

MAY 25, 2012

# Eclipse and Github Tutorial

**Github** is an awesome repository to share your source code. Although there are [numeroustutorials](#) discussing how to use git and eclipse, I got stuck again today while trying to **upload** an existing eclipse project to **github**. This tutorial walks thus through all the steps from signing up for github to uploading an eclipse project to the site!

Please note that the focus of this tutorial is the mere upload of source code and not any of the more sophisticated features git and github offer.

The following steps will be discussed in this tutorial:

1. Sign Up for github
2. Installing EGit
3. Create a DSA Key in Eclipse
4. Register DSA Key with github
5. Create Repository on github
6. Import github Repository into eclipse
7. Link Eclipse Project with github Repository
8. Uploading Project Sources to github

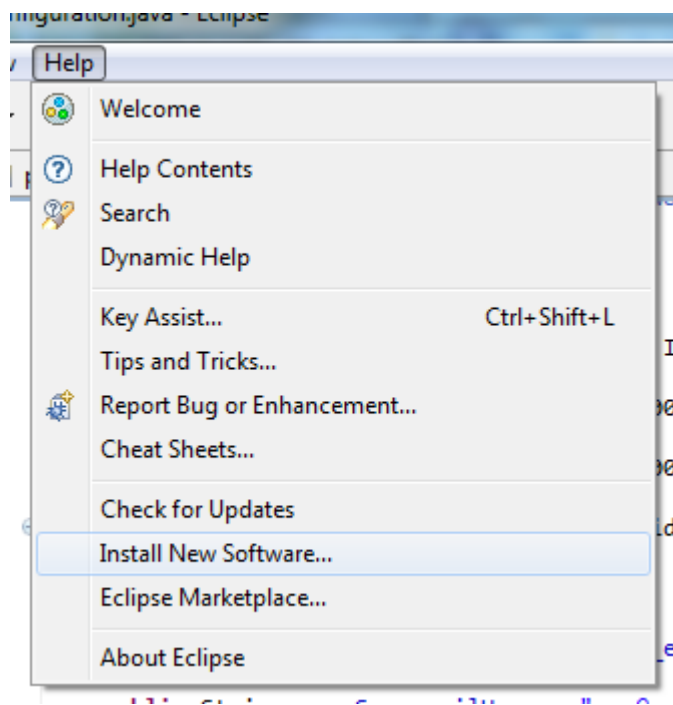
## Step 1: Sign Up for github

That's the easiest part, just go to <https://github.com/> and register!

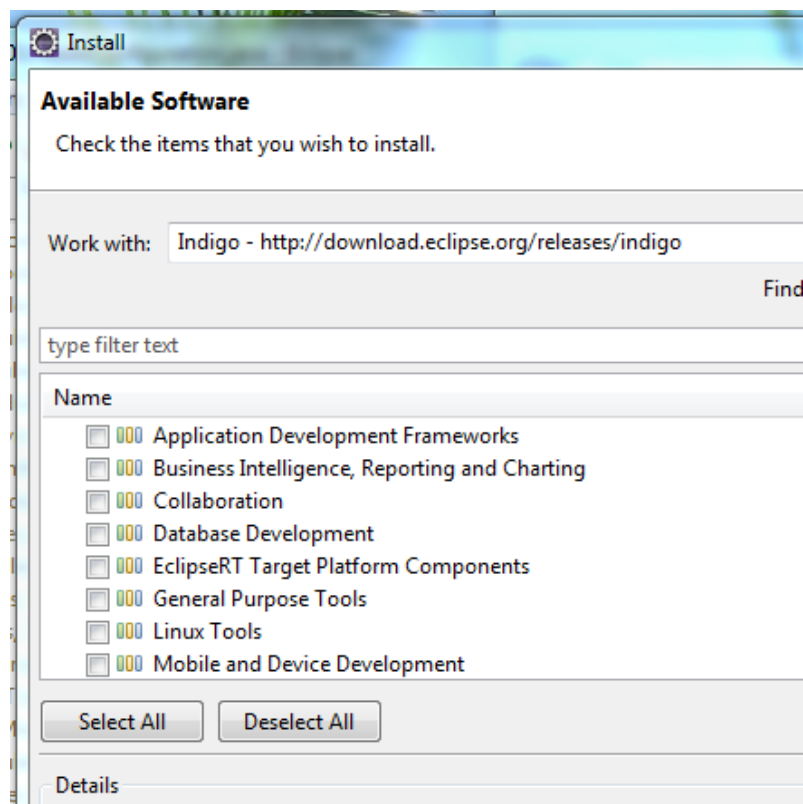
## Step 2: Installing EGit

You will need to install the git plugin for eclipse, EGit, in order to upload code from eclipse projects.

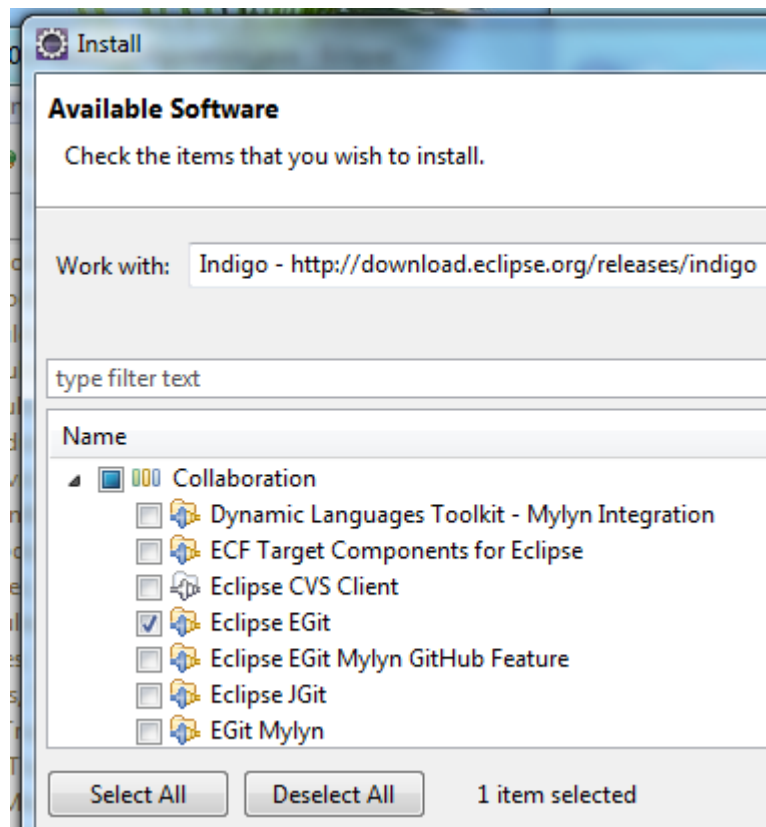
- In eclipse, go to Help / Install New Software



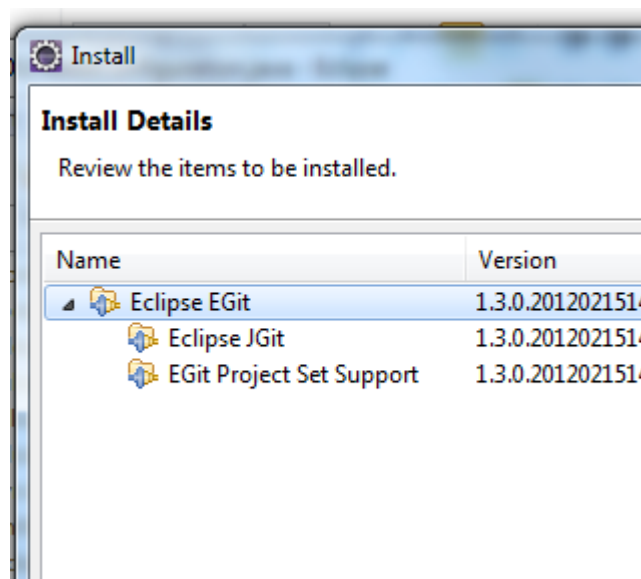
- Open the eclipse Indigo repository



- Select Collaboration / Eclipse EGit



- Click [Next]
- JGit should have been selected for you automatically



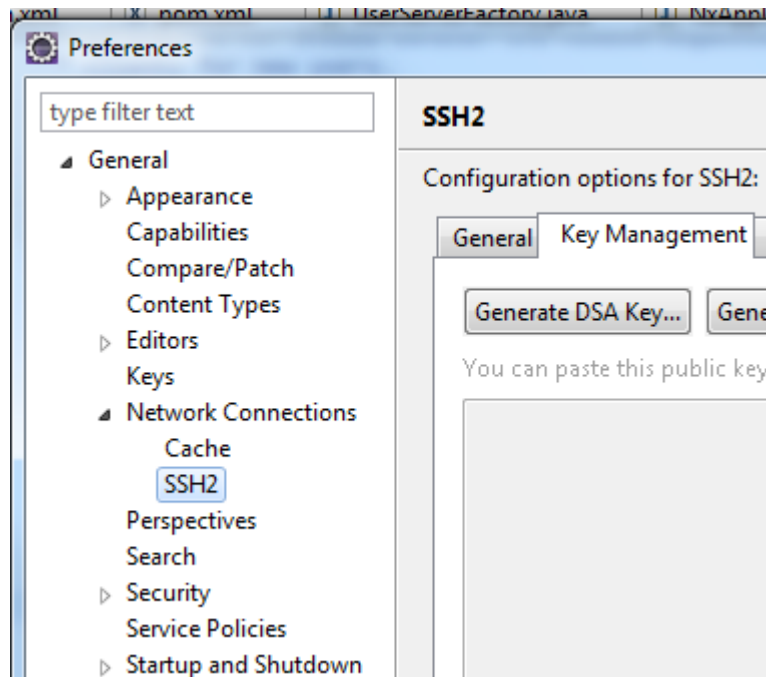
- Click [Next] and confirm the licence agreement
- Restart eclipse and the EGit plugin should be installed

## Step 3: Create a DSA Key in Eclipse

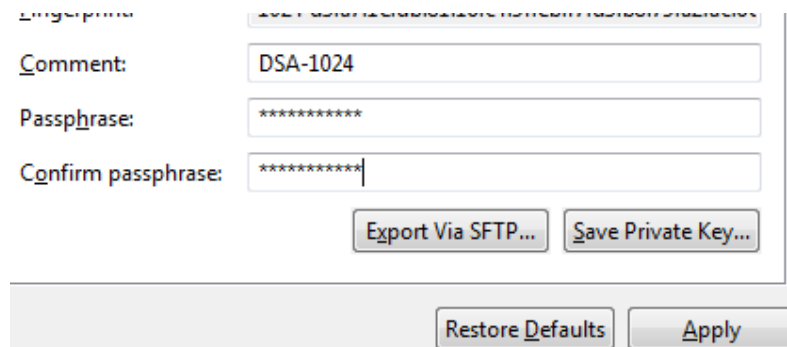
In order to be able to upload source code to github, you need to define a secure key, which must be known both to your local eclipse installation as well as the github

service. Luckily, eclipse provides all the tooling necessary to generate the appropriate key.

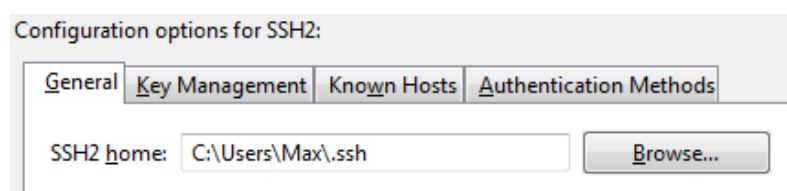
- Open Menu Window / Preferences
- In the preferences, go to General / Network Connections / SSH2
- On the SSH2 page, open the tab 'Key Management'



- Click on [Generate DSA Key ...]
- At the bottom of the page, enter a passphrase of your choosing

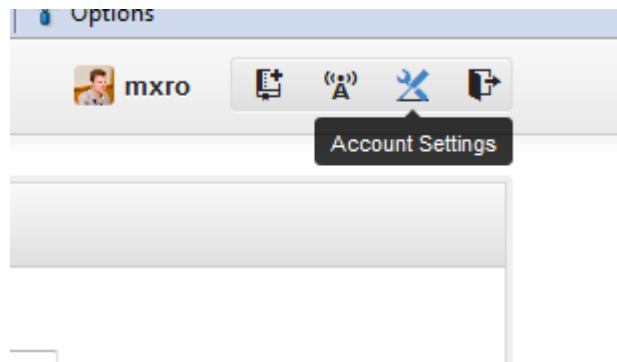


- Click [Save Private Key ...] (what's going on with these three dots in the button captions ... strange)
- Save the key at a location of your choosing (best in the location specified as your SSH2 home on under the tab General)

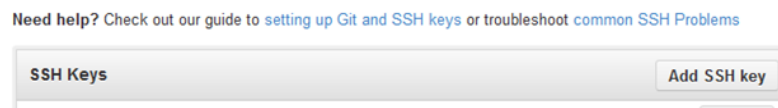


## Step 4: Register DSA Key with github

- Open the file you have saved in the previous step with a text editor (e.g. Notepad on windows)
- Select the contents of the file (Ctrl + A) and copy the complete text
- Go to the github website (<https://github.com>) and login
- On the top right of the screen, click on 'Account Settings'



- On the left hand side of the account settings, click on 'SSH Keys'



- Click on [Add SSH key]
- Provide an appropriate title for your key (e.g. 'EGit 1' ?)
- Paste the contents from the text file containing your DSA key into the text box 'Key'

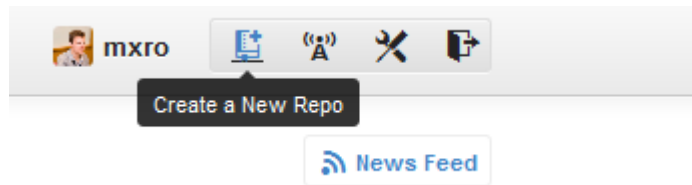
A screenshot of the 'Add an SSH Key' form. It has two input fields: 'Title' with the value 'EGit 3' and 'Key' with the value 'ssh-dss'. The 'Key' field has a red 'x' icon at the bottom right, indicating an error or warning.

- Click [Add Key] at the bottom of the form

## Step 5: Create Repository on github

In order to upload source code from a project in eclipse to github, you will need to create a github repository.

- Go to github homepage (<https://github.com/>) and log in
- At the top right corner, click on 'Create a New Repo'

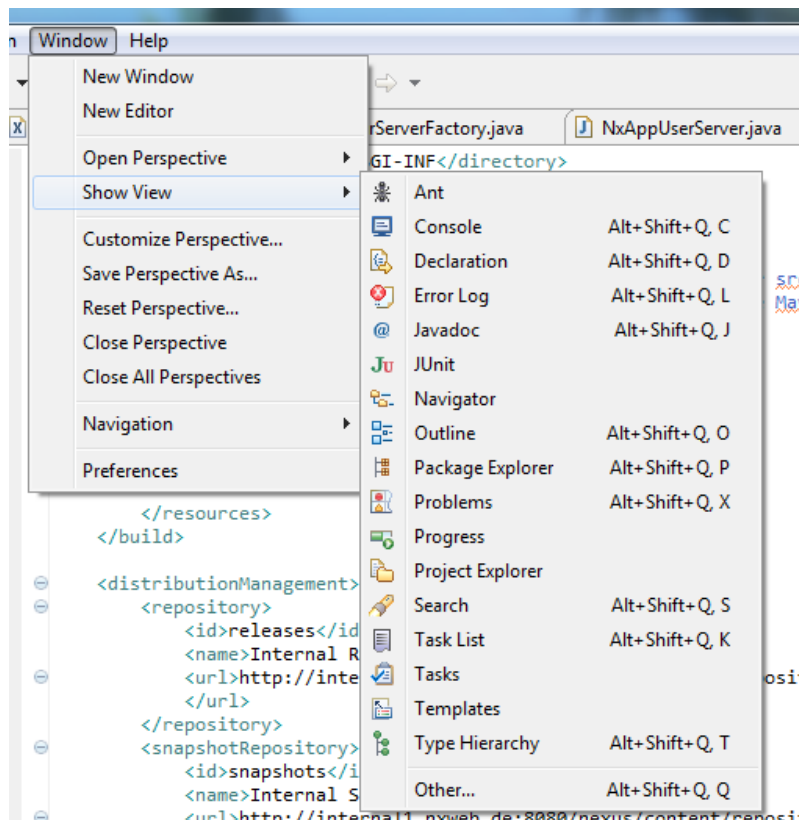


- Chose a repository name and description to your liking and click [Create Repository]

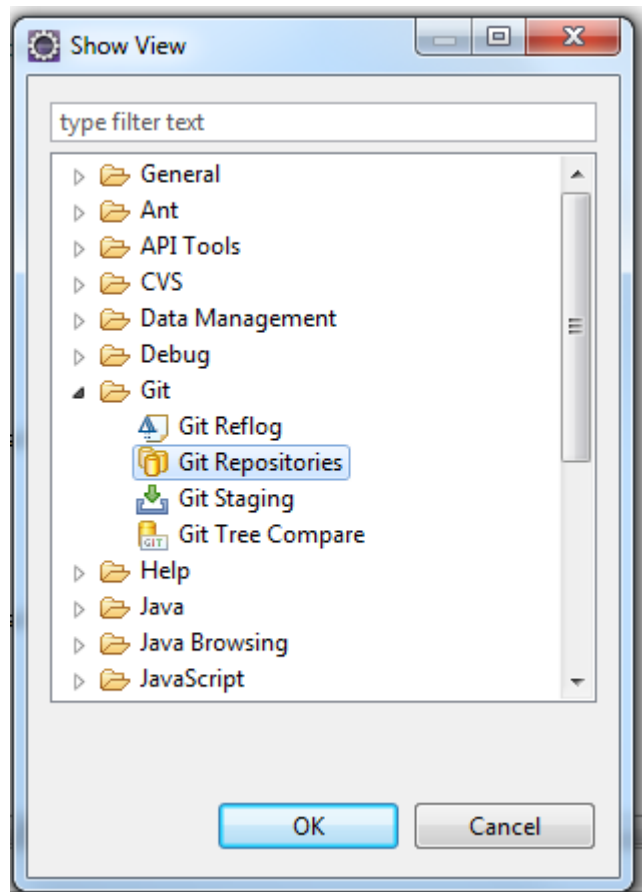
## Step 6: Import github Repository into eclipse

Before you can link an existing eclipse project to a github repository, you must import the repository you have created on github first. For this:

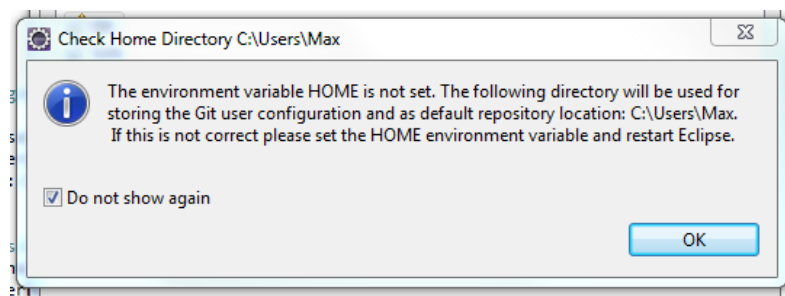
- In eclipse, open Menu / Window / Show View / Other ...



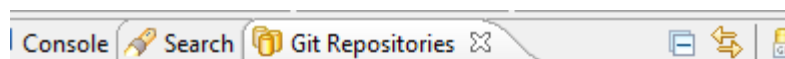
- Select Git / Git Repositories and click [Ok]



- You might see a warning message such as the one shown below (even setting the environment variable did not help me to get rid of the message, but everything seems to be working okay) – you can confirm with [Ok]



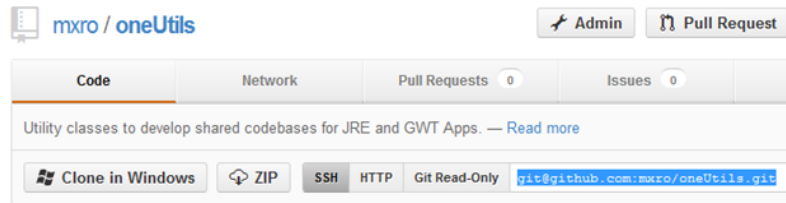
- You should have a new view 'Git Repositories' now
- Click on 'Clone a Git repository' within the new view



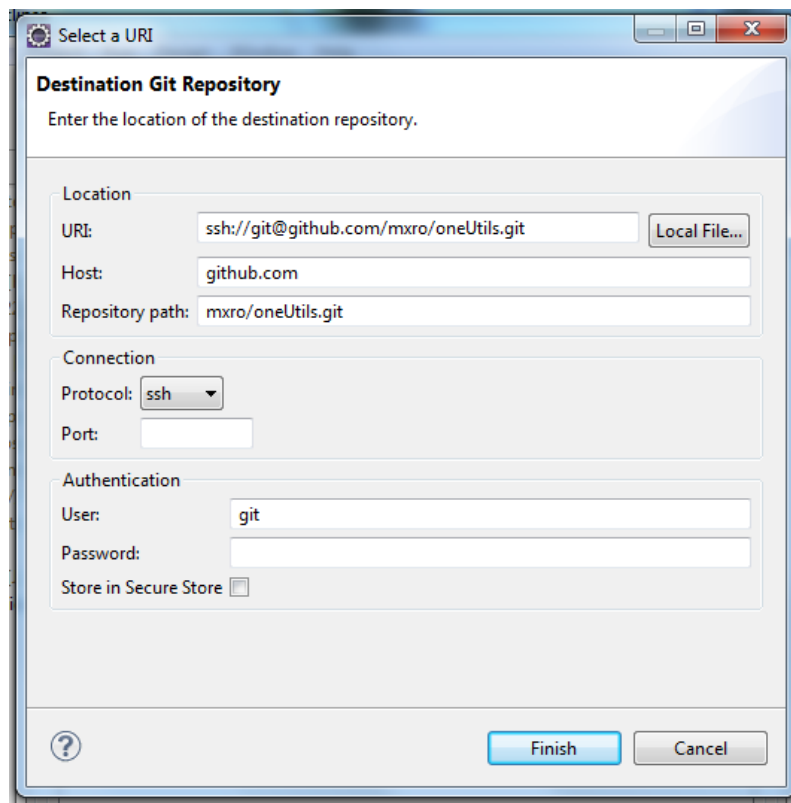
Select one of the following to add a repository to this view:

-  [Add an existing local Git repository](#)
-  [Clone a Git repository](#)
-  [Create a new local Git repository](#)

- Now go back to <https://github.com> and to your newly created github repository
- Under your repository description, you can get the URI for your project. Copy the text starting with 'git@' (make sure that SSH is selected)

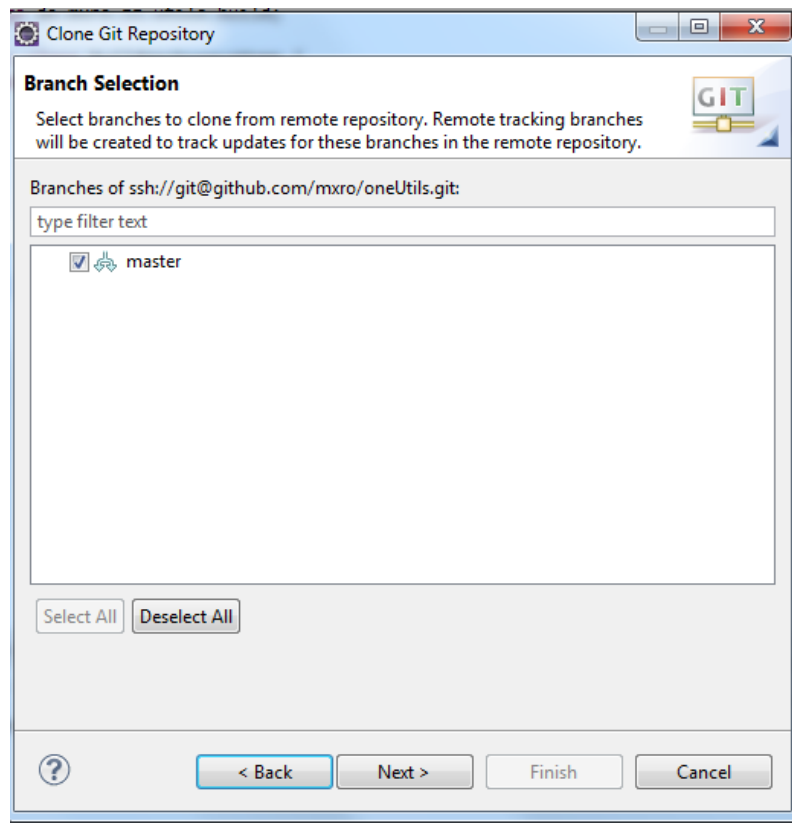


- Go back to eclipse. You can paste the URI you have just copied into the field 'URI'
- Further select as Protocol 'ssh'
- Click [Finish]



- If asked to select a branch, select the 'master' branch

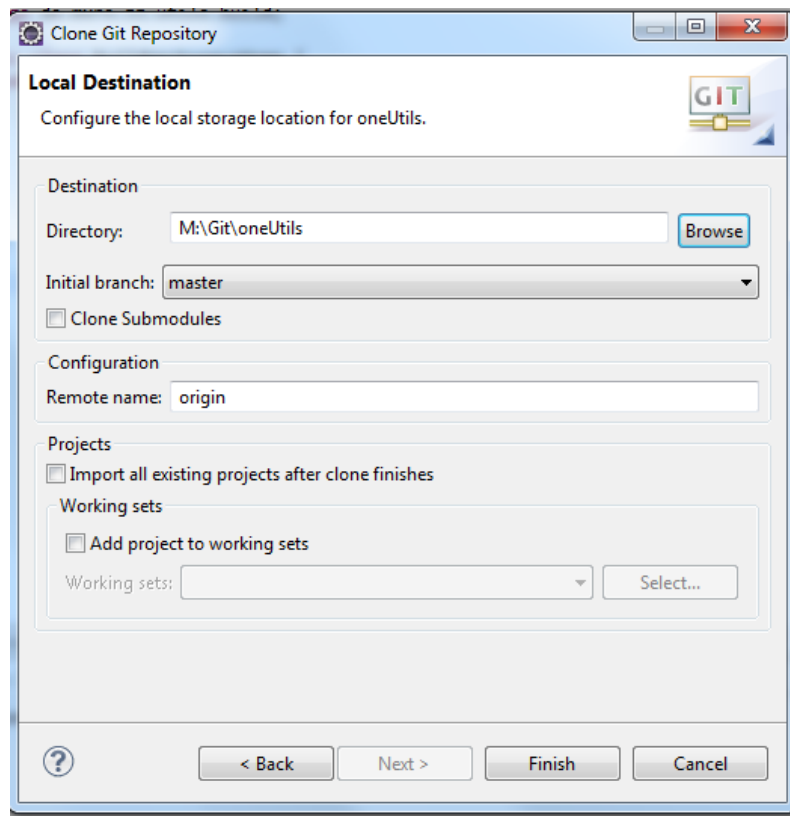




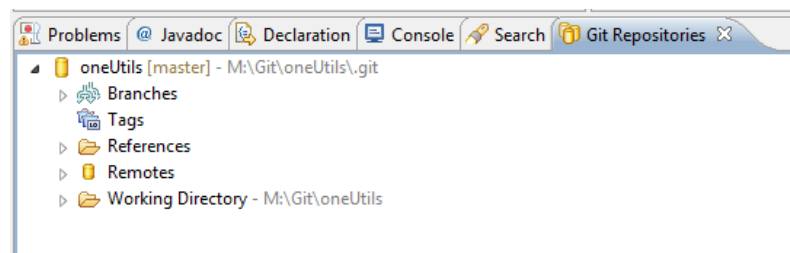
Git (in difference to subversion) allows storing a full blown repository on your local machine rather than just a local copy of the files. This requires to store all source you want to synchronize with git at least **twice** on your local machine: one copy will be stored in the clone of the remote git repository and another one will be stored in your eclipse project.

Hence, when you close the git repository from github, you should define a repository location, which lies outside the eclipse project you want to upload:

- Select one such location and click [Finish]



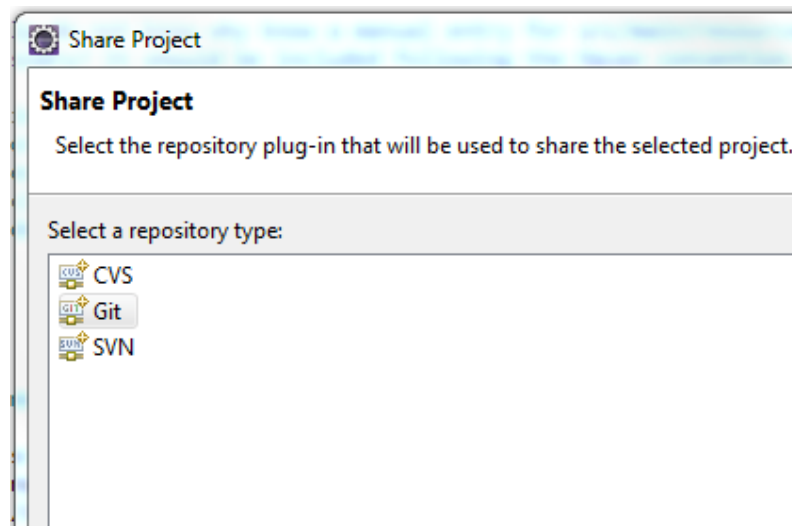
- Now you should have one 'Git Repository'



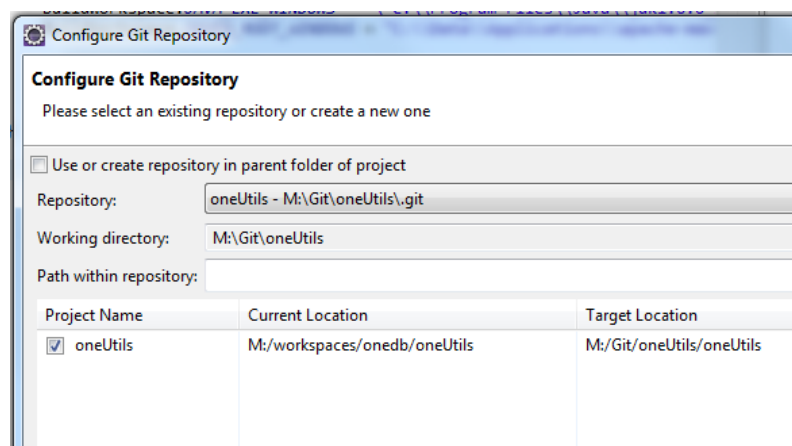
## Step 7: Link Eclipse Project with github Repository

After you have created a local clone of the repository from github, you can link the eclipse project you would like to upload to this local repository.

