1. **Establish local code base.** Do the following:
2. Create a folder named *cs4321\_sample\_project* on your local computer.
3. Open Eclipse and set the workspace to this folder.
4. Create a Java project named: *acmesystem*

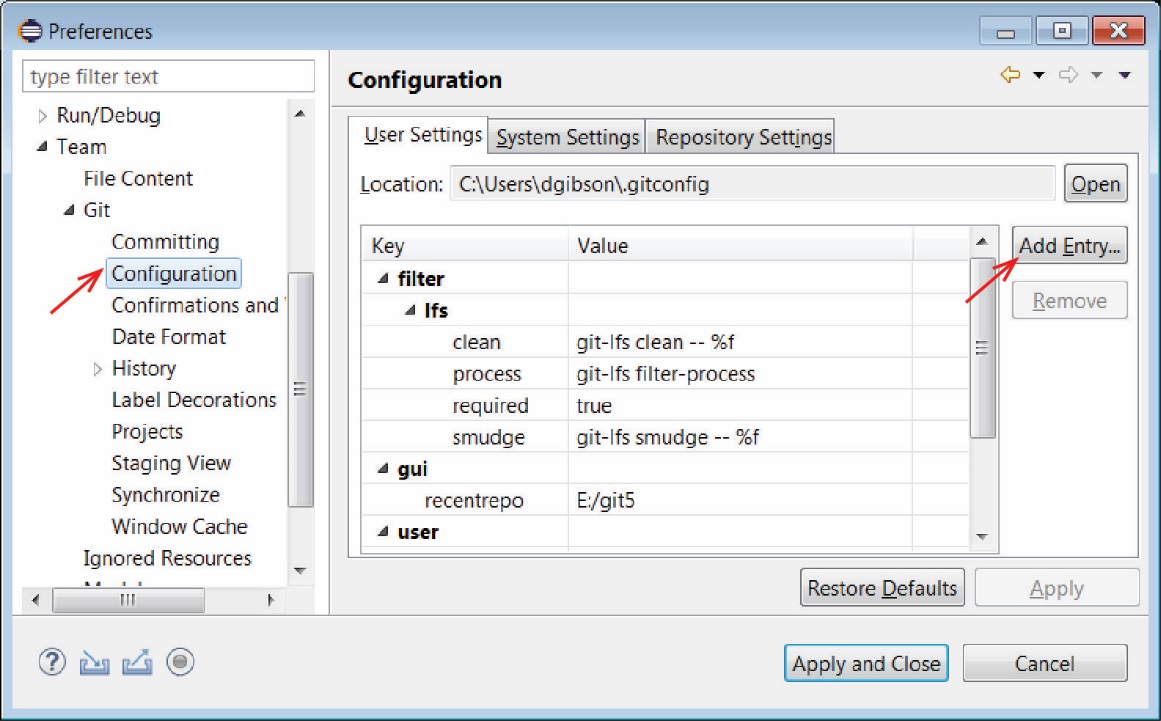
**Warning: The name cannot have any spaces or special characters and should be lower-case**

1. Create a package named: *ver1*
2. Create a class named: *HelloWorld* (select checkbox to create a *main*)
3. Add a print statement to *main*.

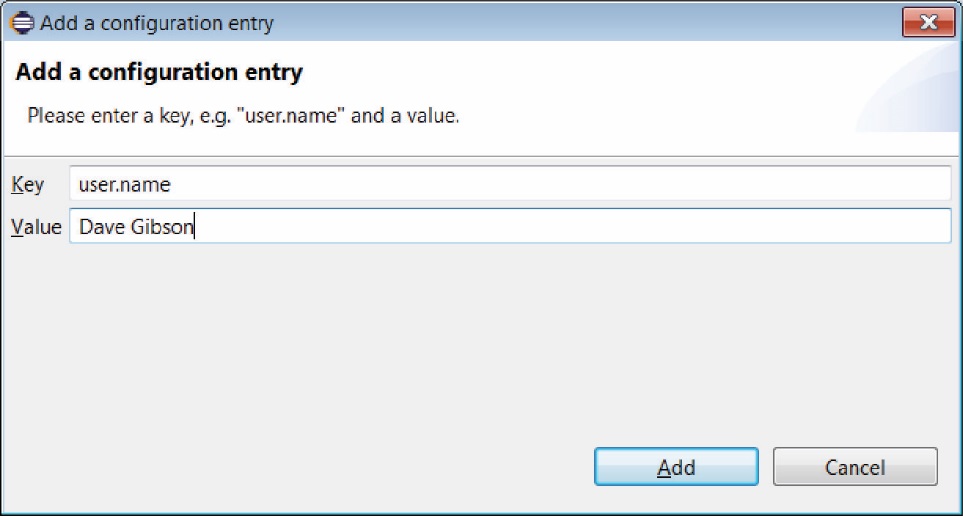
System.***out***.println("Hello from Eclipse");

1. **Initialize Git Credentials.** Do the following:
2. Choose: Window, Preferences, Version Control (Team), Git, Configuration.

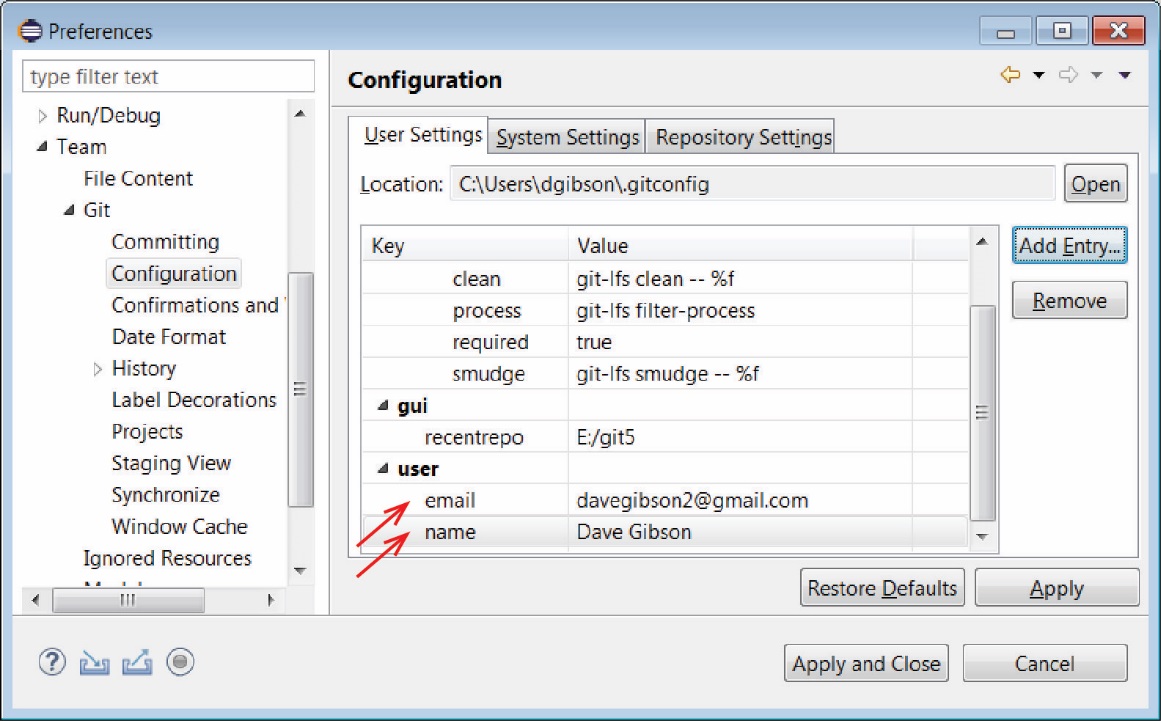
If you see your email and name under *user* you do NOT need to do this step. **Otherwise, choose: Add Entry**



1. Enter *user.name* as Key and your name as Value and Add

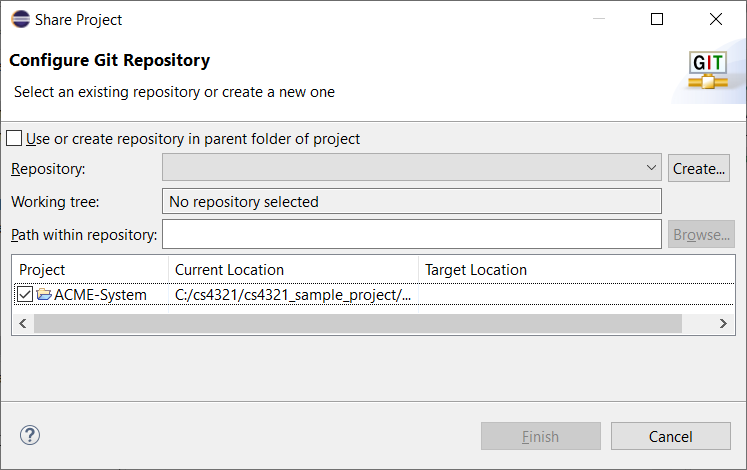


1. Choose: Add Entry and enter *user.email* as Key and your email address as Value. You will see the two entries as shown below. (In many of the figures below, you’ll see a different email and name: aug1west@gmail and August West, as I am now using those credentials to make these tutorials. Previous versions used the credentials below.)

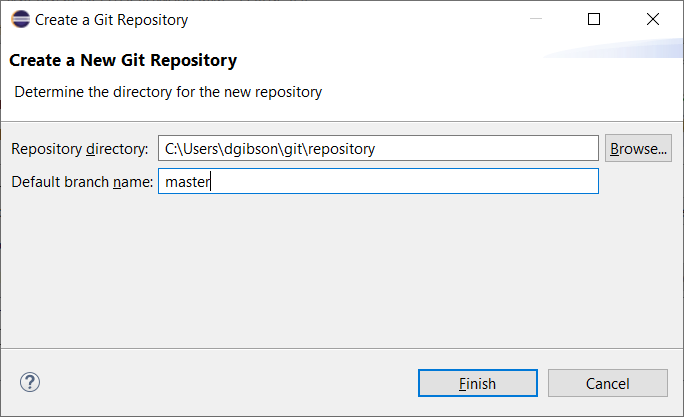


1. Choose: Apply and Close
2. **Setup up Eclipse to use version control locally.** Do the following:
3. Right-click the project node and choose: Team, Share Project

**Note: In diagram below, under Project, yours will say: *acmesystem***

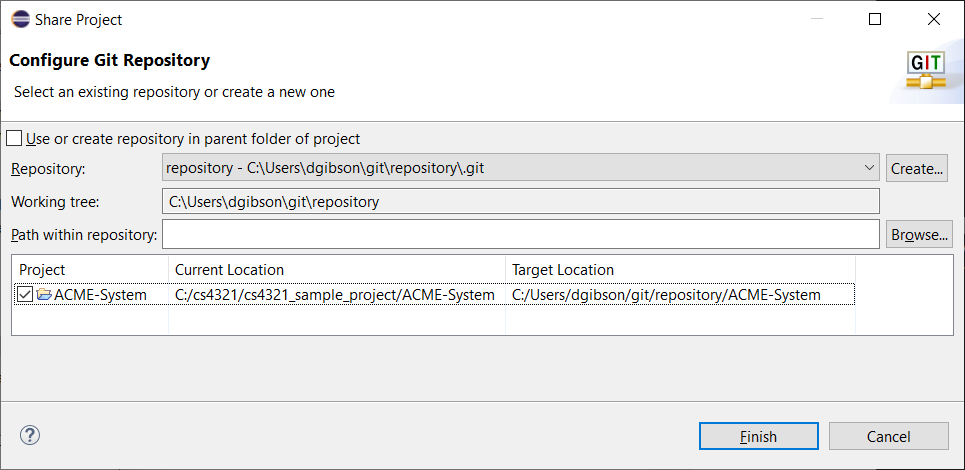


1. Choose: Create. Note that the repository will be stored in a location separate from the workspace, which is the recommended approach. We will accept the default location.



