

Contents

What is EGit ?	2
Installation	2
Clone existing project	4
Basic operations	7
Commit.....	8
Checkout.....	9
Create Branch.....	10
Merge	13

What is EGit ?

EGit is an Eclipse plug-in (software component) which allows you to use the distributed version control system *Git* directly within the Eclipse IDE.

Installation

If you do not have any version of Eclipse IDE. I downloaded the standart java version(Eclipse IDE for Java Developers) for this tutorial. In addition, this tutorial is based on Eclipse Luna 4.4.1. If you have any other version, it may seem different from my screenshots.

The screenshot shows the Eclipse Downloads page on a web browser. The URL is <https://eclipse.org/downloads/>. The page displays several download options for Eclipse Luna (4.4.1). The first item is "Eclipse IDE for Java Developers" at 154 MB, with links for Windows 32 Bit and Windows 64 Bit. Below it is "Package Solutions". The second item is "Eclipse IDE for Java EE Developers" at 254 MB, also with Windows 32 Bit and Windows 64 Bit links. The third item is "Eclipse IDE for C/C++ Developers" at 164 MB, and the fourth is "Eclipse for PHP Developers" at 126 MB. On the right side, there are "RELATED LINKS" for Compare & Combine Packages, Install Guide, Documentation, Updating Eclipse, and Forums. There is also a "MORE DOWNLOADS" section with links to Other Platforms, Eclipse Kepler (4.3), Eclipse Juno (4.2), Eclipse Indigo (3.7), and Older Versions. The browser's address bar shows "Inbox - furkan.tanirverdi@... modelwriter/wp5" and the status bar shows "14:22 19.11.2014".

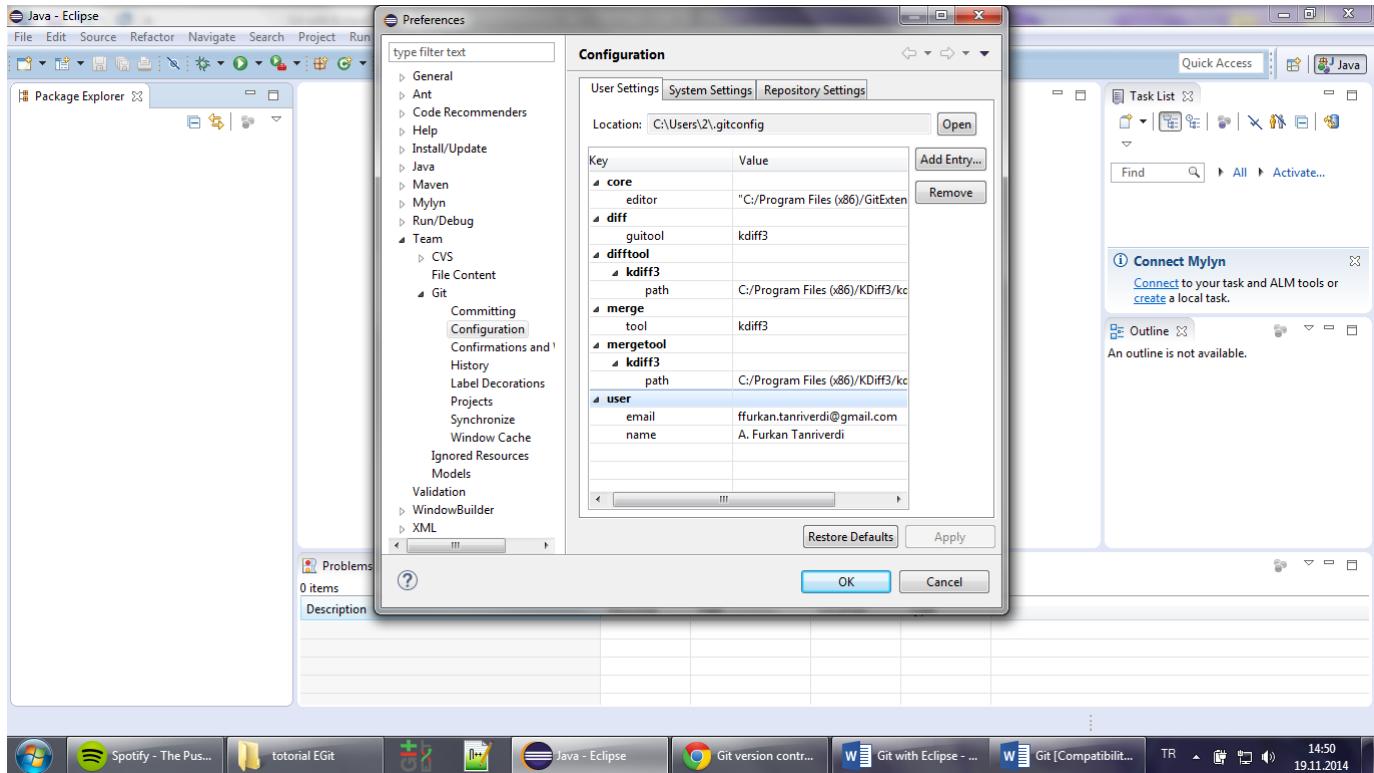
After eclipse starts, you can install Egit to your IDE. To do that, start this manager via the *Help → Install new Software* menu entry. Egit can be installed from the following link:

<http://download.eclipse.org/egit/updates>

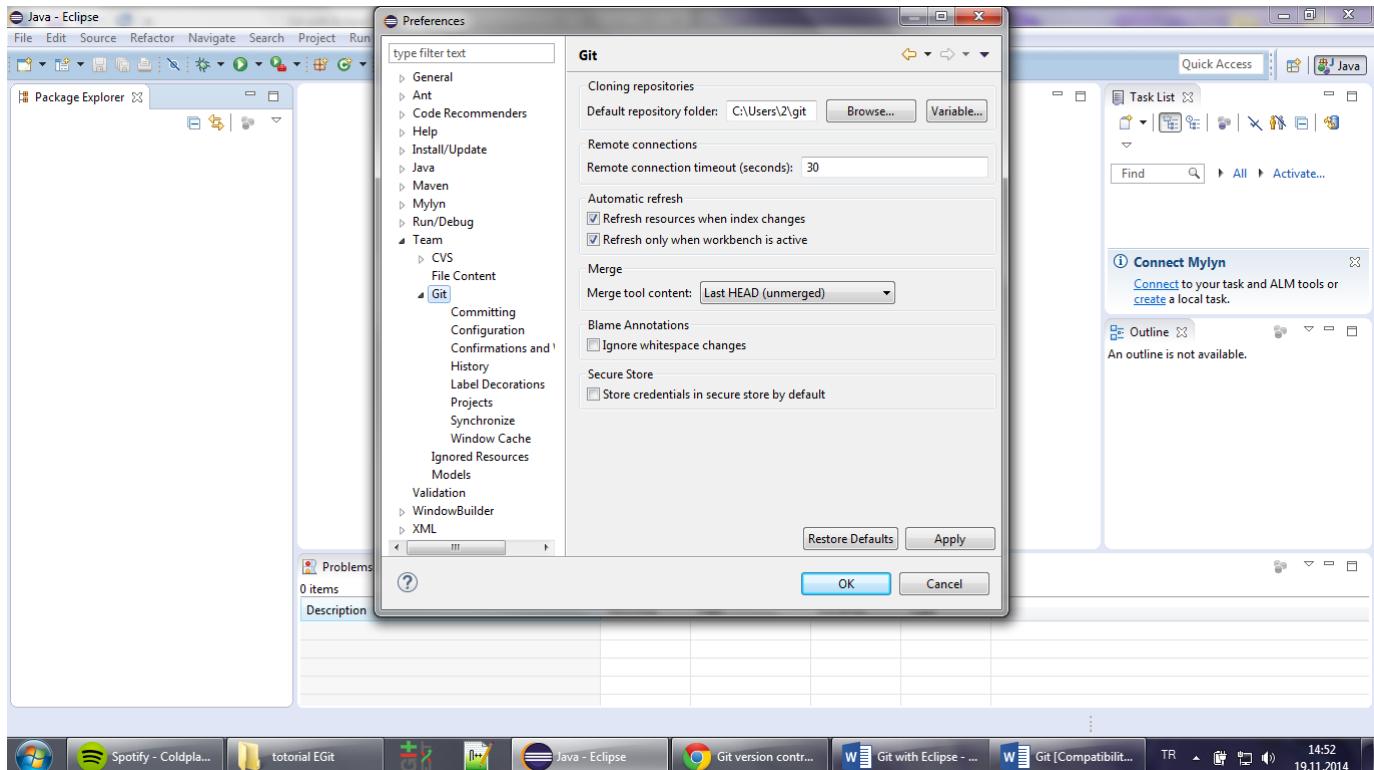
The screenshot shows the Eclipse 'Install' dialog box. The title bar says 'Java - Eclipse'. The main area is titled 'Available Software' with the sub-instruction 'Check the items that you wish to install.' A 'Work with:' dropdown is set to 'http://download.eclipse.org/egit/updates'. Below it is a 'type filter text' input field. A table lists software packages under the 'Name' and 'Version' columns. One item is selected: 'Eclipse Git Team Provider' (version 3.5.2.201411120430-r). Other listed items include 'Eclipse Git Team Provider - Source Code' (version 3.5.2.201411120430-r) and 'Task focused interface for Eclipse Git Team Provider' (version 3.5.2.201411120430-r). At the bottom of the dialog, there are 'Select All' and 'Deselect All' buttons, and a note that '1 item selected'. Below these are several checkboxes: 'Show only the latest versions of available software', 'Group items by category', 'Show only software applicable to target environment', and 'Contact all update sites during install to find required software'. The 'Details' section below the table describes the selected provider as 'An Eclipse Git Team provider in pure Java.'. There are 'More...' and 'Next >' buttons at the bottom. The status bar at the bottom of the screen shows various icons and the time '14:35 19.11.2014'.