- Right click your eclipse project and select Team / Share Project ...
- Select 'Git' as Repository Type



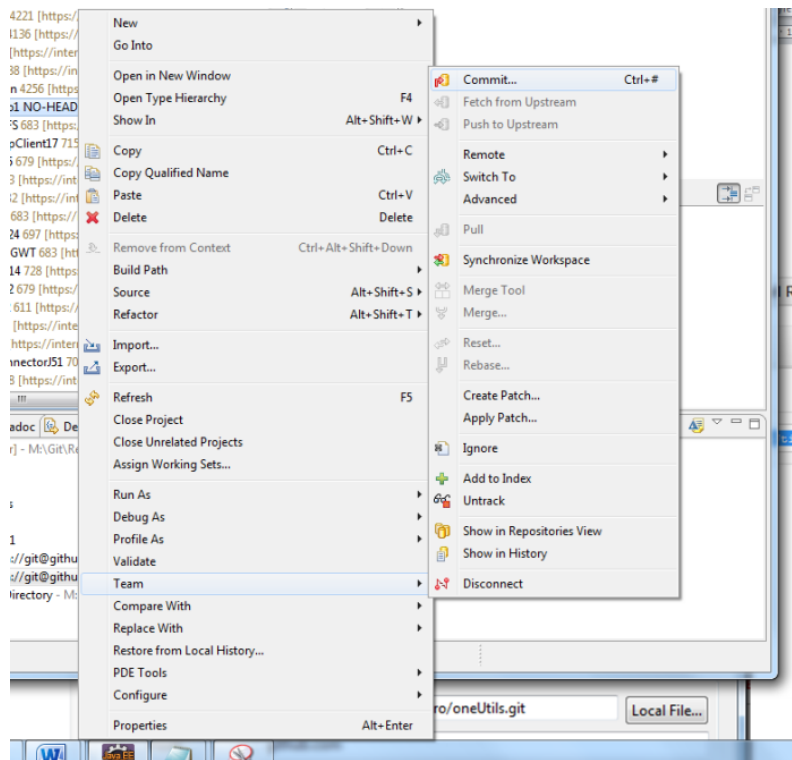
- Select under 'Repository' the repository you have cloned in the previous step and click [Finish]



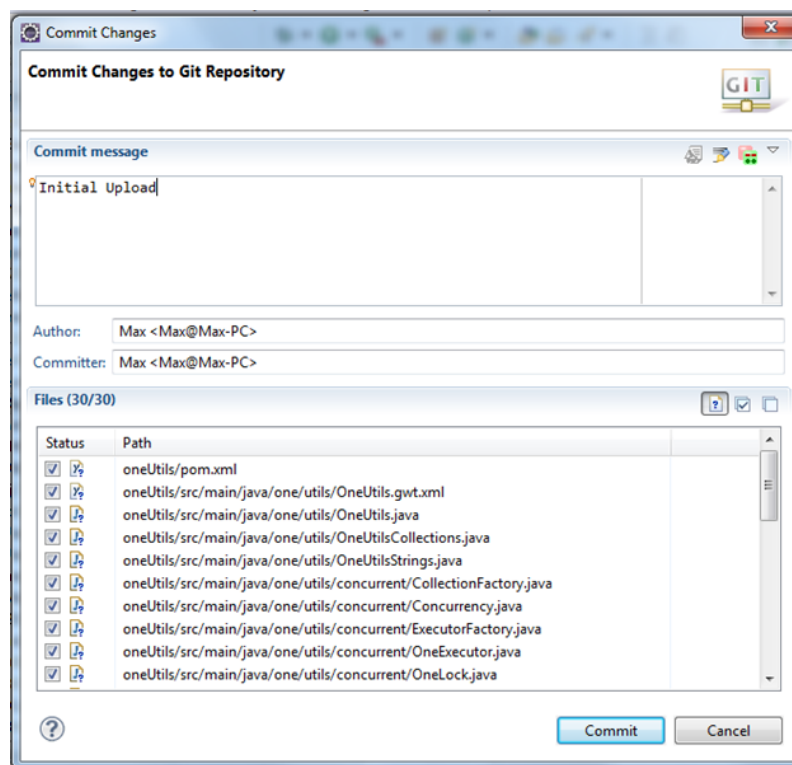
## Step 8: Uploading Project Sources to github

After you have linked your project with the local clone of the github repository, you can 'Commit' all the source files in your existing project to this repository. After you have committed the files to your local repository, you can 'Push' them to the github repository.

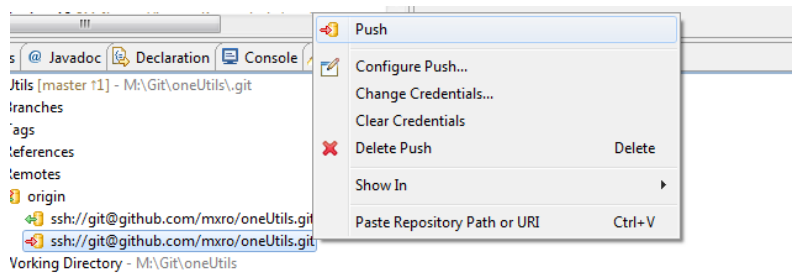
- Right click your project and select Team / Commit ... from the popup menu



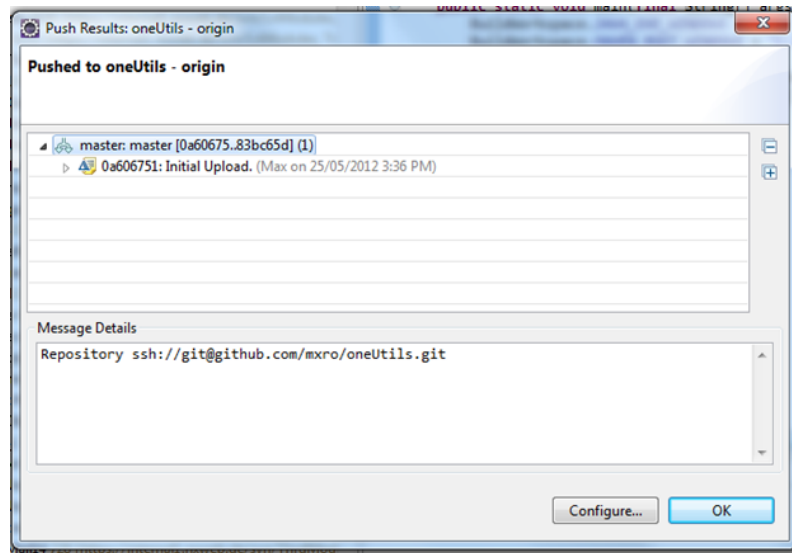
- Write a beautiful commit message and click [Commit]



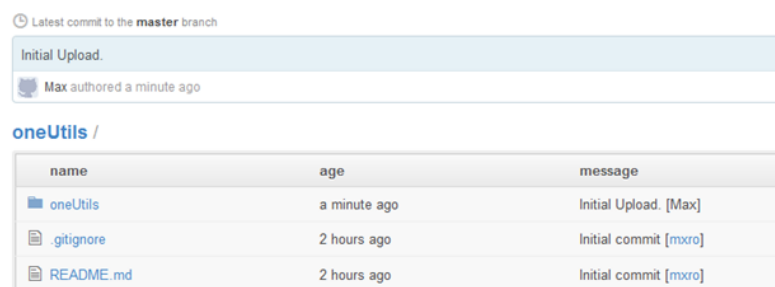
- In the 'Git Repositories' view, open <your repository> / Remotes / origin
- Right click on the second icon (with the red arrow) and select 'Push'



- You should receive the results of the push, click [Ok] to confirm



- You can now go to github and your project sources should be displayed in your repository:



(hmm, did it really took 2 hrs to get this done ...)

## References

[Git with Eclipse \(EGit\) – Tutorial \(vogolla.com\)](#)

[Getting Started with Git, EGit, Eclipse, and GitHub: Version Control for R Projects](#)

[git push rejected \(stackoverflow.com\)](#)

[A Short Tutorial on Eclipse/EGit/GitHub](#)



# Git version control with Eclipse (EGit) - Tutorial

Lars Vogel (c) 2009, 2016 vogella GmbH – Version 4.2, 06.07.2016

---

## Table of Contents

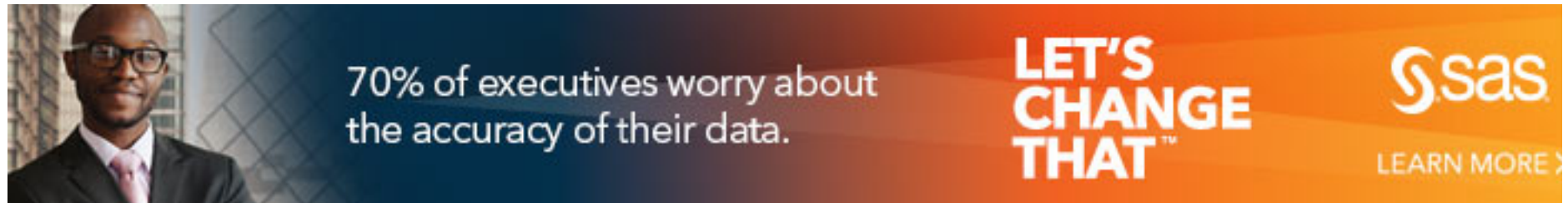
1. Git support for Eclipse
2. Command line Git
3. Installation of Git support into Eclipse
4. How to configure the usage of Git in Eclipse
5. Using the Git Repositories view
6. Git integration into the Package and the Project Explorer
7. Using the Git Staging view
8. Using the Git commit dialog
9. Using the History view
10. Working with Eclipse projects in a Git repository
11. Clone an existing repository
12. Import projects from an existing repository
13. Performing Git operations in Eclipse
14. Branching in Eclipse
15. Starting a merge operation in Eclipse

16. Rebasing a branch onto another branch
  17. Git reset and Git reflog
  18. Using git cherry-pick
  19. Creating patches
  20. See Git information line by line (aka git blame)
  21. Stash via the Git repository view
  22. Adjusting the history with interactive rebase
  23. Using Eclipse Git with GitHub
  24. Eclipse support for SSH based authentication
  25. Eclipse integration with GitHub
  26. Writing good commit messages
  27. Exercise: Working with a local Git repository in Eclipse
  28. Contributing to EGit - Getting the source code
  29. About this website
  30. Eclipse Git Resources
- Appendix A: Copyright and License
- 

*Git with Eclipse (EGit). This tutorial describes the usage of EGit; an Eclipse plug-in to use the distributed version control system Git. This tutorial is based on Eclipse 4.5 (Mars).*



GET THE BOOK!



# 1. Git support for Eclipse

The Eclipse IDE has excellent support for the Git version control system. This support is provided by the *EGit* project via a set of plug-ins (software component).

Eclipse uses the *JGit* library to perform the Git commands. JGit is a library which implements the Git functionality in Java.

The Eclipse Git user guide is bundled with the Eclipse Git installation. You can invoke it via



### *The Eclipse workspace and Git repositories*

It is good practice to place your Git repositories outside the Eclipse workspace. This separates your Git repository from any additional meta-data which Eclipse might create. By default, Eclipse Git uses the *git* folder in the users home directory to clone new repositories. This default location can of course be adjusted, see [Default clone location](#) for more information on this.

---

## 2. Command line Git

This tutorial describes the usage of *EGit*. If you want to learn about the usage of the Git command line, you can use the [Git Tutorial](#) as a reference.

This tutorial also explains the basic Git terminology, e.g., what is a commit, branch, etc.

---

## 3. Installation of Git support into Eclipse

Most Eclipse IDE distributions from Eclipse.org already contain support for Git. In this case

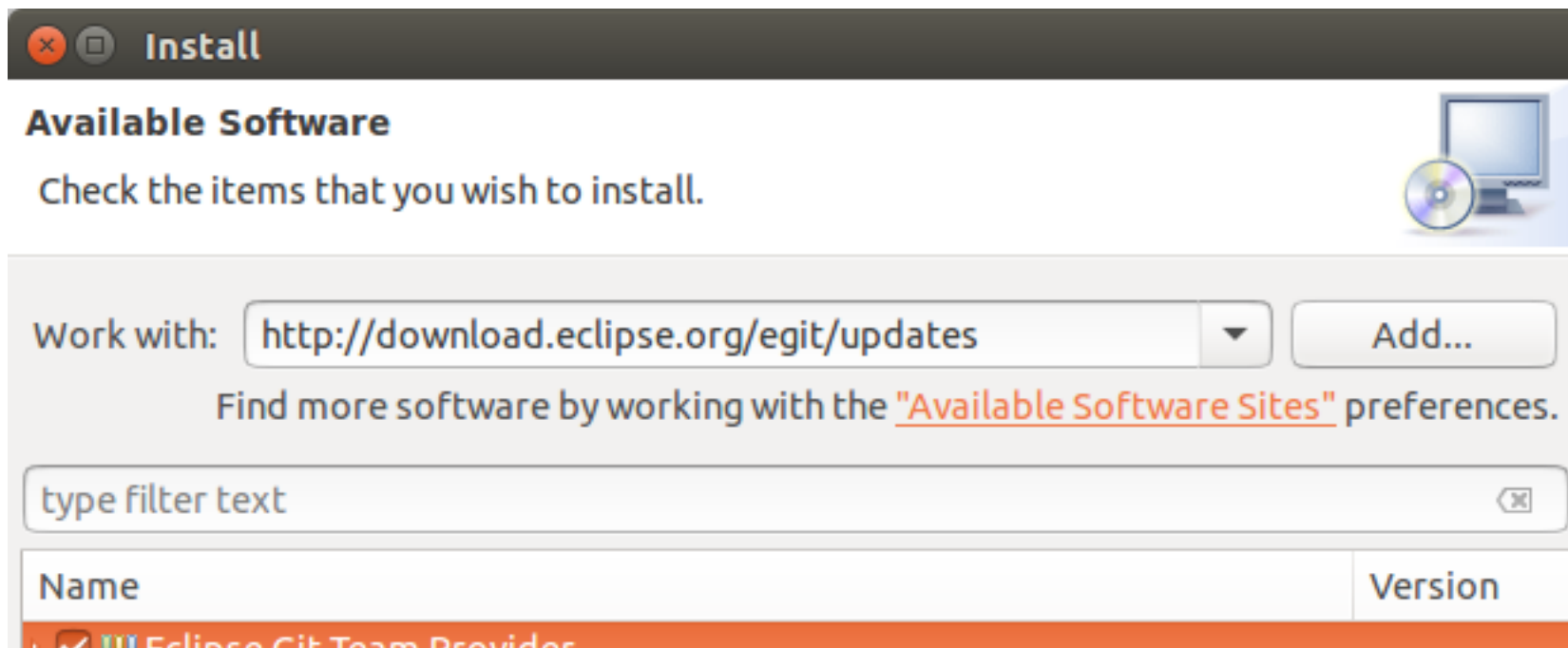
no additional installation is required.

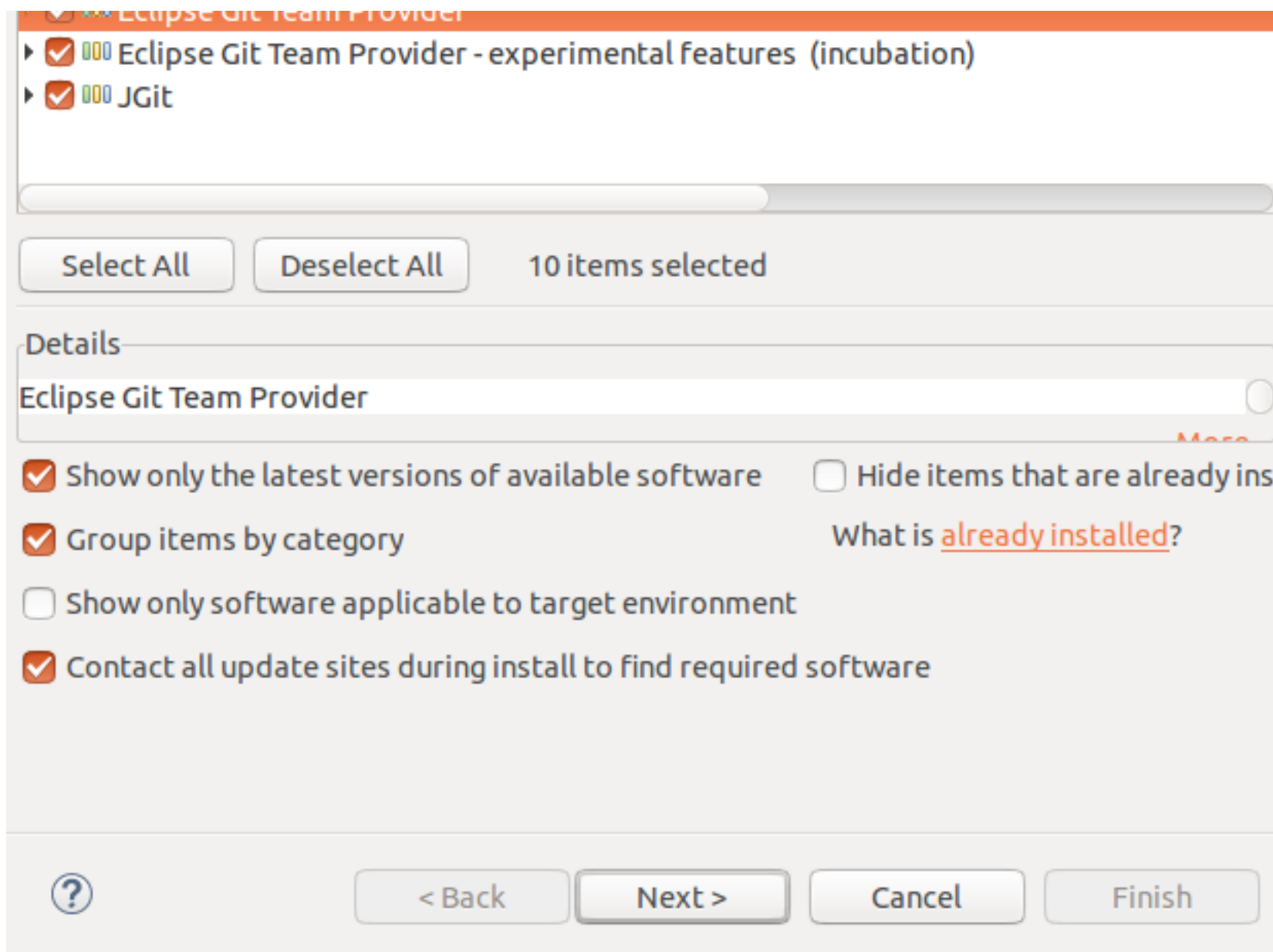
Otherwise you can install it via the Eclipse installation manager. Select the **Help** → **Install new Software** menu entry. Enter one of the following update site URLs:

```
# Use this update site to get the latest release
http://download.eclipse.org/egit/updates

# use this update site to get the night build
http://download.eclipse.org/egit/updates-nightly/
```

The dialog to install the Eclipse Git team provider is depicted in the following screenshot.





## 4. How to configure the usage of Git in

# Eclipse

## 4.1. Interoperability of Git command line settings with the Eclipse IDE

The Git functionality in the Eclipse IDE uses the same configuration files as the Git command line tools. This makes it easier to use the Eclipse Git tooling and the command line tooling for Git interchangeable. One notable exception is currently the support of gitattributes. See [Bug 342372 - support gitattributes](#) for details.

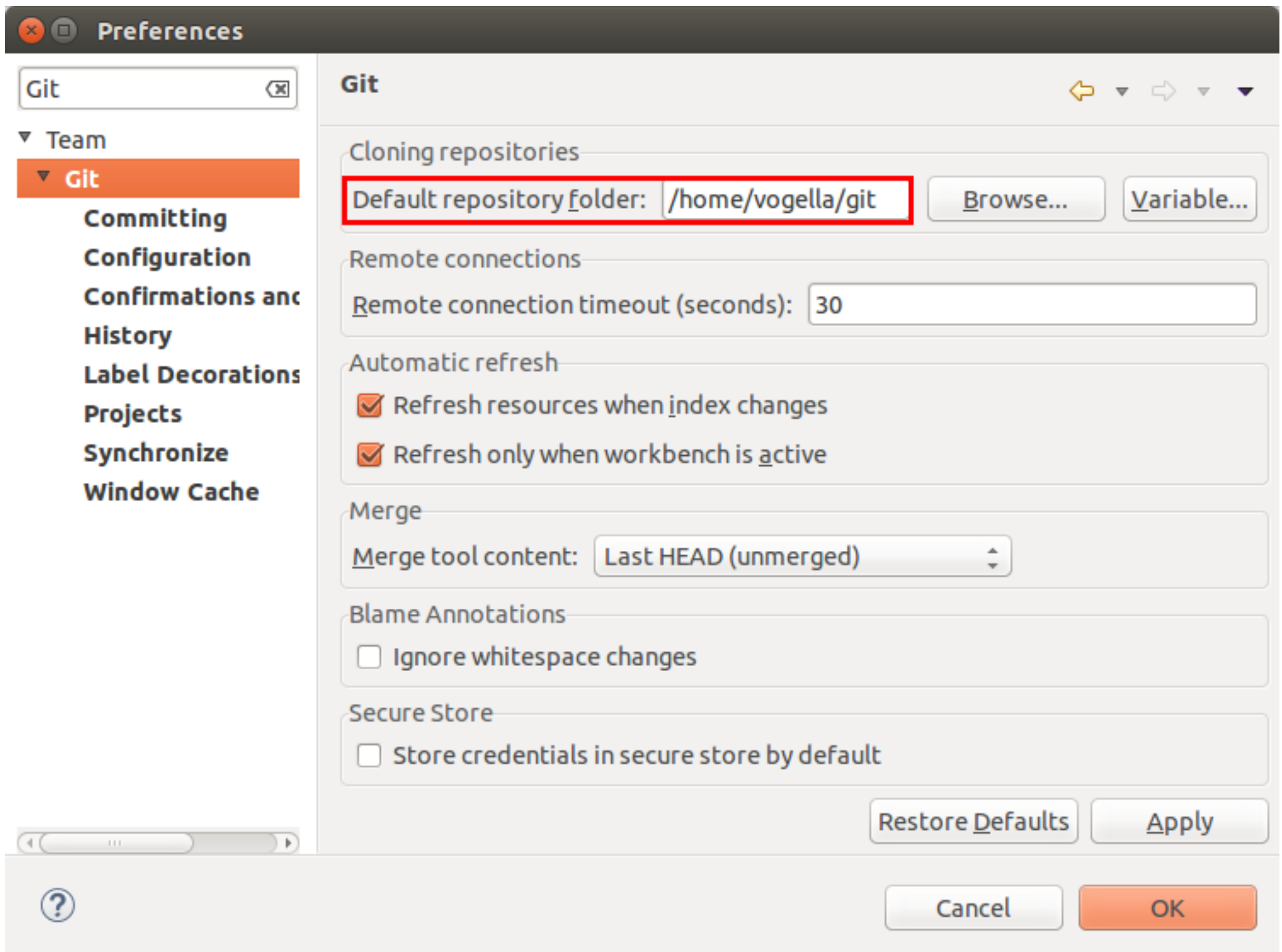
## 4.2. Git user settings in Eclipse

To use Git you must configure your full name and email address. This information is used to fill the author and committer information of commits you create. These Git configuration settings can be adjusted via the Eclipse preference setting. Select Window ▸ Preferences ▸ Team ▸ Git ▸ Configuration to see the current configuration and to change it.

See [\[eclipsegit userconfiguration\]](#) for a detailed description how to configure Git via Eclipse.

## 4.3. Default clone location

If you clone a new repository via Eclipse Git, it will create by default a new sub-folder for the new Git repository in a default directory. This default path can be configured via the Windows ▢ Preferences ▢ Team ▢ Git entry in the *Default Repository folder* field.



You can also use Eclipse configuration variables to define this path, e.g., if you want to store repositories in the folder "git" under the Eclipse workspace you may use `${workspace_loc}/git`.

## 4.4. Configuring the toolbar and the menu for Git usage

To simplify access to the common Git operations you can activate the Git toolbar. For this select `Window` `▢` `Perspective` `▢` `Customize perspective` `▢▢` and check the `Git` and `Git Navigation Actions` entries in the `Action Set Availability` tab.



## Customize Perspective - Plug-in Development










Tool Bar Visibility | Menu Visibility | **Action Set Availability** | Shortcuts

Select the action sets that you want to see added to the current perspective (Plug-in Development). The details field identifies which menu items and/or toolbar items are added to the perspective by the selected action set.












Available action sets:

- ☒ Annotation Navigation
- ☐ Ant Editor Presentation
- ☒ Breakpoints
- ☒ Cheat Sheets
- ☒ Convert Line Delimiters
- ☐ CVS
- ☐ Debug
- ☒ Editor Navigation
- ☐ Editor Presentation
- ☒ External Tools
- ☒ **Git**
- ☒ Git Navigation Actions
- ☐ Java Coding
- ☐ Java Debug
- ☐ Java Editor Presentation
- ☒ Java Element Creation
- ☒ Java Navigation
- ☐ Java Open Actions
- ☐ Java Search
- ☐ Unit

Menubar details:

- ▼  Git
  -  Add to Index
  -  Commit...
  -  Switch to ...
  -  Merge...
  -  Reset...
  -  Push to Upstream
  -  Fetch from Upstream
  -  Pull

Toolbar details:

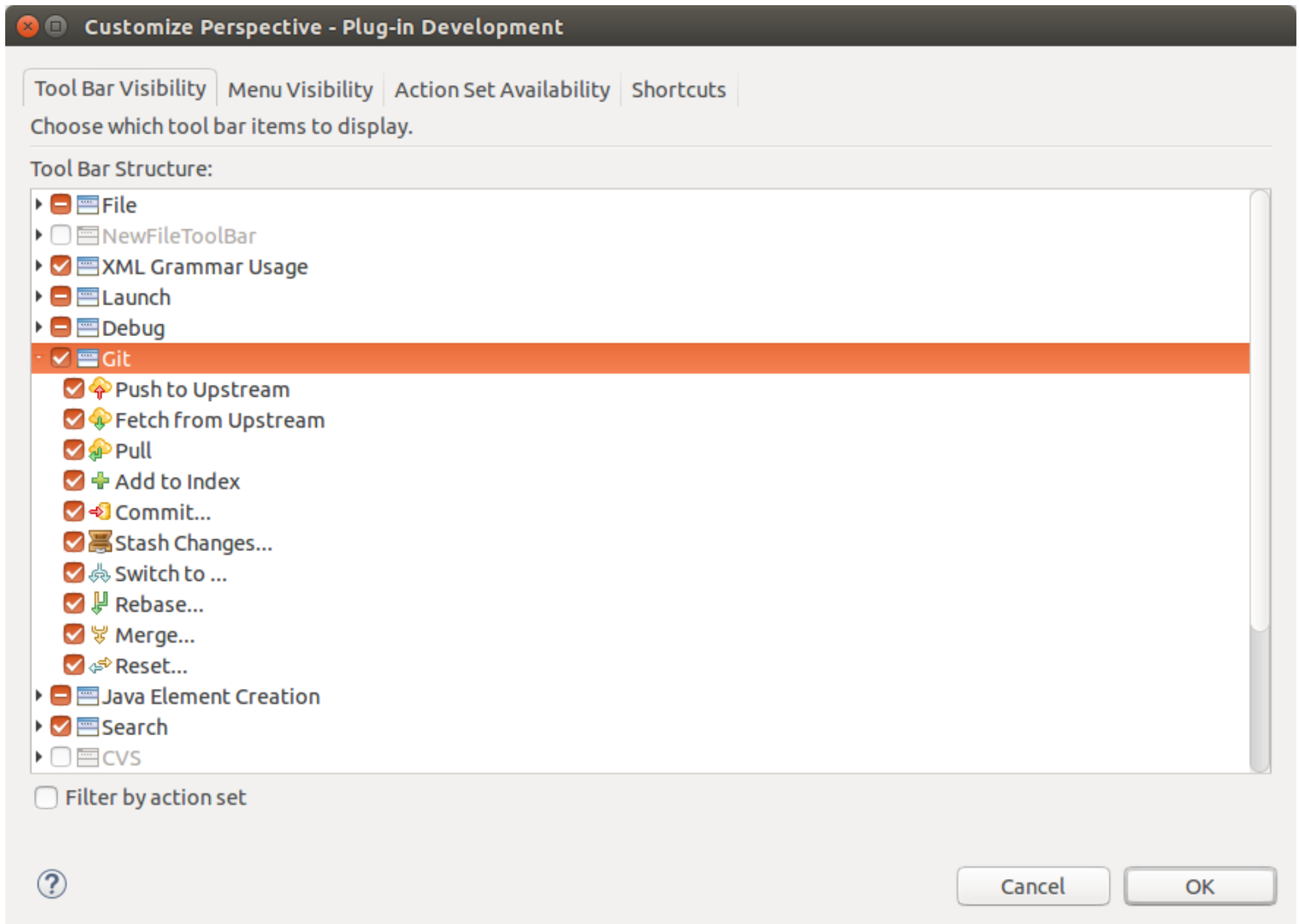
- ▼  Git
  -  Push to Upstream
  -  Fetch from Upstream
  -  Pull
  -  Add to Index
  -  Commit...
  -  Stash Changes...
  -  Switch to ...
  -  Rebase...
  -  Merge...
  -  Reset...



Cancel

OK

Afterwards you can configure which Git operations should be available via the *Tool Bar Visibility* or the *Menu Visibility* tab.



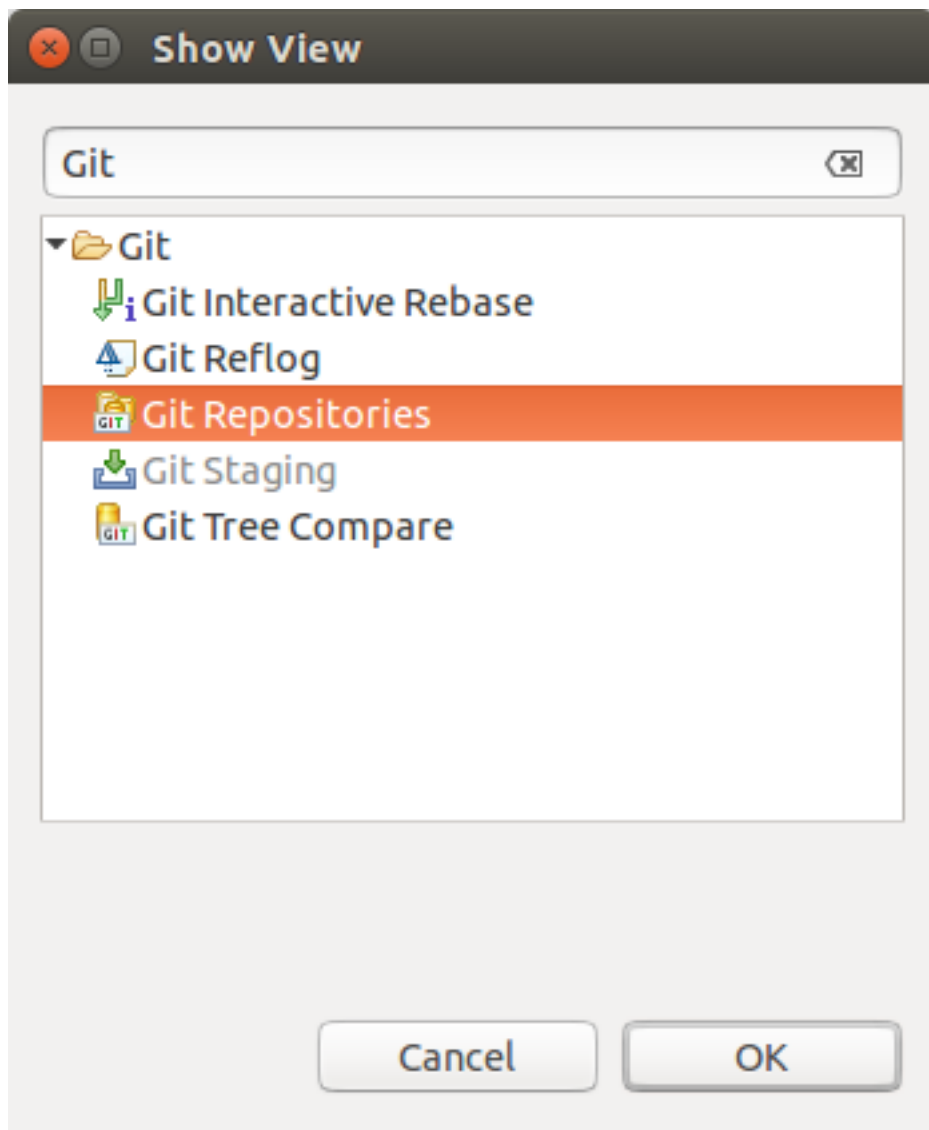


---

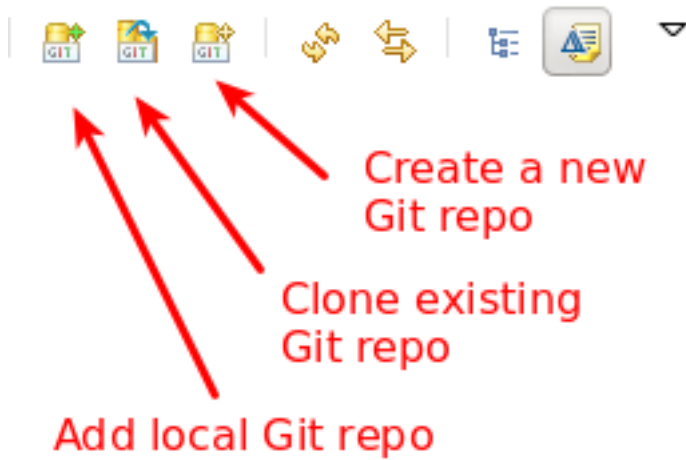
## 5. Using the Git Repositories view

### 5.1. Using the Git Repositories view

Eclipse Git provides the *Git Repositories* view which allows you to browse your repositories, add or initialize local repositories or clone remote repositories, checkout projects, manage your branches and much more. You can open this view via **Window** ▸ **Show View** ▸ **Other...** ▸ **Other...** ▸ **Git** ▸ **Git Repositories**





The toolbar entries allow you to add an existing local Git repository to the view, clone a Git repository and to create a new Git repository.