1. Choose: Finish. The result is shown below:

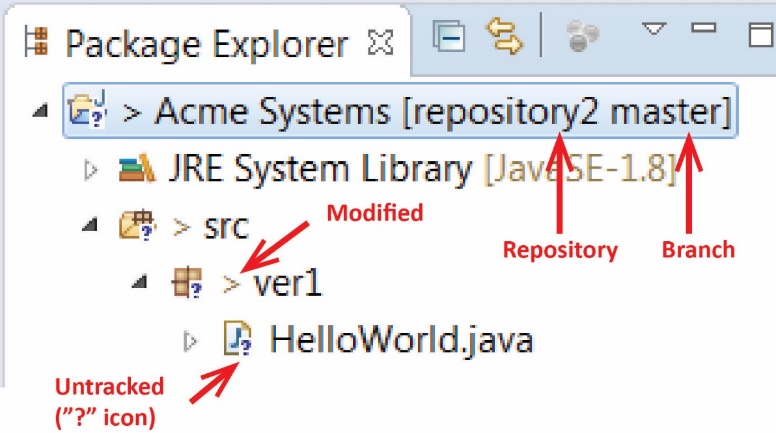
**Note: In diagram below, instead of: *ACME-System,* yours will say: *acmesystem***



1. Choose: Finish. Note:
2. Look at the Package Explorer in Eclipse as shown in the figure below. To see all the different icon annotations, see, “Additional Information” towards the bottom of this page:

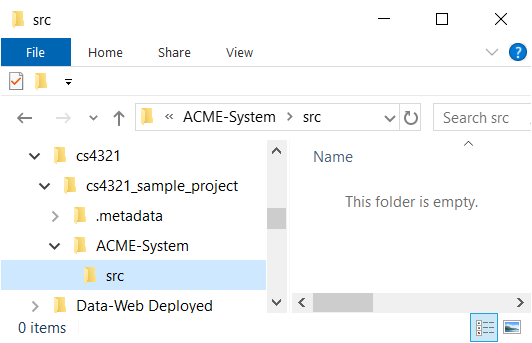
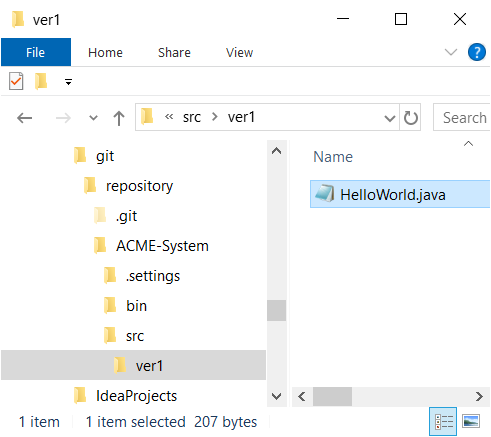
<https://eclipsesource.com/blogs/tutorials/egit-tutorial/>

**Note: In diagram below, instead of: *ACME-System,* yours will say: *acmesystem***



1. Your initial workspace appears on your hard drive as shown below on the left. Note that there are no Java files. They have been moved and/or are in the local repo. On the right is shown the local repo in: *c:\users\yourWindowsID\git.* When you restart Eclipse, you will continue to use the initial workspace, not the one in the repository. Apparently, Eclipse knows where to find the files.

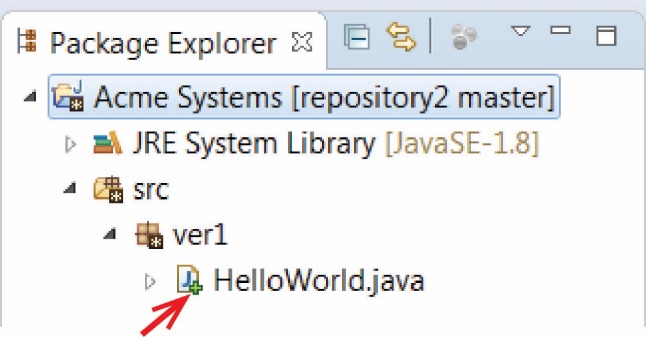
**Note: In diagram below, instead of: *ACME-System,* yours will say: *acmesystem***

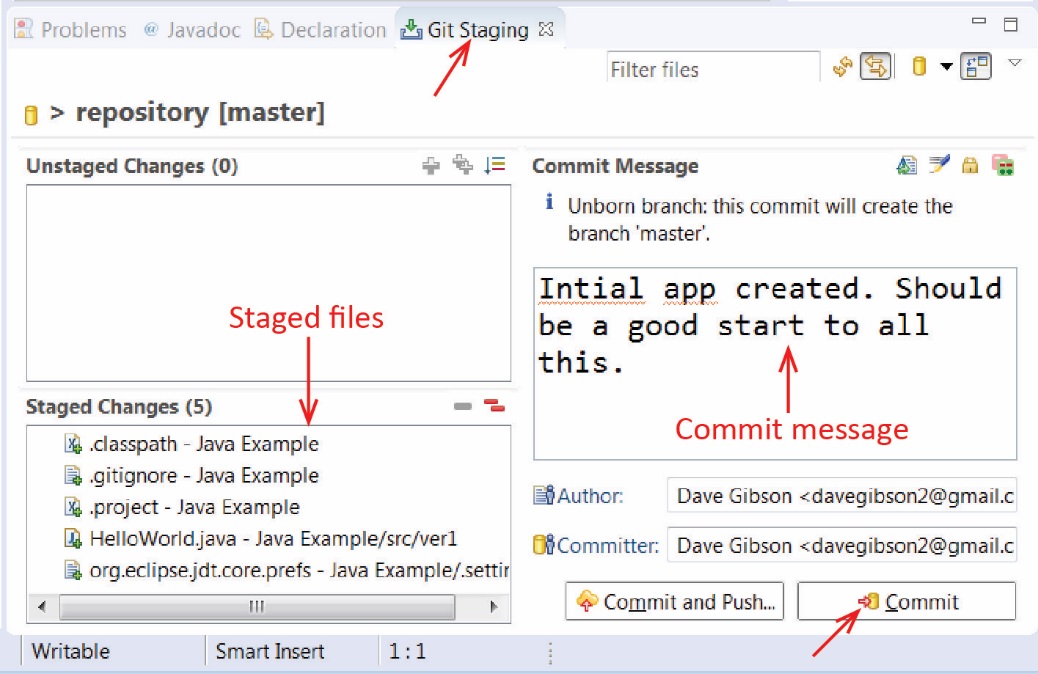
# Person 1 – Commit Locally

1. Next, we need to add the project to the index. Right-click the project node and choose: Team, Add to Index. After this operation, the question mark should change to a plus symbol.

**Note: In diagram below, instead of: *ACME-System,* yours will say: *acmesystem***

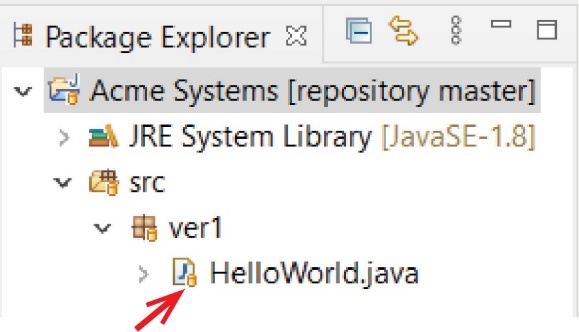


1. Next, we need to commit the files. Do the following:
2. Right-click the project node and choose: Team, Commit…
3. (Read, no action required) Note all the files in the *Staged* area. It has the files that Eclipse uses to recognize this as a project. I think this is not best practice (storing meta-data in a repo), but it is the approach we will take as I haven’t found a way to do it without these.
4. Enter a commit message (the first line should be headline-like, as it will appear in the history view) and press the Commit button (NOT Commit and Push). (The image below shows a different Author as it was from a previous version of this lab).



1. If the commit was successful, the plus symbols will have turned into a repository icon.

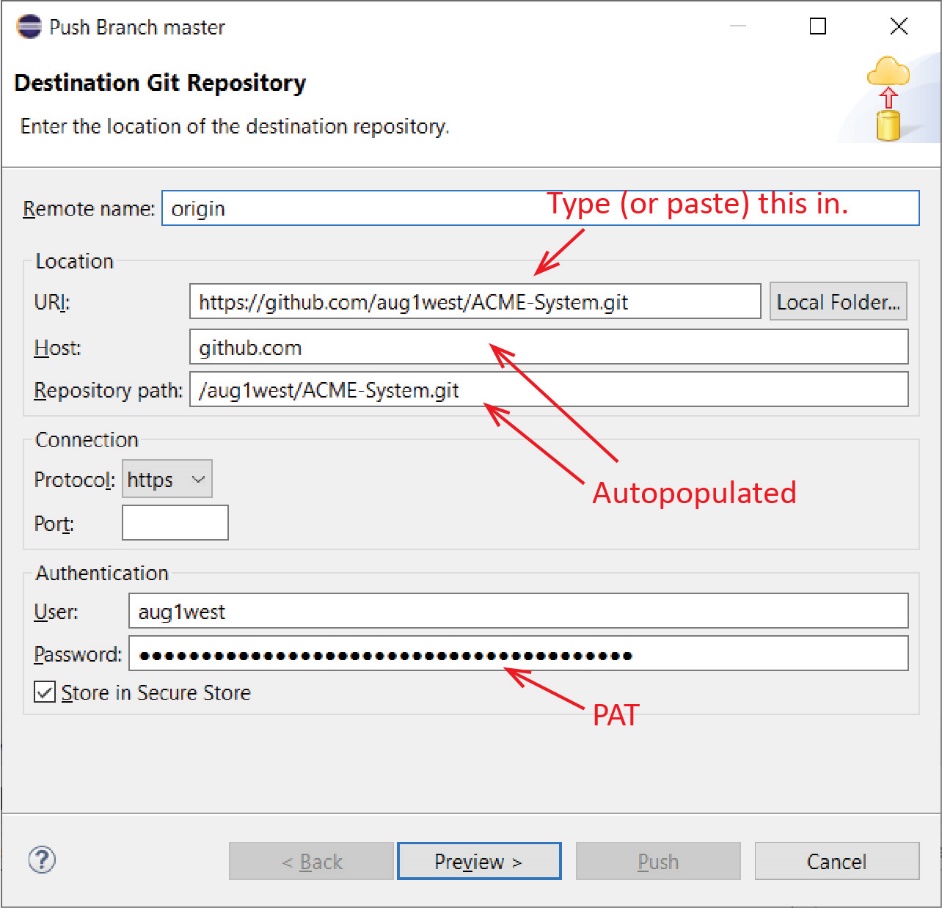
**Note: In diagram below, instead of: *ACME-System,* yours will say: *acmesystem***