Select *Eclipse Git Team Provider*, then finish the installation.

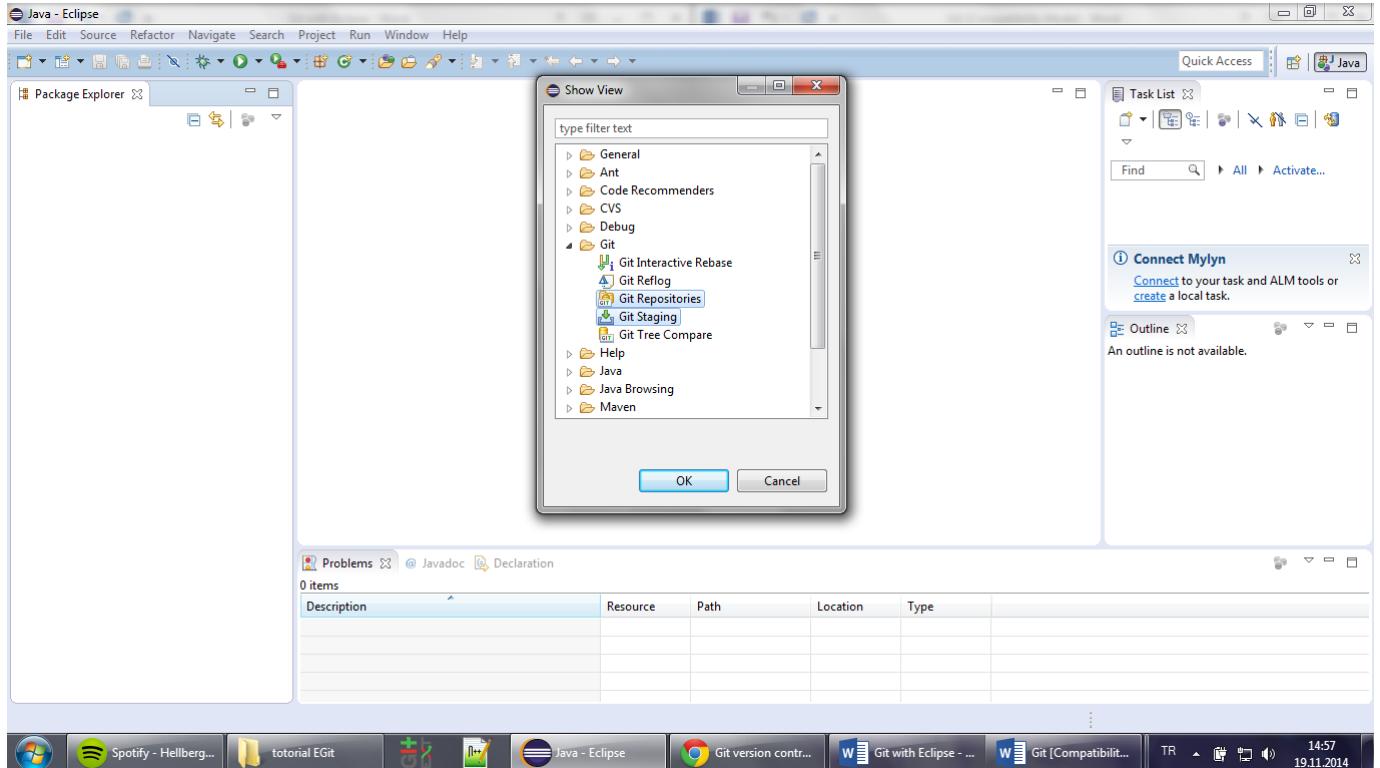
After Eclipse restarts, you should configure your name and email address which is used to fill author and committer information of commits you create. Select *Window -> Preferences -> Team -> Git -> Configuration* to see the current configuration and to change it.



Egit has a default clone location. To change, select *Window -> Preferences -> Team -> Git*. You can change the repository from *Cloning repositories* section.



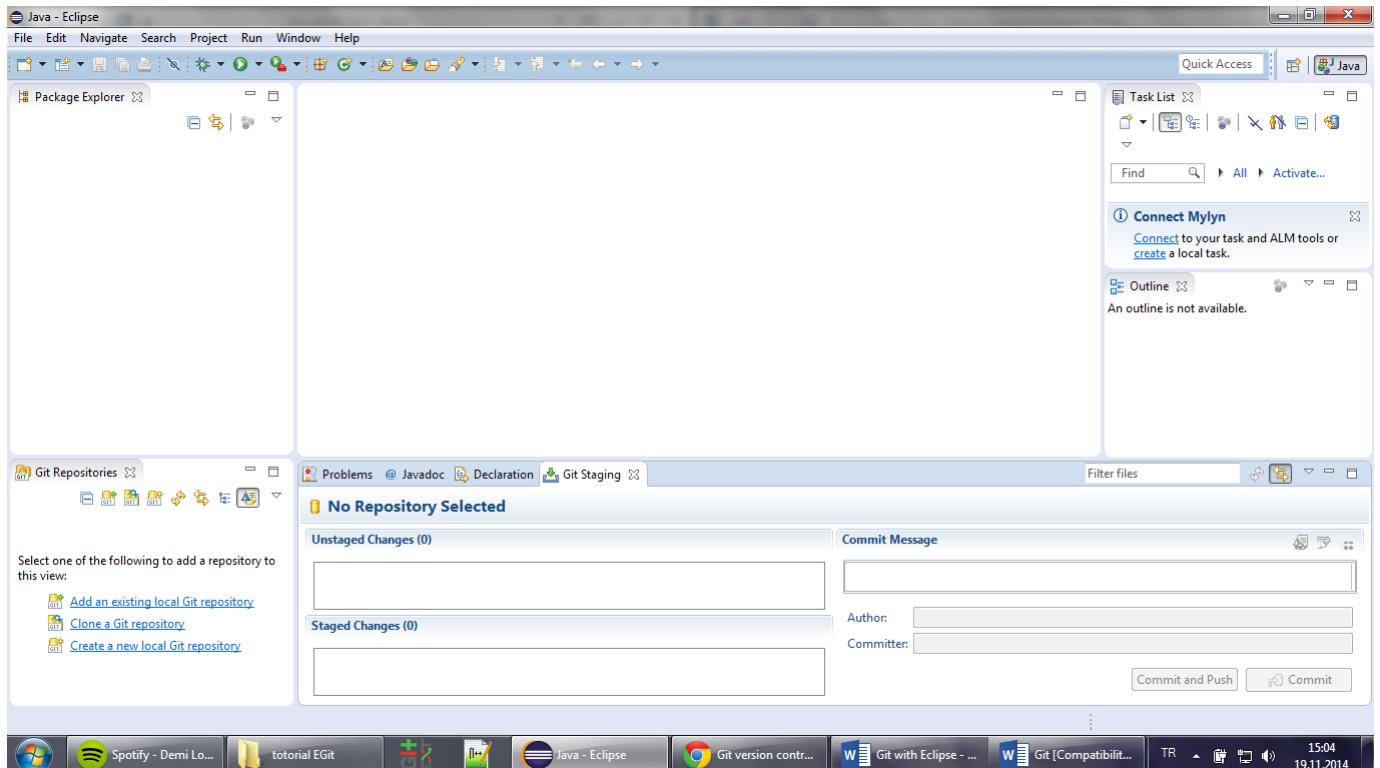
To manage repositories and commits, you can add some views. Select *Window -> Show View -> Other -> Git*. Select *Git Repositories* and *Git Staging* under Git folder.



Finally, we completed Egit installation.

Clone existing project

Eclipse allows you to clone an existing Git repository and to import existing projects from this repository into your Eclipse workspace by using a wizard. Select *Clone a Git Repository*.