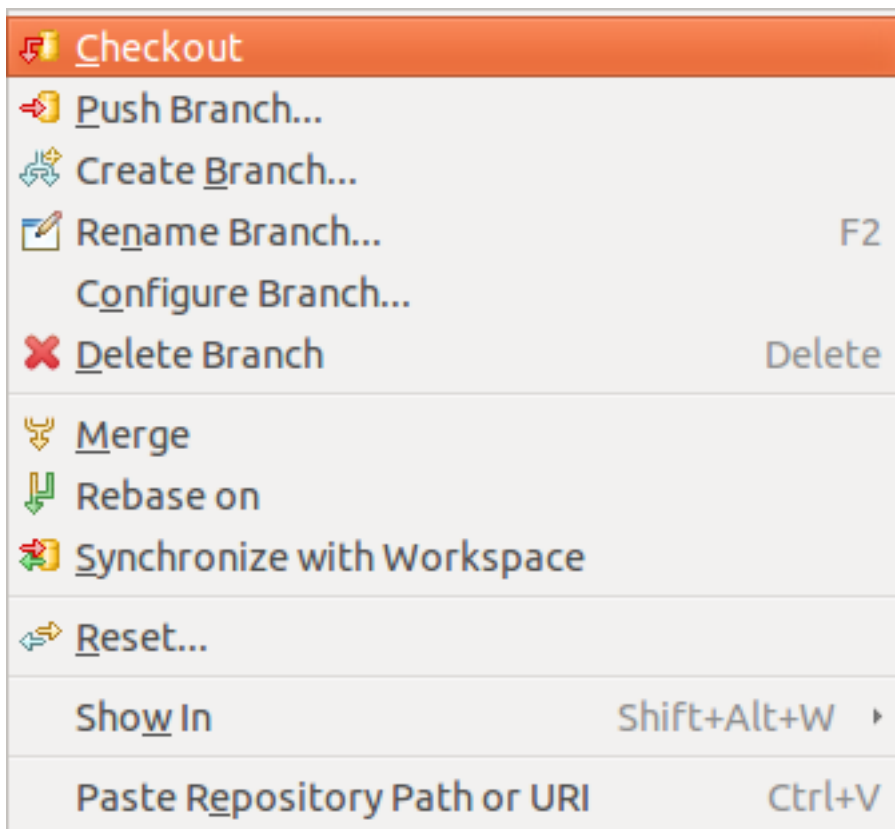


## 5.2. Content in the Git Repositories view

The content area of the *Git Repositories* view shows the existing Git repositories and the important data of each repository. The following screenshot shows an example Git repository.

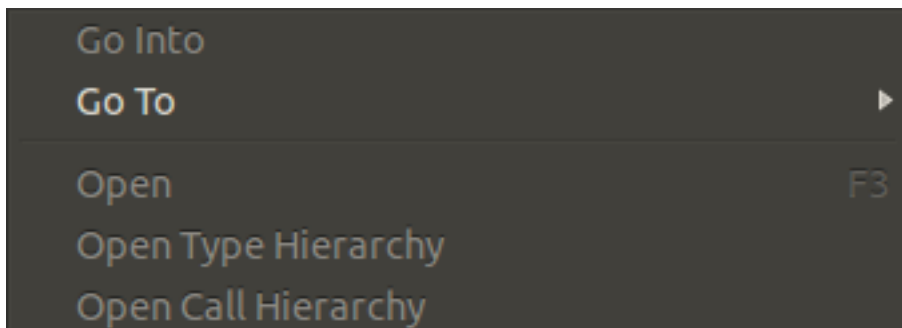
- ▼  com.vogella.tutorials [eclipse44] - /home/vogella/workspace/docu/com.vogella.tutorials/.git
  - ▼  Branches
    - ▶  Local
    - ▶  Remote Tracking
    - ▶  Tags
    - ▶  References
  - ▼  Remotes
    - ▶  origin
  - ▶  Stashed Commits
  - ▶  Working Directory - /home/vogella/workspace/docu/com.vogella.tutorials

A right-click (context menu) on an element in the *Git repositories* view allows you to perform related Git operations. For example if you right-click on a branch you can checkout the branch or delete it.

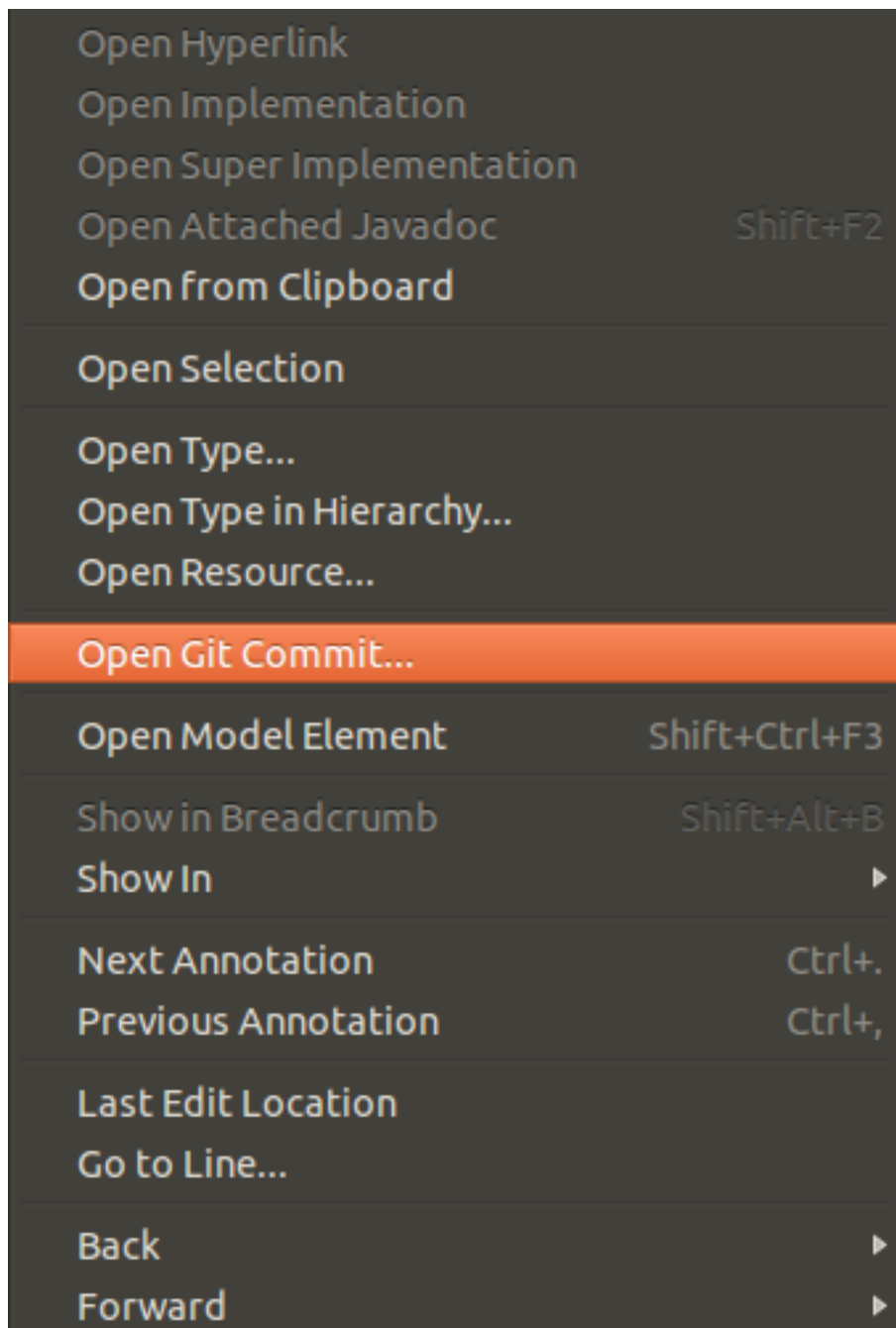


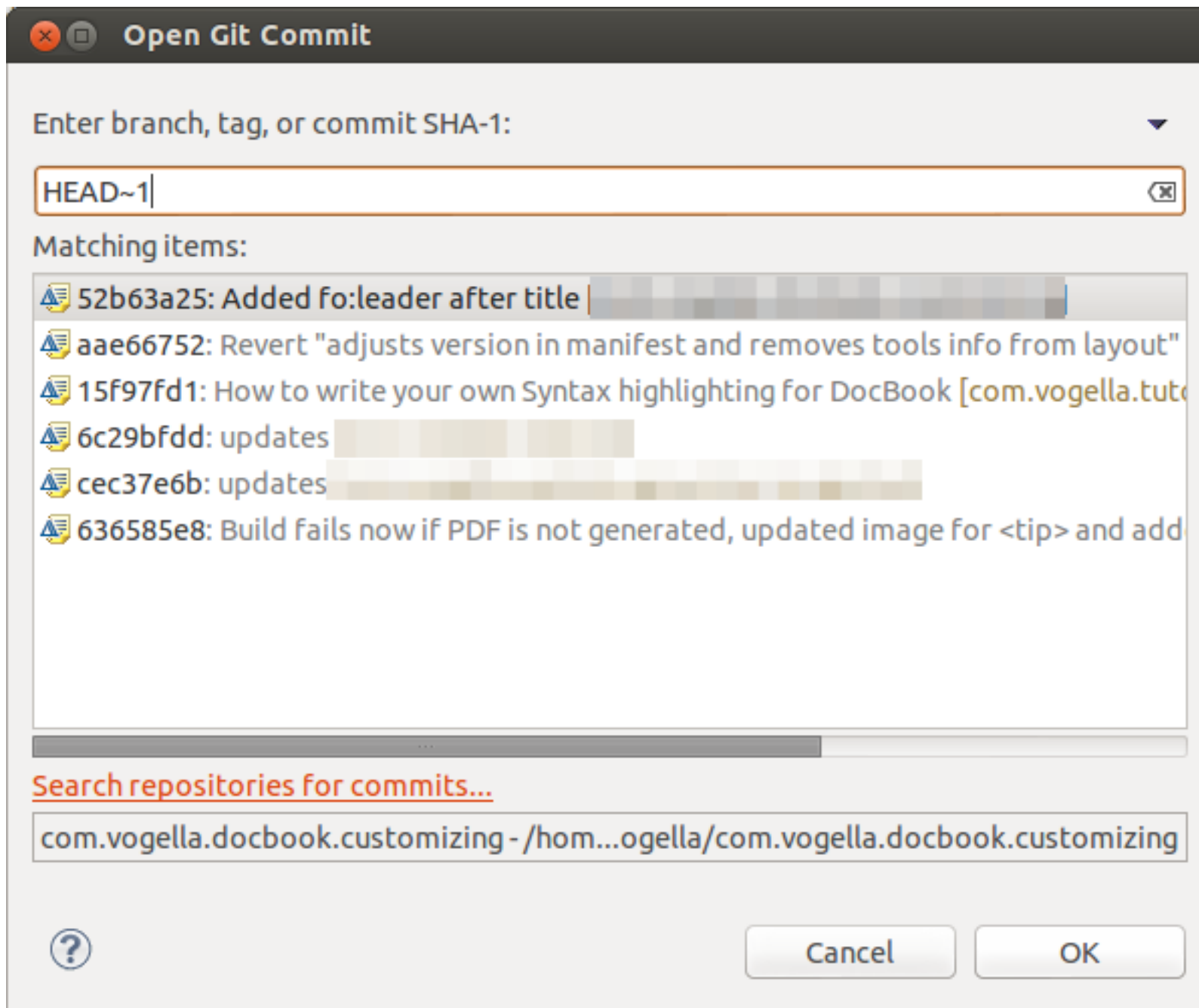
## 5.3. Open a commit

If you are in the *Git repositories* view you can open a commit via the main Eclipse menu. To do this select the **Navigate** ▸ **Open Git Commit** menu entry.



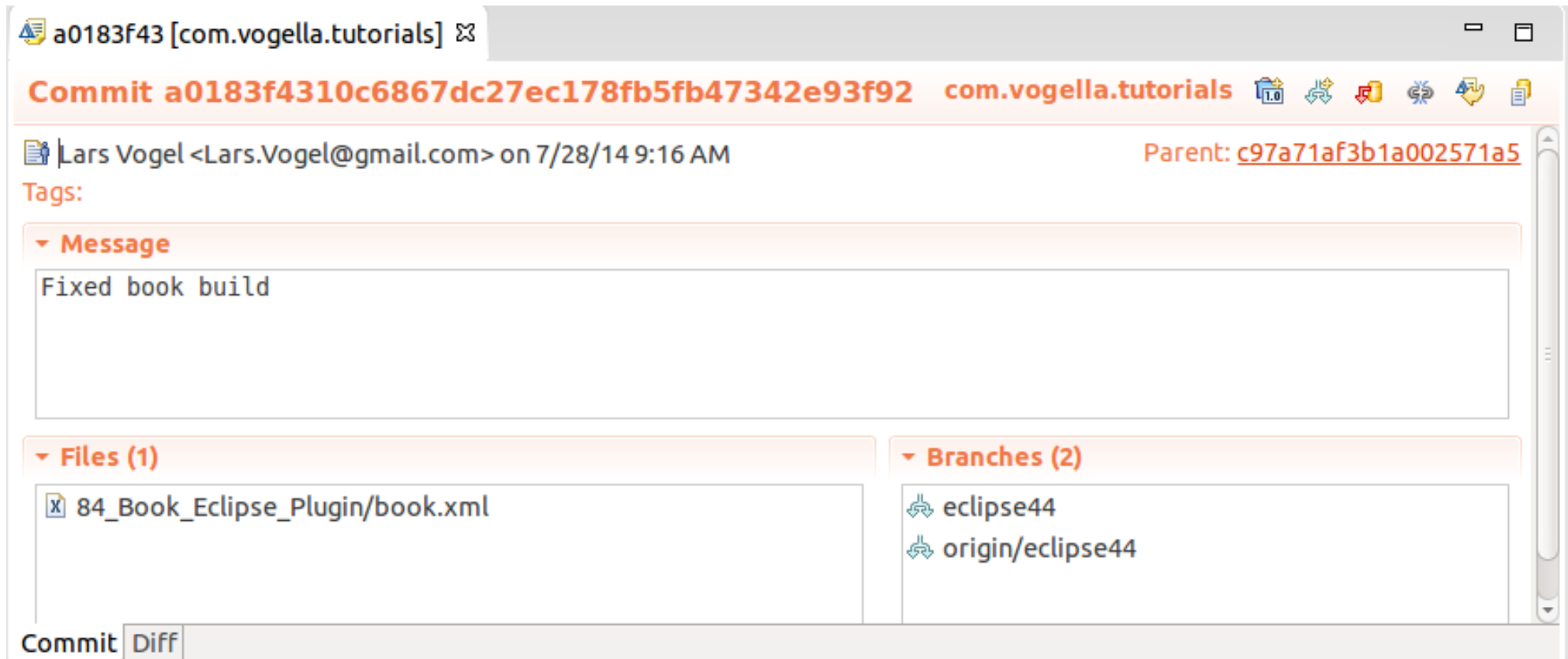






## 5.4. Commands available in the Commit Viewer

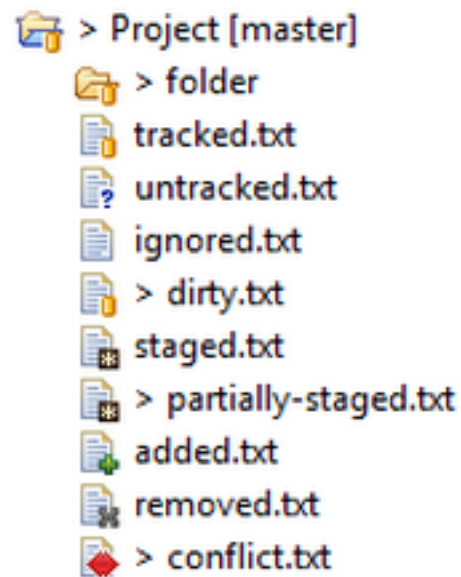
If you open a commit you can create a tag or branch from it. You can also revert it, cherry pick it or check it out. You can also reveal it in the history view.



## 6. Git integration into the Package and the Project Explorer

The *Package Explorer* view shows indicators on the files to show their status. The most

important icon decorators are depicted in the following screenshot.



The file name describes the state of the file from the following table:

*Table 1. Git label decorations*

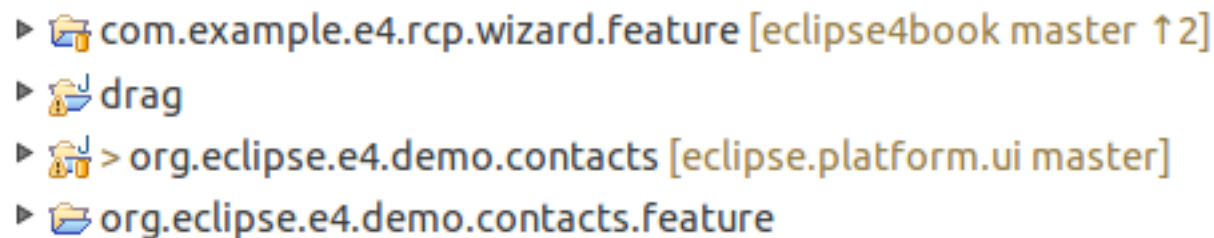
State	Description
tracked	File is committed to the Git repository and has not changed.
untracked	File is neither staged nor committed.
ignored	File is flagged to be ignored by Git operations.

dirty	File has changed since the last commit.
staged	Changes in the file will be included in the next commit.
partially-staged	The resource has changes which are added to the index and additional unstaged changes in the working tree
added	Staged but not yet committed, i.e. snapshot of this file has been stored in the git database. This status is the same as the <i>staged</i> status, but the file wasn't under Git version control before.
removed	The resource is staged for removal from the Git repository.
conflict	A merge conflict exists for the file.

A combination of the staged and dirty status means: some parts of the changed file have been staged while some are still unstaged. This can happen if you stage a file and then again modify the file before recreating the next commit. You can also change the staged parts using the compare

editor opened by double clicking files in the stagingview.

On a project level the explorerview adds the information which Git repository is used to the project name. It also adds the number of commits that are different between local and remote tracking branch. This way you can quickly see if your local branch is ahead or behind the remote branch it is tracking.



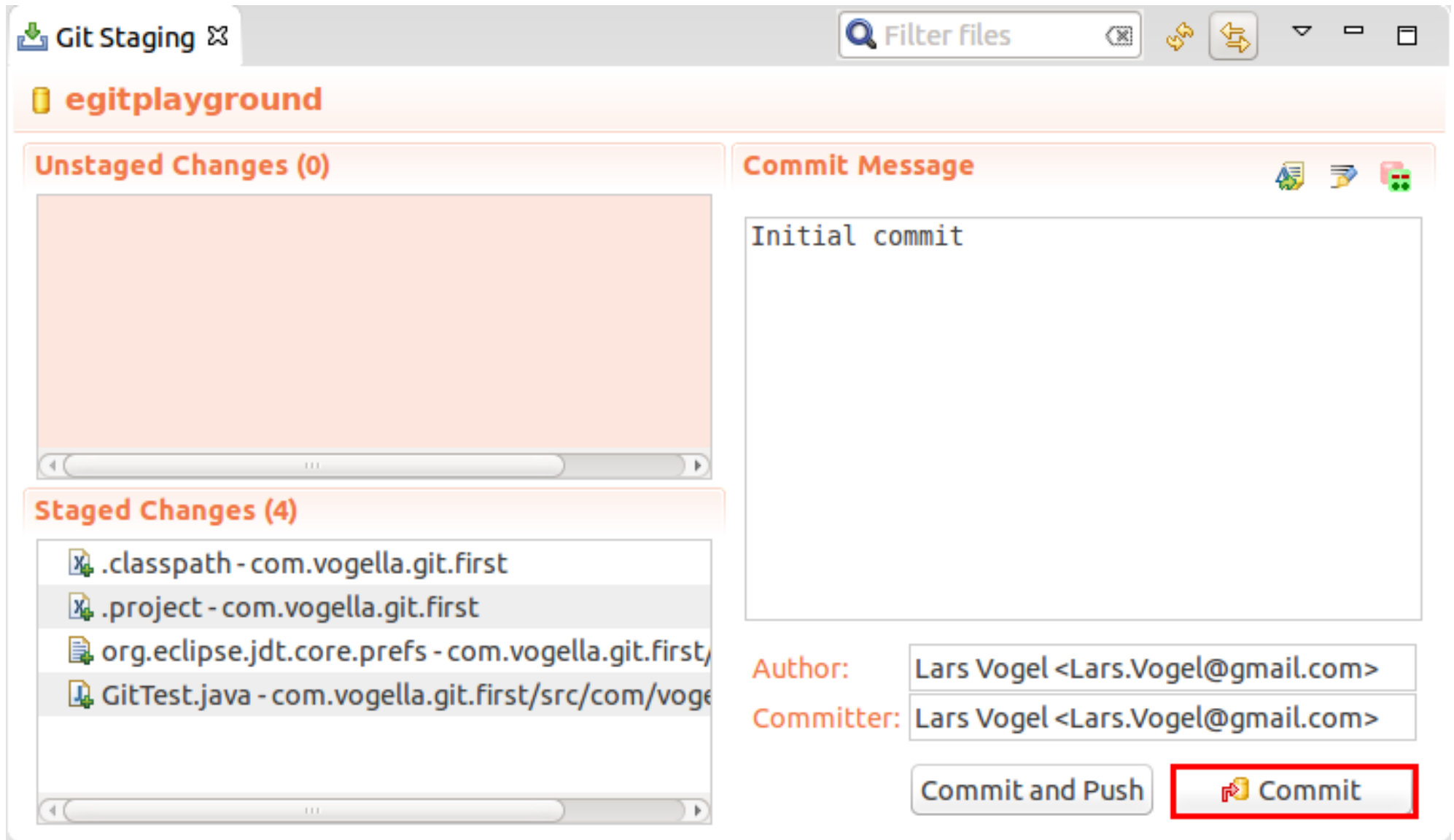
The screenshot shows the Eclipse Explorer view with a project structure. The project name is 'com.example.e4.rcp.wizard.feature' with a status '[eclipse4book master ↑ 2]'. Below it is a folder 'drag'. Then there is a package '> org.eclipse.e4.demo.contacts' with a status '[eclipse.platform.ui master]'. Finally, there is a file 'org.eclipse.e4.demo.contacts.feature'.

## 7. Using the Git Staging view

Eclipse gives you several options to stage and commit your changes. The *Git Staging* view provides a convenient compact overview on all changes you have done compared to the current HEAD revision.

This view presents which files you have touched and which files will be included in the next commit. **Unstaged Changes** lists those changes which you have done locally but which you have not yet added to the staging area. **Staged Changes** list those changes

which you already have added to the staging area. You can drag and drop files from one area to the other. To commit the staged changes you write your commit message and press the `Commit` button which is highlighted in the following screenshot.



You can switch between different repositories or even restart Eclipse without losing a commit message and it allows incremental staging for changes.

You can open the *Git Staging* view via the Window ▸ Show View ▸ Other▹▹ ▸ Git ▸ Git Staging menu.

## 8. Using the Git commit dialog

You can also commit changes with the Git commit dialog. For this right-click on a resource and select Team ▸ Commit from the context menu.

The dialog allows you to add changed and new files to the staging area and commit the changes.





Author:

Lars Vogel <lars.vogel@gmail.com>

Committer:

Lars Vogel <lars.vogel@gmail.com>

Files (1/1)

?

✓

Status	Path
<div><div>✓</div><div><div><div></div><div>J?</div></div></div></div>	de.vogella.git.first/src/de/vogella/git/first/NewExampleClass.java

Push

?

Cancel

Commit



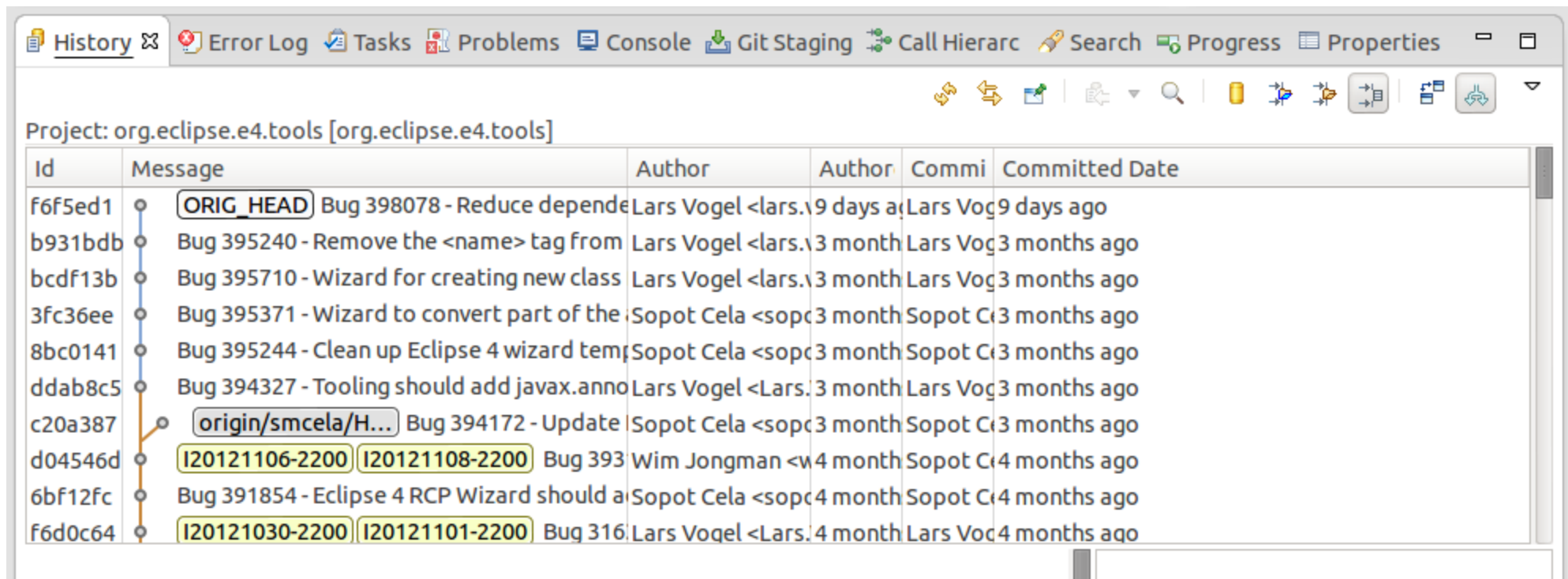
The commit dialog does not allow committing the removal of a file, since the commit dialog ignores the Git index.

# 9. Using the History view

## 9.1. Purpose of the history view

The *History* view allows you to analyze the history of your Git repository and to see to which commits the branches and tags point. This view displays author, date, commit message and the modified files of a commit.

This view is depicted in the following screenshot.



You can open this view via Window ▸ Show View ▸ Other▹▹ ▸ Team ▸ History. Alternatively you can open it via the repository node in the *Git Repositories* view. For this click on the Show In ▸ History entry. Some views, e.g., in the Java EE-Perspective, do not have this shortcut, in this case use Team ▸ Show in History.

## 9.2. Review the repository history via the History view

To see the history of a resource, select your project, a file or a folder, right-click on it and select the Show in> History context menu entry. Alternative you can use the Alt + Shift + W shortcut and select the *History* entry.

You can also configure the *History* view to display the history of the current selection. Select the highlighted button in the following screenshot for that.



If you select a commit you see the commit message and the involved files.

History Git Staging Problems Javadoc Declaration Terminal

Folder: /home/vogella/git/de.vogella.git.first/de.vogella.git.first/ [de.vogella.git.first]

Id	Message	Author	Authored Date	Co
21335c7	<b>master</b> <b>HEAD</b> Changed output message	Lars Vogel <Lars.Vogel@gmail.com>	40 seconds ago	Lar
c82627b	Initial commit	Lars Vogel <Lars.Vogel@gmail.com>	3 minutes ago	Lar

```

commit c82627b06e345ab4daafa21ac6fcdf08c891ed34
Author: Lars Vogel <Lars.Vogel@gmail.com> 2012-07-27 16:59:44
Committer: Lars Vogel <Lars.Vogel@gmail.com> 2012-07-27 16:59:44
Child: 21335c79ece5784fe2762b244a9cf0b73e954c73 (Changed output message)
Branches: master

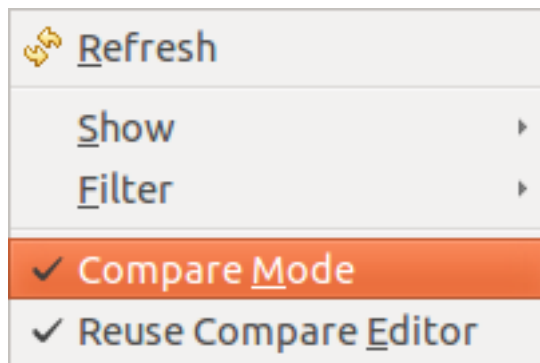
Initial commit
  
```

- de.vogella.git.first/.classpath
- de.vogella.git.first/.project
- de.vogella.git.first/.settings/org.eclipse
- de.vogella.git.first/src/de/vogella/git/fi

Via right-click on an individual file you can compare this file with its ancestor (the commit before that) or with the current version in the workspace.



If the "compare mode" toggle is selected from the view menu of the *History* view you can also doubleclick a file to compare it to the previous version.



## 9.3. The History view filters

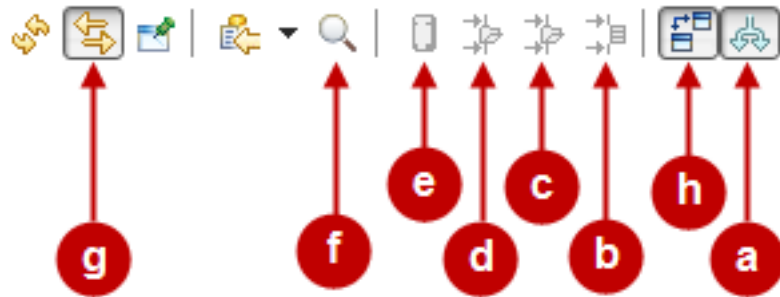
The *History* view has quite some options to configure which commits are displayed. Its toolbar allows you to customize which commits are displayed. By default, the *History* view filters the history based on the current selection and shows only the active branch.

If you work with several branches, e.g., because you are using Gerrit for code reviews, you typically want to see all branch information and remove the filter based on the resource.

The *History* view allows you to filter based on resources. See the tooltips of the toolbar for the meaning of the different filter options. In order to see all commits click the highlighted buttons with the *Show all changes in this repository* and *Show all branches and tags* tooltips.

Problems @ Javadoc Declaration Git Staging History						No filter based on resource selection				Show all branches	
Project: org.eclipse.e4.tools.emf.liveeditor [org.eclipse.e4.tools]											
Id		Message		Author		Authored Date		Committer		Committed Da	
697b66f		master HEAD Bug 427541 - Activate unix line endings for the e4 tools project		Lars Vogel		10 minutes ago		Lars Vogel		9 minutes ago	
547a29e		120140205-1610 120140205-2200 origin/HEAD origin/master FETCH_HEAD		Brian de Alwis		12 hours ago		Brian de Alwis		12 hours ago	
e606d46		Bug 370055 - [ModelEditor] TrimWindow Bounds set to -2147483648		Brian de Alwis		12 hours ago		Brian de Alwis		12 hours ago	
318206f		Bug 427451 - [CSS Spy] Replace Factory usage in CSS spy with EModelService		Lars Vogel		21 hours ago		Lars Vogel		21 hours ago	
d6a158c		Bug 426343 - Activate more save actions for org.eclipse.e4.tools projects		Lars Vogel		21 hours ago		Lars Vogel		21 hours ago	
33c5a59		Bug 426343 - Activate more save actions for org.eclipse.e4.tools projects		Lars Vogel		21 hours ago		Lars Vogel		21 hours ago	

The following listing gives an overview of the purpose of the different buttons.



Depending on your use case you may want to select the following option:

- show only those commits which are reachable from the current branch. Hide all commits on other topic branches.
- see only those commits which changed the selected resource(file, project, subfolder) or it's children. E.g. display only those commits which touched the selected java file. The current selection is shown in the top right corner of the History view.