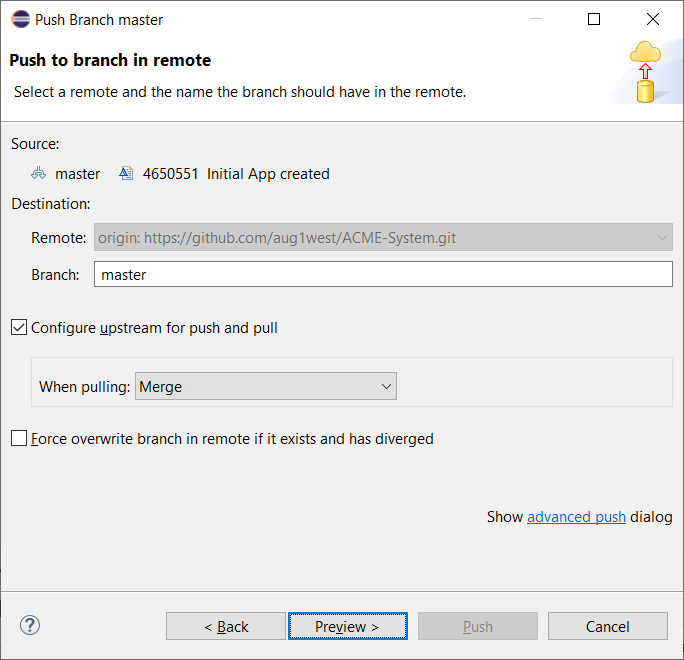


# Person 1 – Push to GitHub *master* Branch

1. Go to GitHub and create a new repository named, *Acme-System.* This is where we will push our local project.­­­

**Note: It is OK that your local system is: *acmesystem* and your remote is: *Acme-System.***

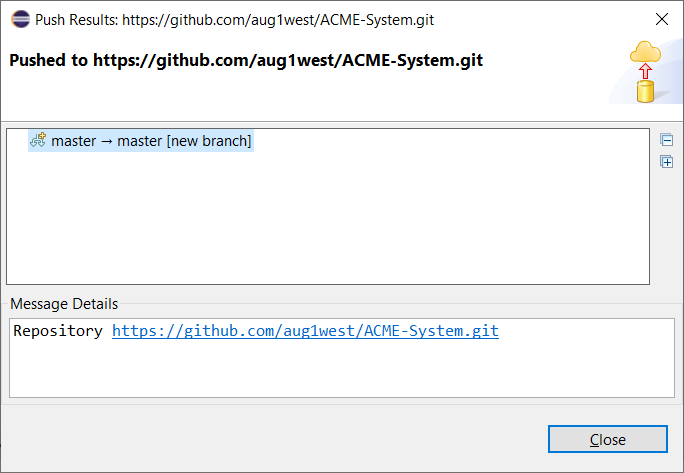
1. Do the following:
2. In Eclipse, right-click the project node and choose: Team, Push Branch ‘master’
3. Paste the URI from GitHub, other fields will be auto-populated.
4. Type your GitHub UserName and Password (PAT).
5. Press Preview, look at dialog,



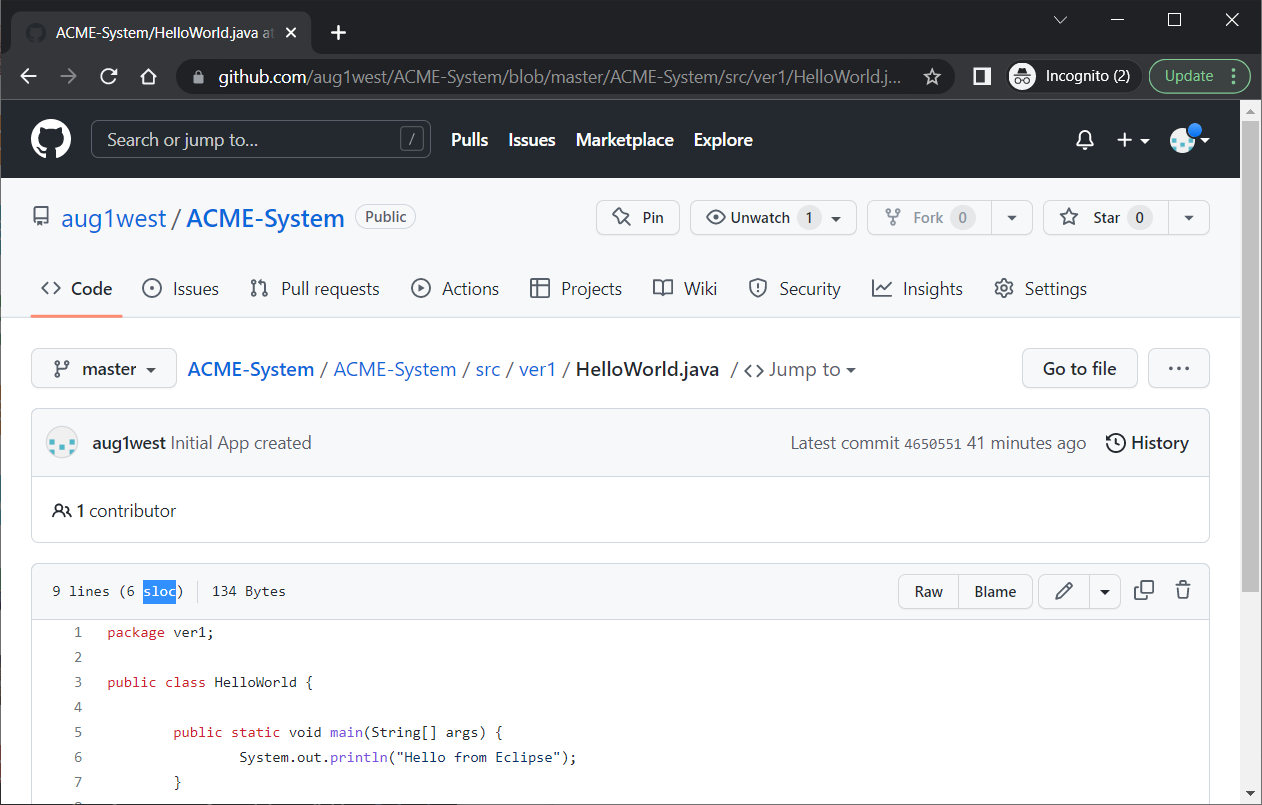
1. Press Preview again.



1. Finally, press: Push. The resulting dialog will be similar to the one shown below. When done, press: Close.



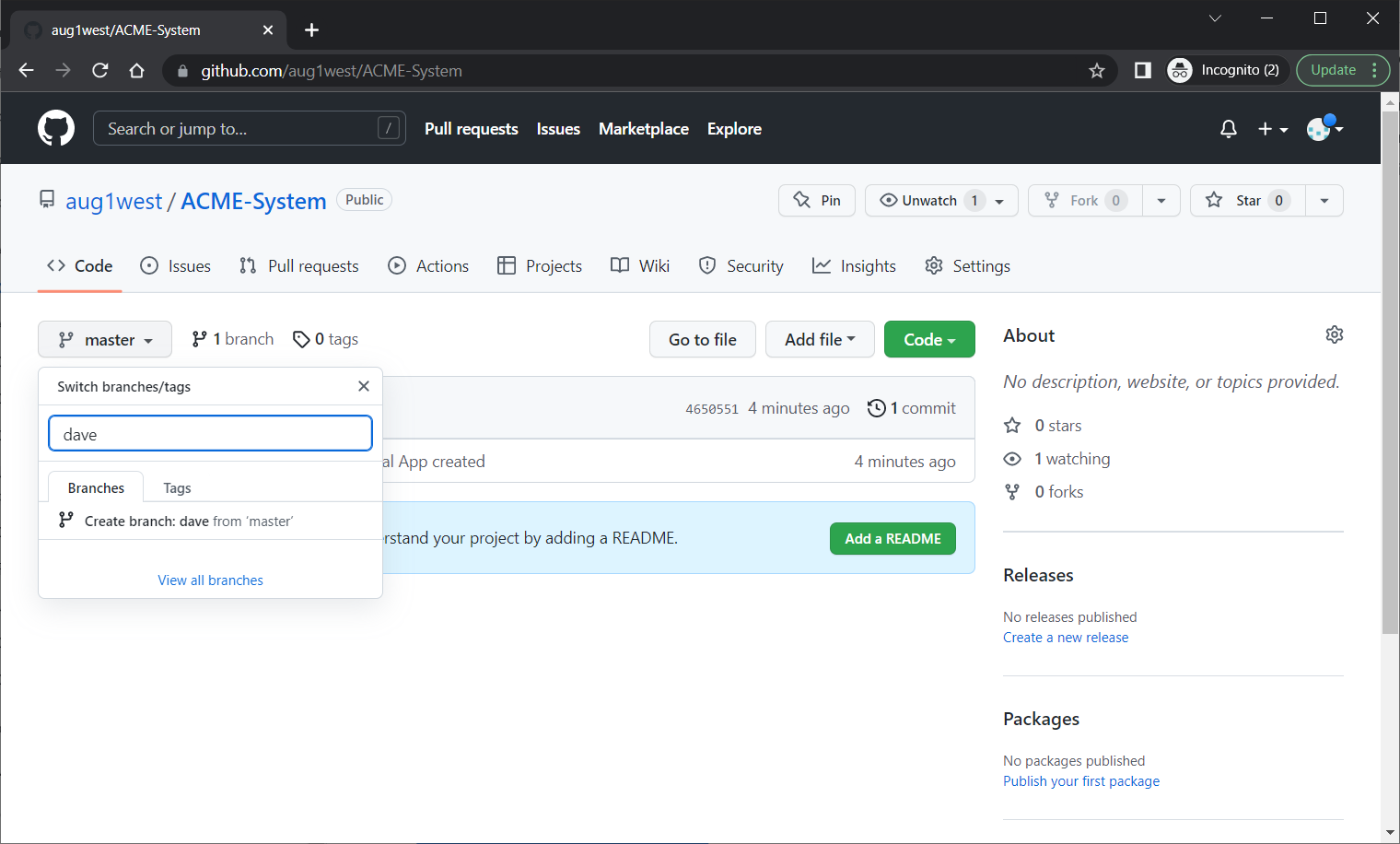
1. Do the following in GitHub: verify that your project was pushed.