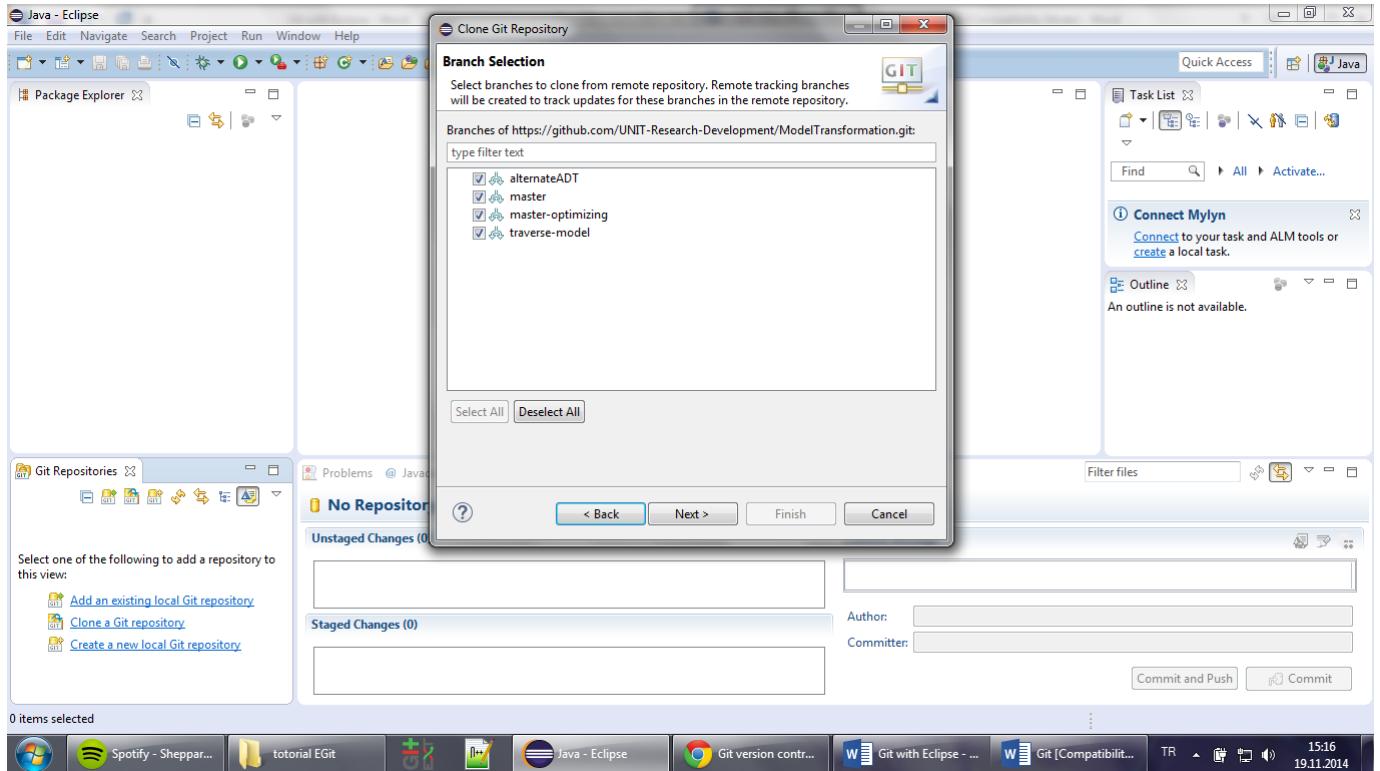
I will clone my current project on GitHub. Just copy the HTTPS URL.

The screenshot shows a web browser window with the URL <https://github.com/UNIT-Research-Development/ModelTransformation>. The page displays basic repository statistics: 24 commits, 4 branches, 0 releases, and 2 contributors. A list of recent commits is shown, along with links to README.md, .gitignore, LICENSE, and README.md. On the right side, there's a sidebar with 'Code' (Issues: 1, Pull Requests: 0), 'Wiki', 'Pulse', 'Graphs', and a 'Clone' section with an 'HTTPS clone URL' field containing the URL and options to 'Clone in Desktop' or 'Download ZIP'. Below the main content, there's a preview of the README.md file which contains the title 'ModelTransformation'.

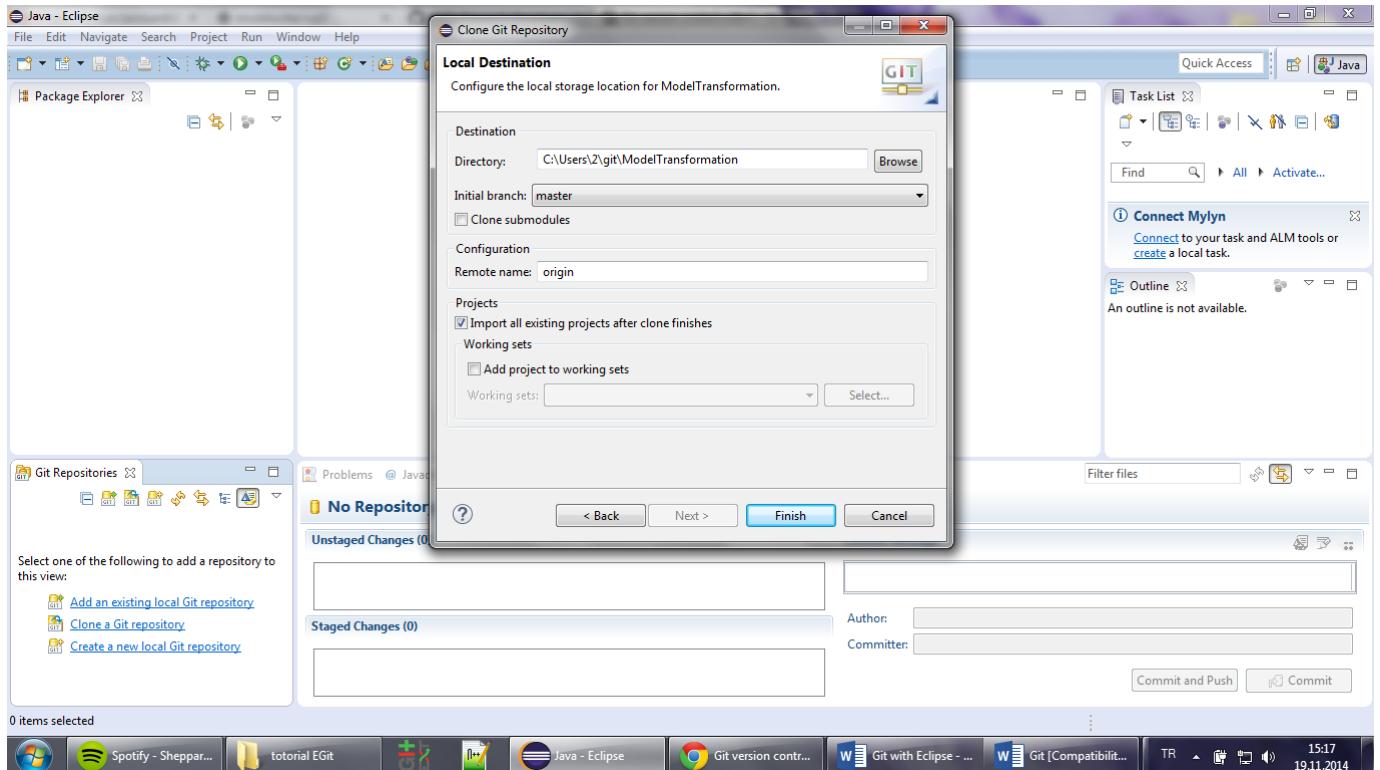
You only have to paste the URL to the first line of the dialog, the rest will be filled out automatically.

The screenshot shows the Java - Eclipse IDE interface. A 'Clone Git Repository' dialog box is open in the center. It has a 'Source Git Repository' header and a 'Location' section where the URI is set to <https://github.com/UNIT-Research-Development/ModelTransformation>, the Host is 'github.com', and the Repository path is '/UNIT-Research-Development/ModelTransformation.git'. The 'Connection' section shows 'Protocol: https'. In the 'Authentication' section, the User is 'ffurkan' and the Password is masked. A checkbox 'Store in Secure Store' is checked. At the bottom of the dialog are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'. The background shows the Eclipse interface with a 'Git Repositories' view and a 'Task List' view. The task list has a message: 'Connect Mylyn' with a sub-instruction 'Connect to your task and ALM tools or create a local task.' The status bar at the bottom indicates '15:13 19.11.2014'.

After pressing the *Next* button the system will allow you to import the existing branches. You should select at least *master* as this is typically the main development branch.