- c. see only those commits which changed anything in the parentfolder of the selected resource (file, project,subfolder) or it'schildren. E.g. display only those commits which changed the samepackage as the selected javasource.
- d. see only those commits which changed anything in the sameproject as the selected resource or it's children.Used when youare working in a repository which contains multiple projects.
- e. don't filter at all. Show all commits of the current repository

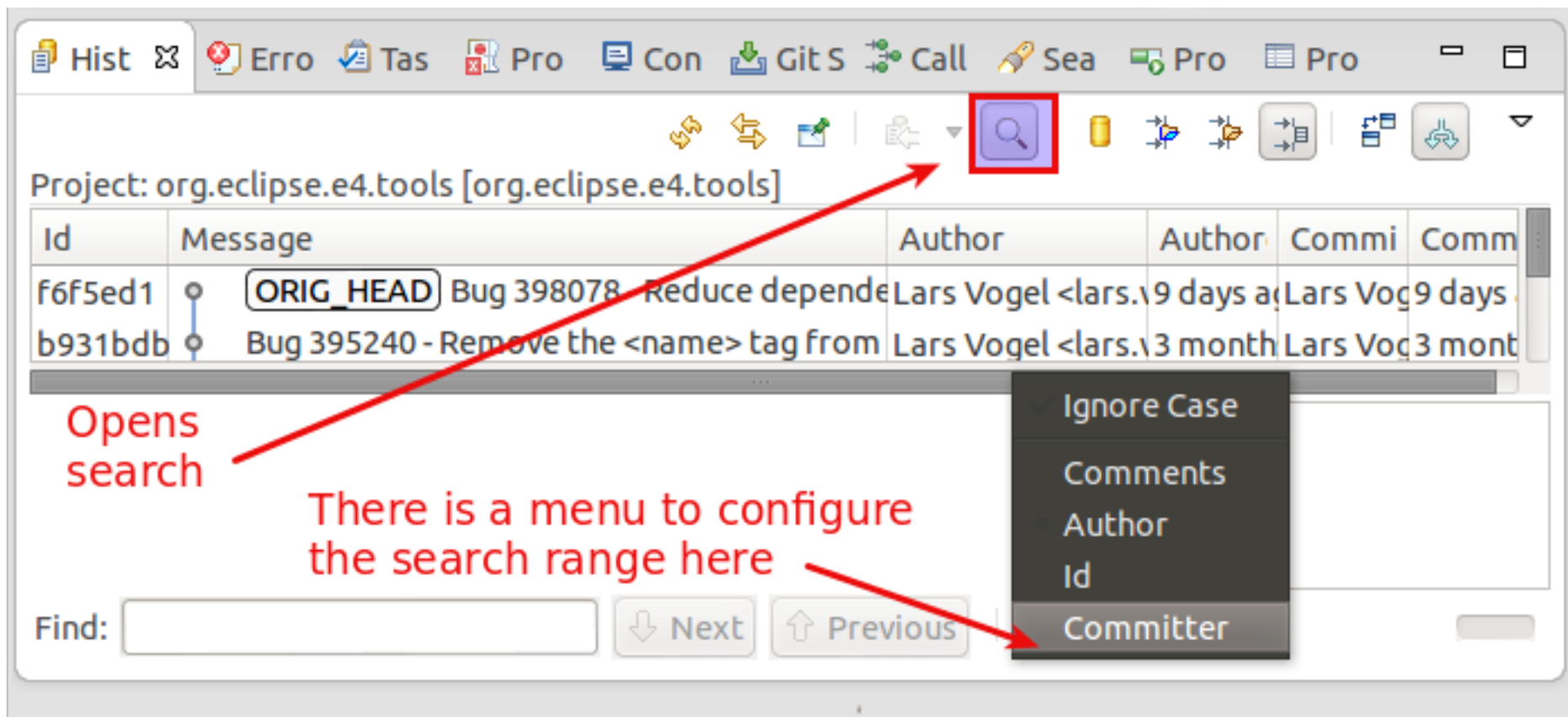
The options b., c. and d. are tied to the currently selectedresource.Button g. allows that the history viewautomatically updates when youchange the selection.



If you got lost with the different filters and the historydoesn't show what you expect, set it back to showeverything.Therefore make sure that**Show all branches and tags(a)** is turned on and**Show all changes in repository(e)** is selected.

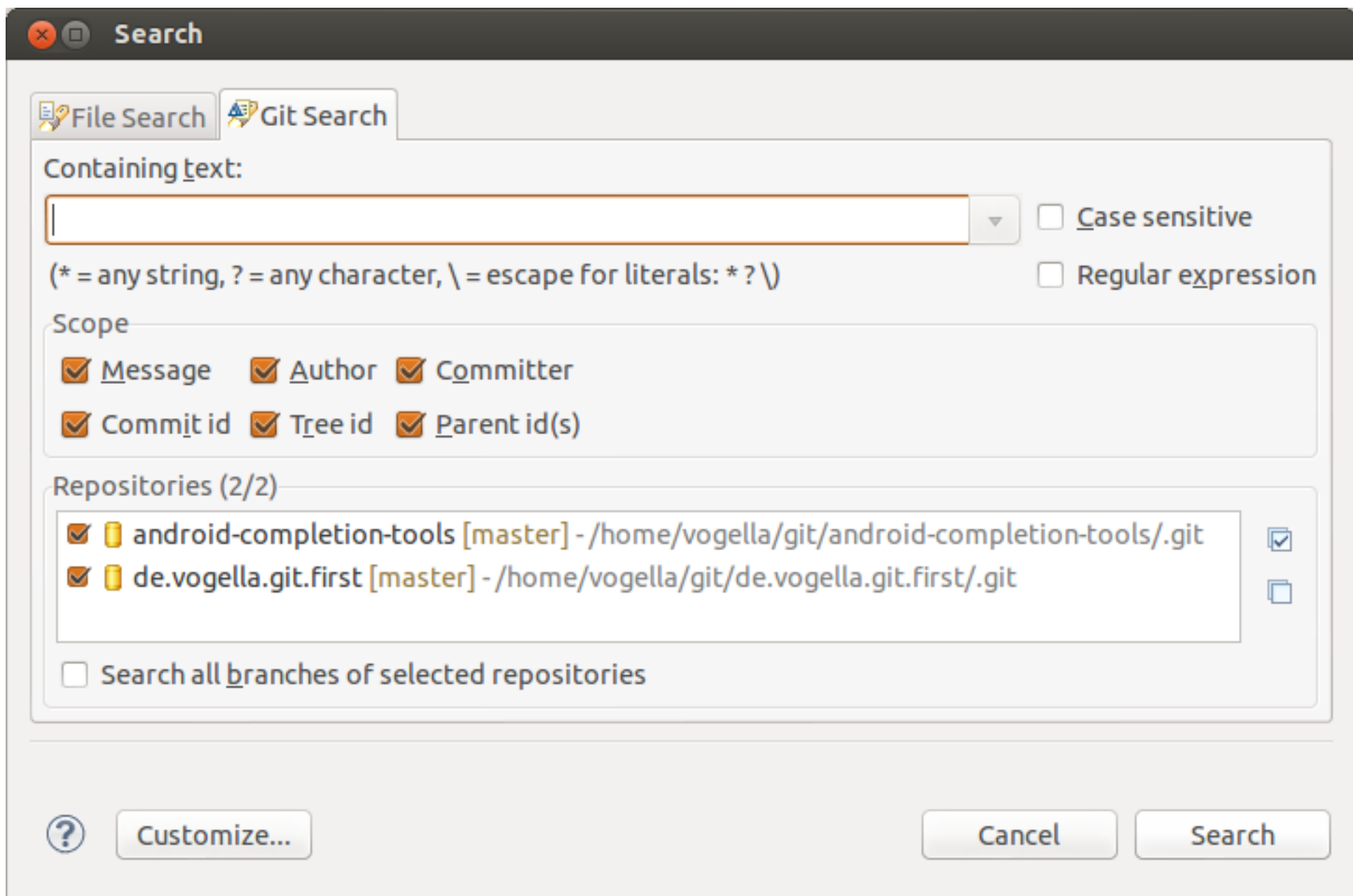
## 9.4. Using search

You can also search for commits basedon committer, author, ID orcomment. For this turn on the**Show Find toolbar(f)** and type in a search string in the**Find**field. Thecommits fitting to your searchare highlighted. You can combine this search with the filtersexplained above.



NOTE: The *Git Search* available in the Search ▢ Search menu is much more powerful and consumes less memory since it doesn't need to also display the history.

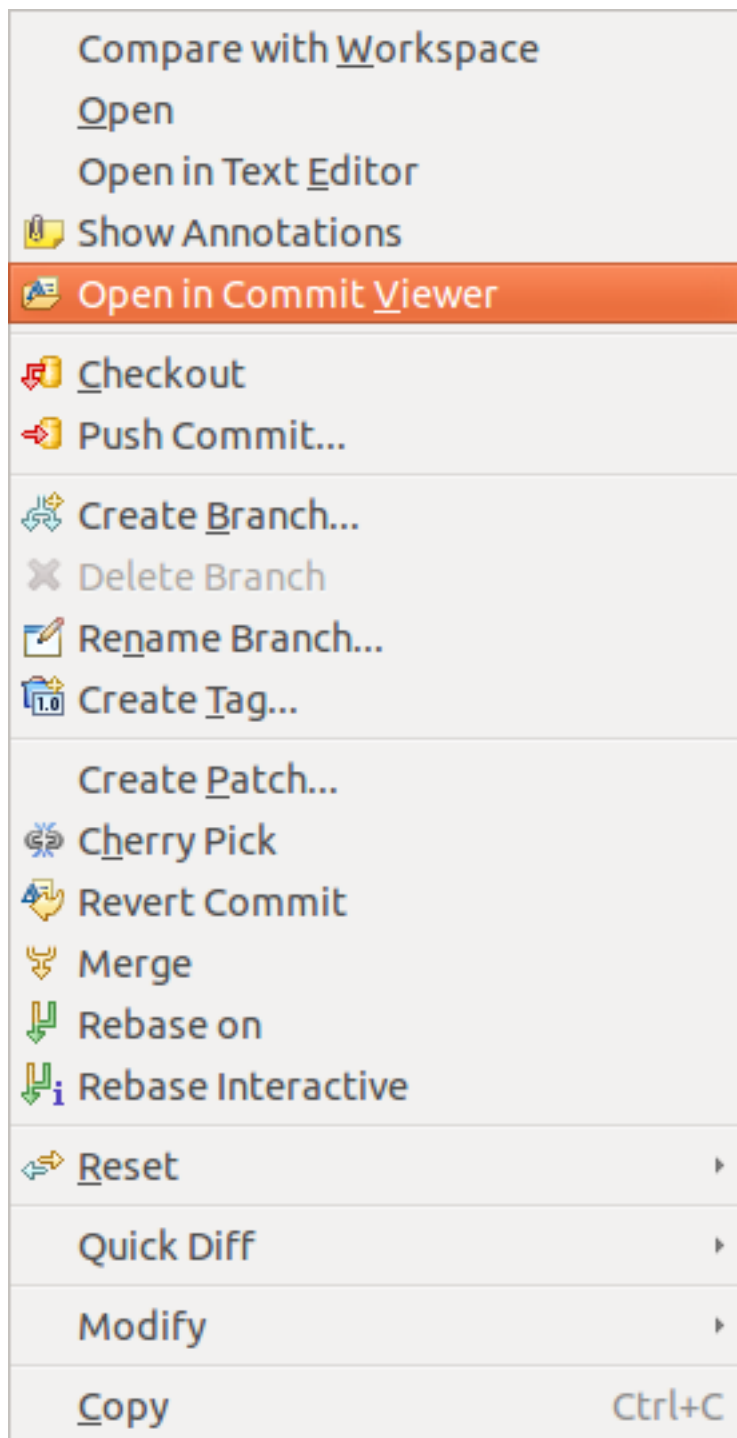




## 9.5. Showing details of a commit

If you want to see more details about a commit, right-click it and select the *Open in Commit*

*Viewer* entry.



3412c6e0 [eclipse.platform.ui]

**Commit 3412c6e0cfff25f7c9d7adf0fd3f3d09d6fa76c2** eclipse.platform.ui

Lars Vogel <Lars.Vogel@gmail.com> on 7/30/14 12:02 PM

Lars Vogel <Lars.Vogel@gmail.com> on 7/30/14 2:26 PM

Tags:

**Message**

Bug 440739 - Add missing @Override annotations to the JFace viewer tests

Change-Id: Id653bc3d28ad7dc93e2889c25630dac11c326dae

**Files (54)**

- tests/org.eclipse.ui.tests/Eclipse JFace Tests/org/eclipse/jface/tests/viewers/AbstractTreeViewTest.java
- tests/org.eclipse.ui.tests/Eclipse JFace Tests/org/eclipse/jface/tests/viewers/Bug138608Test.java
- tests/org.eclipse.ui.tests/Eclipse JFace Tests/org/eclipse/jface/tests/viewers/Bug180504TableViewerTest.java
- tests/org.eclipse.ui.tests/Eclipse JFace Tests/org/eclipse/jface/tests/viewers/Bug180504TreeViewTest.java
- tests/org.eclipse.ui.tests/Eclipse JFace Tests/org/eclipse/jface/tests/viewers/Bug200337TableViewerTest.java
- tests/org.eclipse.ui.tests/Eclipse JFace Tests/org/eclipse/jface/tests/viewers/Bug200558Test.java
- tests/org.eclipse.ui.tests/Eclipse JFace Tests/org/eclipse/jface/tests/viewers/Bug201002TableViewerTest.java
- tests/org.eclipse.ui.tests/Eclipse JFace Tests/org/eclipse/jface/tests/viewers/Bug201002TreeViewTest.java

**Branches (1)**

- master

**Commit** **Diff**

# 10. Working with Eclipse projects in a Git repository

## 10.1. Workspace and projects

Eclipse allows working with projects that are not included in the root folder of the workspace.

Using this functionality your projects can be stored in the working tree of a Git repository.

## 10.2. Adding a new project to a Git repository

To can add Eclipse projects to an Git repository.If you do this the project is moved to the Git repository and linked to from the Eclipse workspace.

A simple way of adding a project to a Git repository is to specify the file location in the *New Project* wizard.This is depicted in the following screenshot.

**New Project**

**Project**  
Create a new project resource.

Project name:

☐ Use default location

Location:

Working sets

☐ Add project to working sets

Working sets:

**Path inside the Git repo**

If you add the Git repository to your *Git repositories* view, you can perform the Git team


operations on the files of this repository.

To add a new Eclipse project to an existing Git repository, select the project, right-click on it and select Team → Share → Git. Afterwards select the desired Git repository.

Configure Git Repository

Configure Git Repository

Select an existing repository or create a new one



☐ Use or create repository in parent folder of project

Repository:

eclipse4book - /home/vogella/git/eclipse4book/.git

Create...


Working directory:

/home/vogella/git/eclipse4book

Path within repository:

Browse...

Project	Current Location	Target Location
<input checked="" type="checkbox"/> com.exa	/home/vogella/workspace/eclipse4t	/home/vogella/git/eclipse4book/com.example.e4.rcp.todo.ownar



< Back

Next >

Cancel

Finish

The Eclipse Git functionality moves the projects to the repository. It also imports the project again into your workspace with the adjusted reference on the file system.

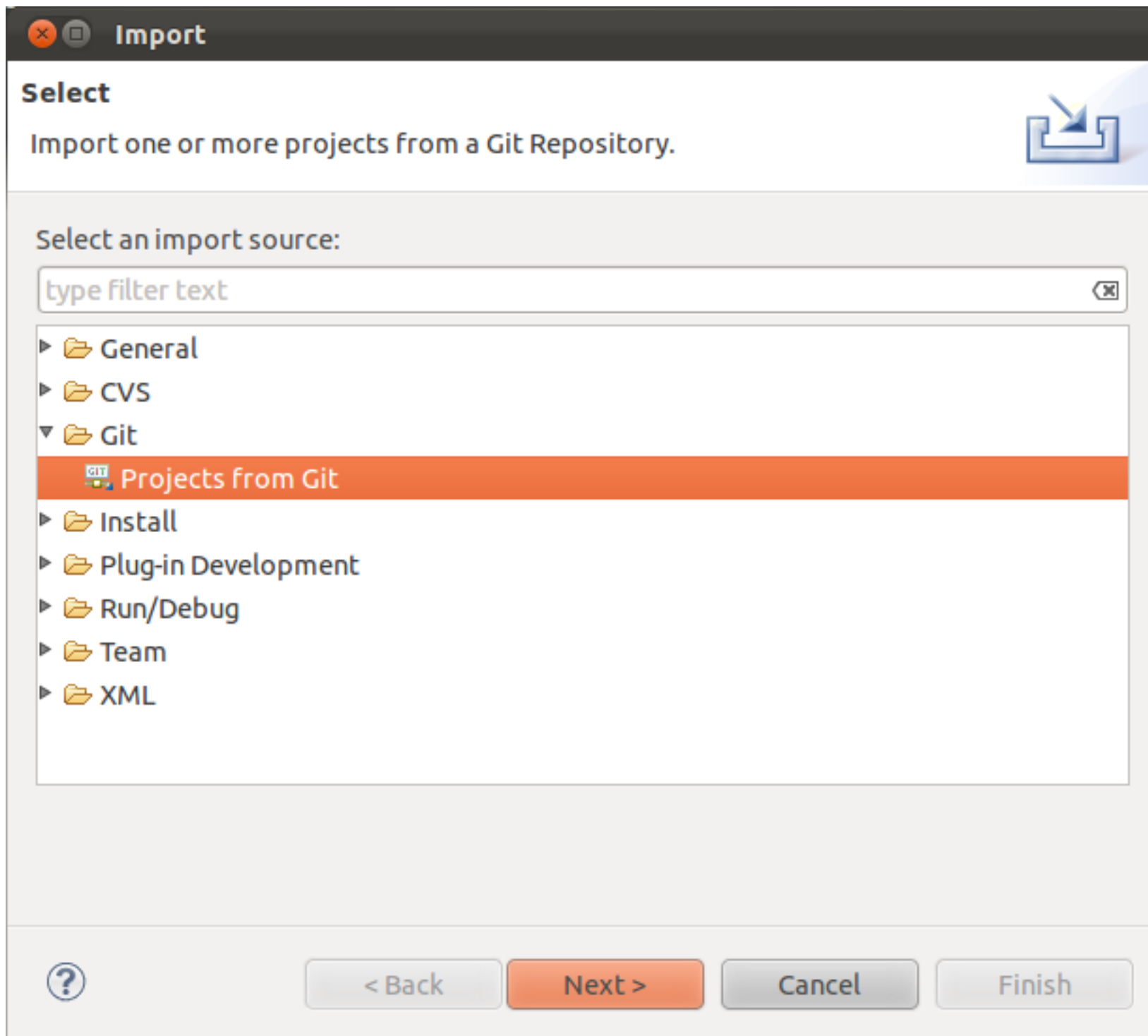


---

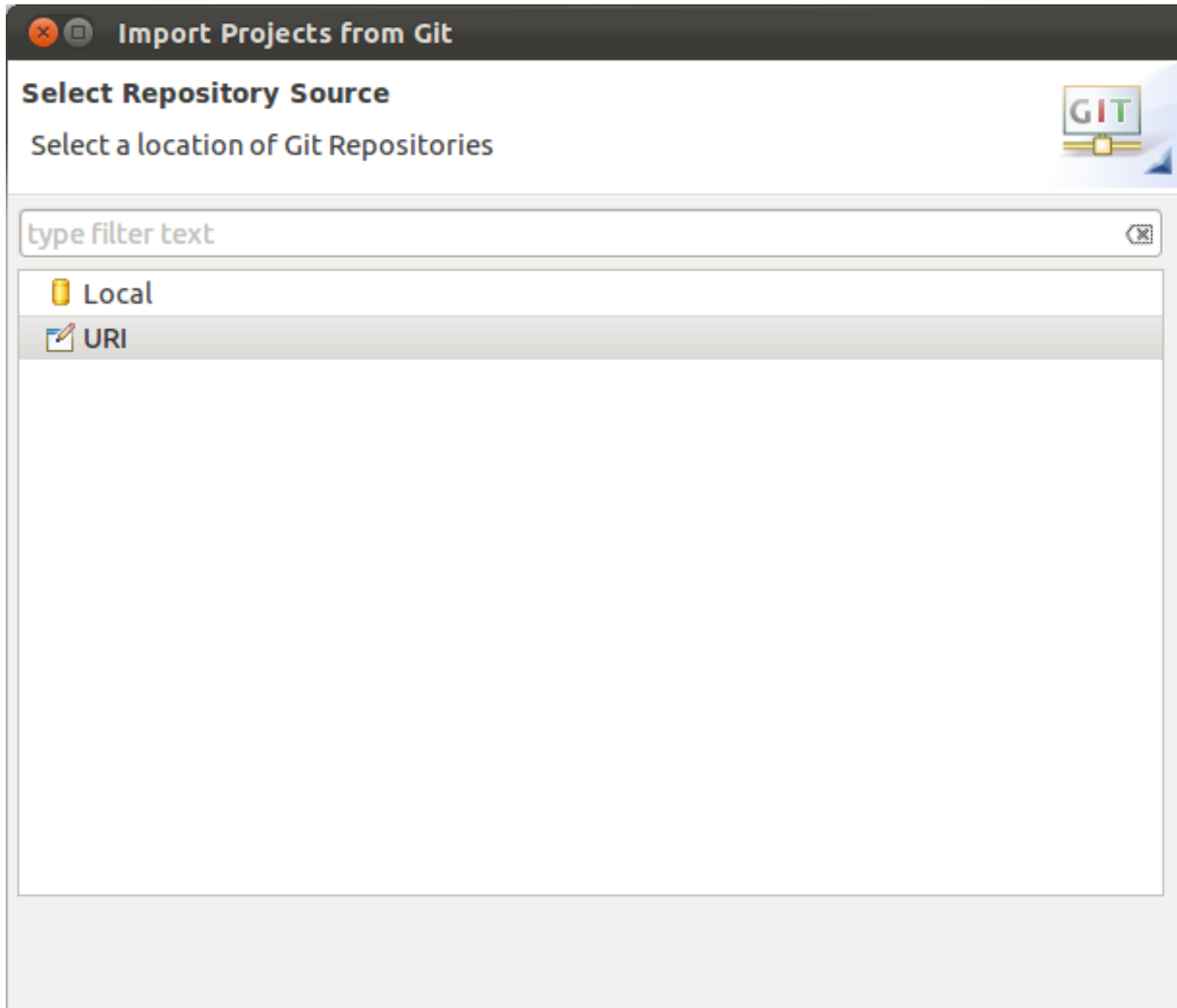
# 11. Clone an existing repository

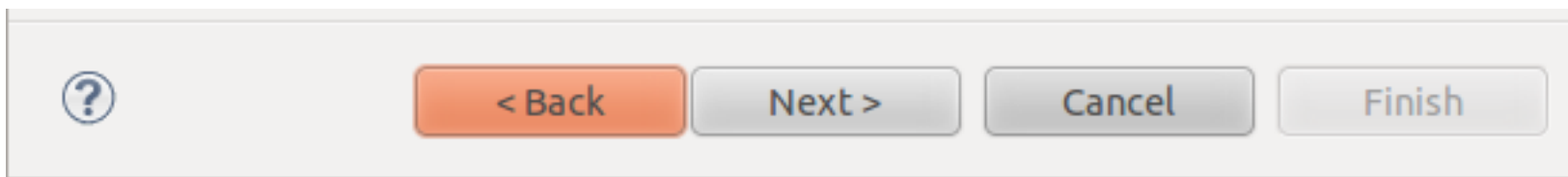
Eclipse allows you to clone an existing Git repository. Afterwards, you can import existing projects from this repository into your workspace.

Select File → Open Projects from File System →



Select *URI* in the next dialog.





Enter the URL to your Git repository. Git supports several protocols, e.g. *git://*, *ssh://* and *https://*. You can paste the clone URL to the first line of the dialog, the rest of the dialog is filled based on this data.

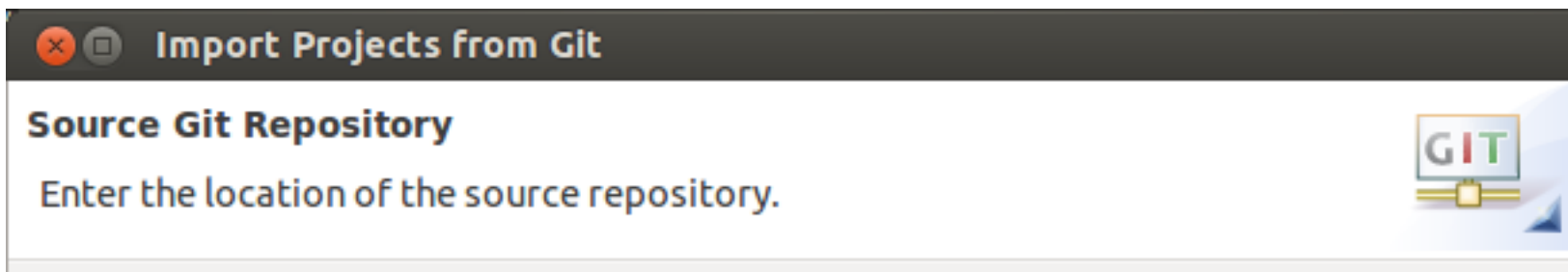


Some proxy servers block the *git://* and *ssh://* protocols. If you face issues, please try to use the *https://* or *http://* protocol.

For example the following URI can be used to clone the example projects of the Eclipse 4 application development book:

`git://github.com/vogella/eclipse4book.git`

The above link uses the git protocol, alternatively you can also use the http protocol:  
<http://github.com/vogella/eclipse4book.git>



**Location**

URI:

Host:

Repository path:

**Connection**

Protocol:


Port:

**Authentication**

User:

Password:

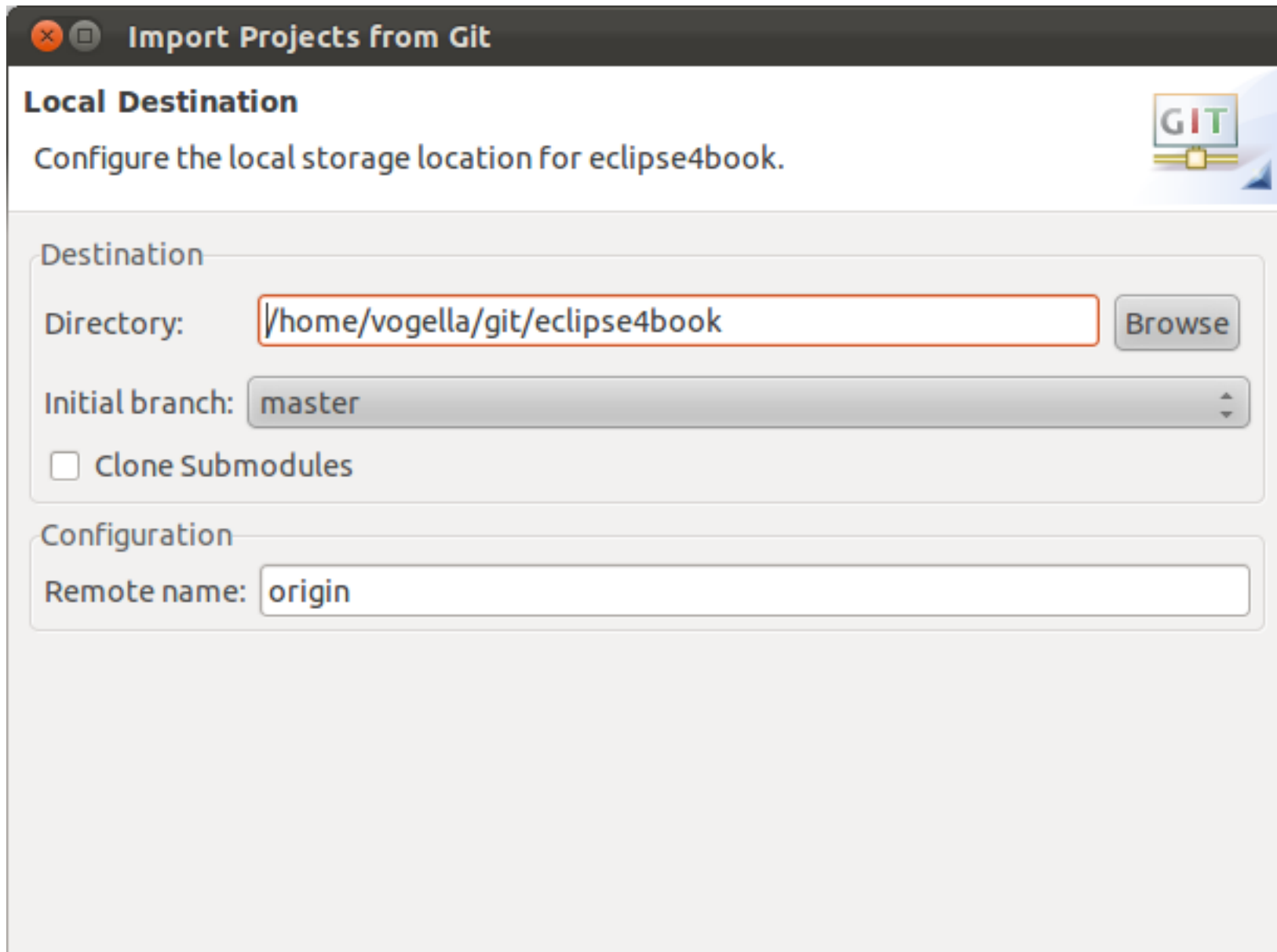
Store in Secure Store ☐

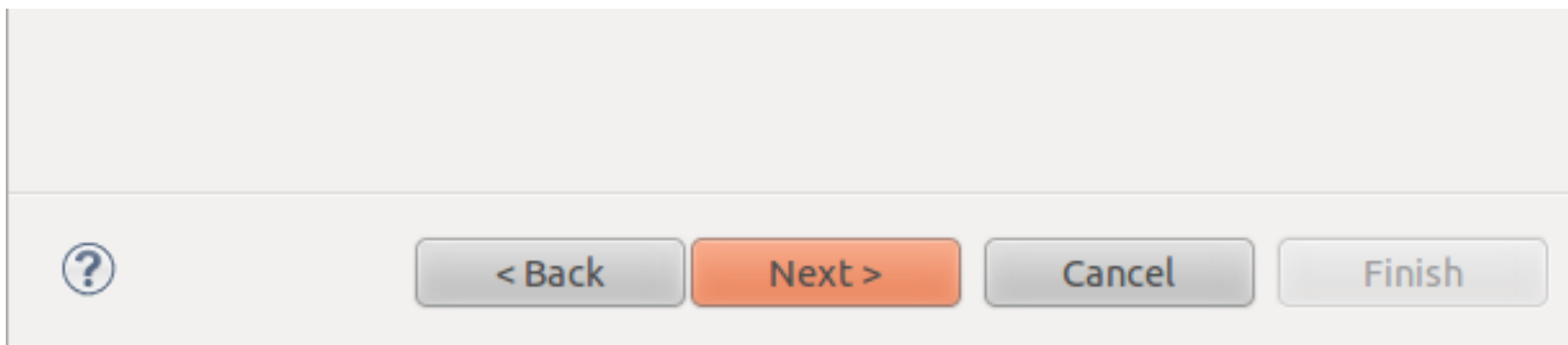


After pressing the  button the system will allow you to import the existing branches. You should select at least *master* as this is typically the main development branch.