At this point, you have an initial (working) code base on the master branch of your repo on GitHub. Next, we show how Person 1 can begin work on their own branch. Following this, we show how another team member (Person 2) can begin working on their own branch.

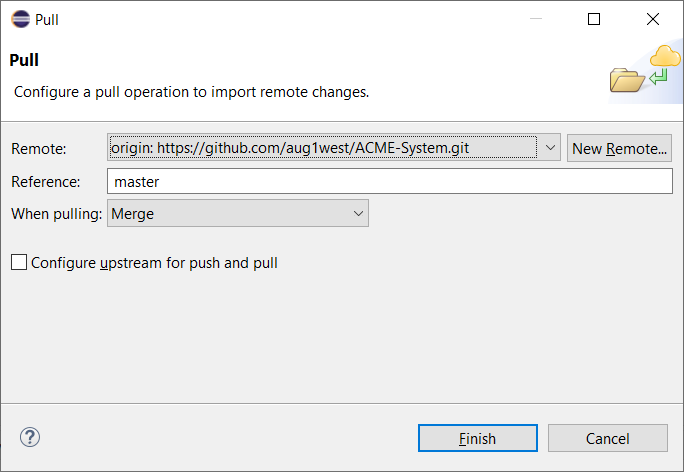
# Person 1 – Establish Personal Branch

1. Create a branch in GitHub (I named mine: *dave*) and press: “Create branch…”

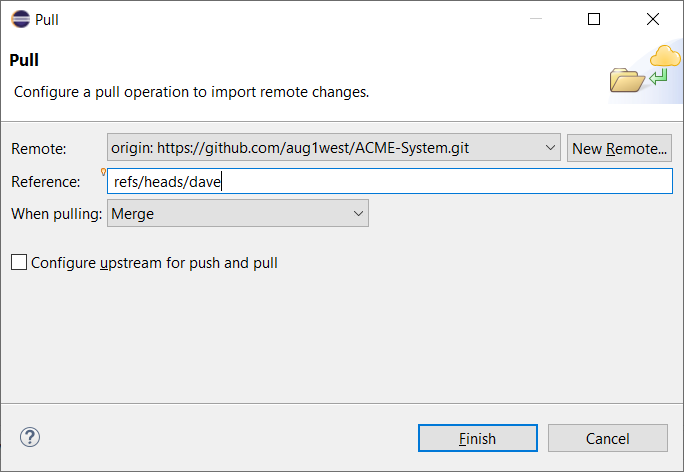


1. Right-click the project node in Eclipse, and choose: Team, Pull…

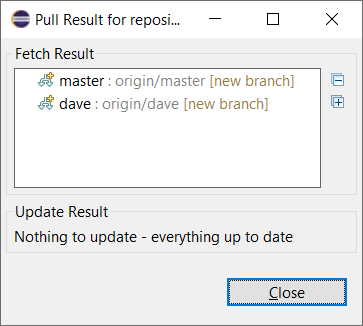
**Note: don’t choose: Team, Pull. Choose: Team, Pull…**



1. Replace the *Reference* with the name of your branch. Start typing it in and the full path will be displayed. Select that.



1. Choose: Finish. The dialog is shown below, and then choose: Close.

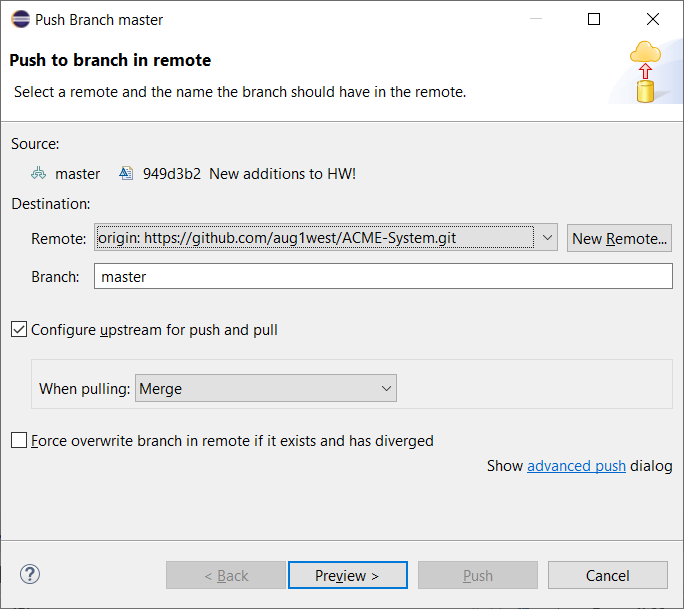


# Person 1 – Write Code & Push to Personal Branch

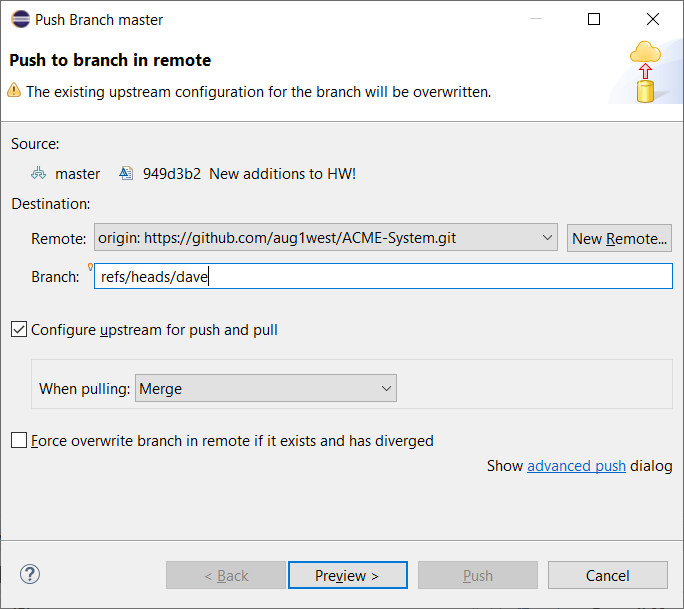
1. In Eclipse, add a line to *HelloWorld*, something like:

System.***out***.println("Dave is working locally on his branch");

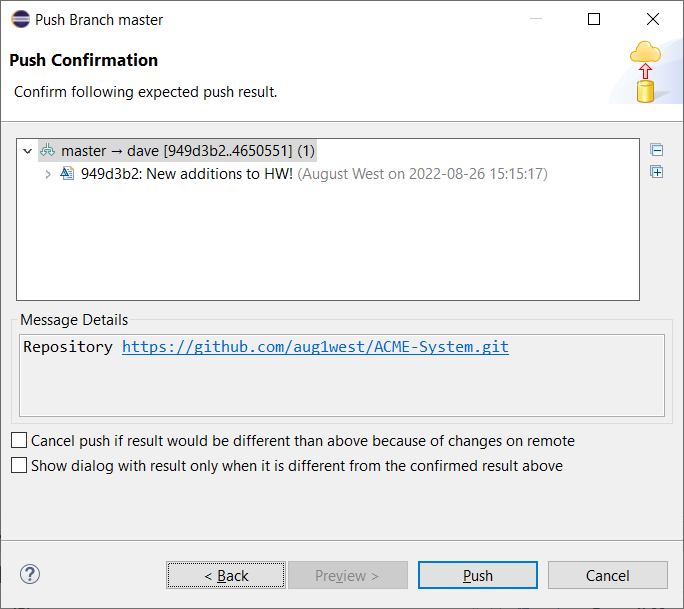
1. Commit the change. (Select *HelloWorld* in Project Explorer and choose: Team, Commit. Type a message and select: Commit.)
2. Push back to remote branch. Do the following:
3. Select the project node and choose: Team, Push Branch ‘master’. In your local repository, your branch is *master*. On the remote, it is (in my case), *dave*.



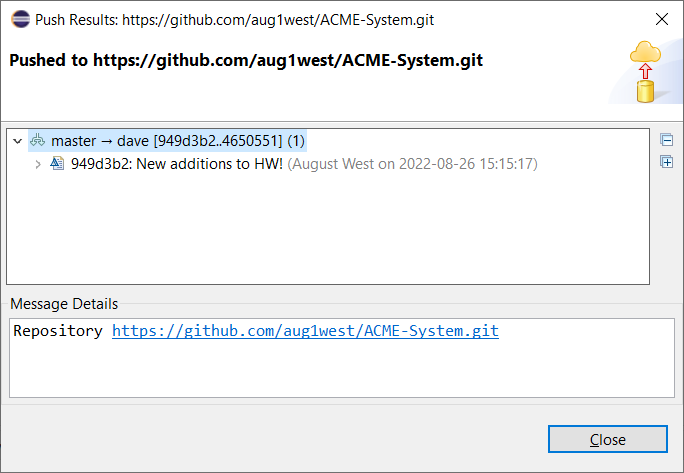
1. Beside *Branch,* type the name of your branch and a pop-up will display the full path. Select that.



