This lab covers the following topics

Contents

[Introduction: 2](#_Toc26198393)

[Local Software Install: 3](#_Toc26198394)

[Setting up an Enterprise GitHub Repository : 5](#_Toc26198395)

[Connecting to the Repository: 8](#_Toc26198396)

[Initial Code Check-in: 12](#_Toc26198397)

# Introduction:

Having a code repository is something that every developer and development team should have. This is a place where code can be saved, versioned, and shared that's independent from local development machines. There are many vendors available; like CVS, SVN, and Rational to name just a few.

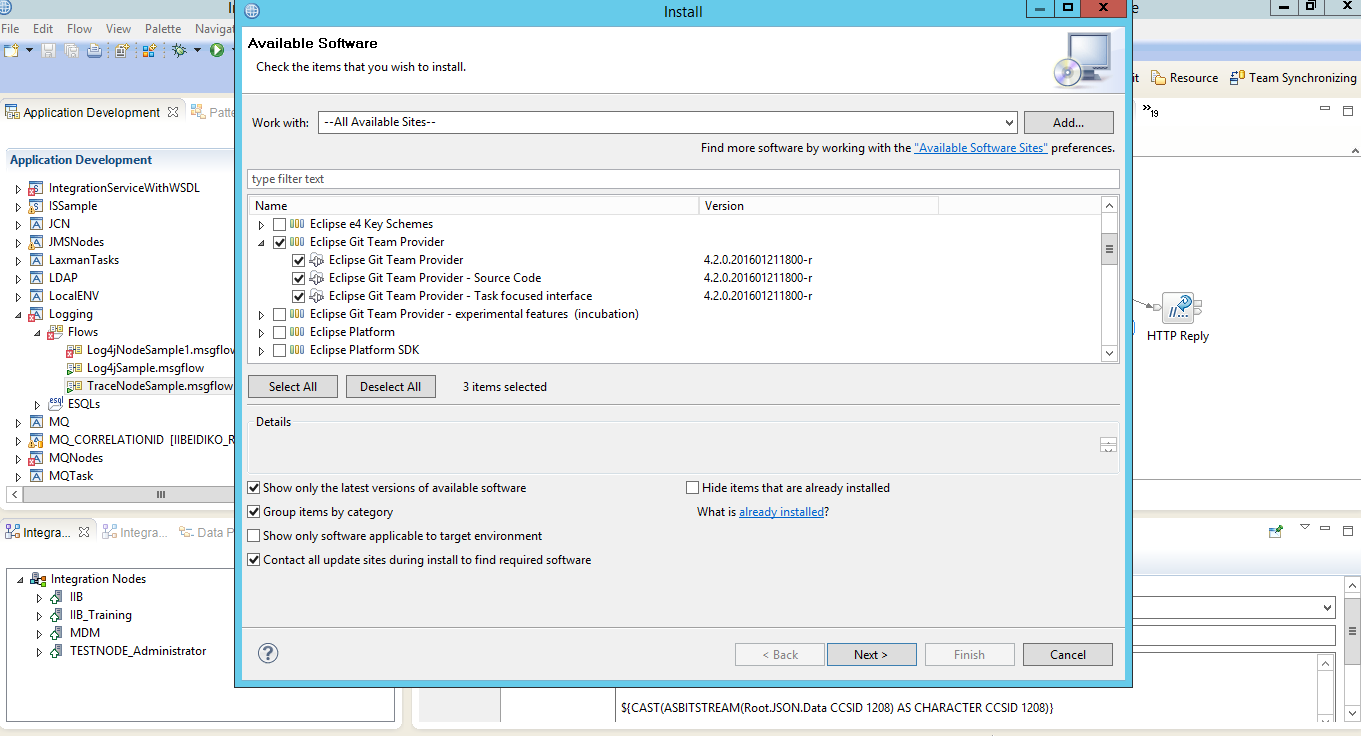
All of these products require a separate machine to be setup somewhere accessible on the developer's network. Then, of course there's the set up and administration of the software and management of the server itself. GitHub as a repository offers the advantage of already being setup and available as far as the software and hardware goes. All that's required is an internet connection, a GitHub account, and for someone to create and administer the desired repositories for your organization to access them.