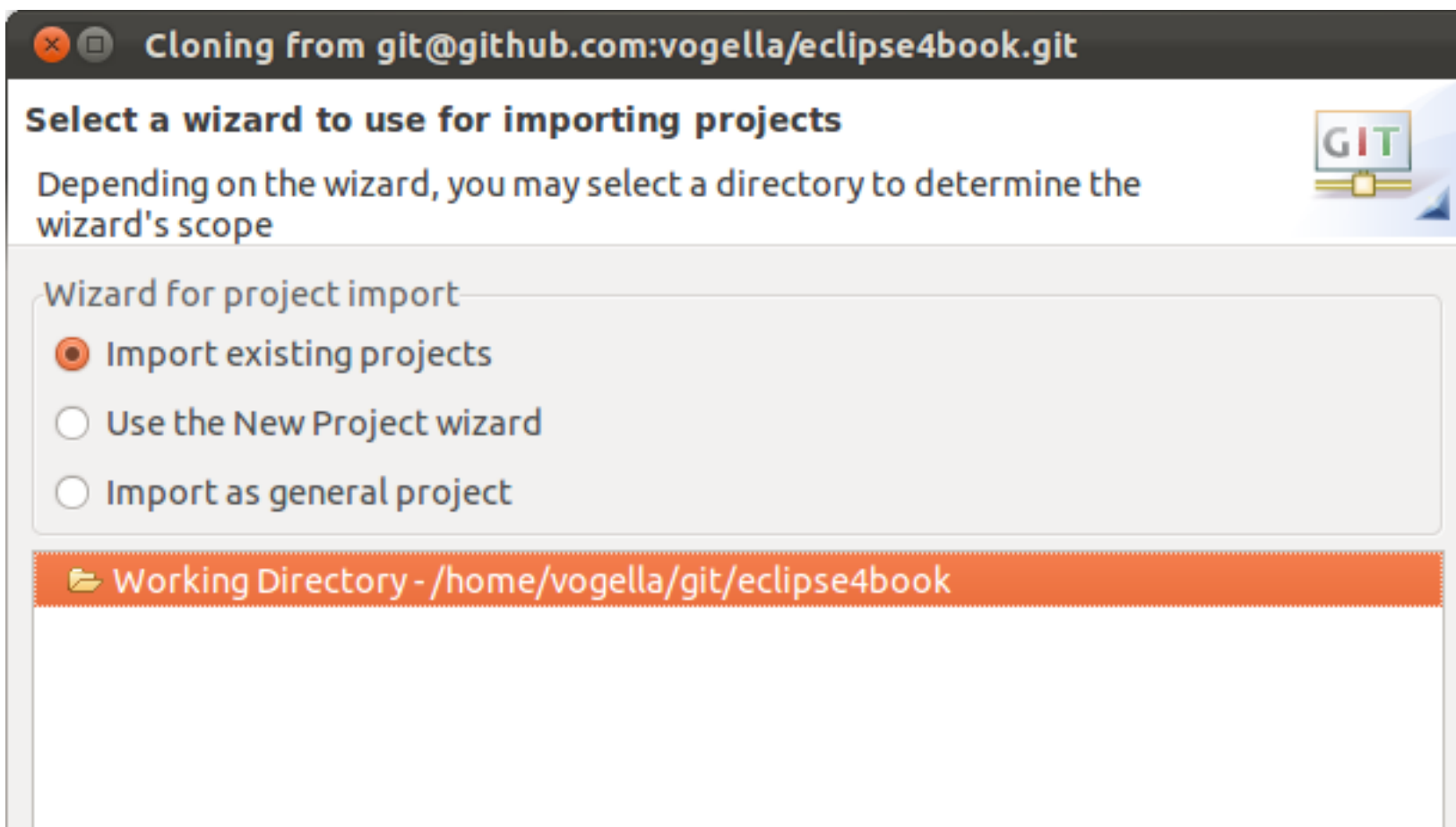
□PN2.png" alt="URI entered in the dialog">

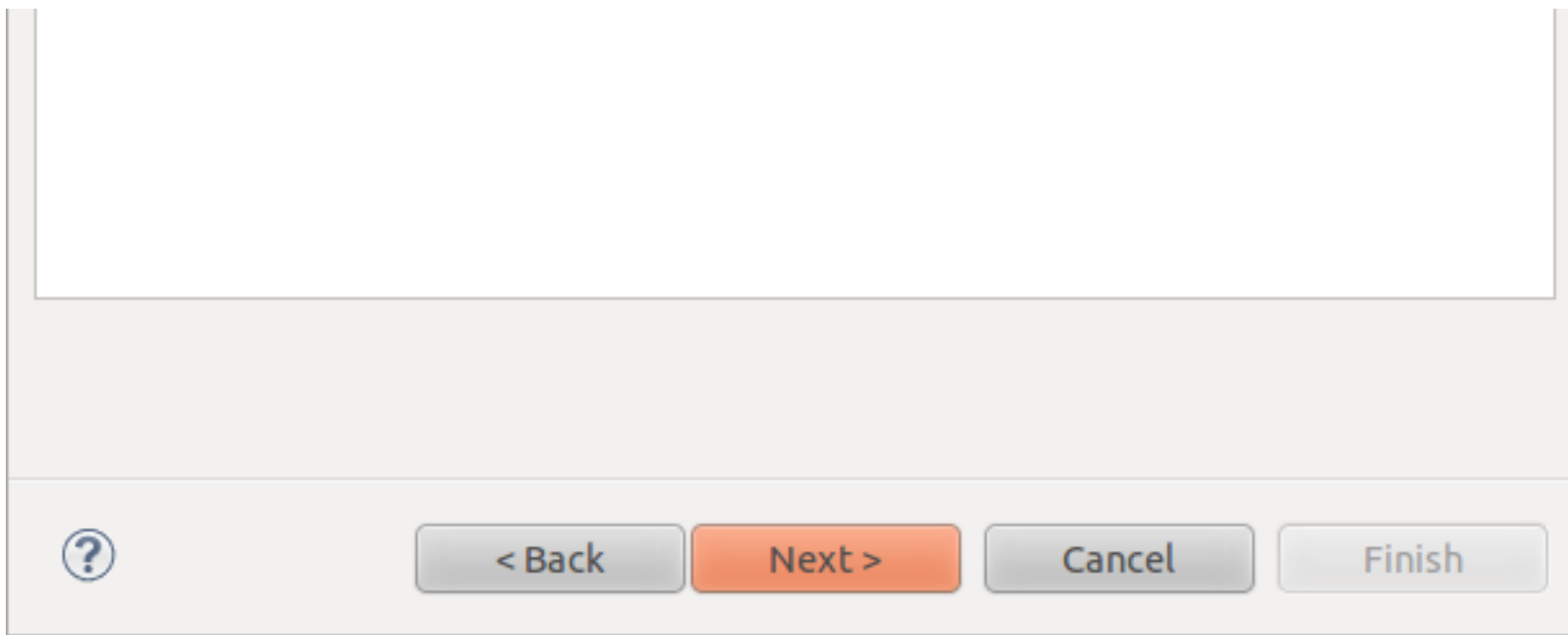
The next dialog allows you to specify where the repository should be copied to and which local branch should be created initially.





After the Git repository is cloned, Eclipse *EGit* opens an additional import dialog which allows importing the Eclipse projects from the Git repository.





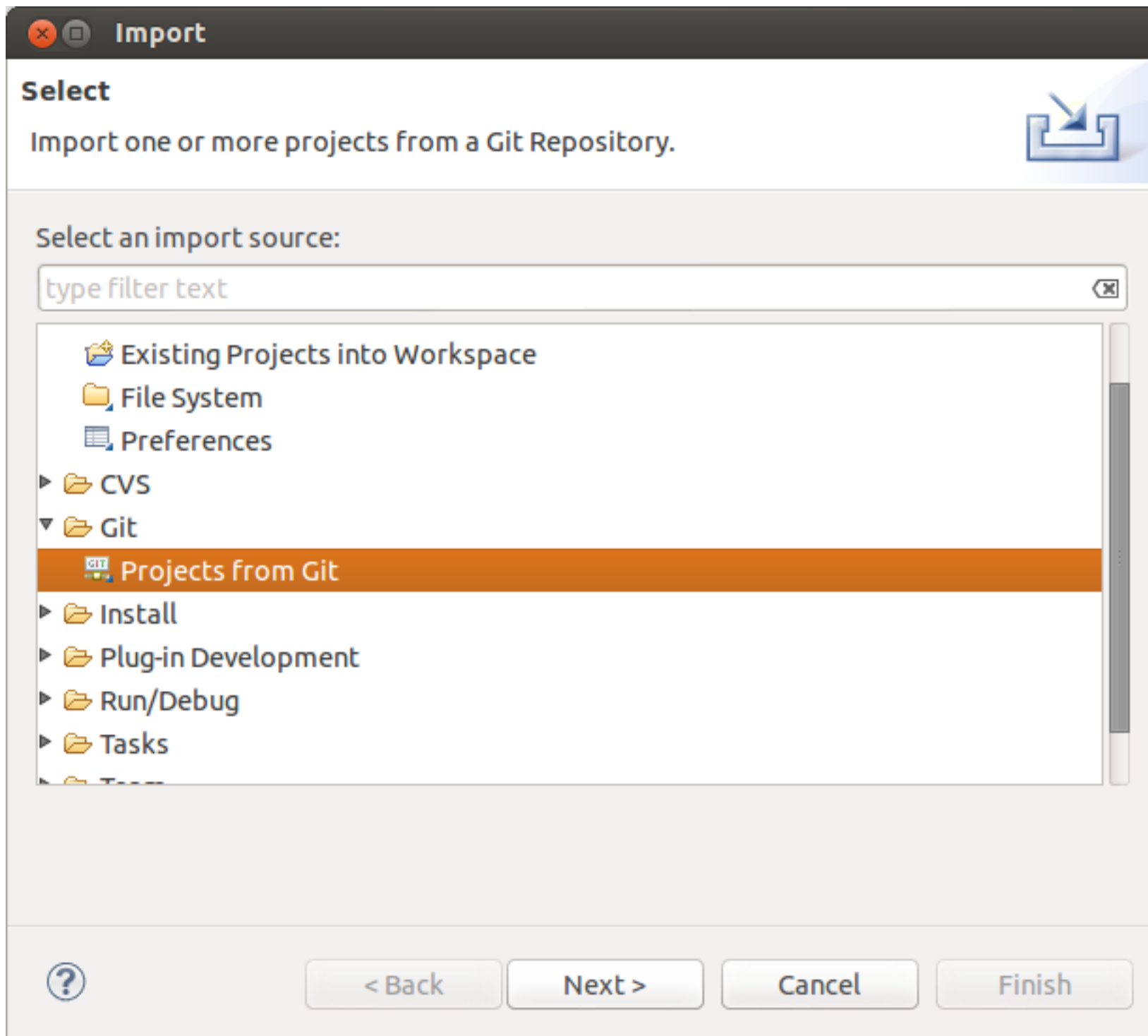
Once this dialog is completed, you have clone the remote repository into a local Git repository. You can use Git operation on these projects.

---

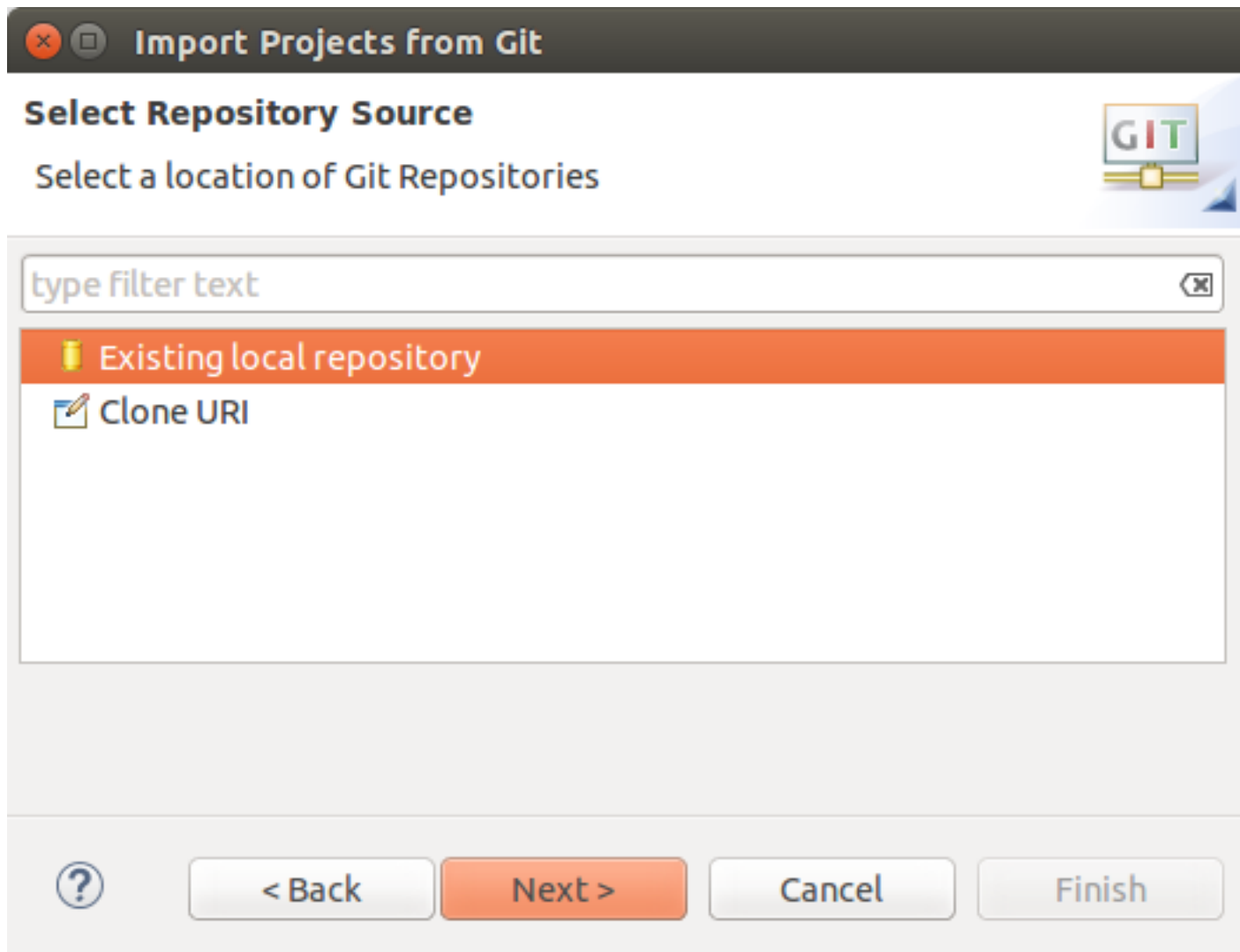
## 12. Import projects from an existing repository

If you have already an existing Git repository you can add it to Eclipse and import the Eclipse projects into your workspace via the `SelectFile` → `Import` → `Git` → `Project from Git` menu entry.



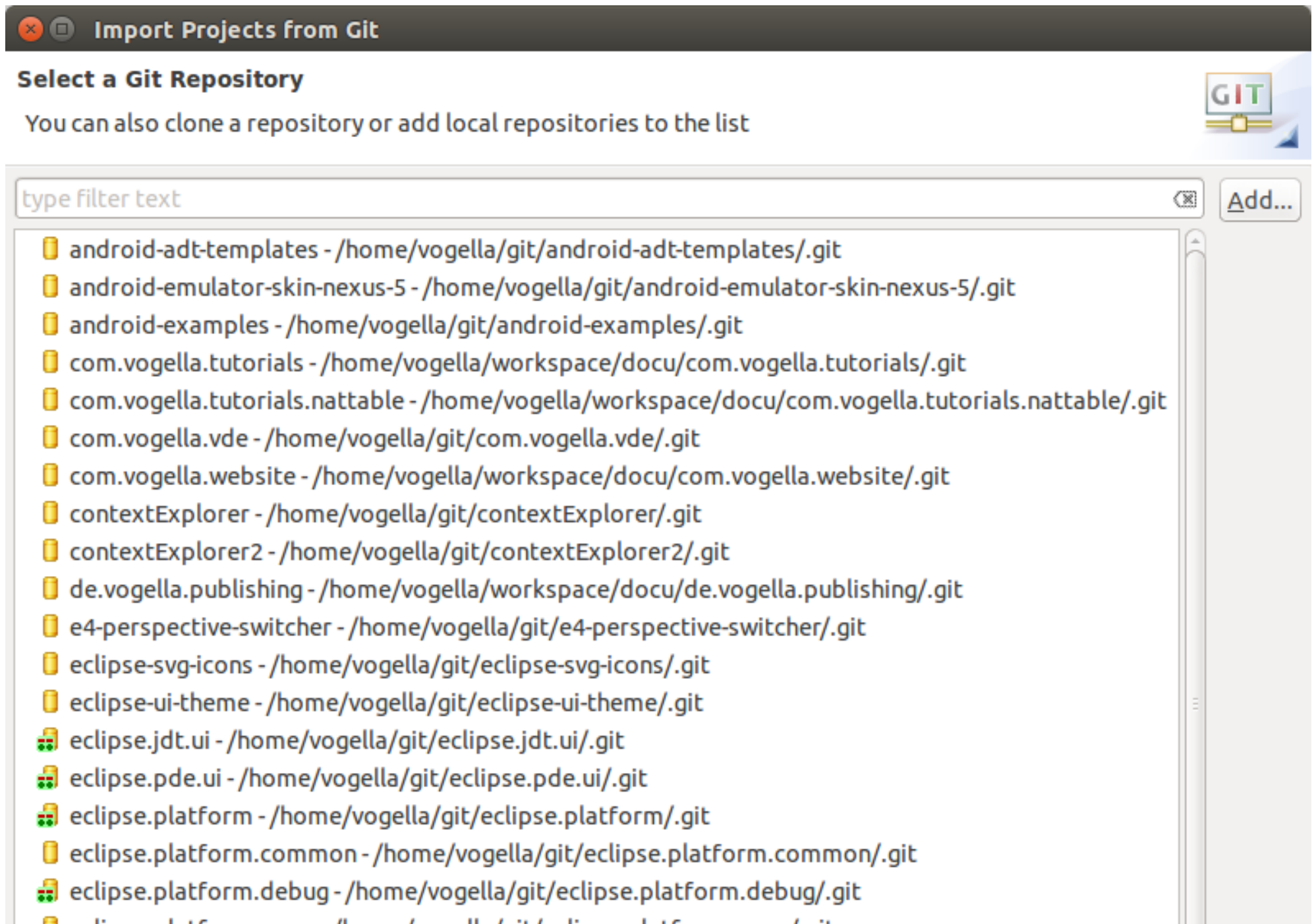


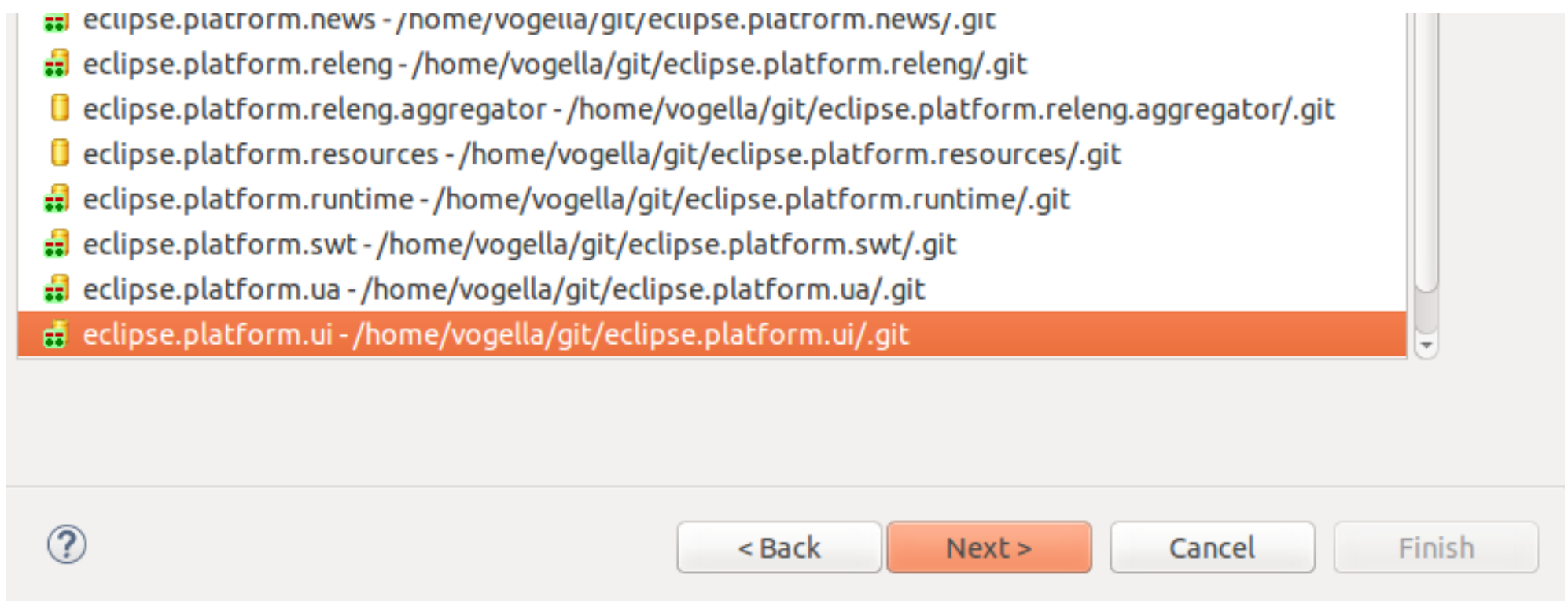
Select *Local* if you want to import from a local repository or *Clone URL* if you first want to clone the repository.



The following screenshot shows several local repositories. To import the project contained in one of them, select one entry and press the **Next** button. To add a new local repository to

this dialog (and the *Git repositories* view) use the  button.



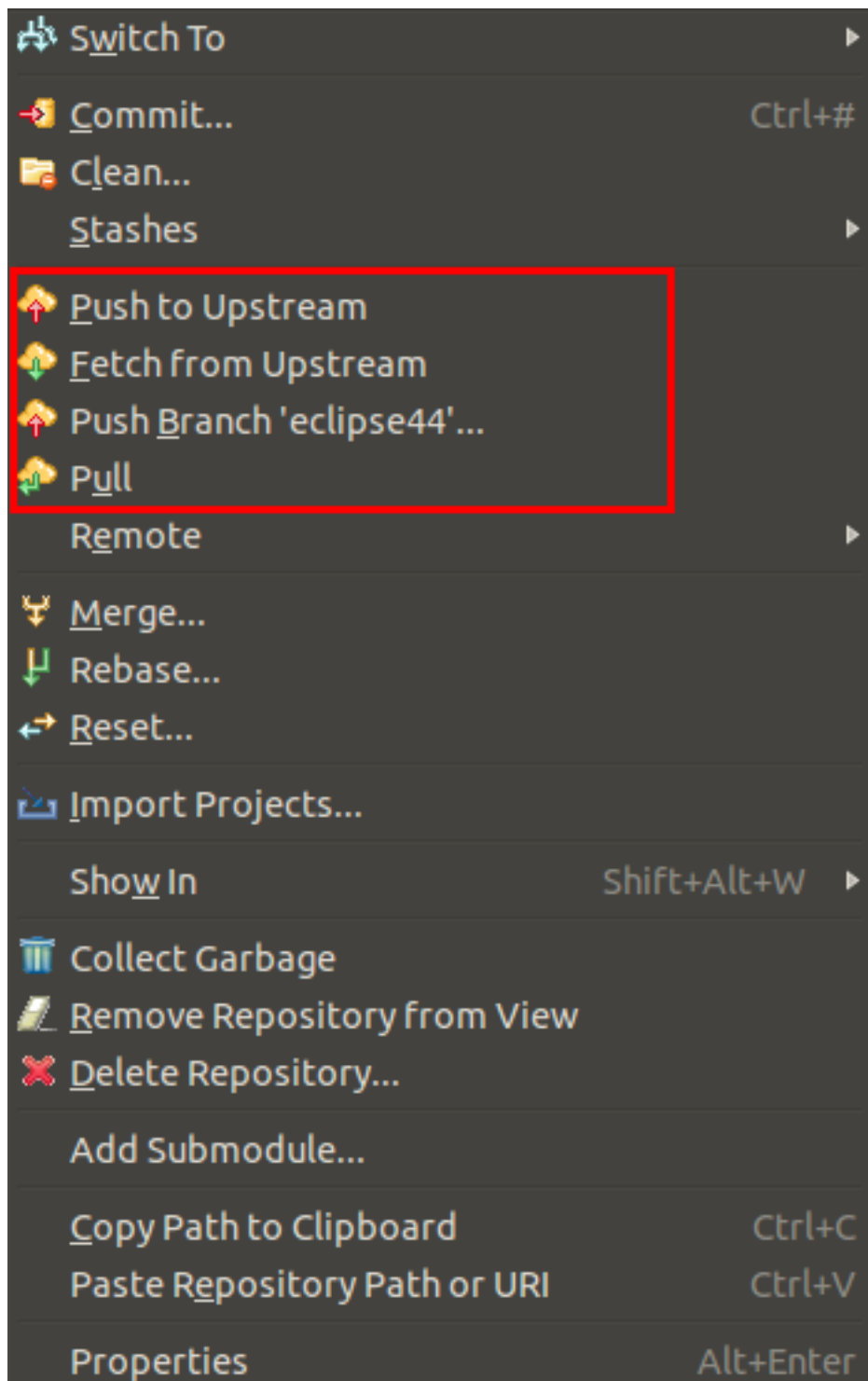


The wizard allows you to import existing projects. After this import the Eclipse IDE makes the projects available and is aware that these projects are part of a Git repository.

## 13. Performing Git operations in Eclipse

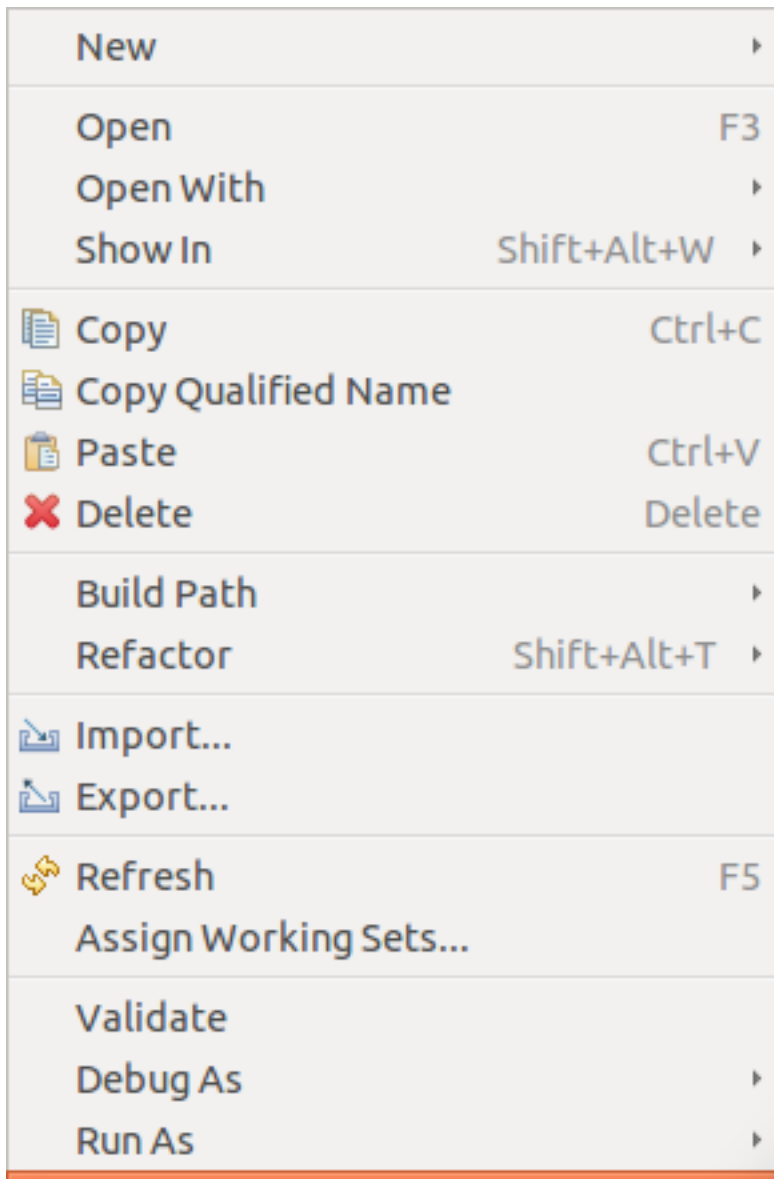
### 13.1. Pull, push and fetch

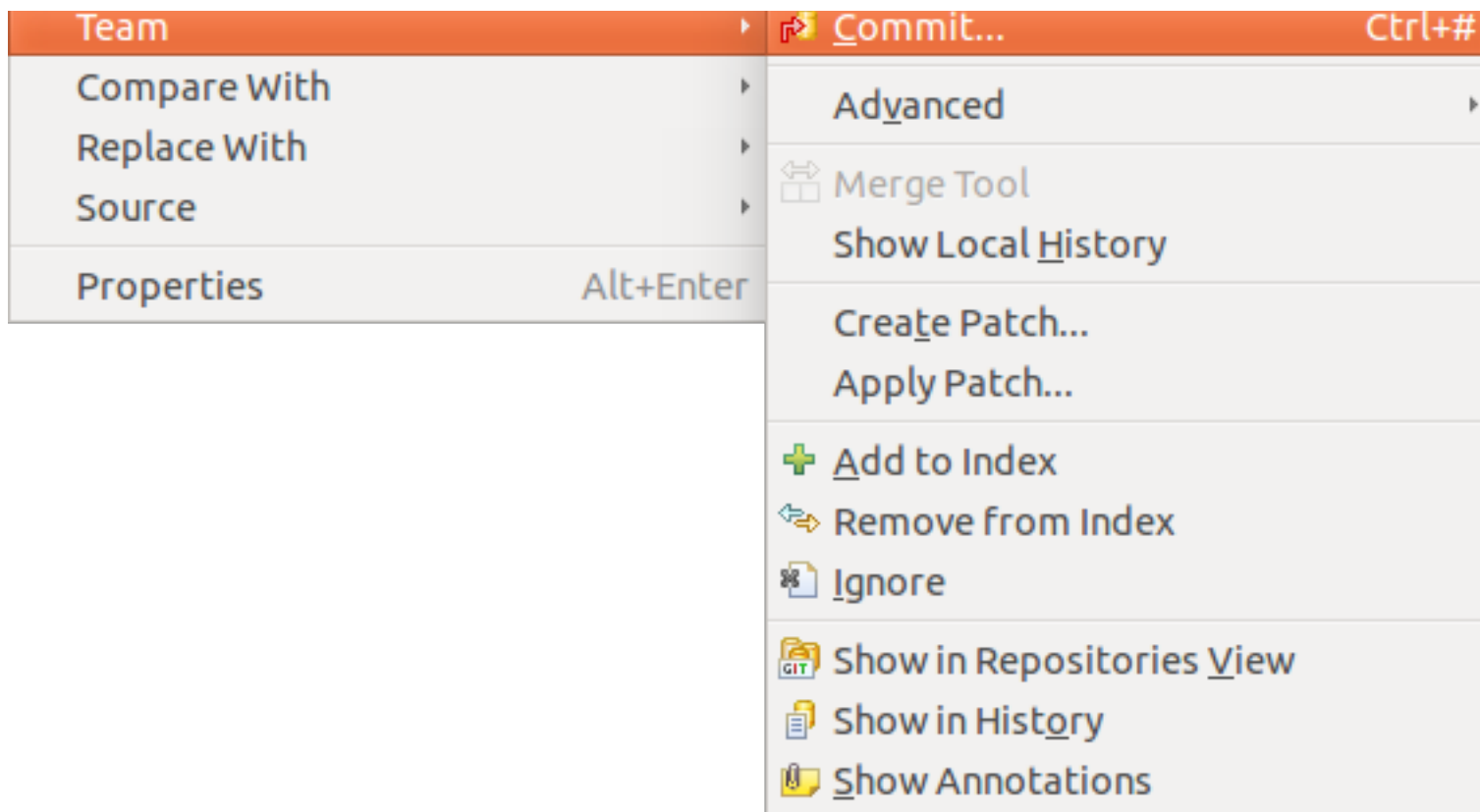
You can use the *Git Repositories* view to pull, push and fetch to remote repositories. Right click on your repository and select the appropriated operation.



## 13.2. Basic team 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 Team menu is also available from the context menu of an opened editor.

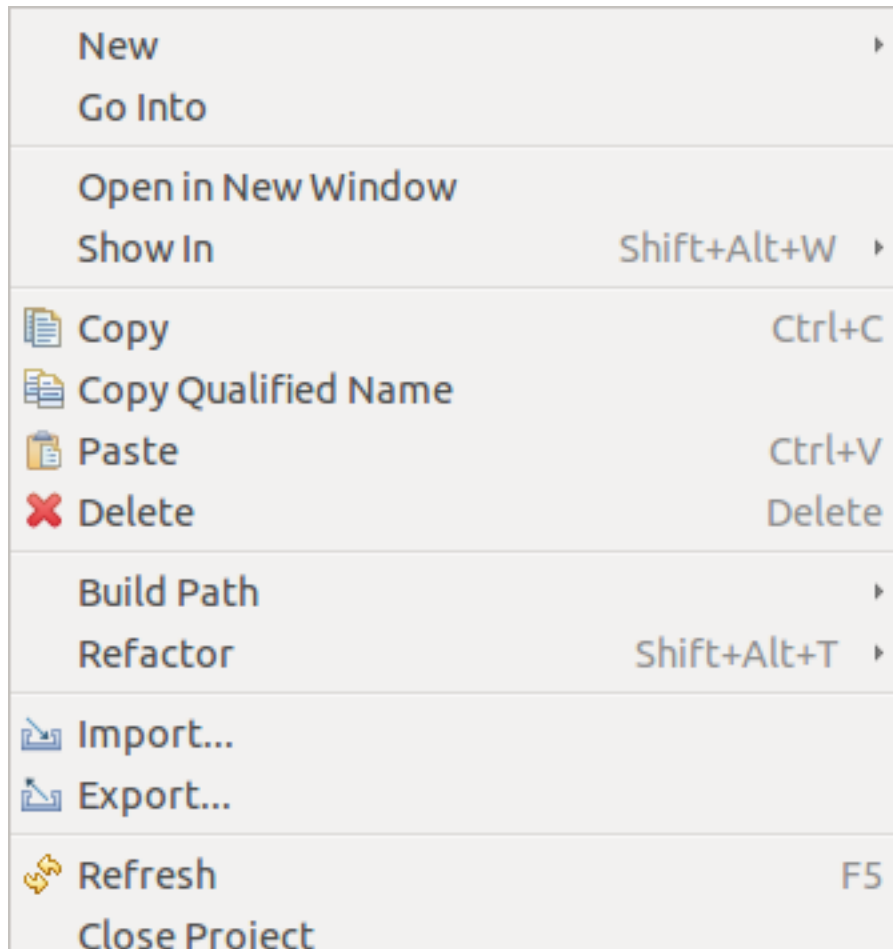
The most important operations are described in the following list. Select:

- Team ▢ Add to index, to add the selected resource(s) to the Git index
- Team ▢ Commit, to open the commit dialog to create a new commit
- Team ▢ Create Patch▢▢, to create a patch

















- Team ▸ Apply Patch▮▮, to apply a patch to your file system
- Team ▸ Ignore, to add a file to the .gitignore file
- Team ▸ Show in History, to display the history of the selected resources(s)

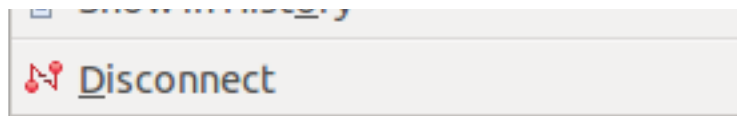
## 13.3. Team operations available on the project

If you select a project you can use additional team operations from the context menu.