1. Select: Preview



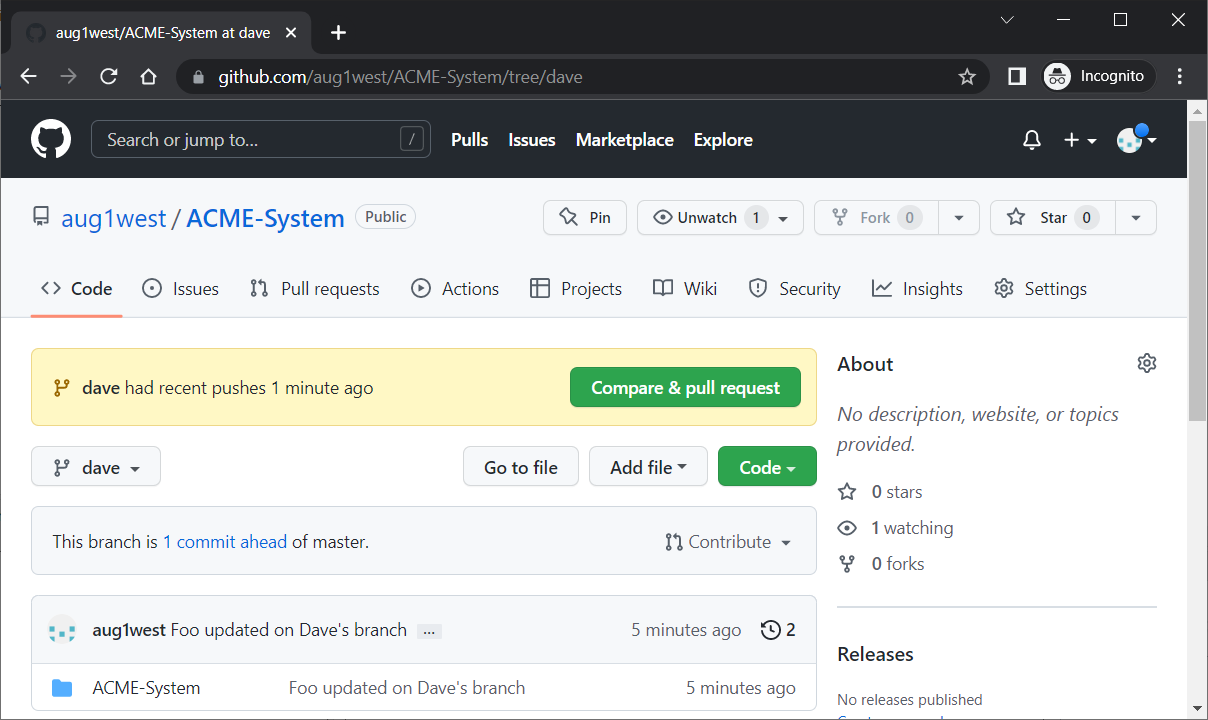
1. Select: Push. Dialog is shown below. Then choose: Close



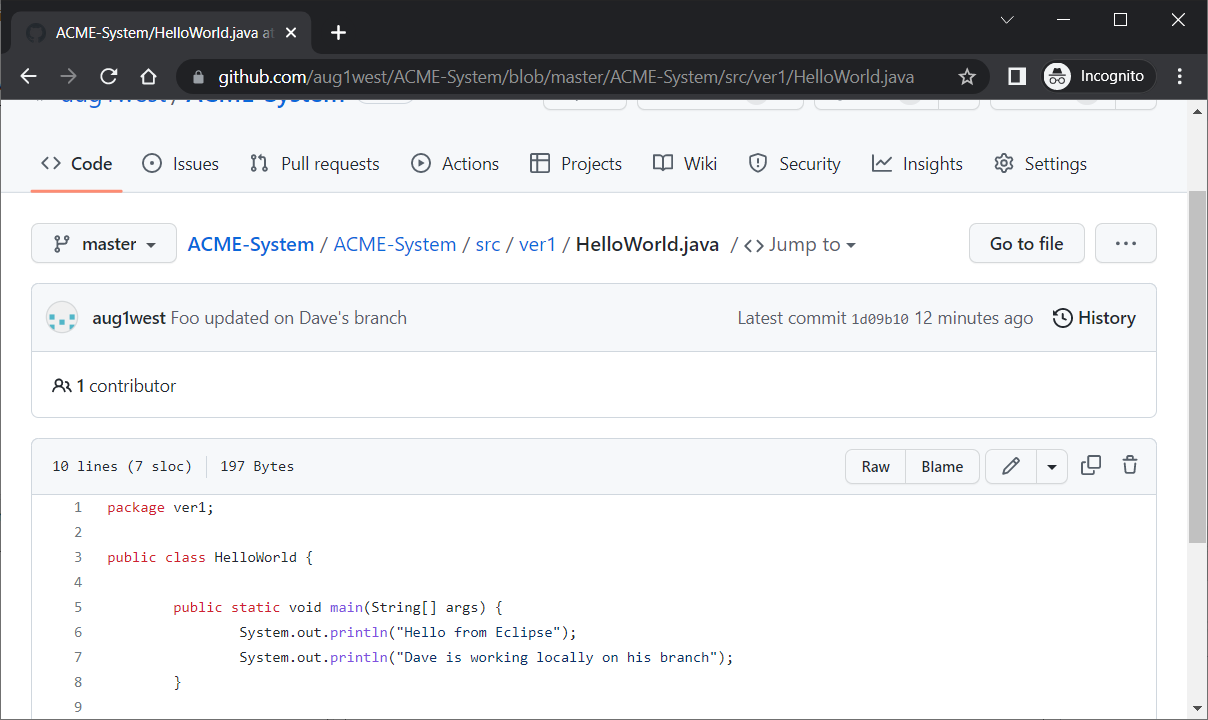
1. Verify changes on your branch in GitHub

# Person 1 – Pull Request – Merge Personal Branch to *master*

1. Do the following:
2. As described in a previous tutorial, display the homepage for your repo.



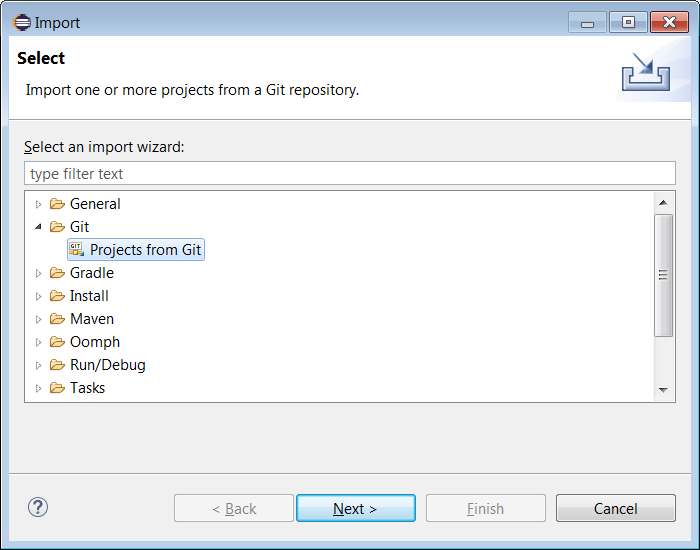
1. Choose: Compare & Pull Request
2. Write a comment about the pull request (overview of the commits)
3. Choose: Create pull request
4. (Read, no action required) Here, other team would see this “Open” Pull request and could review the code.
5. When code review is complete, choose: Merge pull request (or Rebase and merge)
6. Choose: Confirm merge
7. Make sure the *master* branch is displayed and verify that the code has been updated.

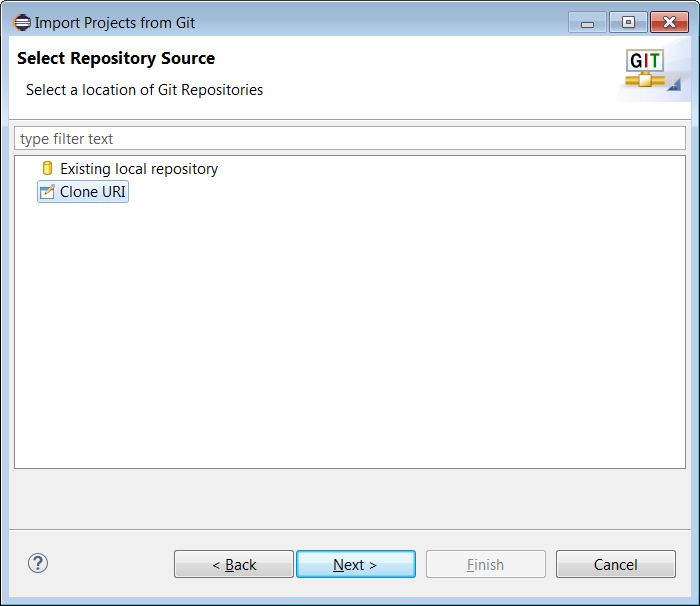


At this point, *master* is updated with working code. Now, let’s simulate Person 2 beginning their work on their own branch.

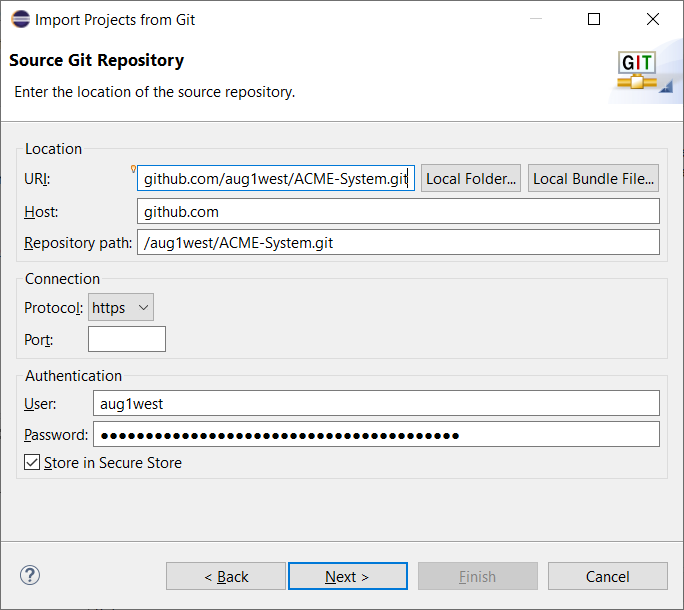
# Person 2 – Create Personal Branch & Clone Remote Repo

We are simulating what Person 2 will do to begin work. Even though you are doing this tutorial, you are playing the role of Person 2. So, you’ll create another Eclipse workspace, create a branch on GitHub, and then clone the repo to obtain your local copy.