In this document, we will show

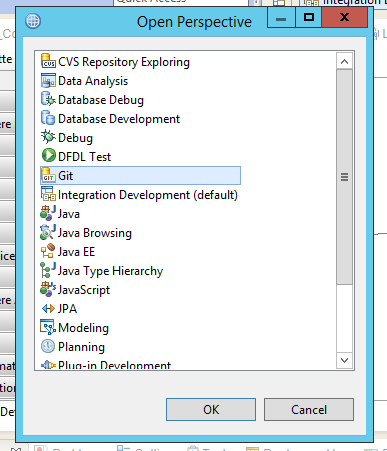
* How to setup a GitHub repository and how to use that repository as an IBM Integration Bus (IIB) code repository within Eclipse.
* The steps necessary to create a GitHub organization and repository.
* The installation of the Git for Windows software and the eGit plugin that's required on the client machine will be covered, including making the connection to a previously created repository via the Eclipse toolkit.
* Additional topics include; how to check-in and check-out code from the repository, creating branches and managing code merges.

# Local Software Install:

* Open your IIB Eclipse Toolkit and go to Help >> Install New Software
* In the interface that pops up, click the Add button in the top right corner so we can add the site for the eGit plugin. Click on All Available Sites >> Eclipse Git Team Provider



* Click "Next" again, then accept the license terms and click "Finish".
* The IIB Eclipse toolkit will need to be restarted when complete.
* Once restarted, you can open the Git perspective by doing the following:
* Go to Window –> Open Perspective —> Other
* Then select "Git" from the list. Click on OK.