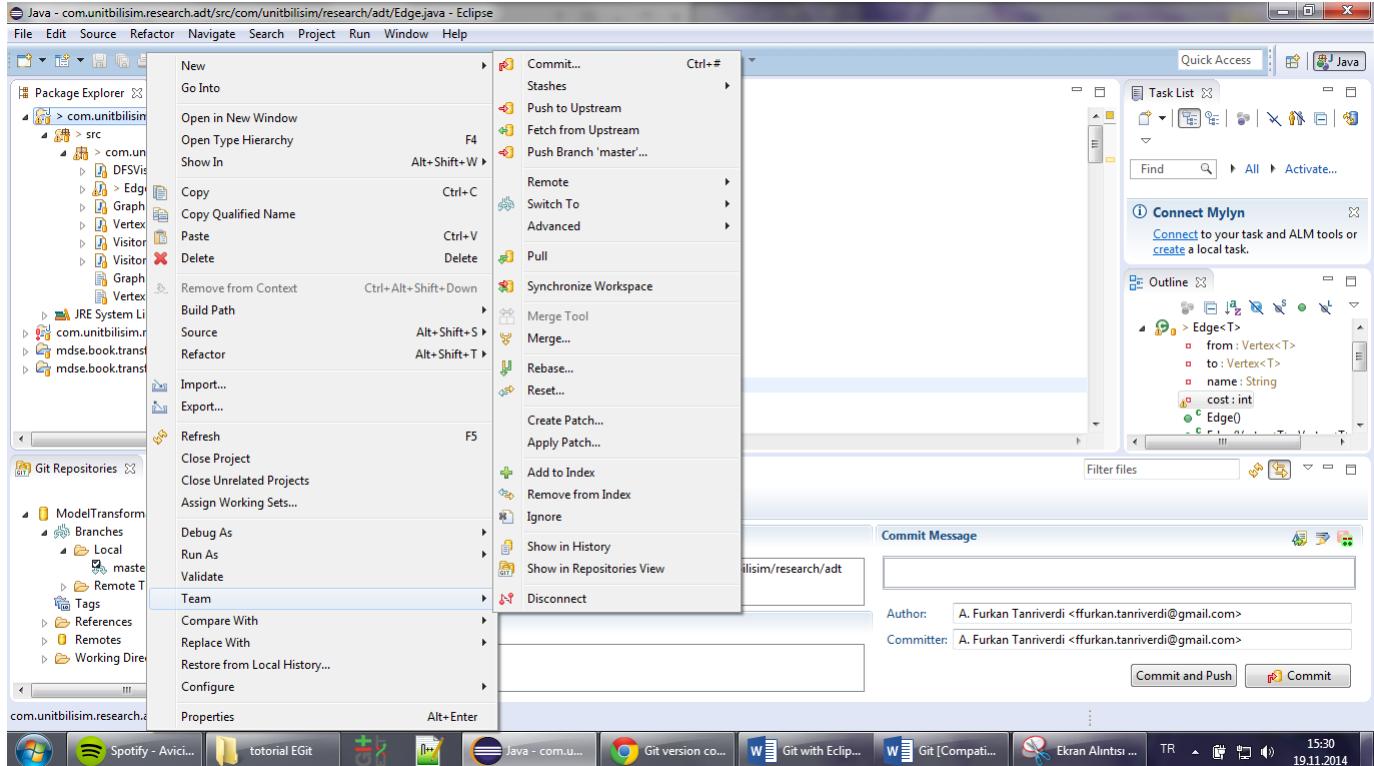
The next dialog allows you to specify where the project should be copied to and which branch should be initially selected. You can also import projects inside the repository at that time with checking *Import all existing projects after clone finishes*.



Once this dialog is completed, you have checked out (cloned) the projects into a local Git repository and you can use Git operation on these projects.

Basic operations

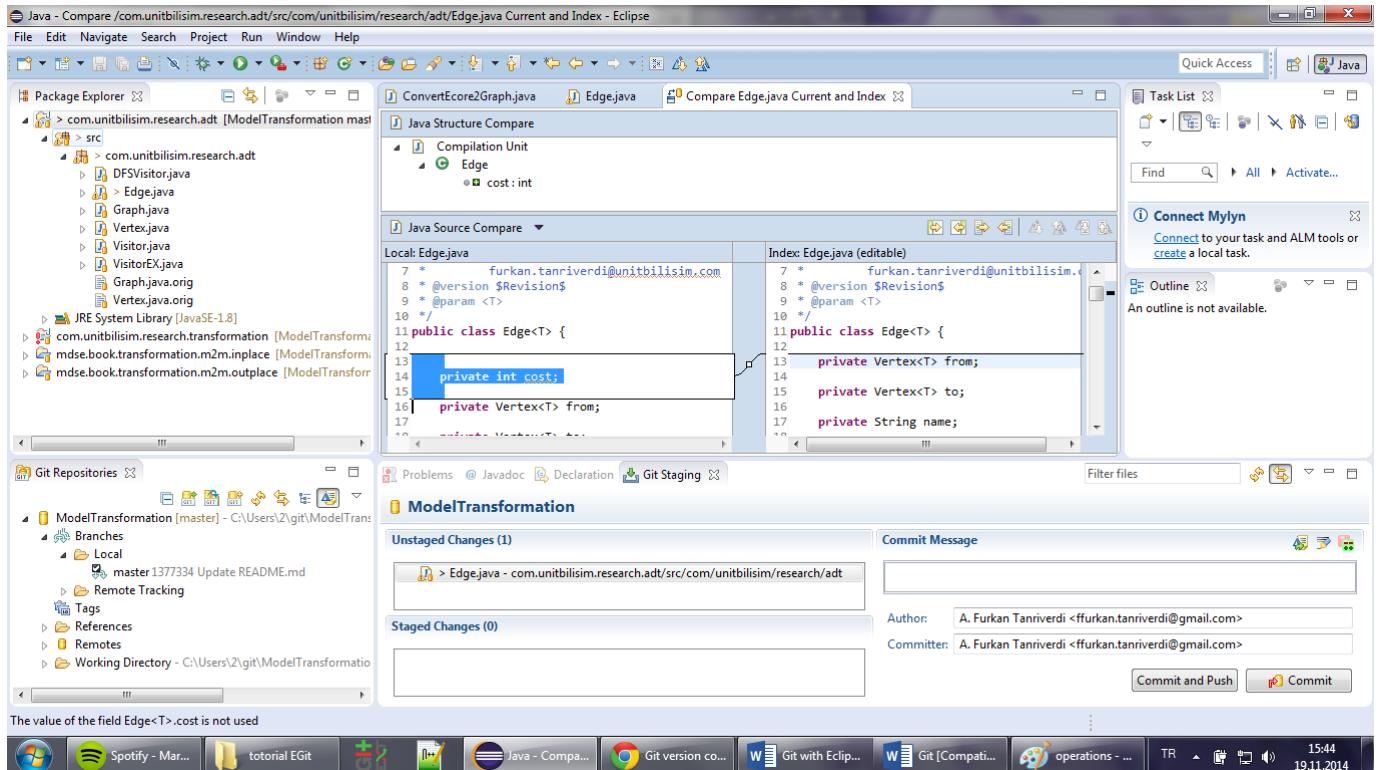
Once you have placed a project under version control you can start using team operations on your project. The team operations are available via right-click on your project or file. The most important operations are described in the following list.