Close Unrelated Projects	
Assign Working Sets...	
Validate	 <u>C</u> ommit... <span>Ctrl+#</span>
Debug As	▶  <u>P</u> ush to Upstream
Run As	▶  <u>F</u> etch <u>f</u> rom Upstream
Team	▶  <u>P</u> ush <u>B</u> ranch...
Compare With	▶ Remote ▶
Replace With	▶  <u>S</u> witch To ▶
Restore from Local History...	▶ Advanced ▶
Configure	▶
Source	▶  <u>P</u> ull
Properties <span>Alt+Enter</span>	 <u>S</u> ynchronize <u>W</u> orkspace
	 Merge Tool
	 <u>M</u> erge...
	 <u>R</u> ebase...
	 <u>R</u> eset...
	Create <u>P</u> atch...
	Apply Patch...
	 <u>A</u> dd to Index
	 Remove from Index
	 <u>I</u> gnore
	 Show in Repositories <u>V</u> iew
	 Show in History

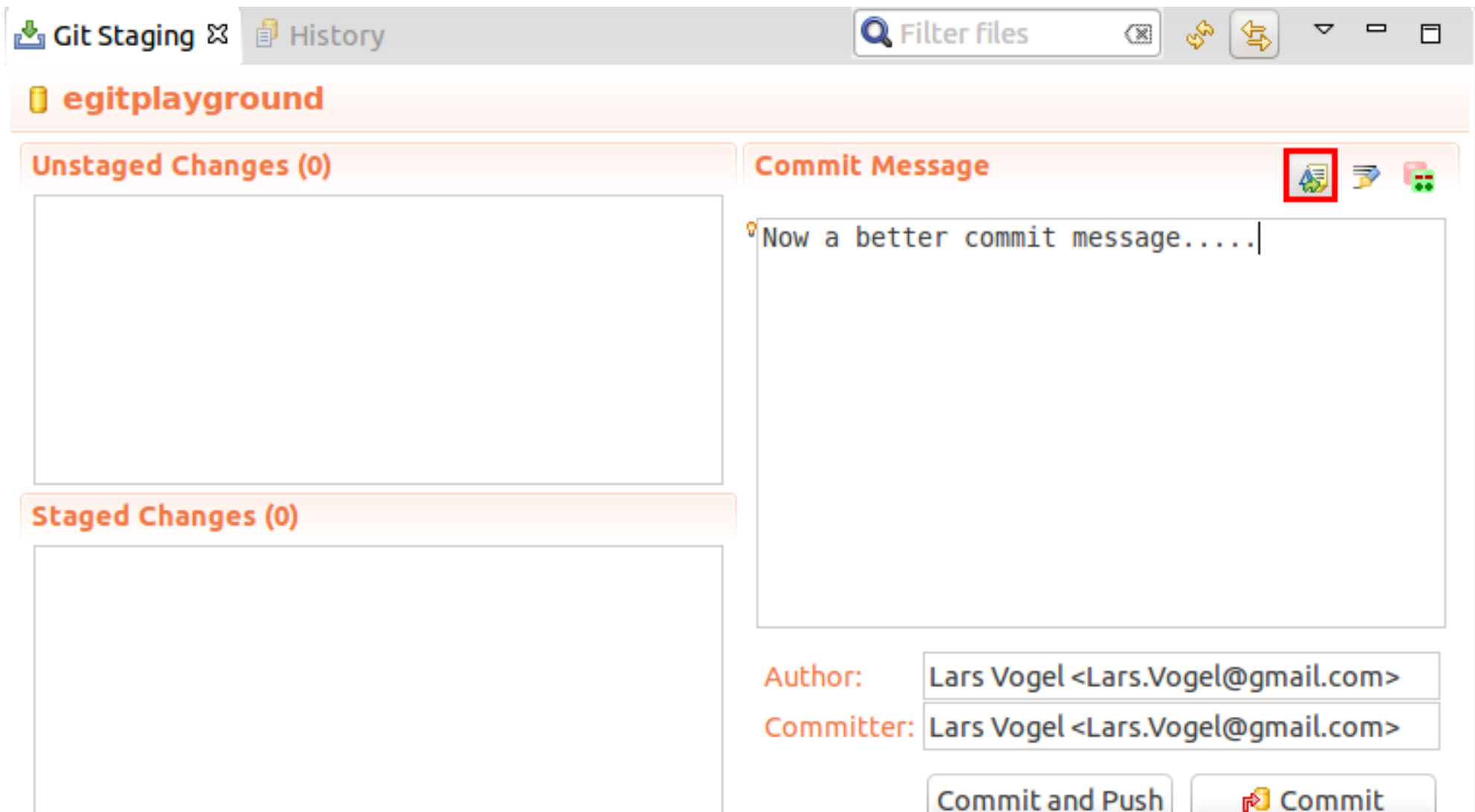


- Team ▢ Pull to pull in changes from your remote Git repository
- Team ▢ Fetch to fetch the current state from the remote repository
- Team ▢ Switch To to checkout existing or create new branches
- Team ▢ Push to push changes to your remote Git repository
- Team ▢ Tag to create and manage tags.

## 13.4. Amending a commit

Git amend allows adjusting the last commit. For example you can change the commit message or add another modification.

The *Git Staging* view allows you to perform the Git amend command via the highlighted button in the following screenshot.



## 14. Branching in Eclipse

Right-click your project and select **Team** → **Branch to** to create new branches or to switch

between existing branches.

You can also switch branches in the *History* view or the *Git repositories* view.



---

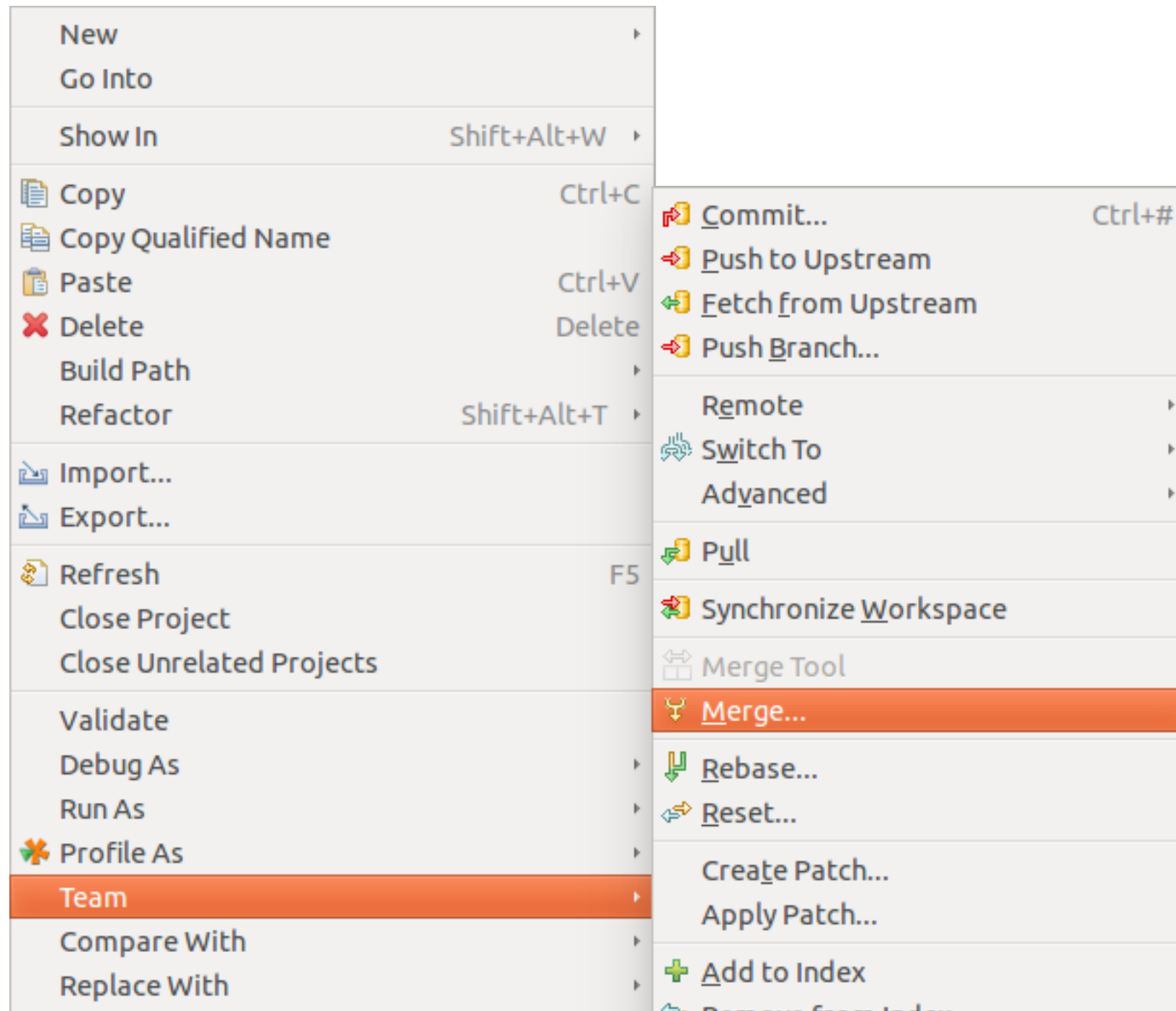
## 15. Starting a merge operation in Eclipse

### 15.1. Merge

Eclipse supports merging of branches to add the changes committed on one branch into another branch.

Checkout the branch into which you want to merge the changes into and select your project

and Team ▸ Mergeto start the merge dialog.



Restore from Local History...		Remove from Index
Plug-in Tools		Ignore
Configure		Show in Repositories View
Source		Show in History
Properties	Alt+Enter	Disconnect

×

□

Merge 'master'

Merge 'master'

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

type filter text

▼ Local

↕ change/27471/1 9278a6d Bug 434991 - [QuickAccess] Ctrl+3 quick access popup, w

☑

↕ master 525372c Bug 439244 - org.eclipse.equinox.launcher.source bundle missing i

▶ Remote Tracking

▶ Tags

Merge options

- ☒ Commit (commit the result)
- ☐ No commit (prepare merge commit, but don't commit yet)
- ☐ Squash (merge changes into working directory, but don't create merge commit)

Fast forward options

- ☒ If a fast-forward, only update the branch pointer
- ☐ If a fast-forward, create a merge commit
- ☐ If not a fast-forward, fail

Cancel Merge

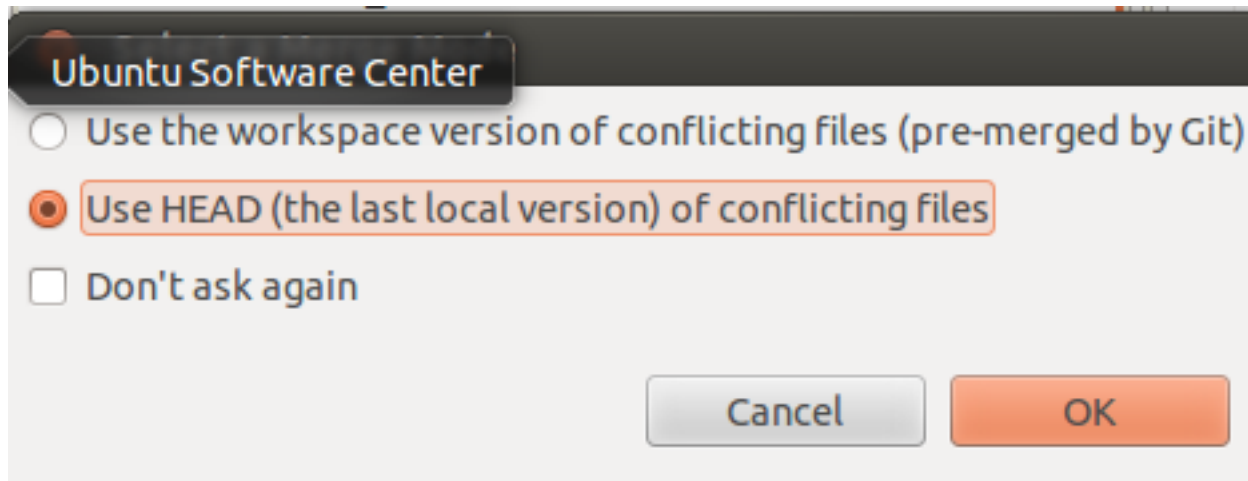
## 15.2. Solving merge conflicts

If during a Git operation, two changes are conflicting, you have to solve these conflicts manually. Eclipse Git highlights the affected files in the *Package Explorer* or *Project Explorer* view.

Eclipse Git supports the resolution of these merge conflicts. To trigger this via the explorer views, right-click on a file with merge conflicts and select **Team** → **Merge Tool**.

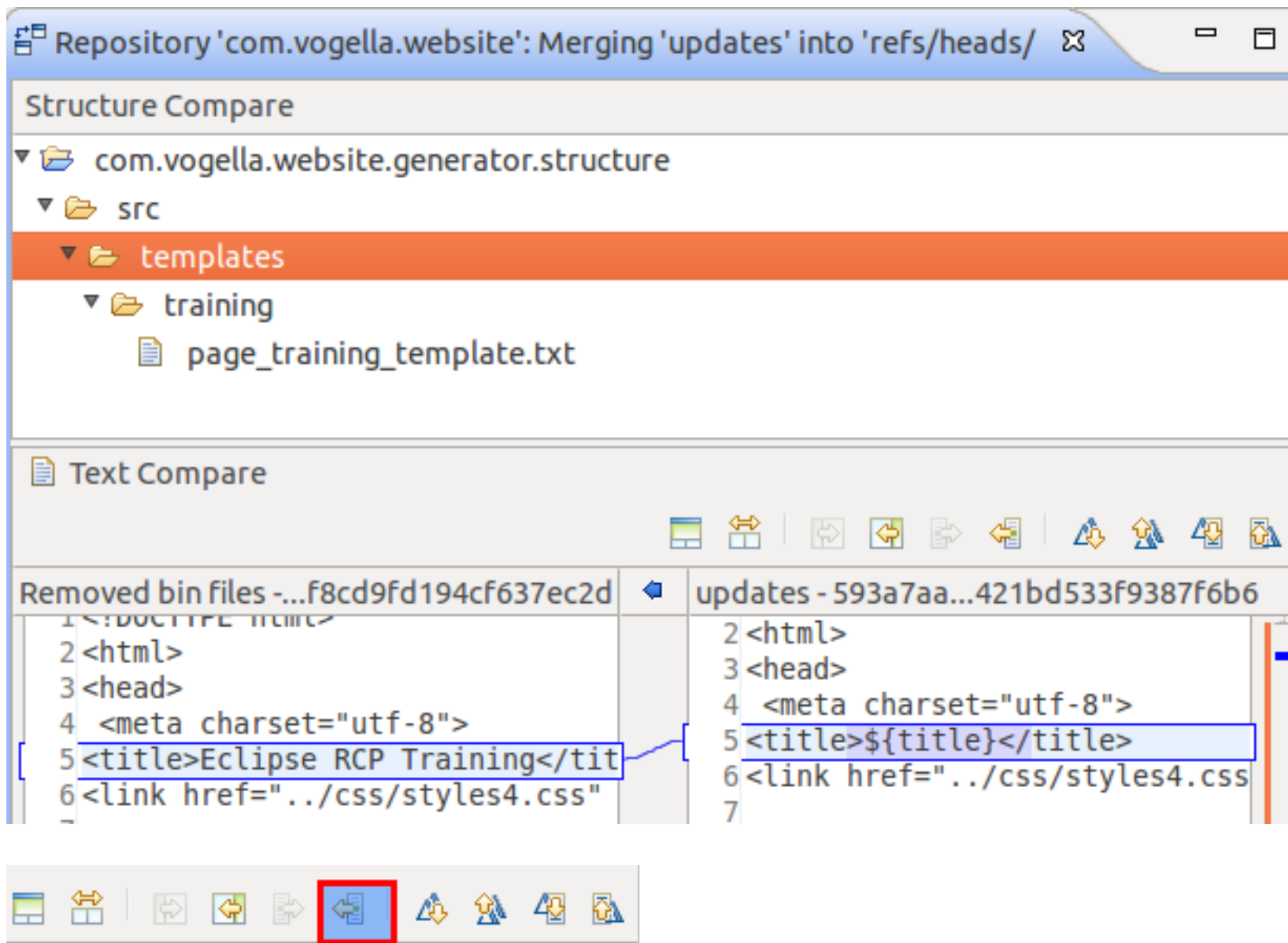
You can also use the *Git staging* view to find the conflicting files. In large projects this is usually faster than navigating the *Package Explorer* or *Project Explorer* view.

This opens a dialog, asking you which merge mode you would like to use. The easiest way to see the conflicting changes is to use the *Use HEAD (the last local version) of conflicting files* as merge mode. This way you see the original changes on the left side and the conflicting and non-conflicting changes on the right side.

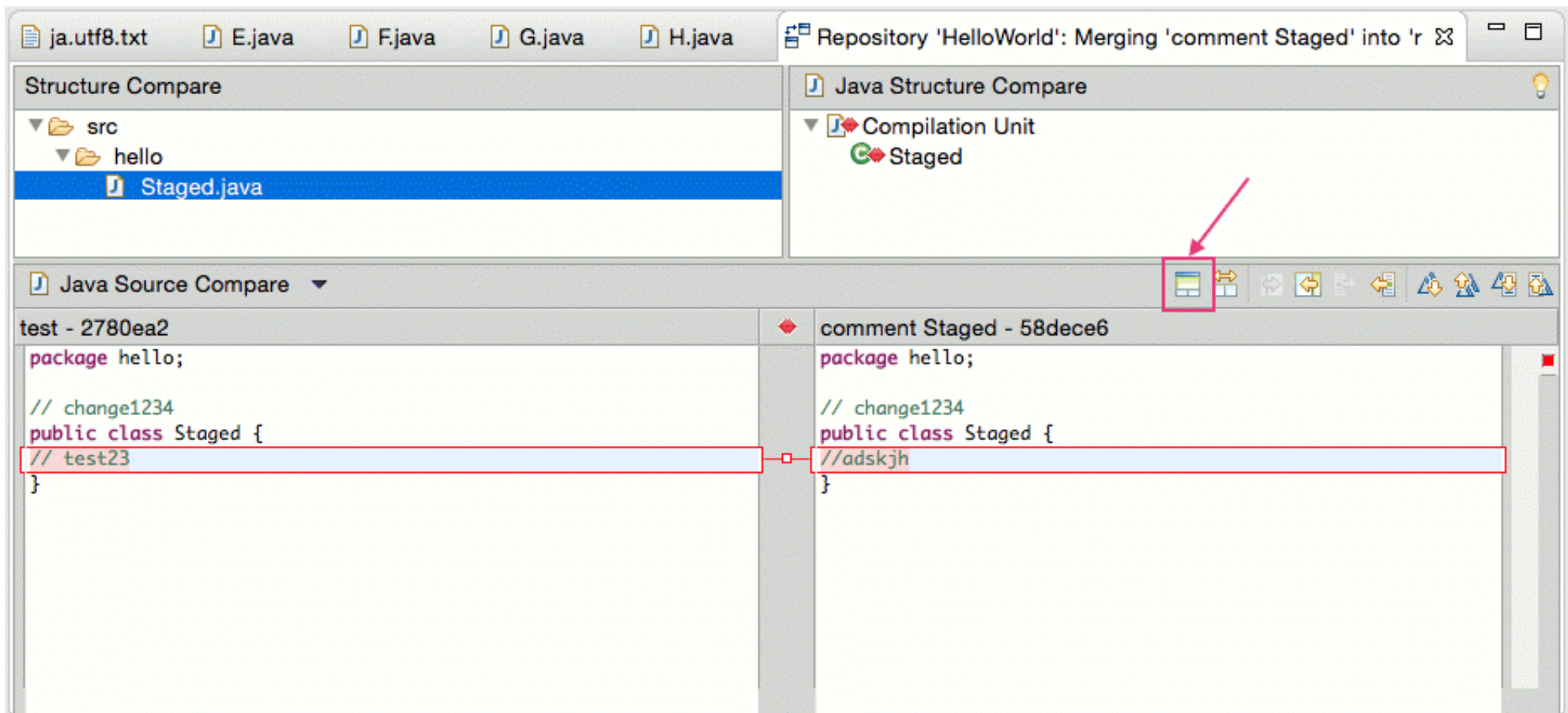


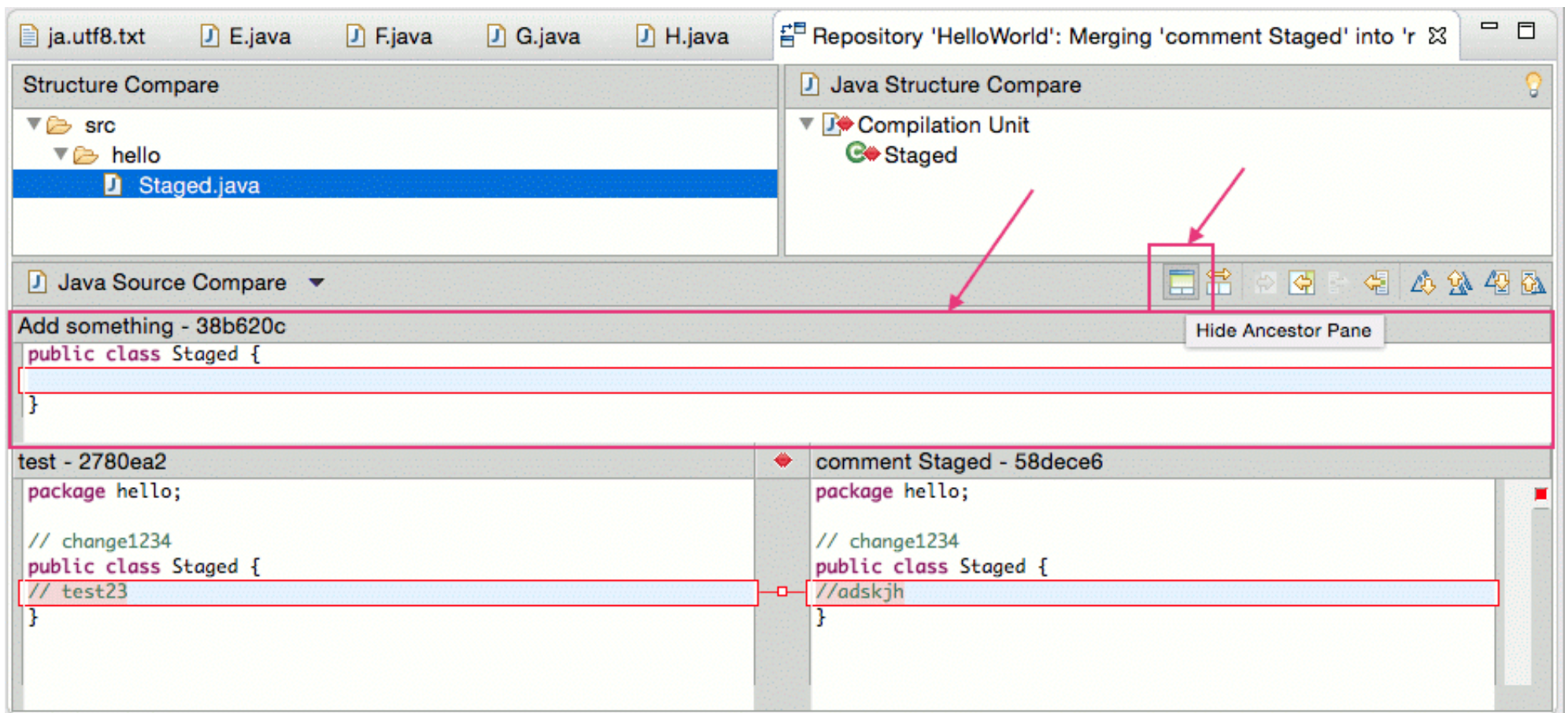
You can manually edit the text on the left side or use the Copy current change from right to left button to copy the changes from right to left.





Eclipse also allows to show the common ancestor of both commits to make the merge easier. Press the Hide/Show Ancestor Pane button for that. This is demonstrated by the following screenshots.





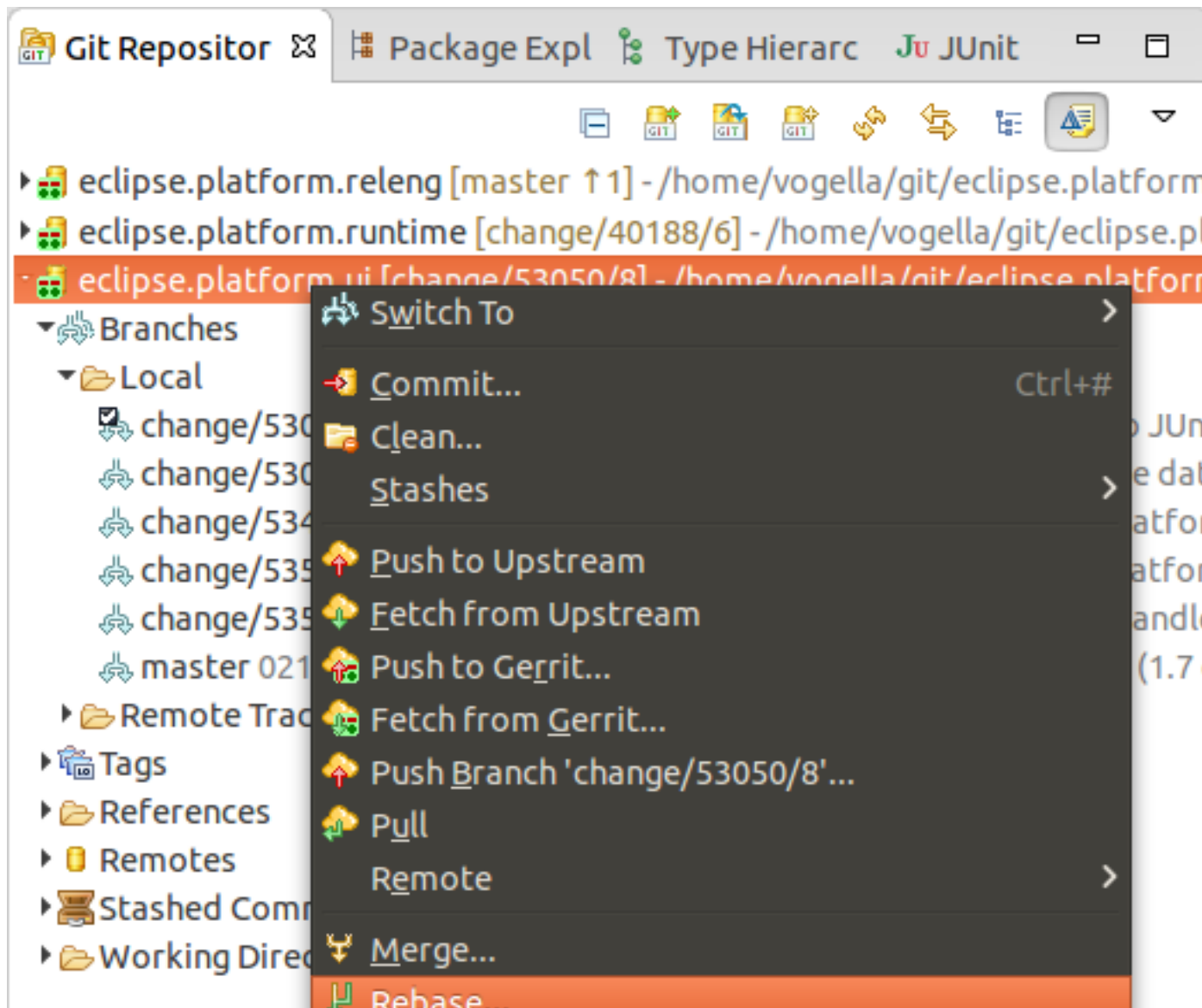
Once you have manually merged the changes, select `Team > Add` from the context menu of the resource to mark the conflicts as resolved and commit the merge commit via `Team > Commit`.

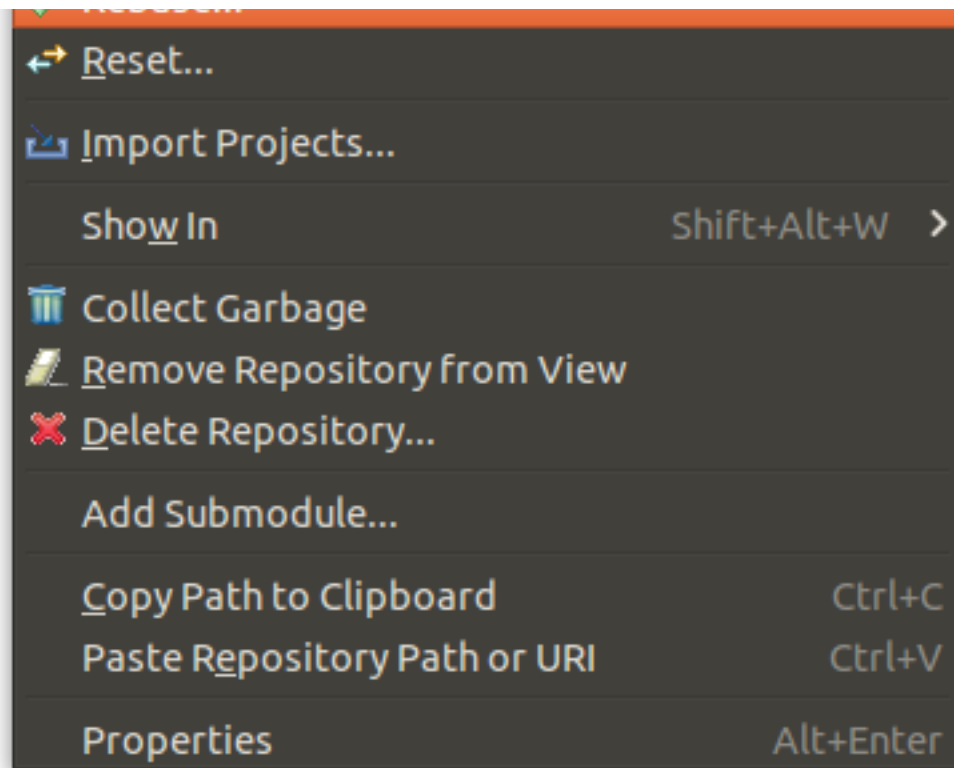
## 16. Rebasing a branch onto another branch



The *Git Repositories* view allows you to rebase your currently checkout branch onto another branch.

Right-click on a repository node and select **Rebase** as depicted in the following screenshot.





In the following dialog you can select the branch onto which you want to rebase.