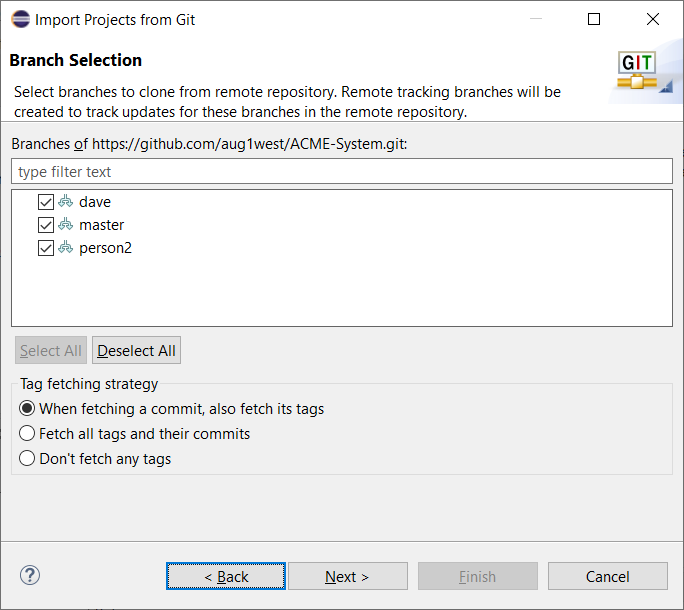
1. Create a branch on GitHub named *person2*
2. Do the following:
3. Create a folder named *cs4321\_sample\_project2* on your local computer.
4. Open Eclipse and set the workspace to this folder.
5. Next, we will *clone* the *Acme-System* repository on GitHub to our local workspace, *cs4321\_sample\_project2.* Do the following:
6. Choose: File, Import, Git, Projects from Git
7. Choose: Next, then Select: Clone URI



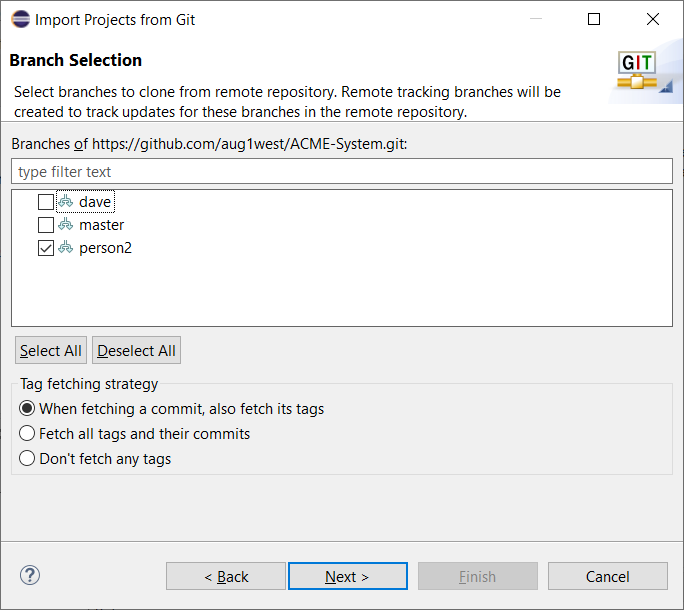
1. Choose: Next, and then supply the URI for the repository, and supply your UserName and PAT. Choose: Next



1. Choose: Next

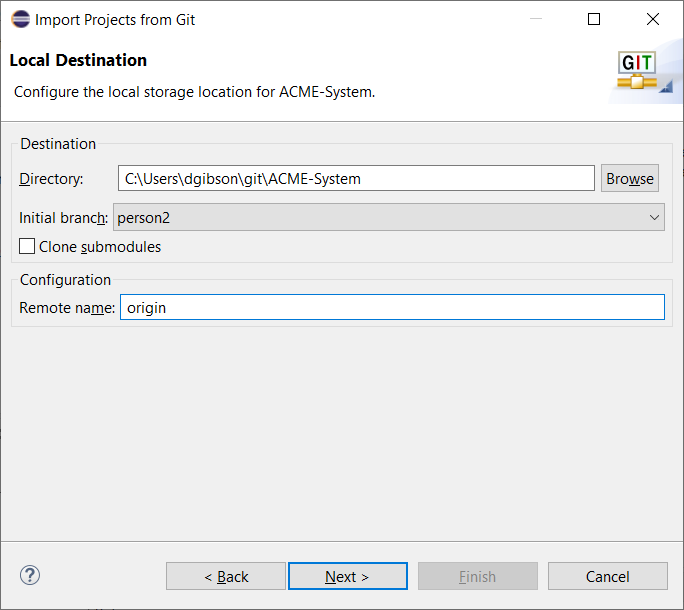


1. Uncheck all branches except *person2*

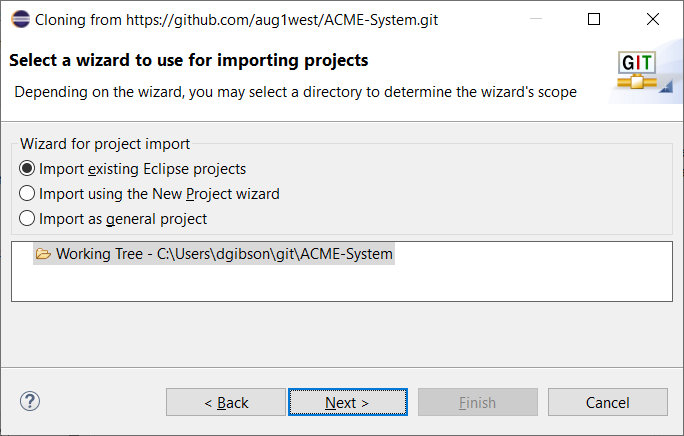


1. Choose: Next. Note the location. I think this is where your files are stored.

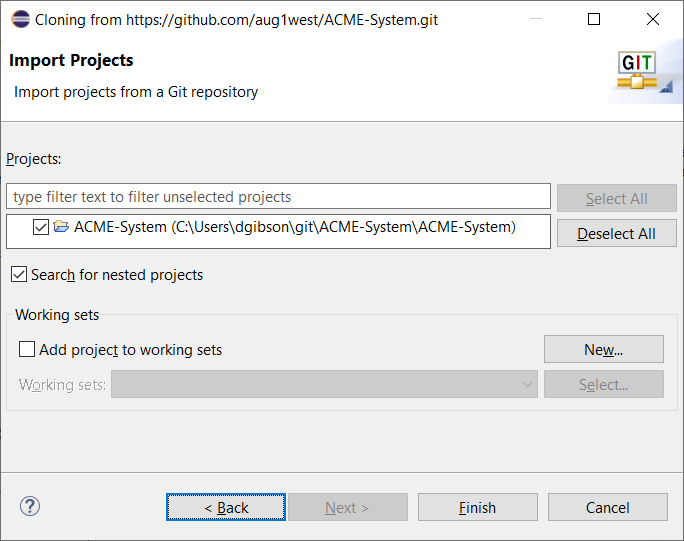
**Note: if you get an error at the top of this dialog that says: “C:\...\git\Acme-System is not an empty directory”, then just append a “2” on the end, so that the Directory says**



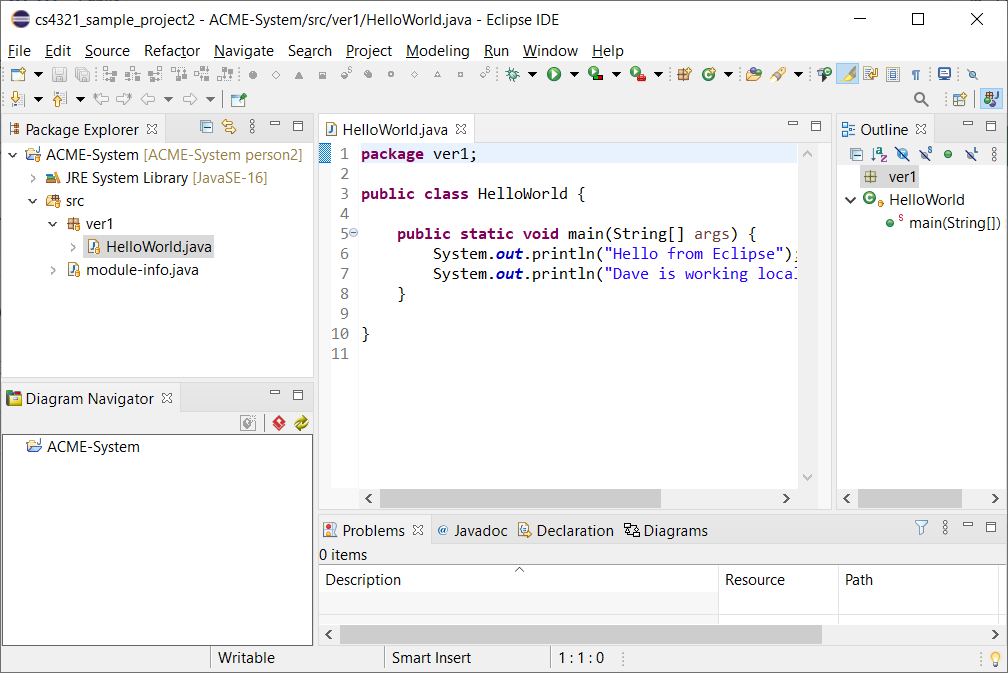
1. Choose: Next



1. Choose: Next



1. Choose: Finish.
2. The code should be displayed in the package explorer now. Verify that the contents are:

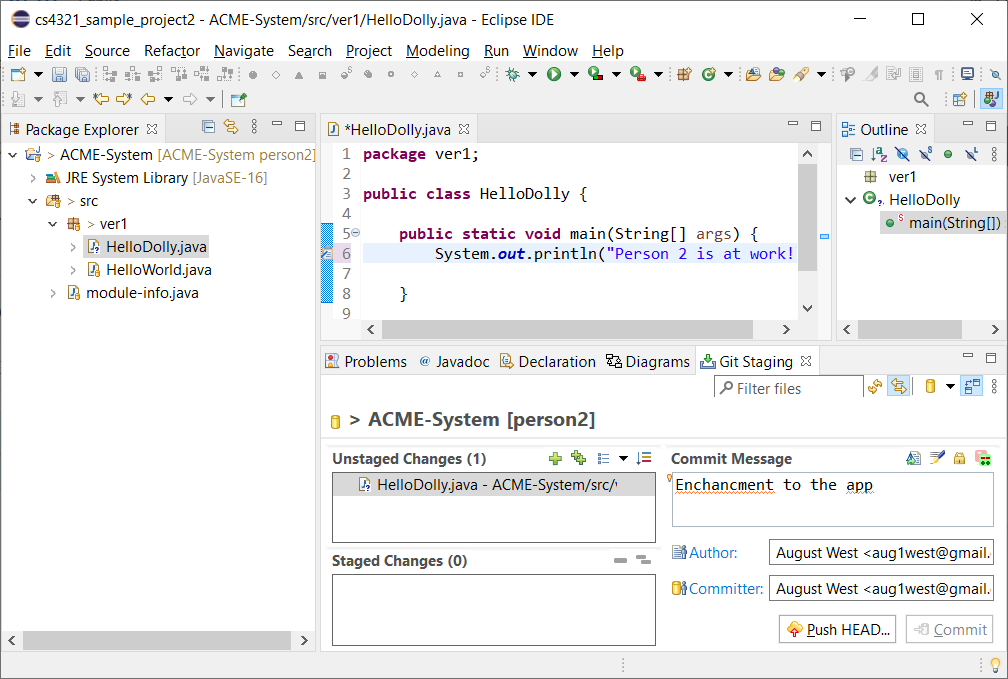


# Person 2 – Write Code

1. Do the following in Eclipse:
2. Add a class named *HelloDolly* with a *main* and add this line of code below and save.