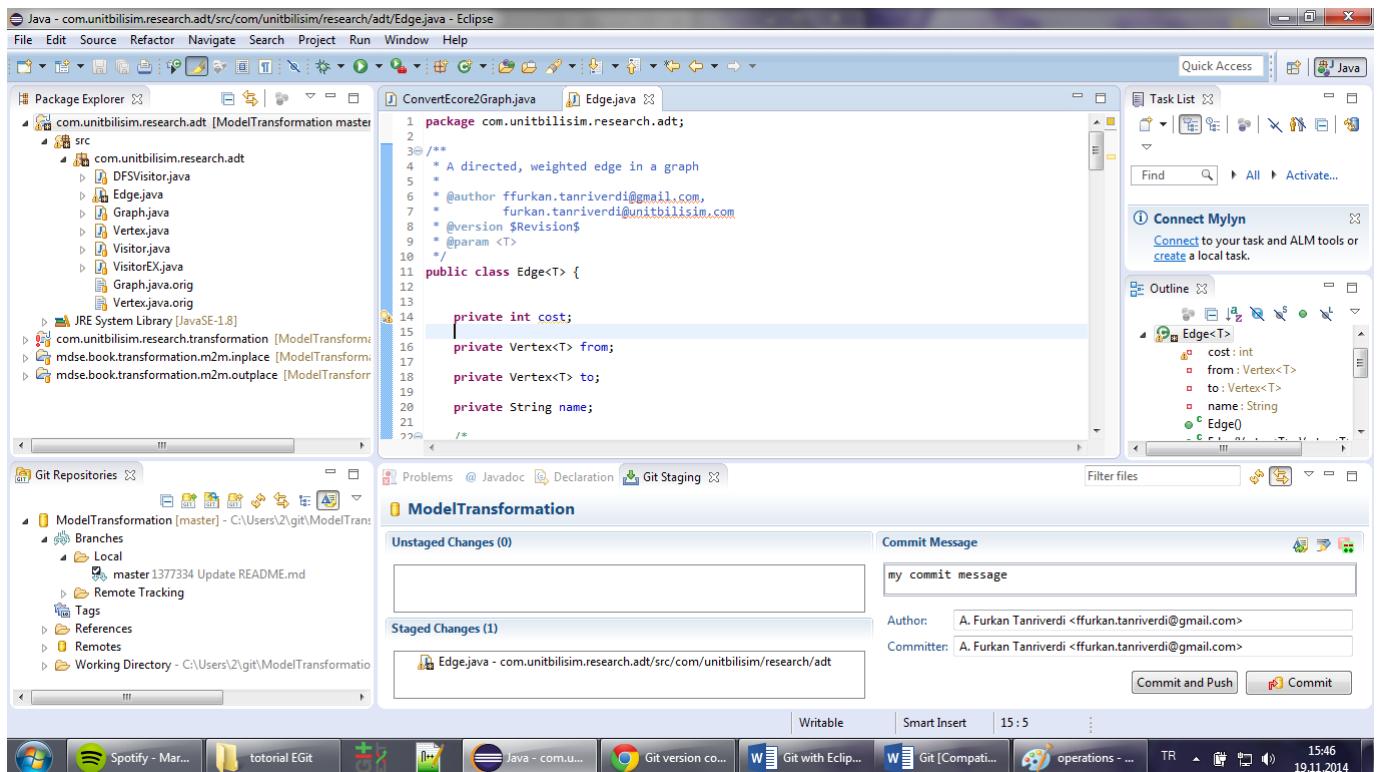
- *Team* → *Add to index*, to add the selected resource(s) to the index of Git
- *Team* → *Commit*, to open the commit dialog for committing to your Git repository
- *Team* → *Create Patch...*, to create a patch
- *Team* → *Apply Patch...*, to apply a patch to your file system
- *Team* → *Ignore*, to add the file to a .gitignore file
- *Team* → *Show in History*, to display the selected files and folders in the *History* view
- *Team* → *Pull* to pull in changes from your remote Git repository
- *Team* → *Fetch* to fetch the current state from the remote repository
- *Team* → *Switch To* to switch or create new branches
- *Team* → *Push* to push changes to your remote Git repository

Commit

When you make changes on project, it appears under the Git Staging view. I added a property to Edge class named *cost*. You can see below.



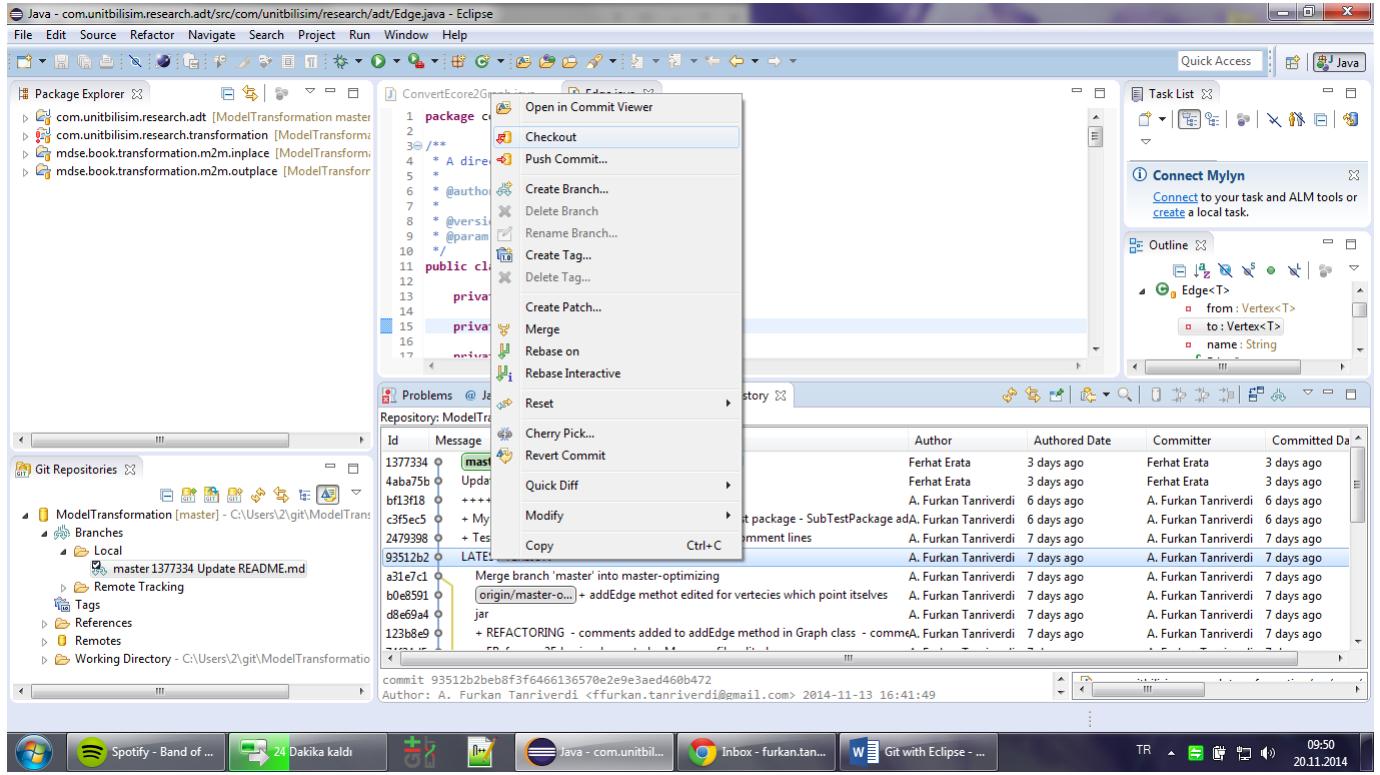
If you want to stage changes, right-click on file then click *Add to Index*. Do not forget to add commit message(s).



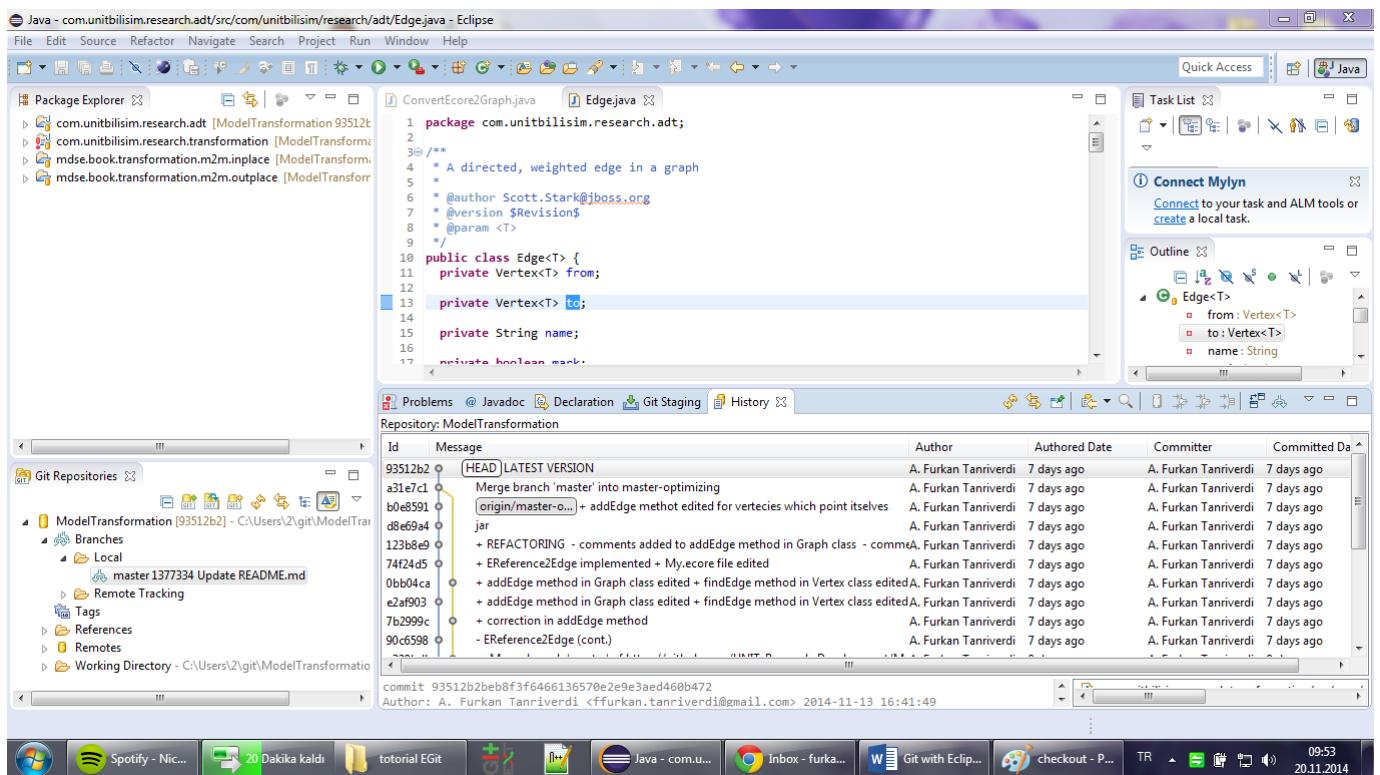
Click on *Commit* to store changes on local. Other option is *Commit and Push* which is for commit and push to remote.