## Rebase the 'change/53050/8' branch onto another branch

Select a branch other than the 'change/53050/8' branch

type filter text

### Local

- ☒ change/53050/8 42bac7c Bug 474132 - [Tests] Move o.e.ui.tests to JUnit 4
- ☐ change/53096/2 21c2232 Bug 472673, 335792 - Add generics to the databinding.p
- ☐ change/53457/2 26b158a Bug 474544 - Move all test plug-ins of platform.ui to Ja
- ☐ change/53574/1 3b4f69d Bug 474544 - Move all test plug-ins of platform.ui to Ja
- ☐ change/53574/3 ab2d3a1 [Bug 468773] Perspective should be a HandlerContain
- ☐ master 021fed3 Bug 475361 - Remove redundant type arguments (1.7 or higher)

### Remote Tracking

- ☐ origin/bdealwis/353628-cocoa-bindingtable c4ca0bc Bug 353628 - No binding ta
- ☐ origin/bdealwis/359778-activities e7e8ffb A first cut at providing activity suppor
- ☐ origin/bdealwis/362147-feature-split 969a67d Revise to include equinox.console
- ☐ origin/bdealwis/380570-package-versions de6c8ea Bug 393739 - Remove packag

☐ Rebase interactively

☐ Preserve merges during rebase

Cancel

Rebase



You can also select the branch to rebase onto from the *Branches* node of the tree directly.

If the rebase was successful a dialog is shown. You have to resolve rebase conflicts if they occur. After resolving them, select **Rebase** ☐ **Continue**.

If you want to skip the conflicting commit and continue with the rebase operation use **Rebase** ☐ **Skip**.

To cancel the rebase operation select **Rebase** ☐ **Abort**.

---

## 17. Git reset and Git reflog

### 17.1. Moving the branch pointer with Git reset

The *History* view allows you to reset your current branch to a commit. Right-click on a certain commit and select **Reset** and the reset mode you would like to use.

Project: org.eclipse.e4.tools [org.e...

Id	Message
459b023	handleraddon vogella/handler...
f6f5ed1	Bug 398078 - Reduce dependencies in
b931bdb	Bug 395240 - Remove the <name> tag
bcdf13b	Bug 395710 - Wizard for creating new
3fc36ee	Bug 395371 - Wizard to convert part

## 17.2. Finding "invisible" commits with the Reflog view

Commits are not visible in the Git history if they can't be reached from a branch or tag. This might happen during a reset, commit amend or rebase operation. By default, such invisible commits are removed after two weeks by the Git system.

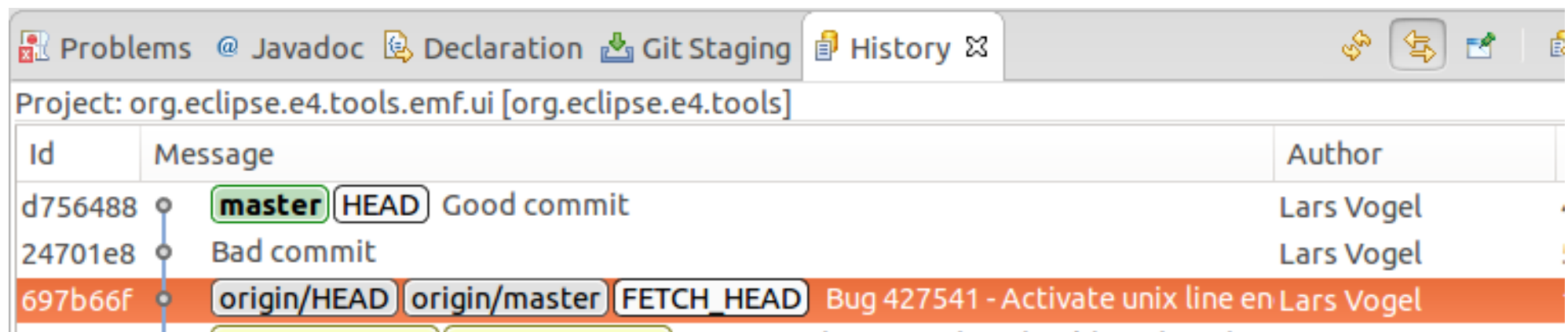


The *Git Reflog* view keeps track of the movements of the HEAD pointer and the movements of each branch. This view allows you to find a commit again, e.g., if you used the `git reset --hard` command to remove certain commits.

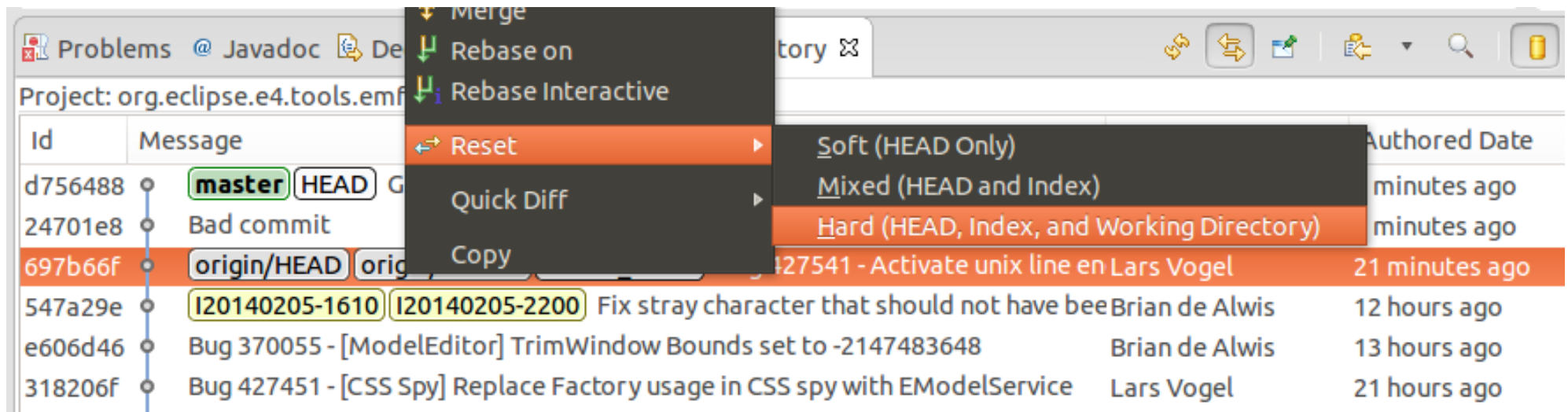
## 18. Using git cherry-pick

In the *History* view, you can cherry-pick a commit via the context menu.

A combination of `git reset` and `git cherry-pick` allows you to move the changes done in a commit to another branch. Assume you have a bad commit which you would like to remove from the history of branch followed by a good commit. This situation is depicted in the following screenshot.



For this you would make a hard reset on the commit of origin/master.



Afterwards you can cherry-pick the good commit.

Problems @ Javadoc Declaration Git Staging History

Project: org.eclipse.e4.tools.emf.ui [org.eclipse.e4.tools]

Id	Message
d756488	ORIG_HEAD Good
24701e8	Bad commit
697b66f	master origin/HEAD Bug 427541 - Ac
547a29e	I20140205-1610 I2
e606d46	Bug 370055 - [Mode
318206f	Bug 427451 - [CSS S
d6a158c	Bug 426343 - Activa
33c5a59	Bug 426343 - Activa

commit d75648803de00ccb90979  
 Author: Lars Vogel <Lars.Vogel@cloudera.com>  
 Committer: Lars Vogel <Lars.Vogel@cloudera.com>  
 Parent: 24701e868189d6c2c3a7

Open in Commit Viewer  
 Checkout  
 Push Commit...  
 Create Branch...  
 Delete Branch  
 Rename Branch...  
 Create Tag...  
 Create Patch...  
 Cherry Pick  
 Revert Commit

Problems @ Javadoc Declaration Git Staging History

File: org.eclipse.e4.tools.emf.ui/src/org/eclipse/e4/tools/emf/ui/internal/imp/RegistryUtil.java [org.eclipse.e4.tools.emf.ui]

Id	Message	Author
47878fe	master HEAD Good commit	Lars Vogel
b902205	ORIG_HEAD Good commit	Lars Vogel
24701e8	Bad commit	Lars Vogel
697b66f	origin/HEAD origin/master FETCH_HEAD Bug 427541 - Activate unix line	Lars Vogel

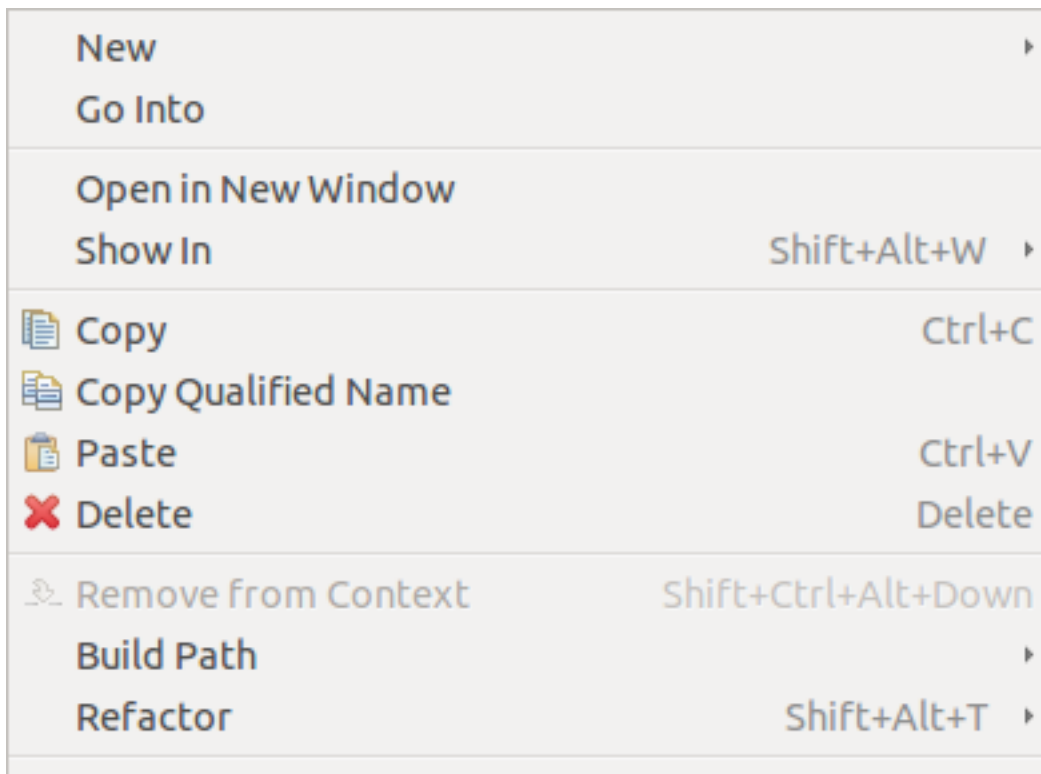
This results in a history without the bad commit.



















You can do the same with [Adjusting the history with interactive rebase](#).

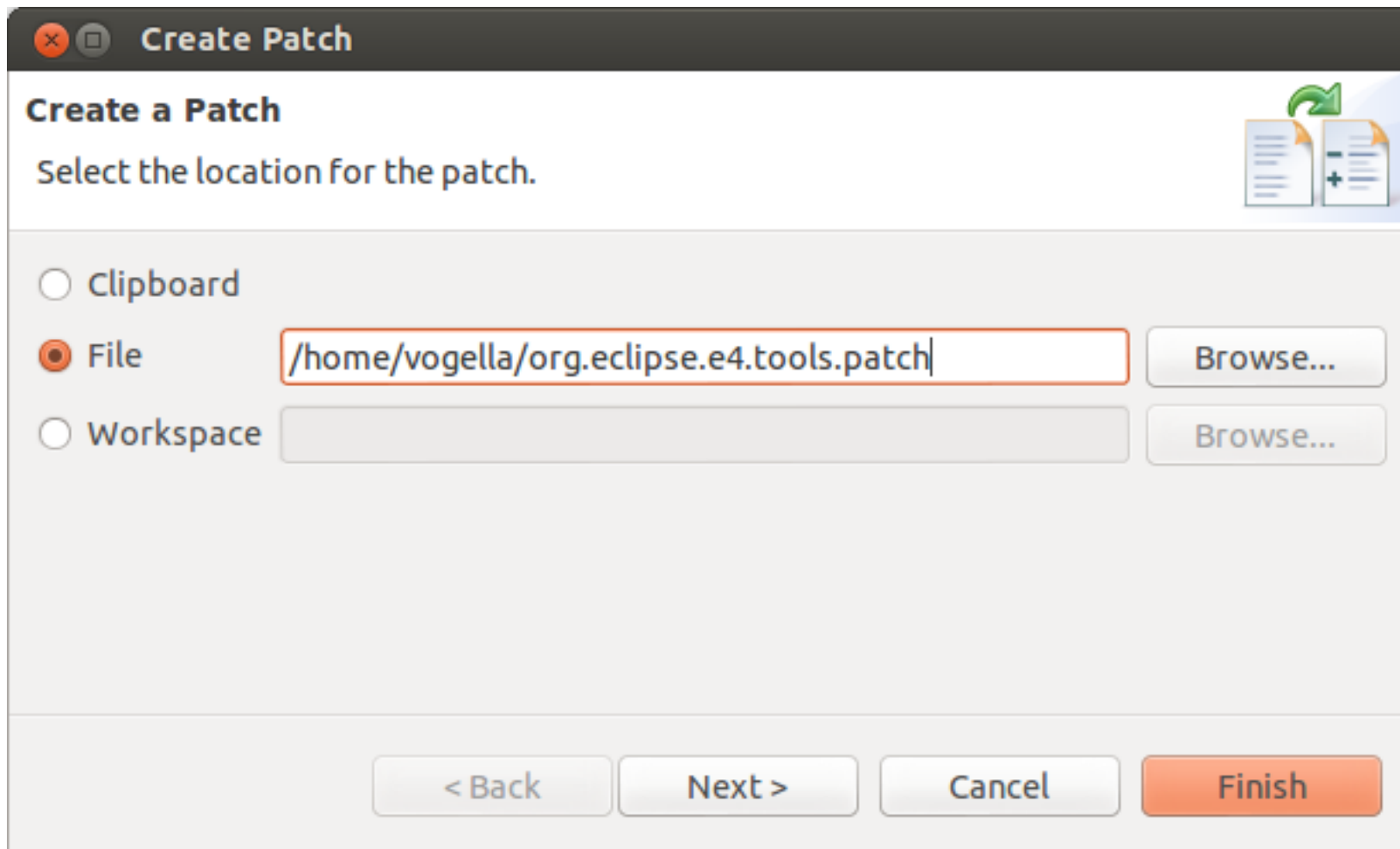
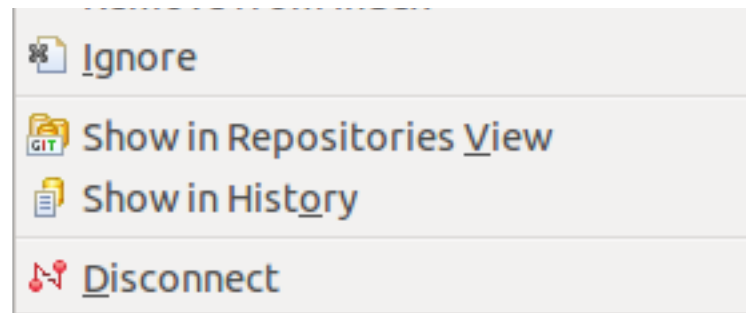
## 19. Creating patches

To create a patch for a set of changes with Eclipse Git, select the resources for which you want to create a patch in the *Package Explorer* view, right-click and select **Team** → **Create Patch**.



 Import...	
 Export...	
 Refresh	F5
Close Project	
Close Unrelated Projects	
Assign Working Sets...	
Validate	
Debug As	
Run As	
Team	
Compare With	
Replace With	
Restore from Local History...	
Configure	
Source	
Properties	Alt+Enter

 Commit...	Ctrl+#
 Push to Upstream	
 Fetch from Upstream	
 Push Branch...	
Remote	
 Switch To	
Advanced	
 Pull	
 Synchronize Workspace	
 Merge Tool	
 Merge...	
 Rebase...	
 Reset...	
Create Patch...	
Apply Patch...	
 Add to Index	
 Remove from Index	



The resulting file can be used to get applied to another Gitrepository, viaTeam ☐ Apply

Patch. You can also apply the patch on a system where Git isn't installed at all, i.e., you don't need a Git repository to apply a patch.

---

## 20. See Git information line by line (aka git blame)

Eclipse allows to display the information which commit and person change a line. To enable this, right-click on your file and select Team ▸ Show Annotations.

Afterwards, you can place the mouse on the left side of the editor. A popup dialog shows the commit information and the change applied by the shown commit.



```

56 * <code>null</code>.
57 *
58 * @return the working set icon or <code>null</code>.
59 * @since 2.1
60 * @deprecated use {@link #getImageDescriptor()} instead
61 */
62 @Deprecated

```

Commit 7e26a4a ([open commit](#)) ([show in history](#))

Author: Lars Vogel <Lars.Vogel@gmail.com> 3/27/14 12:48 PM

Bug 431179 - Add missing @Override and @Deprecated annotations to org.eclipse.ui.workbench

Change-Id: I7a59eb6f206a8687d4c72a748554a969a7d9ef40

Signed-off-by: Lars Vogel <Lars.Vogel@gmail.com>

Diff to 3a3b6e0 Bug 429885 - [Import/Export] Import wizard doesn't work well with "Copy projects i

@@ -63,3 +63,4 @@

```

74     */
75 -    public ImageDescriptor getImage();
76 +    @Deprecated
77 +    public ImageDescriptor getImage();

```



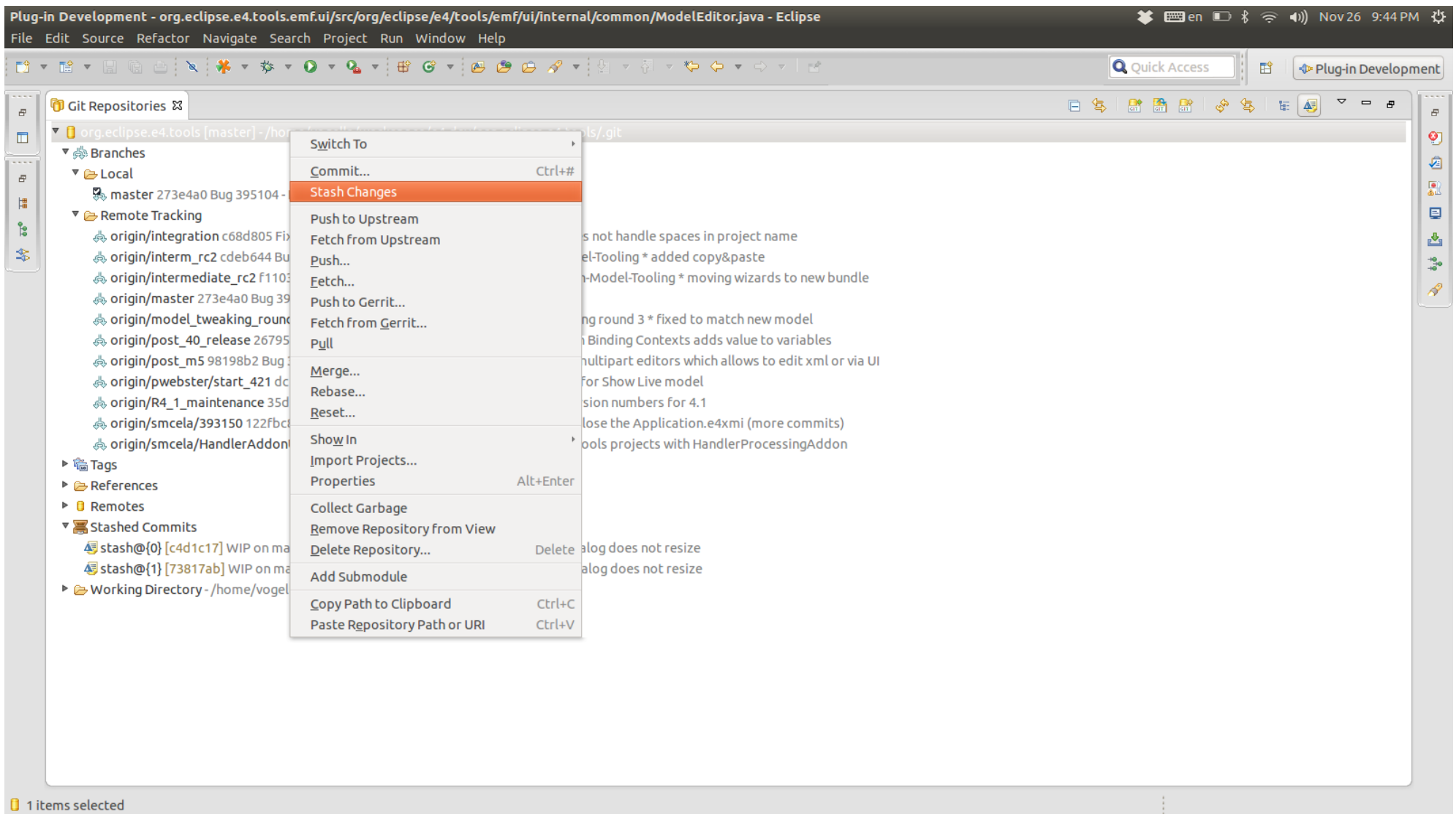
To ignore whitespace changes in the Git blame annotations in Eclipse, select Window ▸ Preferences ▸ Team ▸ Git and select *Ignore whitespace changes*.



---

# 21. Stash via the Git repository view

The `git stash` command is available in the *Git repositories* view. Right-click on your Git repository and select *Stash Changes*.



## ▼ Stashed Commits

- stash@{0} [c4d1c17] WIP on master: 273e4a0 Bug 395104 - Find Handler Class Dialog does not resize
- stash@{1} [73817ab] WIP on master: 273e4a0 Bug 395104 - Find Handler Class Dialog does not resize

Apply Stashed Changes	
Delete Stashed Commit...	Delete
Paste Repository Path or URI	Ctrl+V

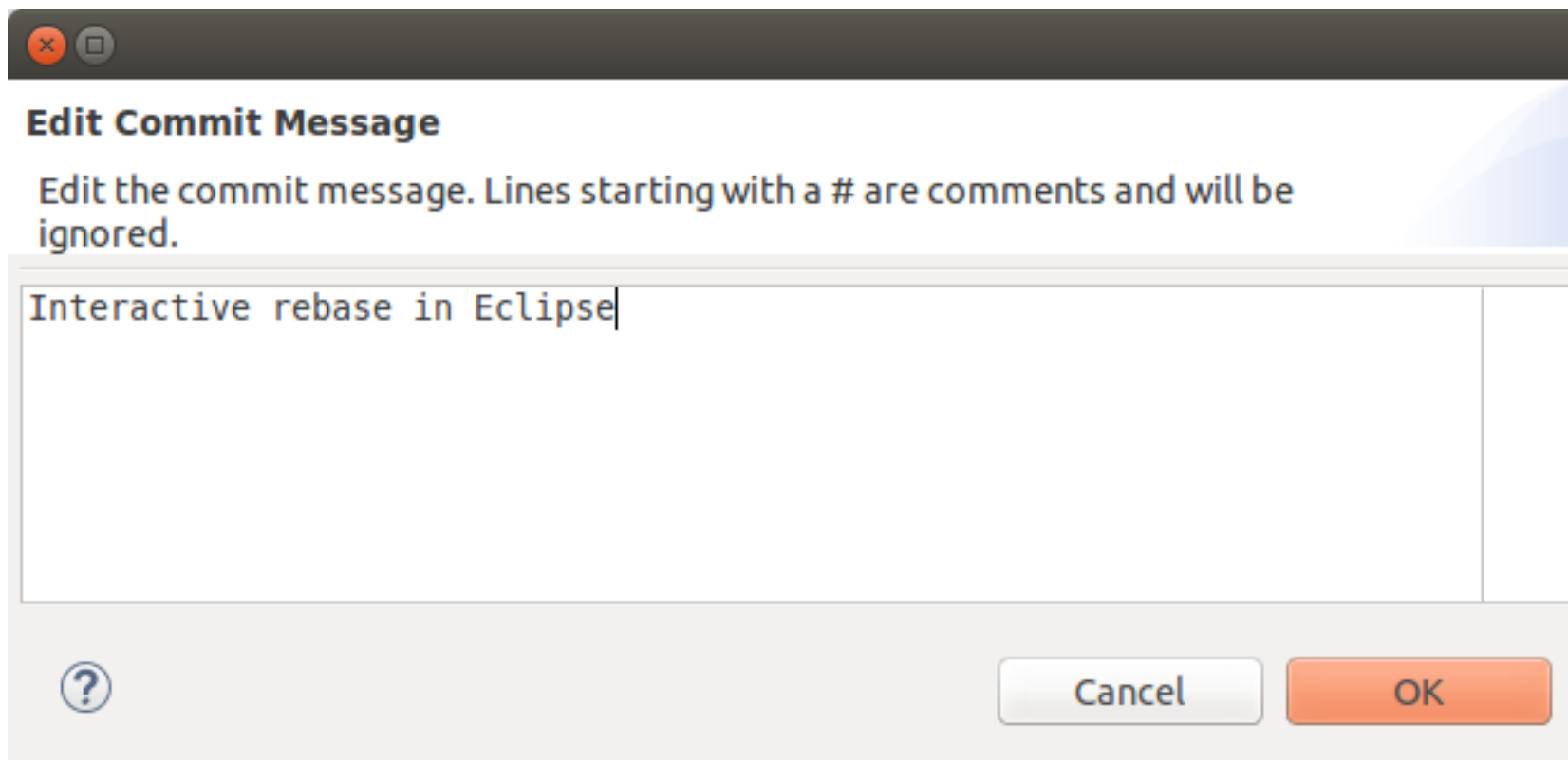
## 22. Adjusting the history with interactive rebase

### 22.1. Support for interactive rebase in Eclipse

Git allows to adjust the existing commit history the interactive rebase functionality. You can start an interactive rebase via the *History* view. The execution is done via the *Git Interactive Rebase* view.

### 22.2. Actions available via the History view

To reword a commit, right-click on it in the *History* view and select **Modify** ▢ **Reword** to change the commit message.



You can squash several commits by selecting them in the *History* view. Select afterwards the Modify ▾ Squash menu entry from the context menu.

History Problems Search Console Progress Git Staging Call Hierarch Plug-in Depe Error Log Git Inter

File: com.vogella.tutorials/82\_Book\_Eclipse\_IDE/gitexercises/052\_rebase.xml [com.vogella.tutorials]

Id	Message	Author	Authc
ab87fca [git] E		Lars Vogel	2 seco
0aa7e0d [git] G		Lars Vogel	2 min
297c1e0 [asciid		Lars Vogel	8 hou
894f864 [ide] A	g to the book	Lars Vogel	2 days
e4b6110 [rcp] R		Lars Vogel	2 days
8e264d3 [rcp] A		Lars Vogel	2 days
3d1363t [rcp] P		Lars Vogel	2 days
6e127e3 [rcp] H		Lars Vogel	5 days

- Open
- Open in Text Editor
- Open in Commit Viewer
- Compare with Each Other
- Revert Commit
- Quick Diff
- Modify
- Import Changed Projects
- Copy

Ctrl+C

Squash

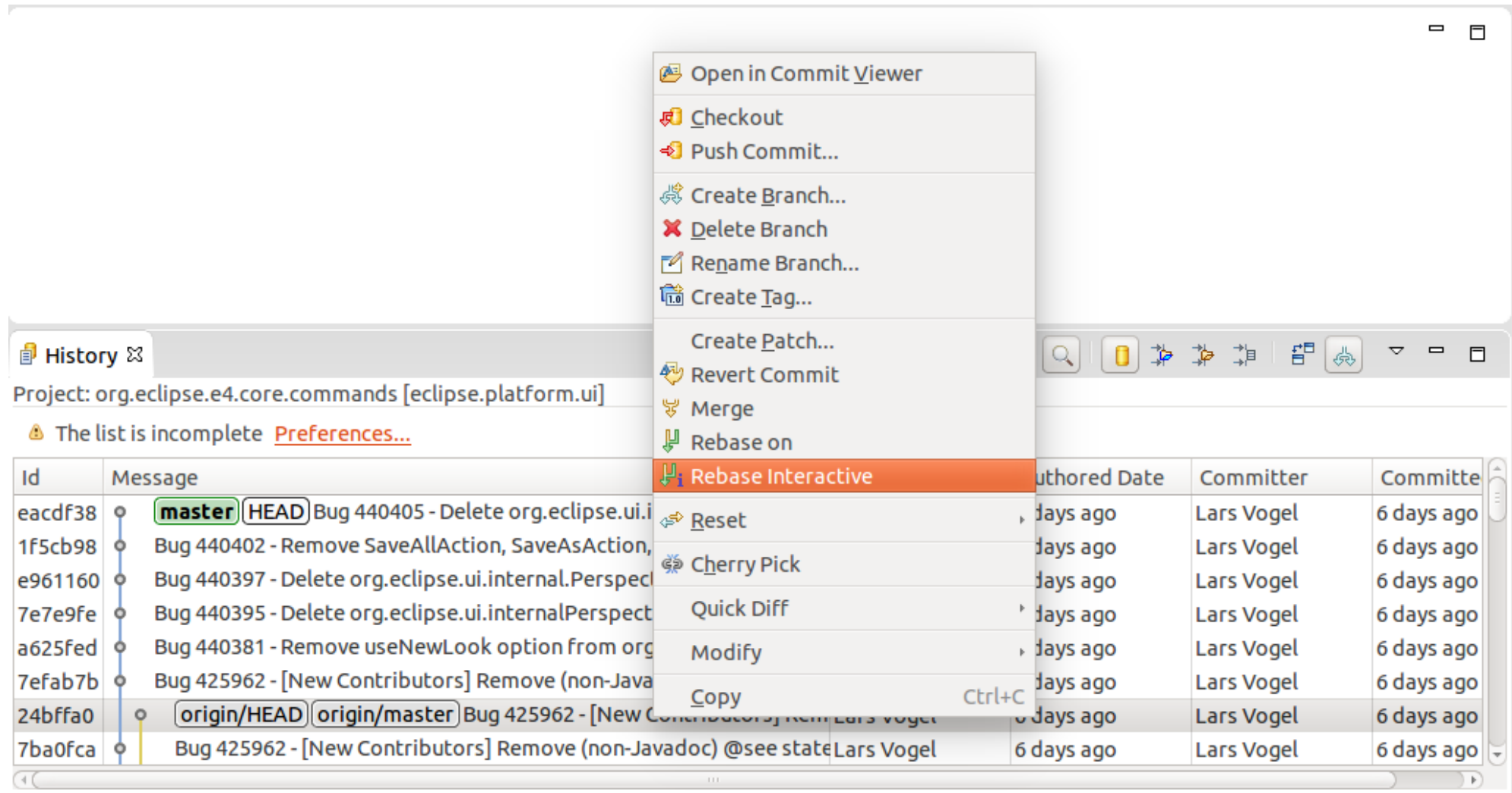
The above options are simplified ways to do an interactive rebase.

## 22.3. Using the Git Interactive Rebase view

The *Git Interactive Rebase* view allow you perform the full interactive rebase functionality. This includes changing the order of commits or combining, removing and adjusting commits.

To start the full interactive rebase open the *History* view and click *Rebase Interactive* on the context menu. Select the last commit preceding the oldest commit you want to rewrite.

Often this is the one origin/master points to.



This opens the *Git Interactive Rebase* view. It shows the rebase plan populated with the commits to be modified. They are sorted in topological order of the sequence in which they will be processed. This order is the reverse order which you see via the `git log` command or in the *History* view. The initial action for all commits is "Pick".

Git Interactive Rebase
 Git Staging
 History

eclipse.platform.ui

Refresh
 Abort
 Skip
 Continue
 Start

Pick
 Skip
 Edit
 Squash
 Fixup
 Reword
 Move up
 Move down

Status	Action	Commit ID	Message	Author	Authored Date	Commit
*	PICK	7ba0fca	Bug 425962 - [New Contributors] Remove (non-Javadoc) @see statements if @Ove	Lars Vogel	6 days ago	Lar
*	PICK	7efab7b	Bug 425962 - [New Contributors] Remove (non-Javadoc) @see statements if @Ove	Lars Vogel	6 days ago	Lar
*	PICK	a625fed	Bug 440381 - Remove useNewLook option from org.eclipse.e4.ui.workbench.swt.i	Lars Vogel	6 days ago	Lar
*	PICK	7e7e9fe	Bug 440395 - Delete org.eclipse.ui.internal.PerspectiveSwitcher	Lars Vogel	6 days ago	Lar
*	PICK	e961160	Bug 440397 - Delete org.eclipse.ui.internal.PerspectiveBarManager	Lars Vogel	6 days ago	Lar
*	PICK	1f5cb98	Bug 440402 - Remove SaveAllAction, SaveAsAction, SaveAction	Lars Vogel	6 days ago	Lar
*	PICK	eacdf38	Bug 440405 - Delete org.eclipse.ui.internal.BaseSaveAction	Lars Vogel	6 days ago	Lar

The Eclipse Git tooling supports the following actions.