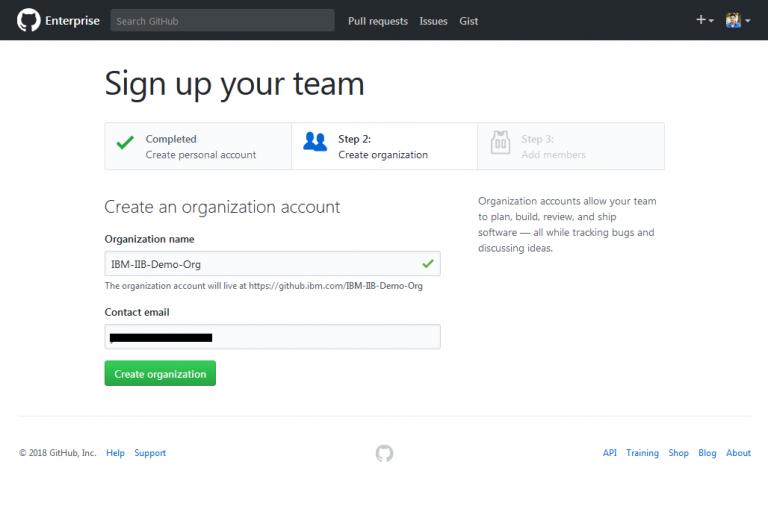


# Setting up an Enterprise GitHub Repository:

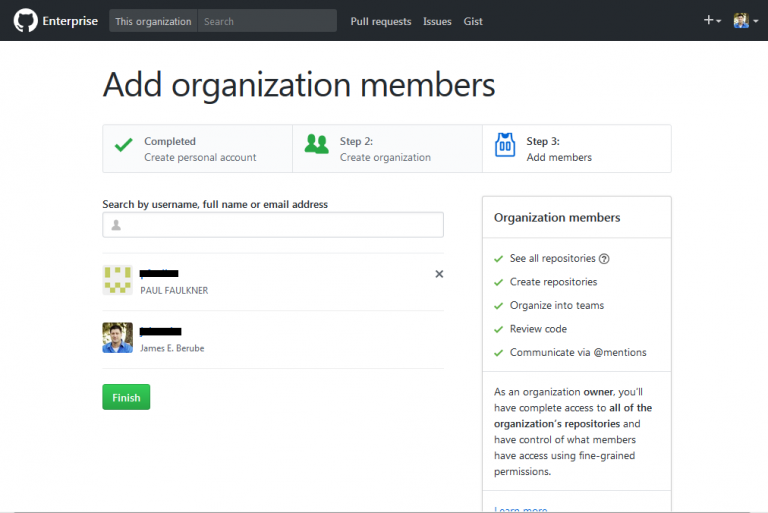
Now we'll create a repository to store code in. This activity does not rely on the local software setup, but you will not be able to connect your IIB Eclipse toolkit to your GitHub repository until that has been completed.

In GitHub, you can create repositories directly under your profile, which you will own and can add other members to. Another way is to create an Organization and create your repositories under that instead.

* To start with, go to the Organizations tab under Settings in GitHub Enterprise. Then click on the "New Organization" button in the top right corner. Enter the name of your Organization and your contact email address, as shown:

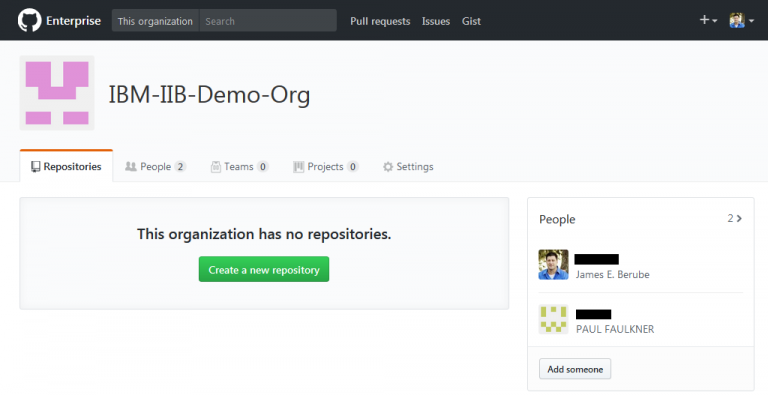


* Then click the "Create organization" button.
* Once that is complete, you can add members to your Organization. Just search for members and click their information to add them. GitHub members outside your Enterprise can be added



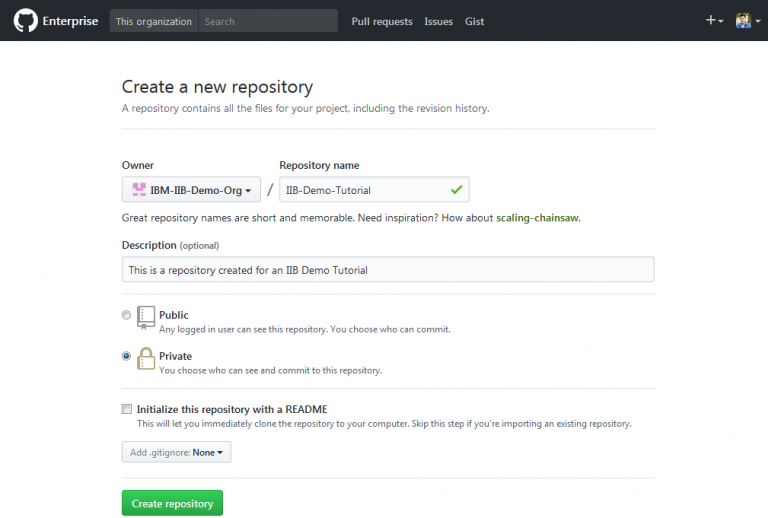
* Once all members have been added, click the "Finish" button.

You should now see the home page for the Organization that was just created. There are several tabs available; to create and manage the Organization's repositories, manage the members and their access, create and manage teams, create and manage projects and the general settings for the organization.