Checkout

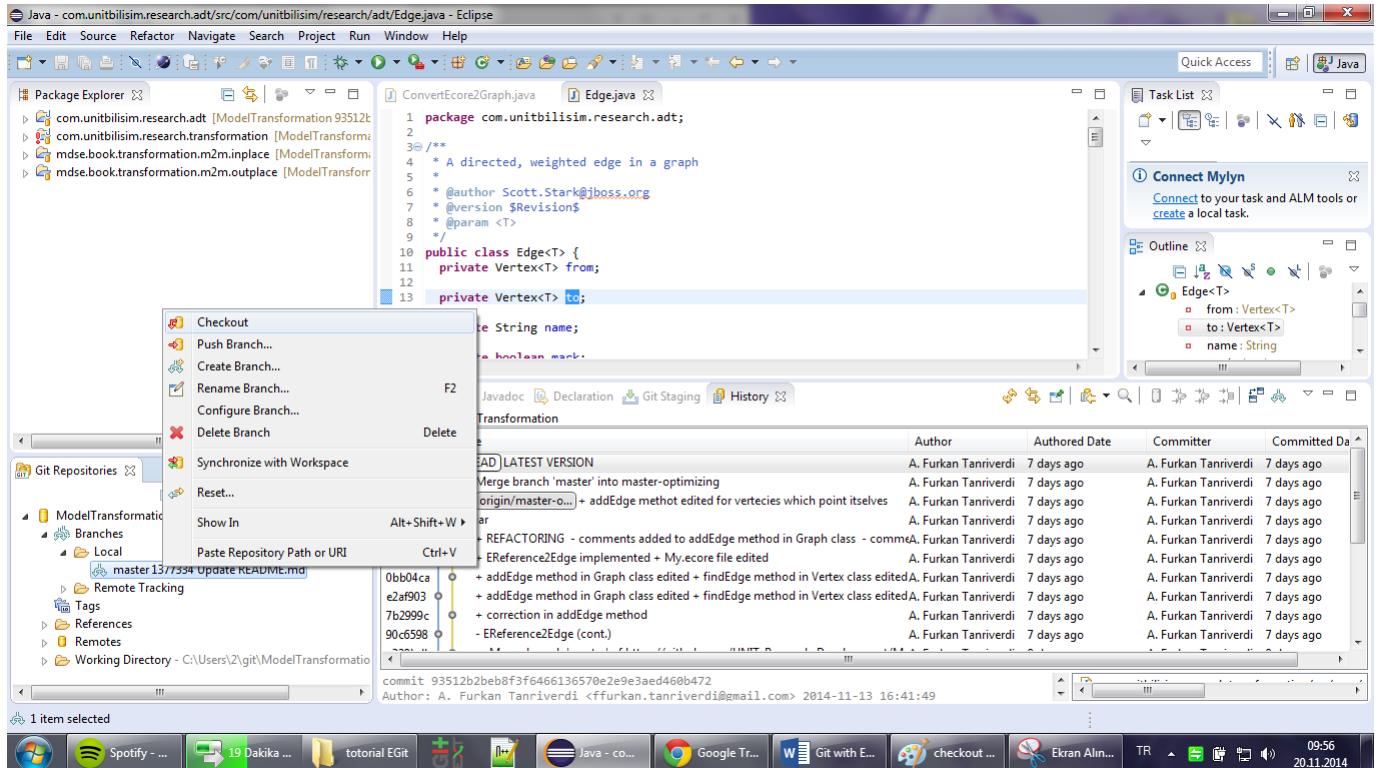
Checkout is used if you want to take a look your previous versions and see the differences. Right-click on repository or branch that you want to checkout, then *Show In -> History*. You will see commit history of that particular branch. Right-click on a commit and click *Checkout*.



Now, *HEAD* is on *LATEST VERSION* as you can see below. This means I returned the 7 days prior version of my codes.

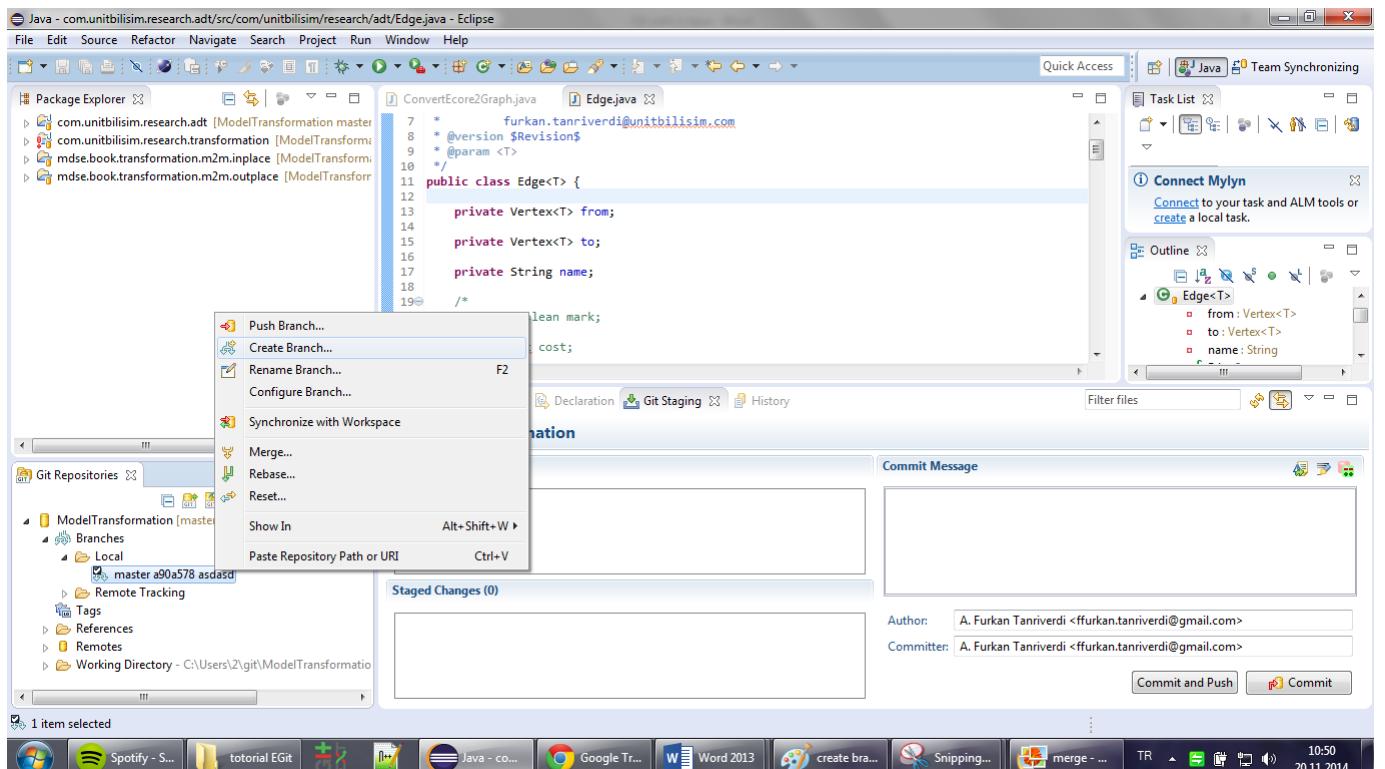


You can return back to your current position with checking-out master.