System.***out***.println("Person 2 is at work!");

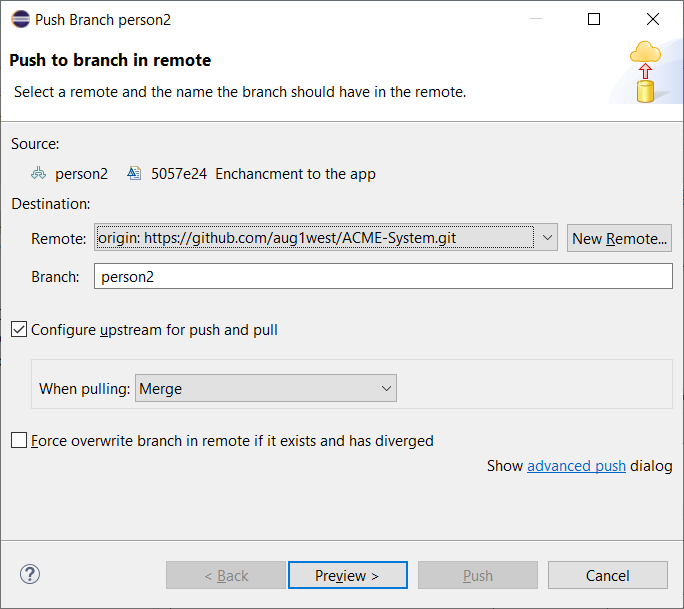
**Note: In the following steps, you will add this file to the index and then commit. If you do this, as shown below, while the file is not saved (note the \* on the file name at the top), the Commit button (lower right) will be grayed out. To fix this, save the file, and *HelloDolly.java* will be moved to the *Stage* are and the Commit button will be enablved.**



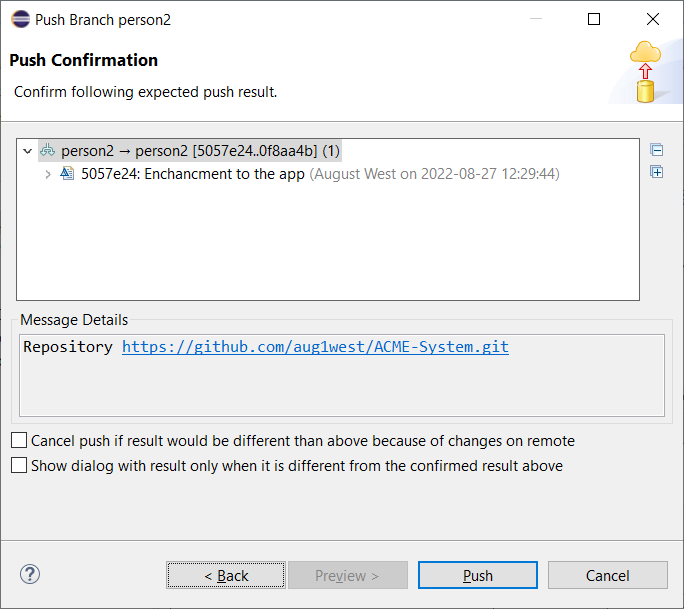
1. Add this file to the index (Team, Add to index)
2. Commit the change (Team, Commit, type message, then press: Commit)

# Person 2 – Push to Personal Branch

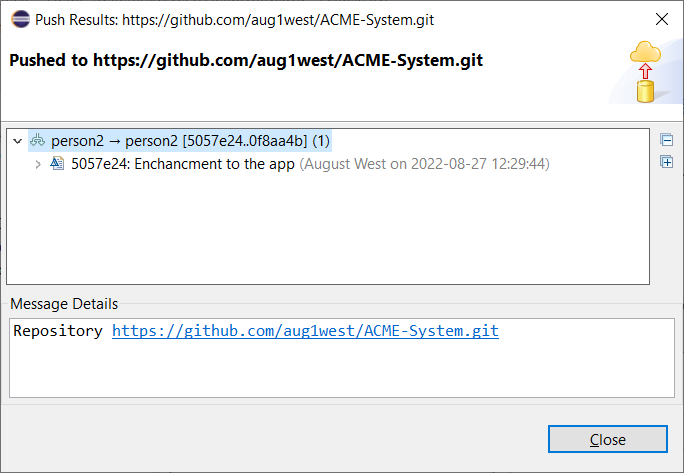
1. Right-click the project node in the Package Explorer and choose: Team, Push Branch ‘person2’