For now, we're just going to focus on creating a repository for our organization.

* Click the button to "Create a new repository".
* Give your repository a name and a description. We'll be creating this as a Private repository, which means it won't be seen by anyone who's not a member of the Organization.

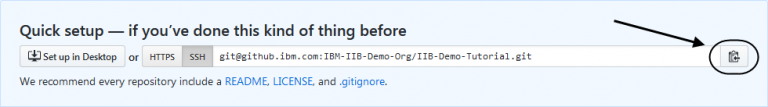


Now that a repository has been created, we'll add it to the IIB Eclipse Toolkit.

# Connecting to the Repository:

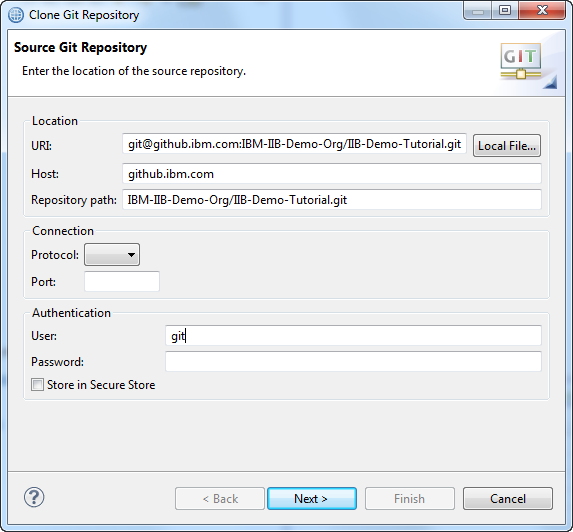
The Git software has been installed and configured. A GitHub repository has been created. Now, you need to connect your IIB Eclipse toolkit to the repository so you can start checking code in.

* Go to your Enterprise Organization Repository page and click the button to copy the information in the first box, under "Quick setup".

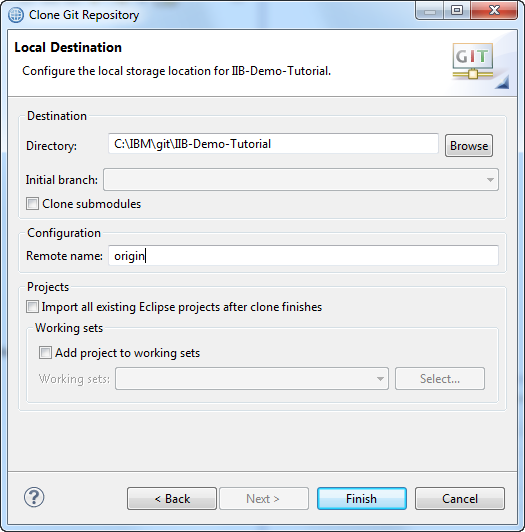


* Open your IIB Eclipse Toolkit and go to the Git perspective.
* Click the link to "Clone a Git repository" and a new window will open.
* Click Paste.

All of the information should be auto-populated in the correct places in the window to add your repository link.



* Click "Next". If you get a popup about the site not being trusted yet, click "Yes" to accept and "Yes" again to create a known hosts file. Enter the passphrase for the SSH key you created earlier and click "OK".
* You will see a message that the Branch is currently empty under Branch selection. This is fine, since we haven't added anything to the repository yet. Click "Next".
* If other users are adding this connection later, after code has been added to the repository, we strongly recommend clicking the checkbox to "Import all existing Eclipse projects after the clone finishes" when making this connection. If they do not, they will not see anything added to their workspace.



* Then click "Finish".
* You should now see a connection to your repository listed under Git Repositories.

10.png

* There is a Git toolbar that can be added to the Git perspective. To add the Git commands. Open the Git perspective, then go to the Window menu bar and select Customize Perspective.
* Go to the Command Groups Availability tab, then scroll down and check the Git command groups.
* The Git commands will be added as a toolbar.