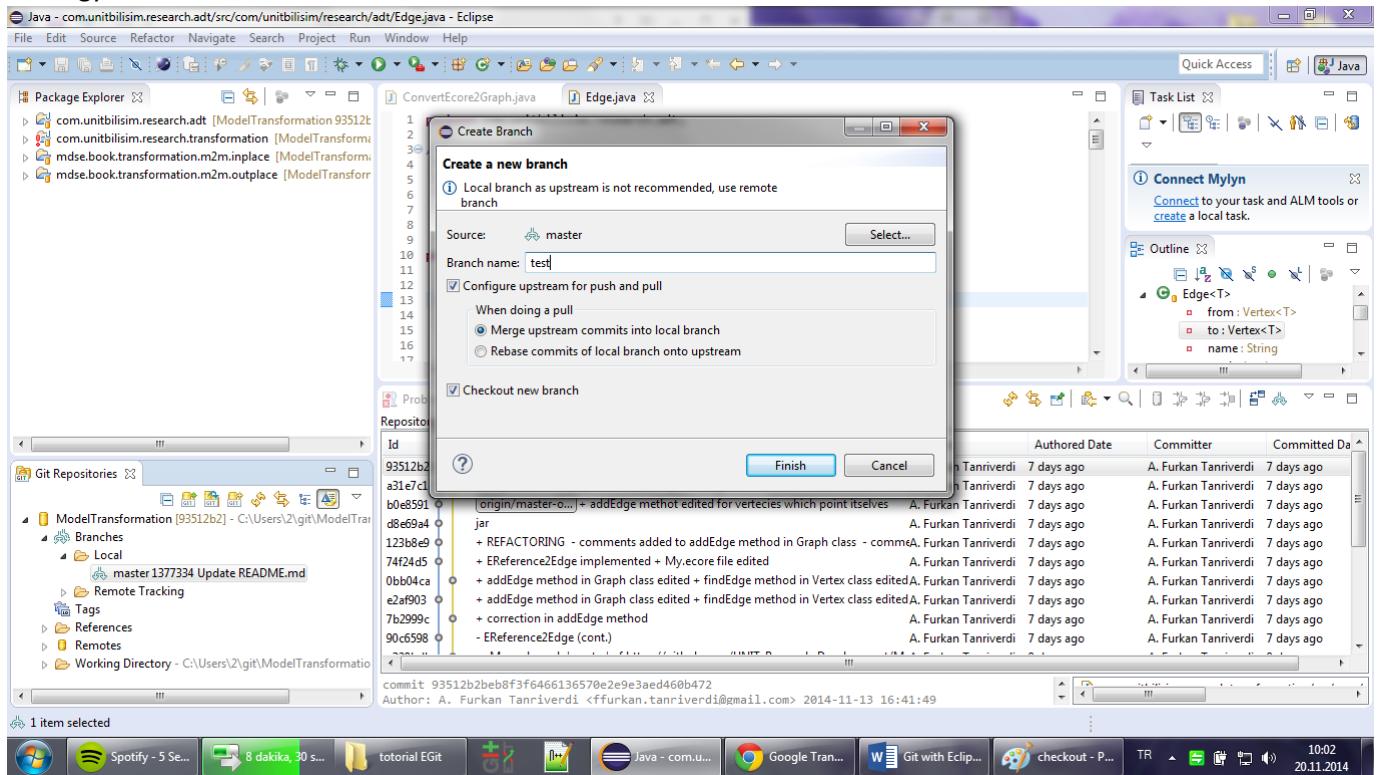


Create Branch

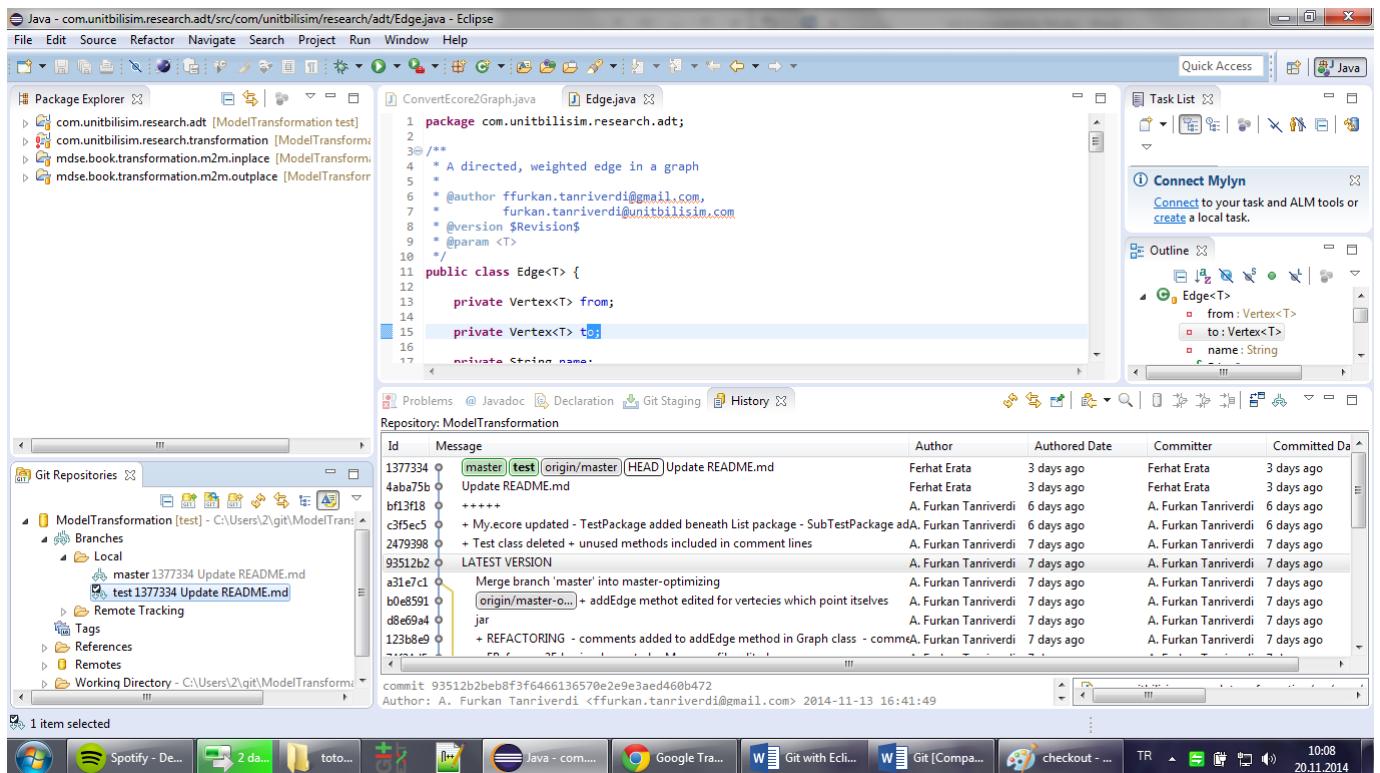
Right-click on a branch(in our case *master*) and click *Create Branch*.



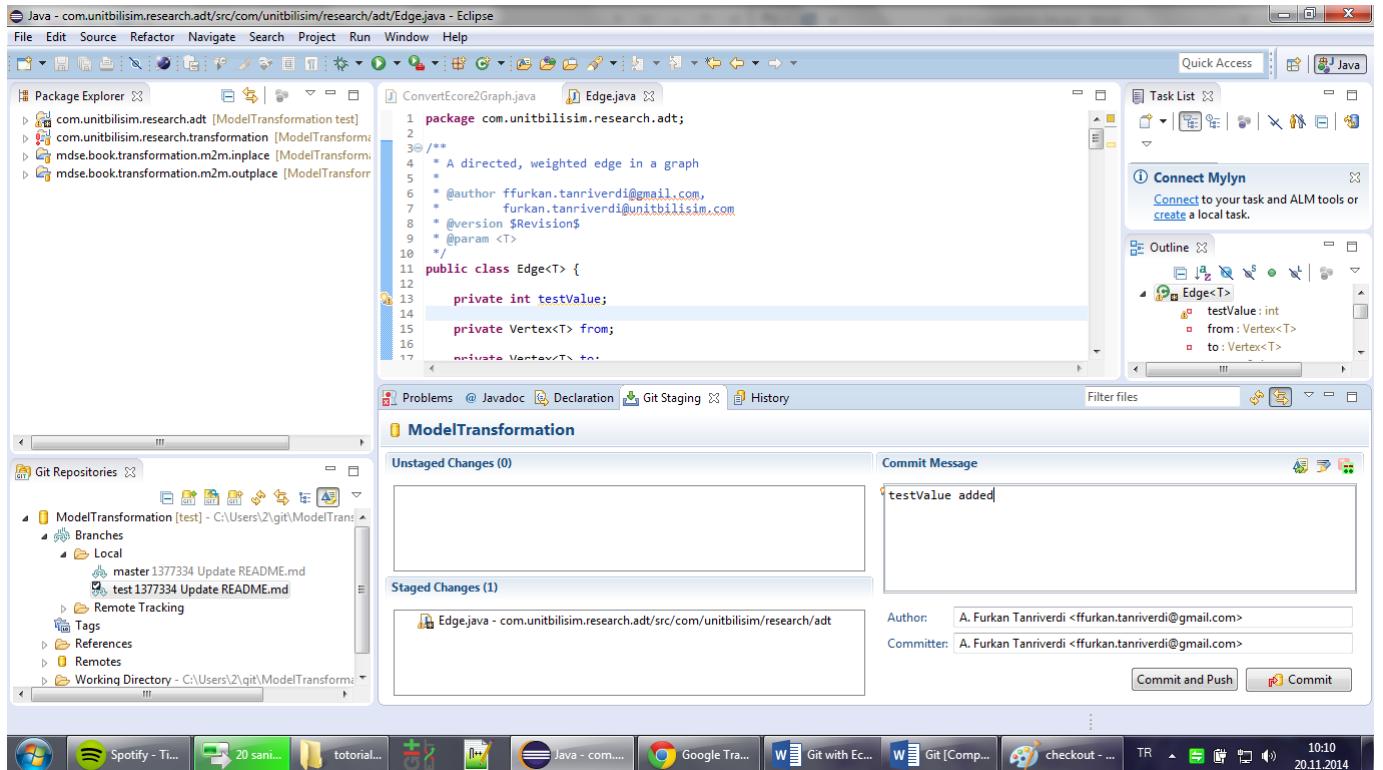
In the dialog window, enter your branch name and check *Merge upstream commits into local branch* as your pull strategy.