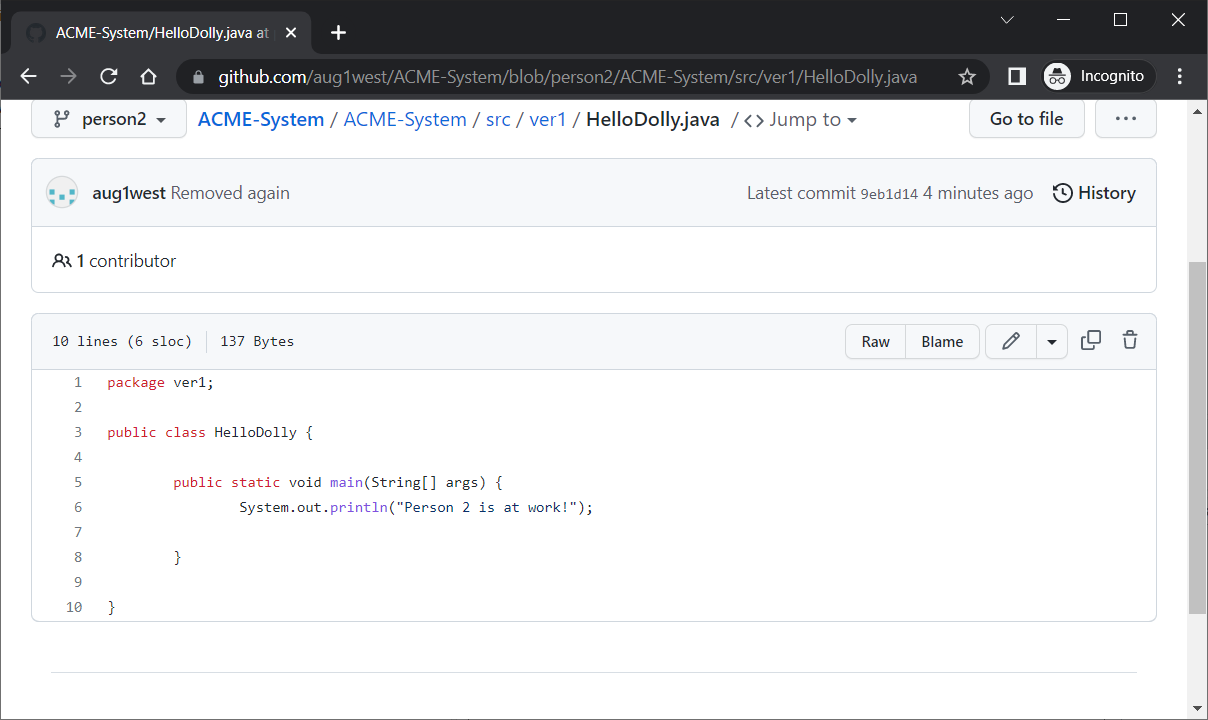
1. Choose: Preview



1. Choose: Push and then Close.



1. Verify the new class is in GitHub.

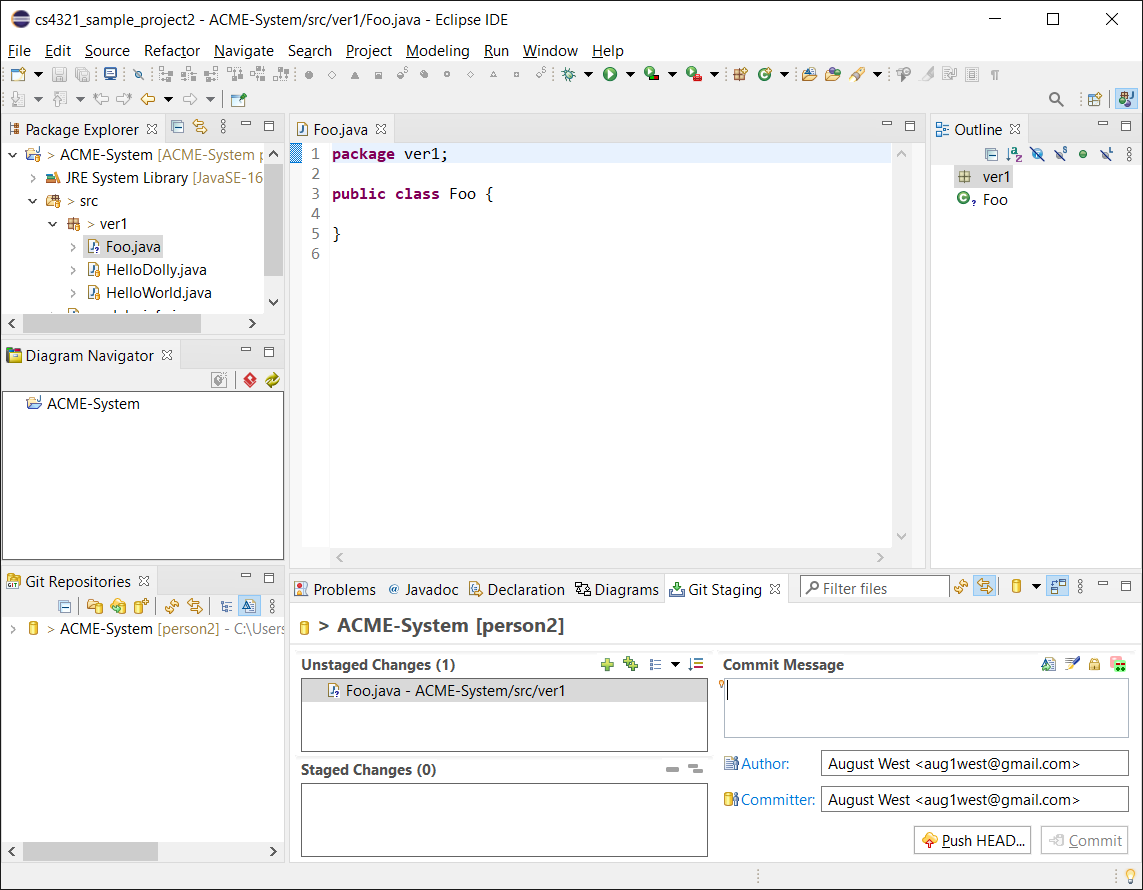


# Person 2 – Delete a File

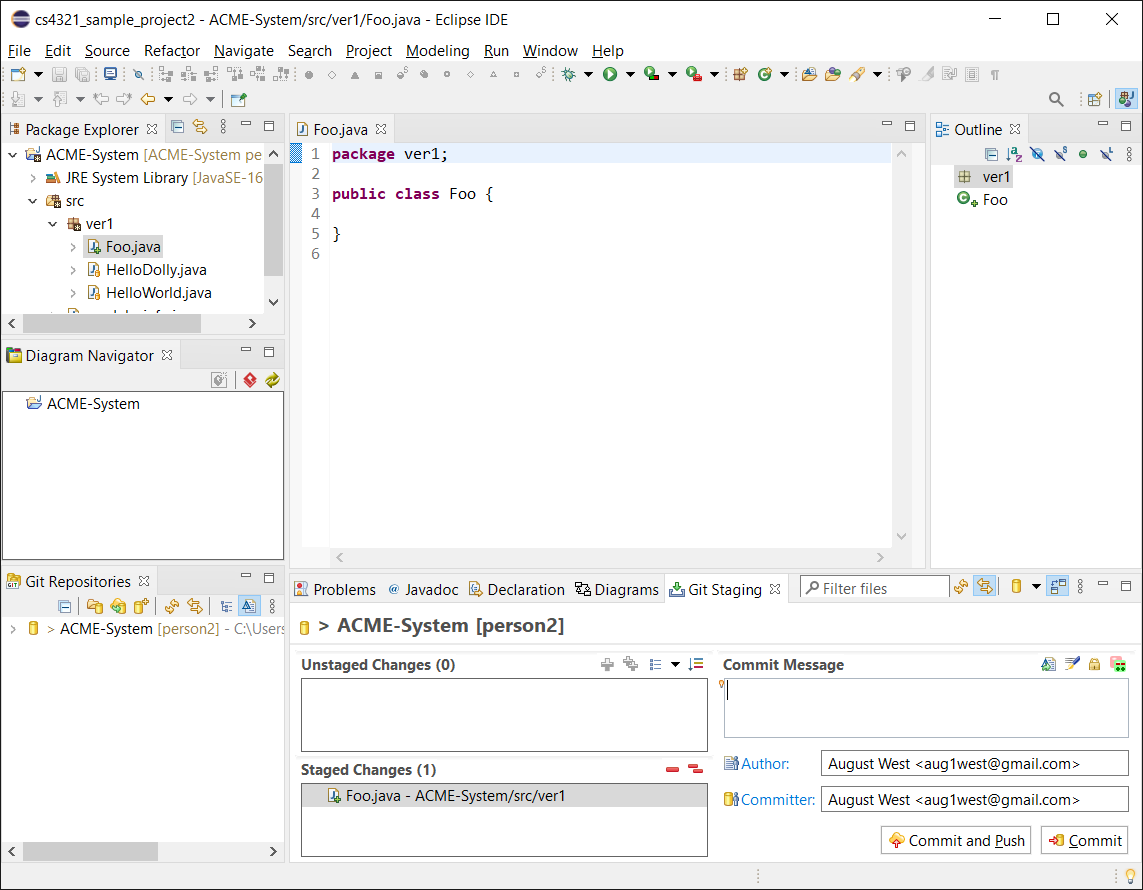
Deleting a file caused me some confusion. This is what I figured out. First, we will add a file locally, push to GitHub, verify it is there. Then, we will delete the same file locally and push to GitHub, then verify it is deleted.

1. In Eclipse, add a class named *Foo* (don’t need to add any code)
2. Display the *Git Staging* window by choosing: Window, Show View, Other, Git, Git Staging

As shown below, it appears in *Unstage Changes.* You can add it to the *Staged Changes* by either: (a) Team, Commit, or (b) press the green “+”.

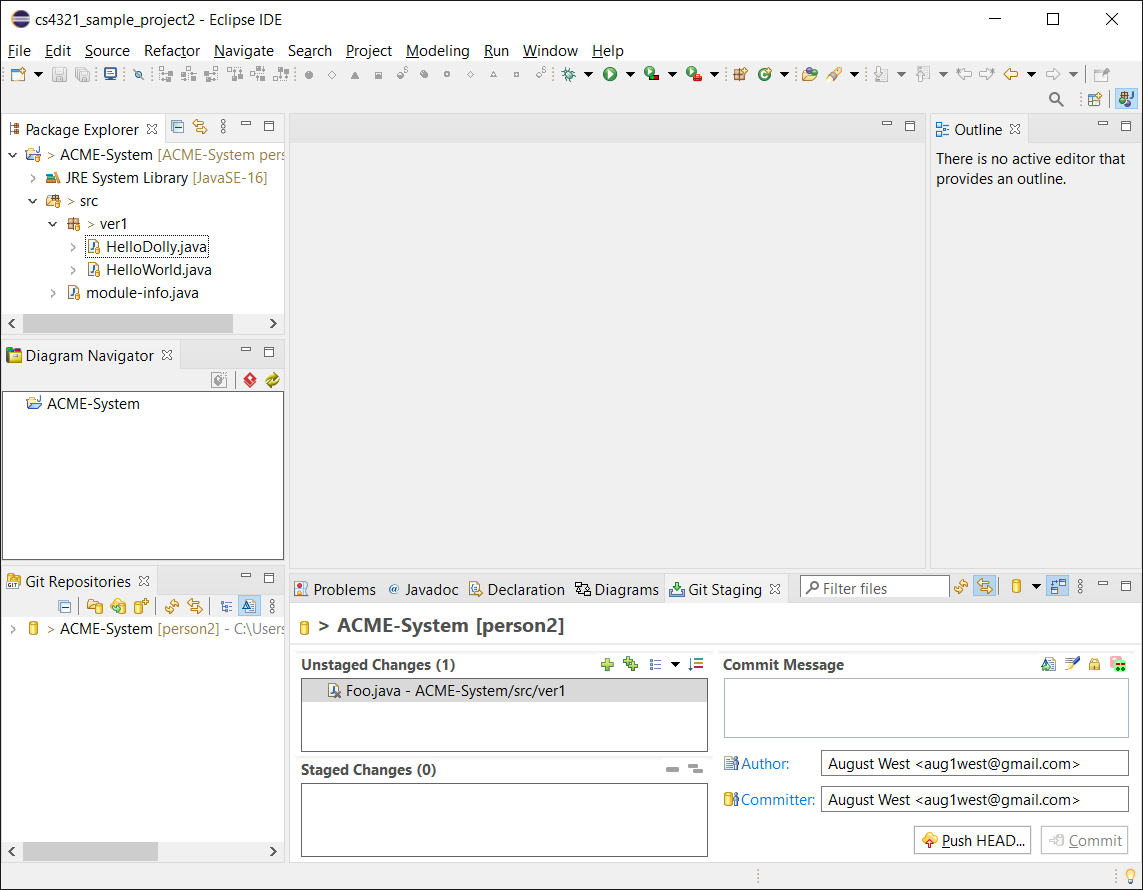


1. Commit the file with either of the methods above. Notice that the Commit button is enabled now.

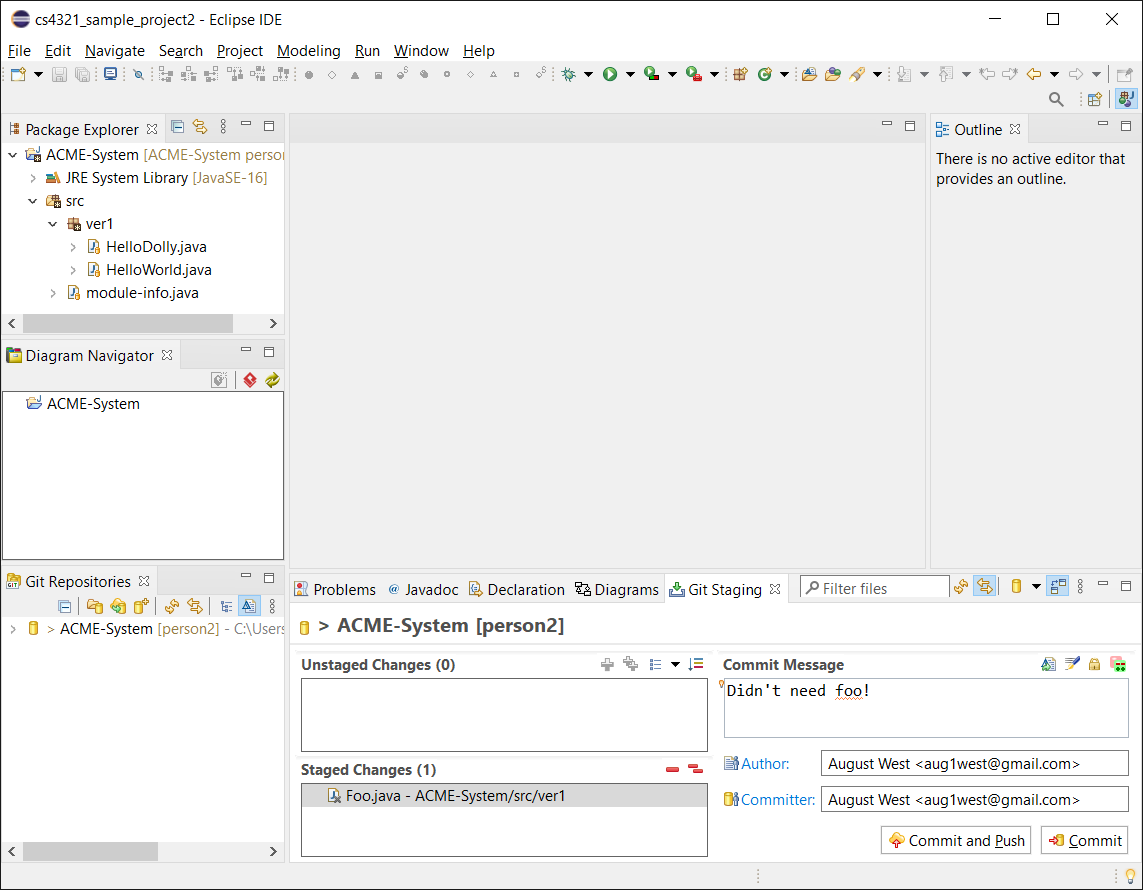


1. Type a commit message and choose: Commit
2. Push to GitHub
3. Verify that *Foo* is in GitHub
4. In Eclipse, select *Foo* in the Package Explorer and press Delete

Note that it is now in *Unstaged Changes*.



1. The only way to stage it is to choose the green “+”. Do this now, type a commit message, then choose: Commit and Push (or Commit and then Team, Push branch ‘person2’)



1. Verify that *Foo* is now removed from GitHub.

# Person 2 – Pull Request – Merge Personal Branch to *master*

1. Follow the steps in [Section 7](#_Person_1_–) to do a pull request.

# Submission

1. **Do the following:**
2. Make a screen shot similar to step 28 above.
3. Place the image in the *HW VCS* document in the appropriate place.
4. The image should easily readable without zooming in or out.