11.png

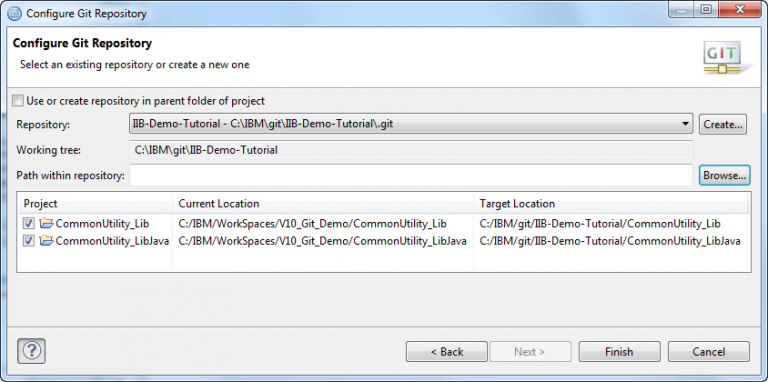
# Initial Code Check-in:

Your IIB Eclipse toolkit is now connected to the GitHub repository and it's time to start checking code in for the first time. This section describes the process to check code in that doesn't yet exist in your repository.

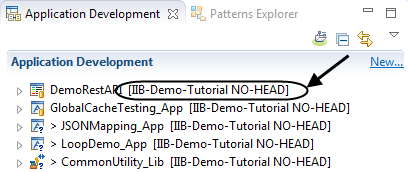
* To initially check code into the Repository, go to the Integration Development Perspective.

**NOTE:** We recommended that you close any open objects that are going to be checked in. The files will still appear in your workspace, but the physical location of the files will change. The files will be removed from the workspace on your file system and moved to your local git repository file system location.

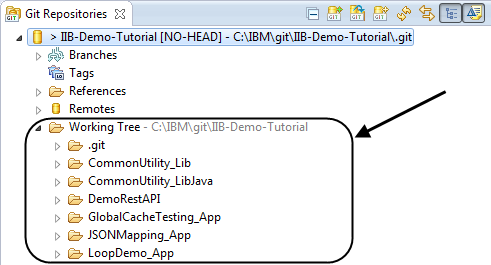
* Right click on the code that you'd like to check in and select Team –> Share Library (or Application) in the window that pops up.
* Share the code and select "Git" for the repository and click "Next".



* In the window that opens, select your Git repository from the drop down and make sure your projects are checked, then click "Finish".
* Once you've shared the project(s) with your repository, they will be moved, under the covers, from your workspace to your local git repository directory, indicated by the Target Location directory. Now, when opening the toolkit, you will still go to the same workspace, but all the code will actually be physically located in the target location.
* Now, in your toolkit, notice that all of the code has markers indicating it's associated with a repository.

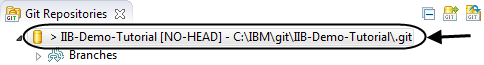


* Switching over to the Git perspective, the code that was just shared will now be visible under Working Tree.