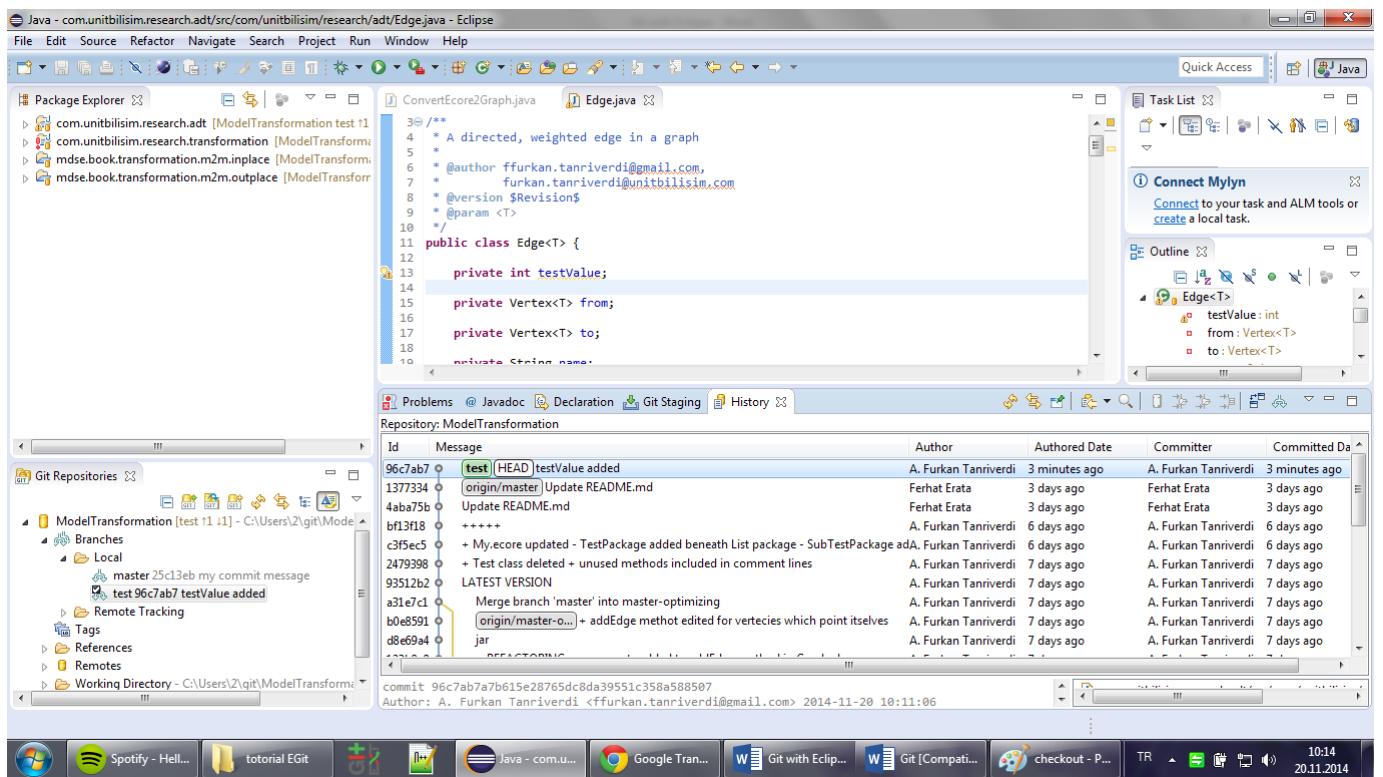
You can see the new branch named *Test*. Let's make some changes on it.



I added a new property which is *testValue*.



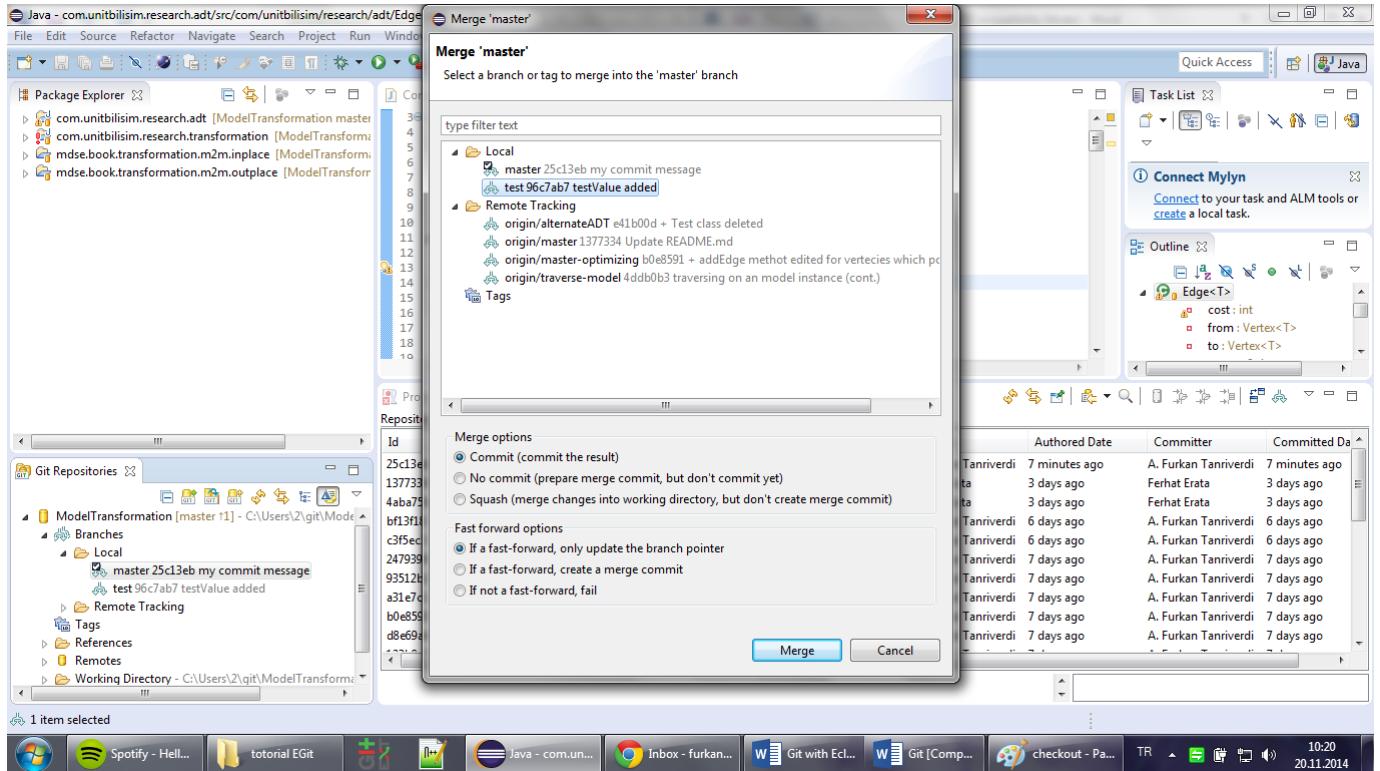
After commits the changes, it is shown in history as below.



Merge

It combines two branches into one. In our case, merging *master* with *test*. You have to checkout *master* first.

Right-click on *master* branch and select *Merge*. Select the branch you want to merge with.



As you can see below, *test* and *master* branches point the same commit.