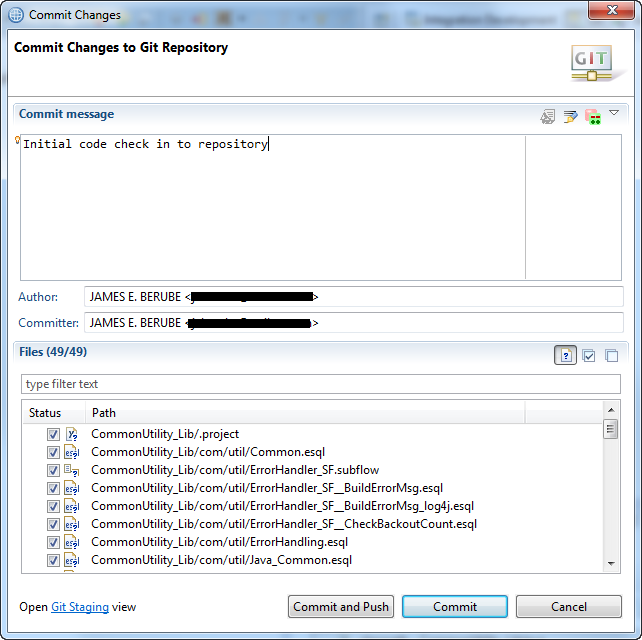


* The code still needs to be Committed and Pushed to get it to the actual GitHub repository as all work is always done locally first. Using a Git repository, you're operating in a disconnected mode until you request a connection to the remote repository to push your changes or when pulling or fetching changes from the remote GitHub repository. The Commit is to your local repository, where the push is sending your changes up to the GitHub remote repository.

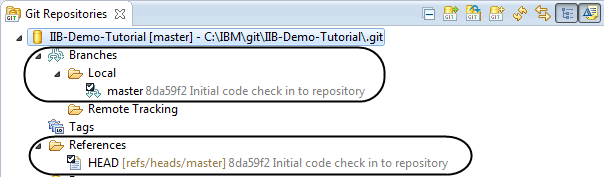
There are multiple ways to accomplish this. Either by clicking the "Commit" icon on the Git toolbar, right clicking on the repository and selecting "Commit" from the dropdown menu, or going to the "Git Staging" panel in the lower right hand portion of the toolkit and pressing the "Commit" or "Commit and Push" buttons. We'll be doing the commit based on right clicking the repository. From the Git perspective, right click on your repository name and select "Commit".



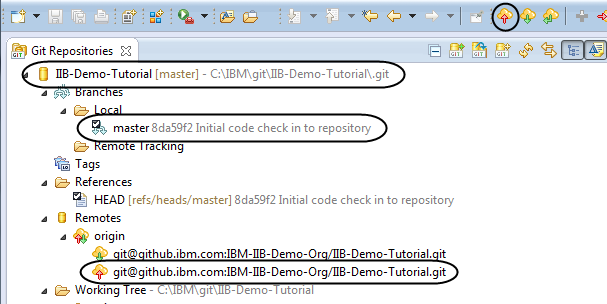
* This will open up a new window, where you will select the files to be committed and add a check in comment. The checkmark button in the lower half of the window on the right will allow you to select all files. If there are any files that should not be checked in (like java class files or anything that’s generated by the toolkit), make sure to deselect them.



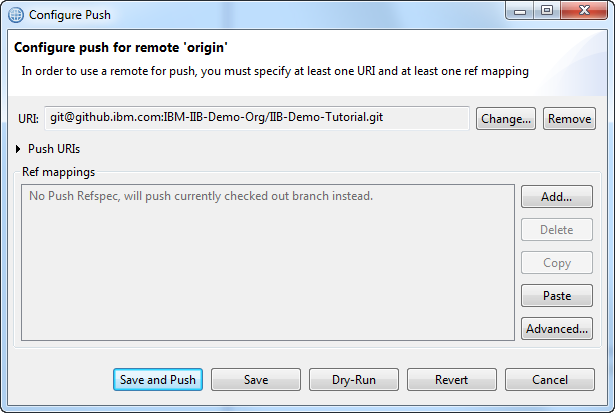
* Once you've added your comment and selected the files, click the "Commit" button.
* Now that the code has been checked in, you'll see that 2 new entries have been added to the Git Repositories view, under Branches –> Local and References.



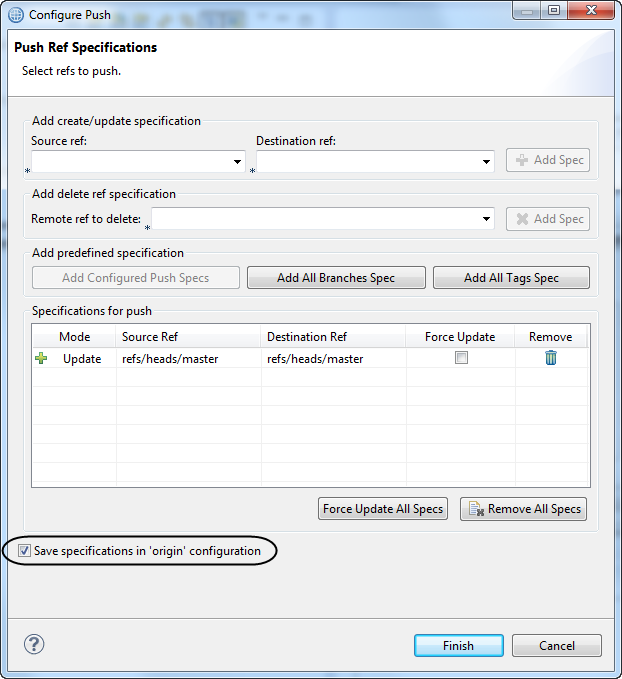
* This has committed the changes from your workspace to your local copy of the repository. In order to get code from your local repository up to the GitHub repository, the code still needs to be "pushed". To push code to GitHub, there are multiple ways to do that as well. By right clicking on the Repository name and selecting "Push Branch to master", right clicking on the Local Branch and selecting "Push Branch", clicking the toolbar button for push, or by right clicking on the red arrowed line to push under Remotes –> origin and selecting "Push".



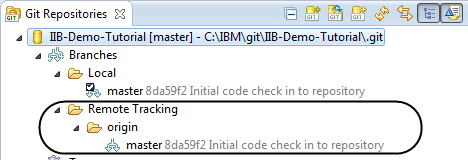
* Since this is the first time we're pushing code to GitHub, we'll want to configure the Push first. So, let's start by right clicking on the "red arrow" line under Remotes –> origin and selecting "Configure Push". This will bring up a new window.



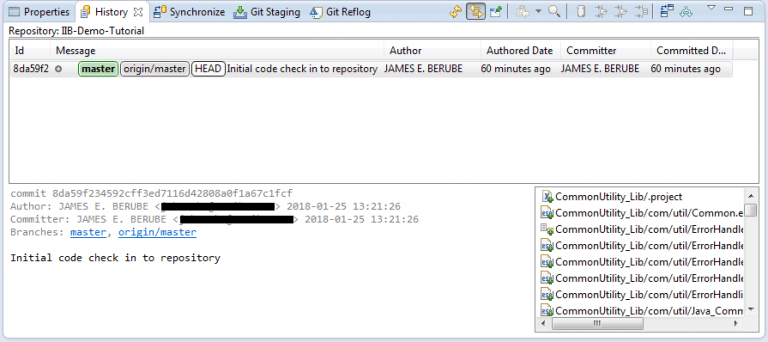
* As you can see, no "Ref mappings" have been configured yet. You can click "Add" and manually enter all the information required or click on the Advanced button and use the wizard. We're going to click Advanced. (If prompted, enter your ssh password created earlier.)
* In the Push Ref Specifications, select the Source ref as your master. the Destination should be auto populated. Then click the Add Spec button. The spec will be added under "Specifications for push". Also check the "Save specifications in 'origin' configuration" checkbox before clicking Finish.



* Now, from the "Configure Push" window, click the "Save" button to save this configuration. (Save and Push is an option, and would be a shortcut, I want to show them individually here.)
* Now that the configuration has been saved, right click on the same line under Remotes –> origin and select "Push".
* A dialog will appear giving the status of the code being pushed to the repository and a Push Results window will appear when complete. (If there are any merge conflicts, you would see them here. There shouldn't be any now, since this is our initial code check in.) Click "OK" to exit. You should now see something created under Branches –> Remote Tracking



* Notice that the hash values for each of these are all the same. In the case shown, 8da59f2.
* Right click on your code, select Show In, then select History. In the History view, you can see the history of your check-ins as well as the id associated with the branch.



* Now, looking at GitHub, you can also see that the code has been successfully pushed to the repository.