*Table 2. Interactive rebase actions*

Action	Description
pick	includes the selected commit, moving pick entries enables reordering of commits
skip	removes a commit

edit	amends the commit
squash	combines the changes of the commit with the previous commit and combines their commit messages
fixup	squashes the changes of a commit into the previous commit discarding the squashed commit's message
reword	similar to pick but allows modifying the commit message

Use this view to finalize the rebase plan. For example, you can reorder commits with the arrow buttons and select the rebase action you want to apply to the commit. The following screenshot demonstrates a possible selection.



Git Interactive Rebase
 Git Staging
 History

eclipse.platform.ui

Refresh
 Abort
 Skip
 Continue
 Start

Pick
 Skip
 Edit
 Squash
 Fixup
 Reword
 Move up
 Move down

Status	Action	Commit ID	Message	Author	Authored Date	Commit
•	SKIP	7ba0fca	Bug 425962 - [New Contributors] Remove (non-Javadoc) @see statements if @Ove	Lars Vogel	6 days ago	Lar
•	PICK	7efab7b	Bug 425962 - [New Contributors] Remove (non-Javadoc) @see statements if @Ove	Lars Vogel	6 days ago	Lar
•	REWORD	a625fed	Bug 440381 - Remove useNewLook option from org.eclipse.e4.ui.workbench.swt.i	Lars Vogel	6 days ago	Lar
•	PICK	7e7e9fe	Bug 440395 - Delete org.eclipse.ui.internal.PerspectiveSwitcher	Lars Vogel	6 days ago	Lar
•	FIXUP	e961160	Bug 440397 - Delete org.eclipse.ui.internal.PerspectiveBarManager	Lars Vogel	6 days ago	Lar
•	PICK	1f5cb98	Bug 440402 - Remove SaveAllAction, SaveAsAction, SaveAction	Lars Vogel	6 days ago	Lar
•	SQUASH	eacdf38	Bug 440405 - Delete org.eclipse.ui.internal.BaseSaveAction	Lars Vogel	6 days ago	Lar

When the rebase plan is finalized, click the Start button to start the interactive rebase command. Eclipse Git processes the plan. It stops at all commits with an action which needs user feedback. For example, the reword action which requires entering the new commit message. The dialog for changing the commit message is depicted in the following screenshot.

Edit Commit Message

Edit the commit message. Lines starting with a # are comments and will be ignored.

Bug 440381 - Remove useNewLook option from  
org.eclipse.e4.ui.workbench.swt.internal.copy.FilteredTree  
  
Also adjust the usage in ShowViewDialog  
  
Change-Id: I3ccf6c1b3adeb5de26965ca2a08f692108ceae71  
Signed-off-by: Lars Vogel <Lars.Vogel@gmail.com>

?

CancelOK

PickSkipEditSquashFixupRewordMove upMove down

Status	Action	Commit I	Message
✓	PICK	7efab7b	Bug 425962 - [New Contributors] Remove (non-Javadoc) @see statements if @Override is used
▶	REWORD	a625fed	<b>Bug 440381 - Remove useNewLook option from org.eclipse.e4.ui.workbench.swt.internal.copy.FilteredTree</b>
•	PICK	7e7e9fe	Bug 440395 - Delete org.eclipse.ui.internal.PerspectiveSwitcher
•	FIXUP	e961160	Bug 440397 - Delete org.eclipse.ui.internal.PerspectiveBarManager
•	PICK	1f5cb98	Bug 440402 - Remove SaveAllAction, SaveAsAction, SaveAction
•	SQUASH	eacdf38	Bug 440405 - Delete org.eclipse.ui.internal.BaseSaveAction

Here is the result of the rebase operation displayed in the *History* view.

Git Interactive Rebase   Git Staging   **History**

Project: org.eclipse.e4.core.commands [eclipse.platform.ui]

⚠ The list is incomplete [Preferences...](#)

Id	Message	Author	Authored Date	Committer	Committe
90c8144	<b>master</b> <b>HEAD</b> Bug 440402 - Remove SaveAllAction, SaveAsAction, S	Lars Vogel	6 days ago	Lars Vogel	10 minutes
7b06066	Bug 440395 - Delete org.eclipse.ui.internalPerspectiveSwitcher	Lars Vogel	6 days ago	Lars Vogel	11 minutes
b42d91f	Bug 440381 - Remove useNewLook option from org.eclipse.e4.ui.wor	Lars Vogel	6 days ago	Lars Vogel	11 minutes
7358227	Bug 425962 - [New Contributors] Remove (non-Javadoc) @see statem	Lars Vogel	6 days ago	Lars Vogel	12 minutes
24bffa0	<b>origin/HEAD</b> <b>origin/master</b> Bug 425962 - [New Contributors] Remov	Lars Vogel	6 days ago	Lars Vogel	6 days ago
a401b44	Bug 279902 - [Import/Export] WizardResourceImportPage.createDes	Marc-Andre Laper	4 months ago	Dani Megert	7 days ago
8736f40	<b>origin/R4_4_mai...</b> Increased bundle version for 4.4.1 fixes	Dani Megert	7 days ago	Dani Megert	7 days ago
9fd5799	Bug 436247 - [Themes] "Colors and Fonts" pref page fails with NPE	Cornel Izbasa	8 weeks ago	Dani Megert	7 days ago



If something goes wrong during the rebase operation, you can select *Abort* in order to stop the rebase operation and roll back to the starting point.

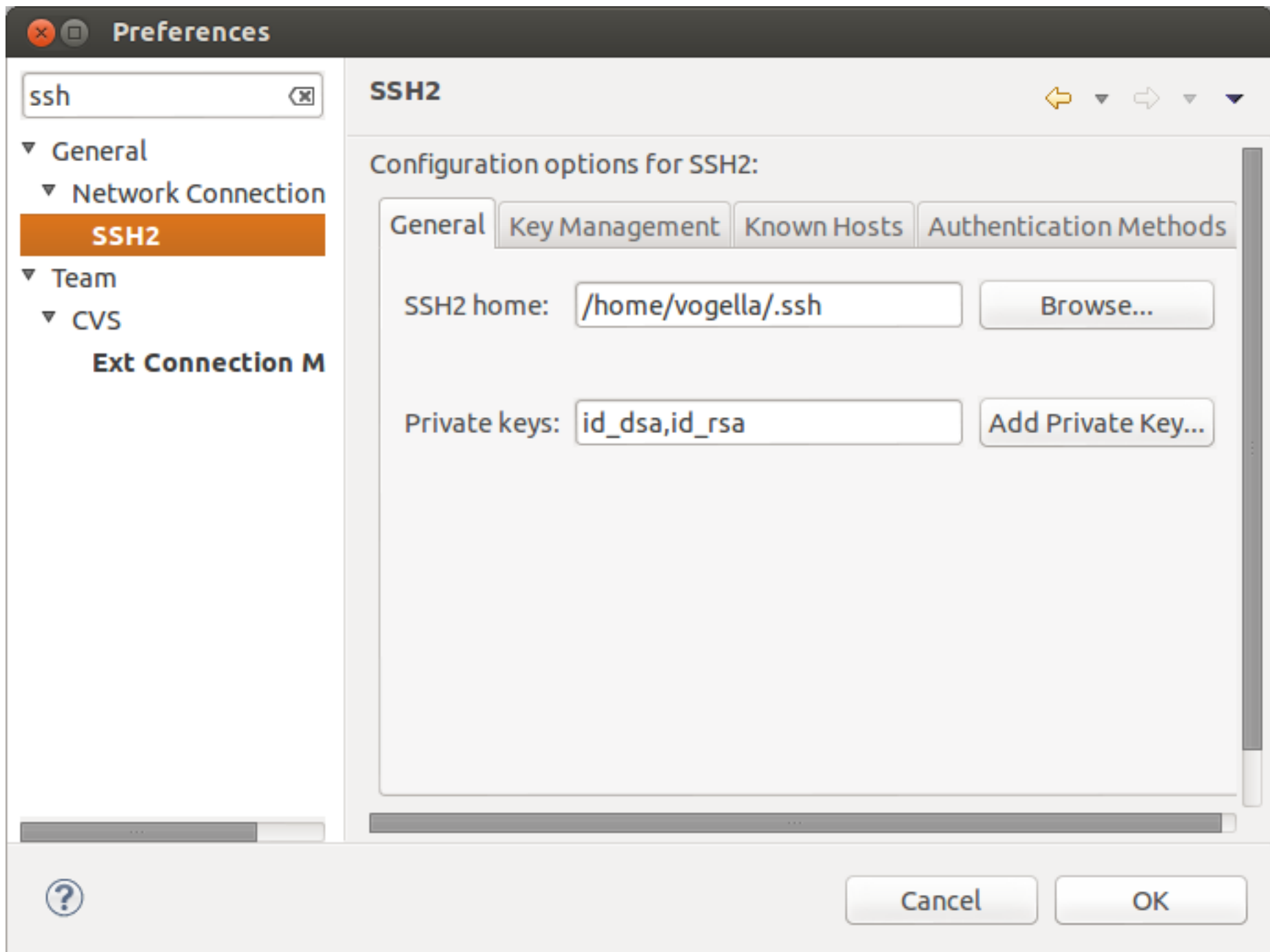


## 23. Using Eclipse Git with GitHub

### 23.1. Clone project

Copy the URL from GitHub and select in Eclipse from the menu the **File** **Import** **Git** **Projects from Git**

Eclipse fills out most of the fields based on the URL in the clipboard. Enter your user and password to be able to push to GitHub. Alternative you can also use an SSH key. You can configure Eclipse to know your SSH via the **Window** **Preferences** **General** **Network Connection** **SSH2** preference setting. This setting is depicted in the following screenshot.



## 23.2. Push changes

After you made changes and committed them to your local repository, you can select **Team** → **Push to upstream** on the project folder, to push your changes to your GitHub. This requires write access to the GitHub repository.

---

## 24. Eclipse support for SSH based authentication

You can create an SSH key pair in Eclipse for SSH based communication. This can be done via **Window** → **Preferences** → **General** → **Network Connection** → **SSH2**.

---

## 25. Eclipse integration with GitHub

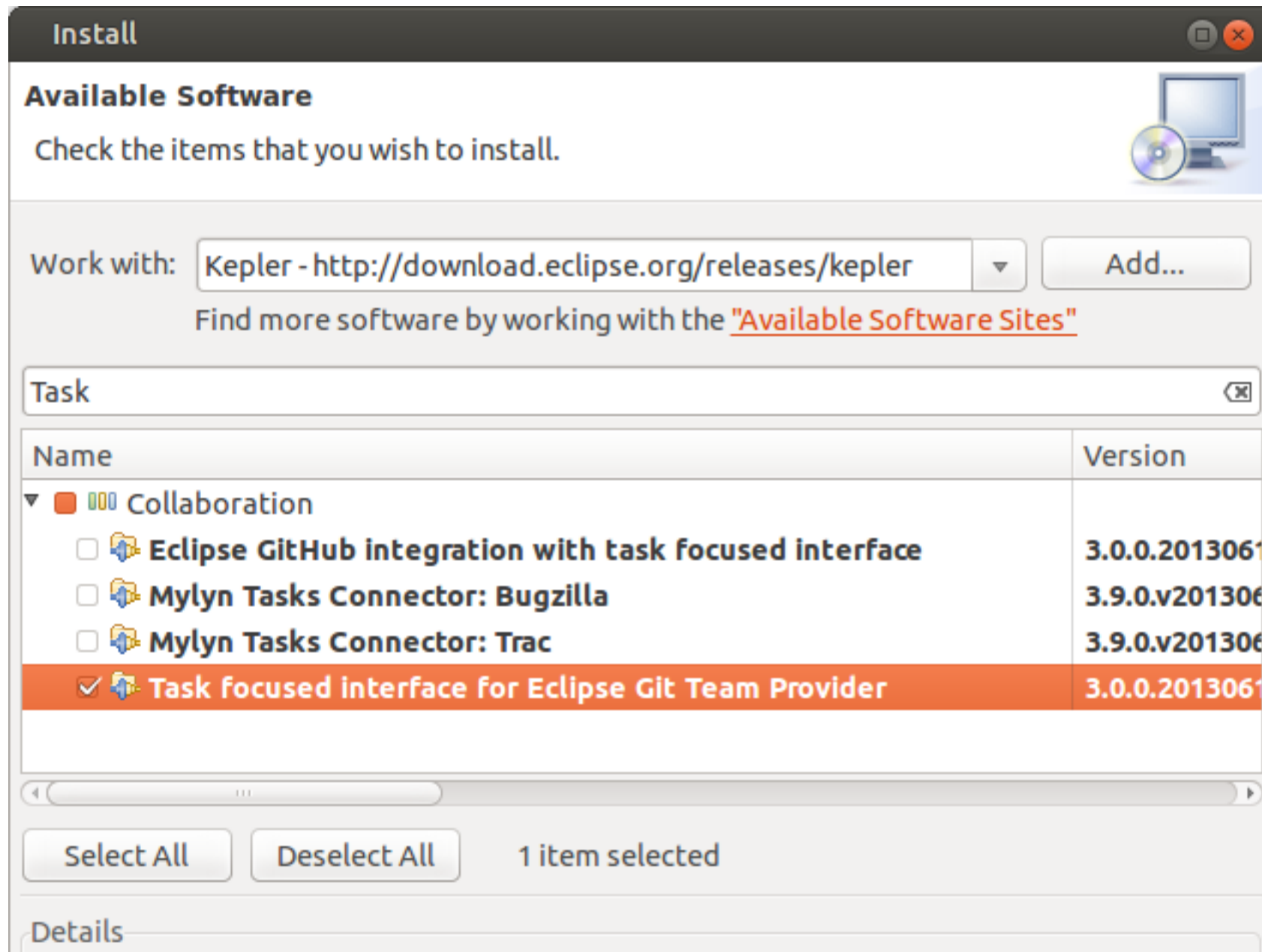
=== The Eclipse Mylyn plug-in

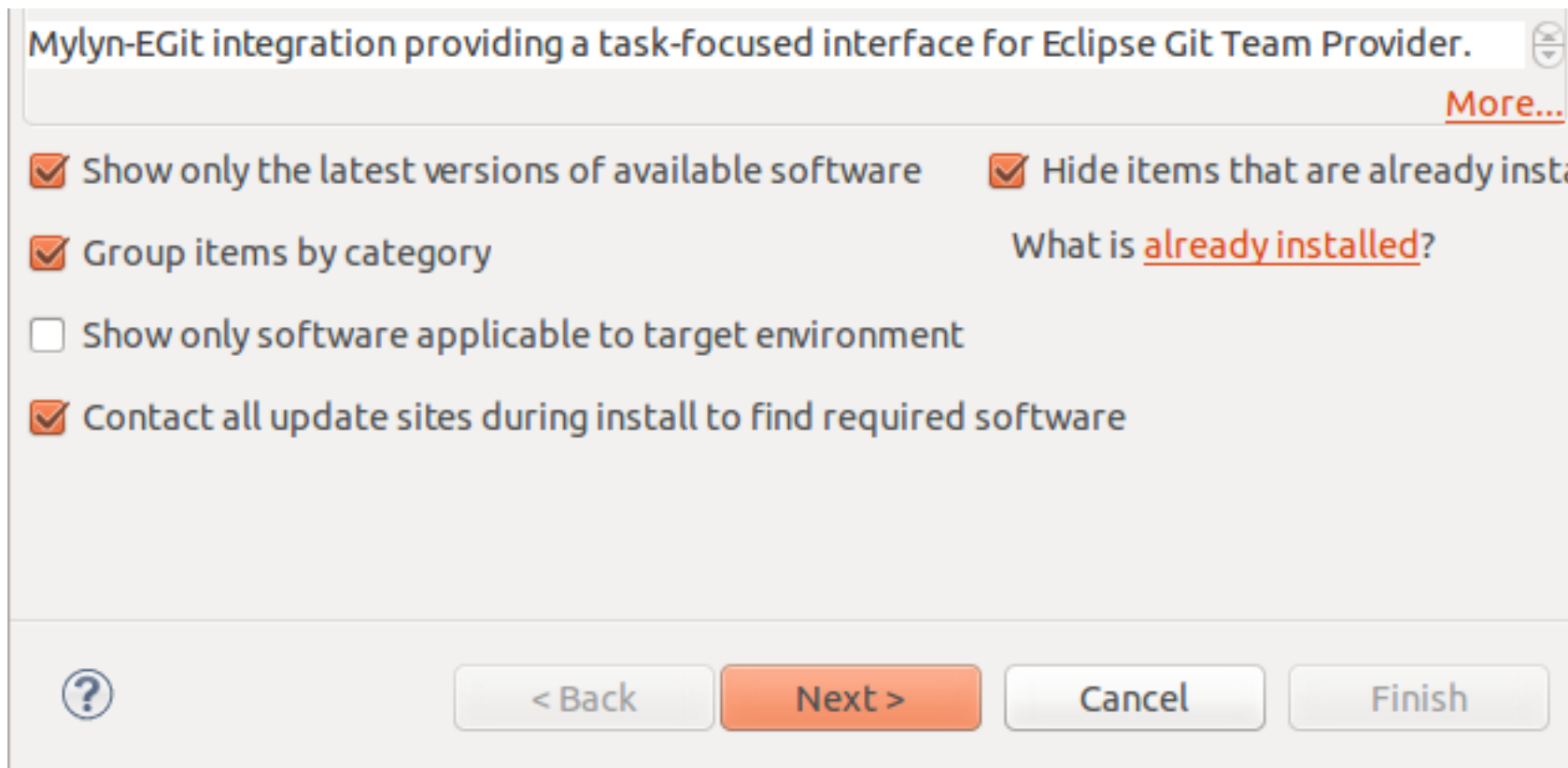
Eclipse Mylyn provides task integration for GitHub issues, GitHub pull and Gist (short text snippets) into the Eclipse IDE.

There is a GitHub connector for Mylyn available, please see [GitHub Mylyn User Guide](#) for

details.

You install it via Help → Install new Software and the update site of your release.

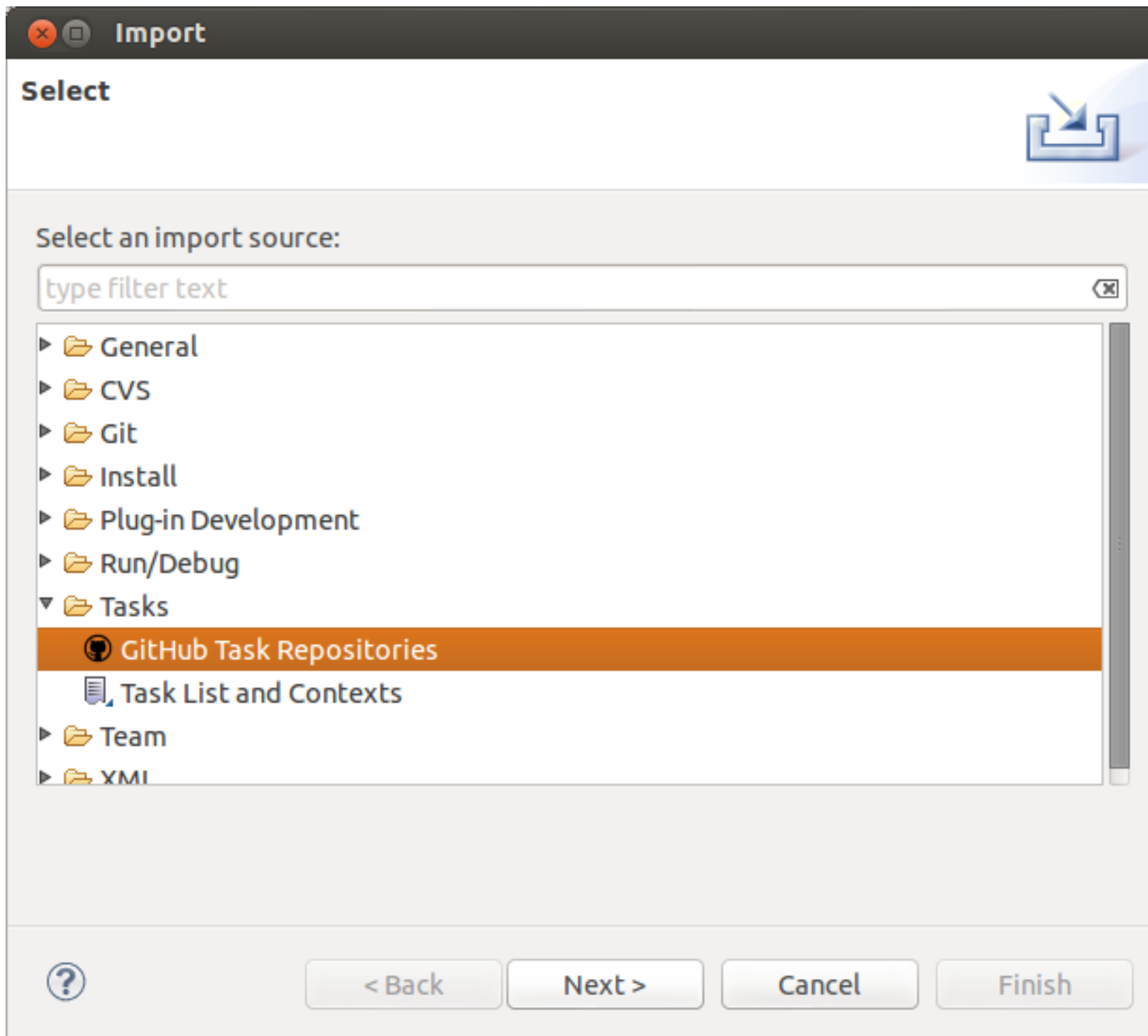




## 25.1. GitHub issue integration

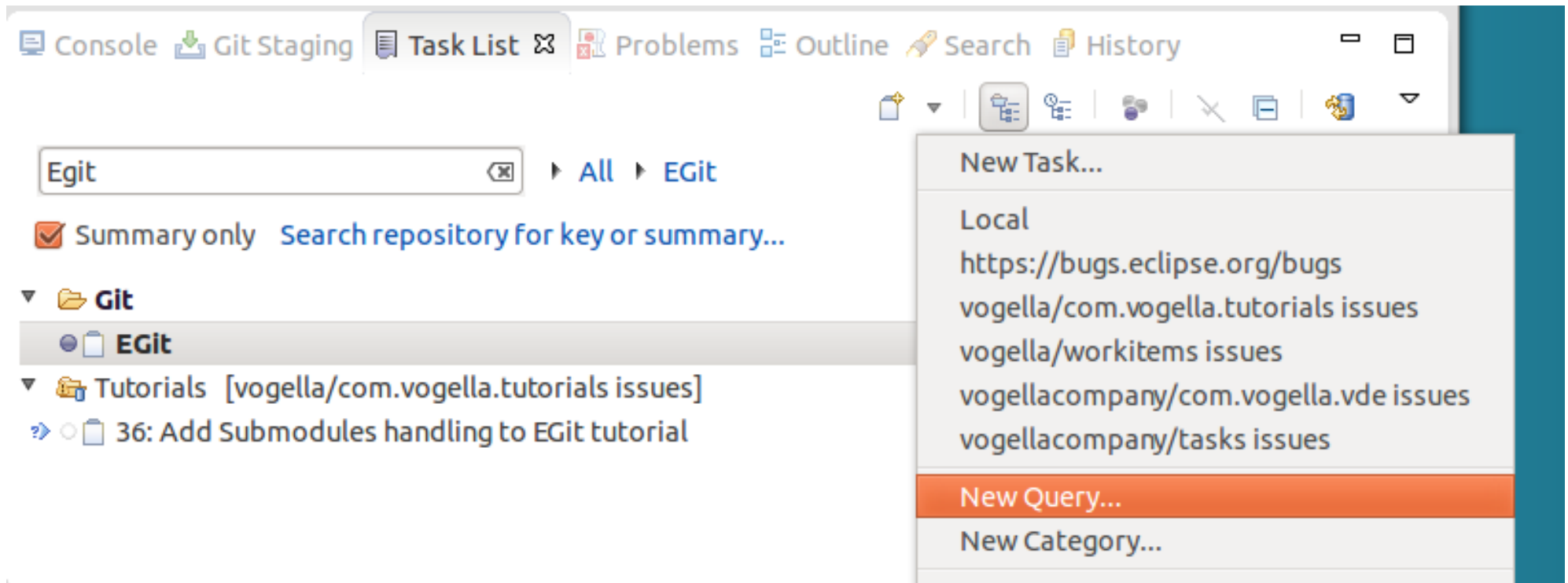
You can integrate your GitHub issues into Eclipse via **File** → **Import** → **Task** → **GitHub Task Repositories** and by following the wizard.

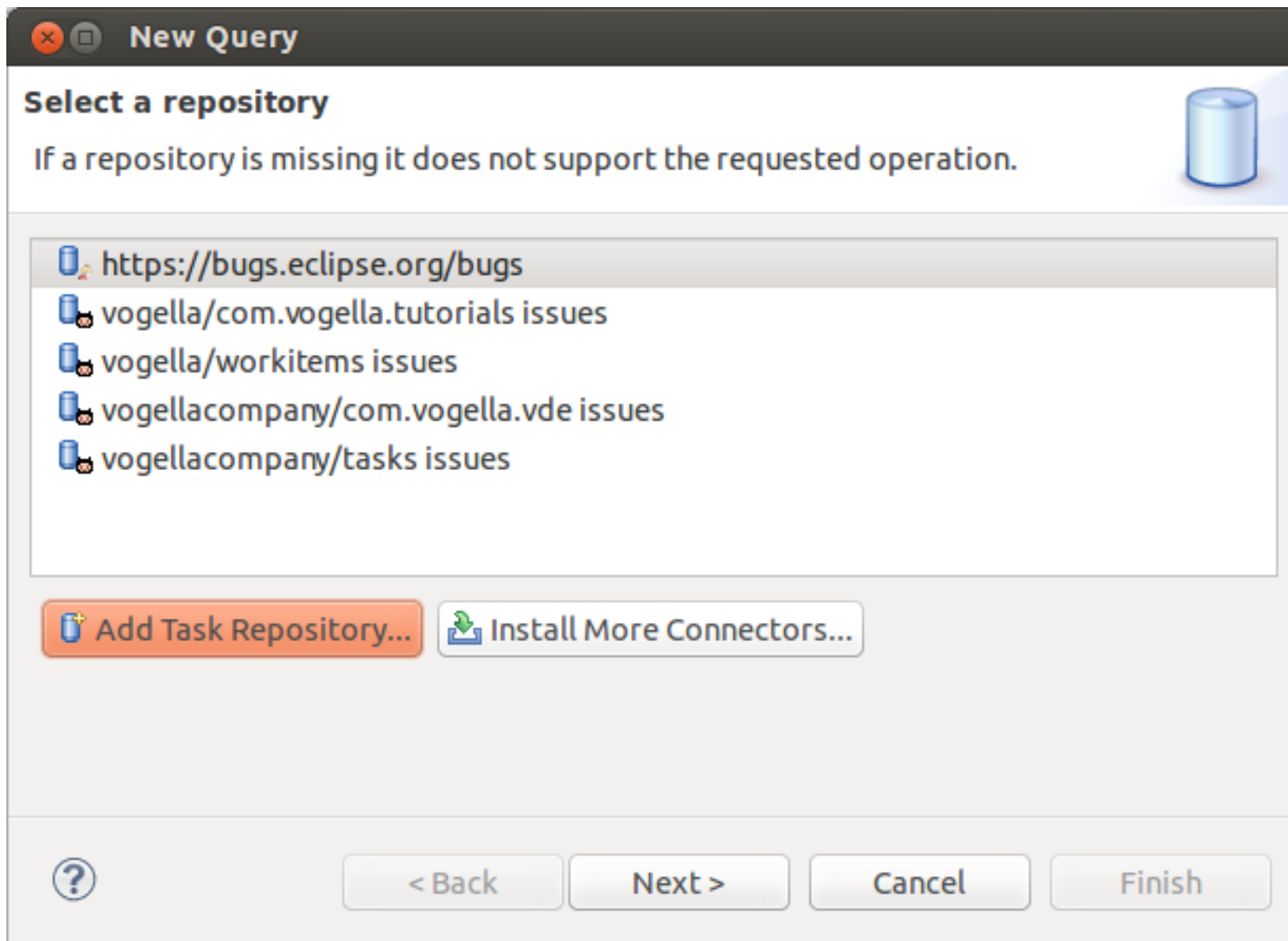


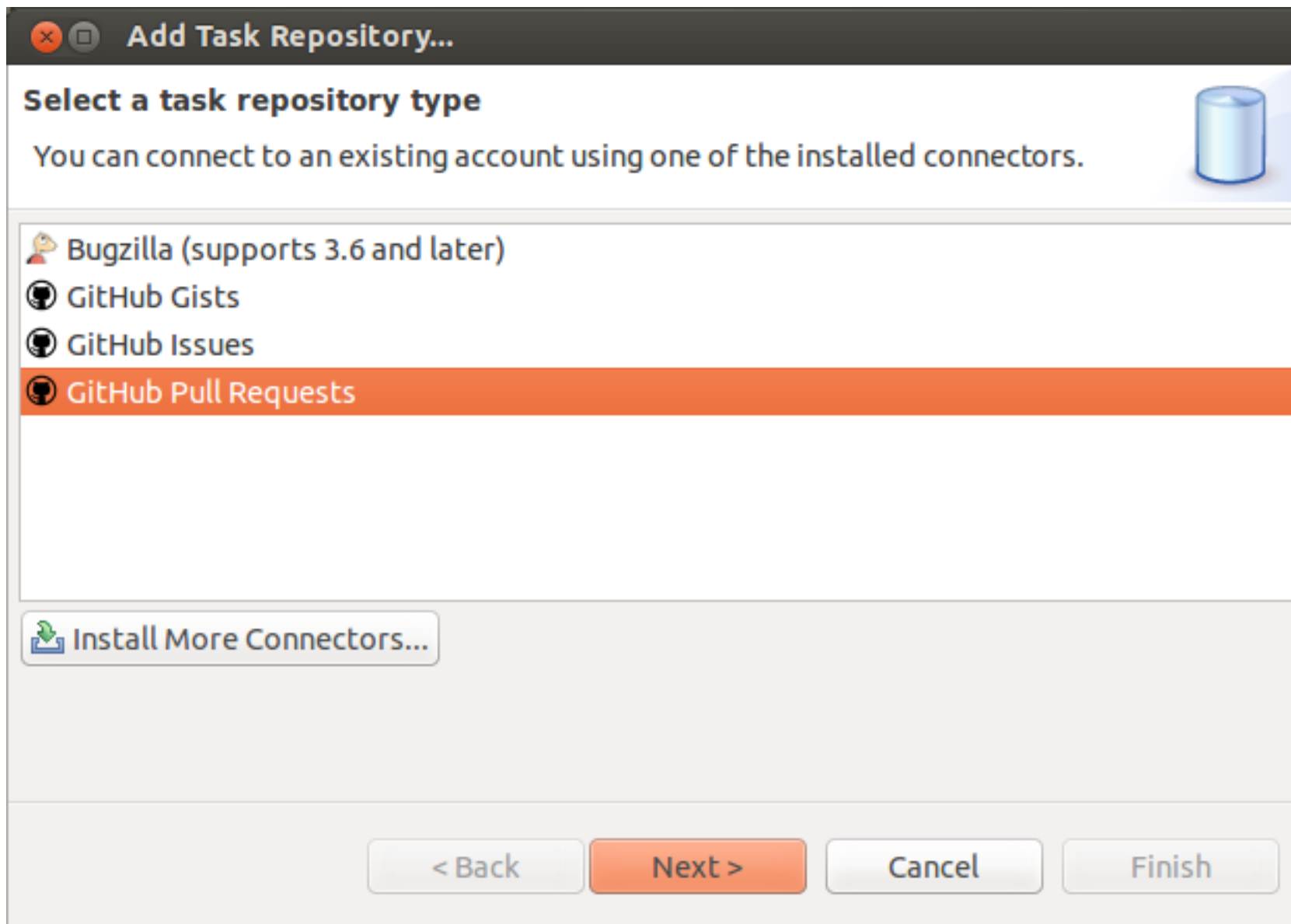


## 25.2. Manage pull requests in Eclipse

You can integrate your pull requests at GitHub into Eclipse by creating a new query from the *Task List* view. This is demonstrated via the following screenshots.










Edit Query

Enter query parameters

Pull request query




Title

Open Pull Requests

Status:

☒ Open

☐ Closed



< Back

Next >

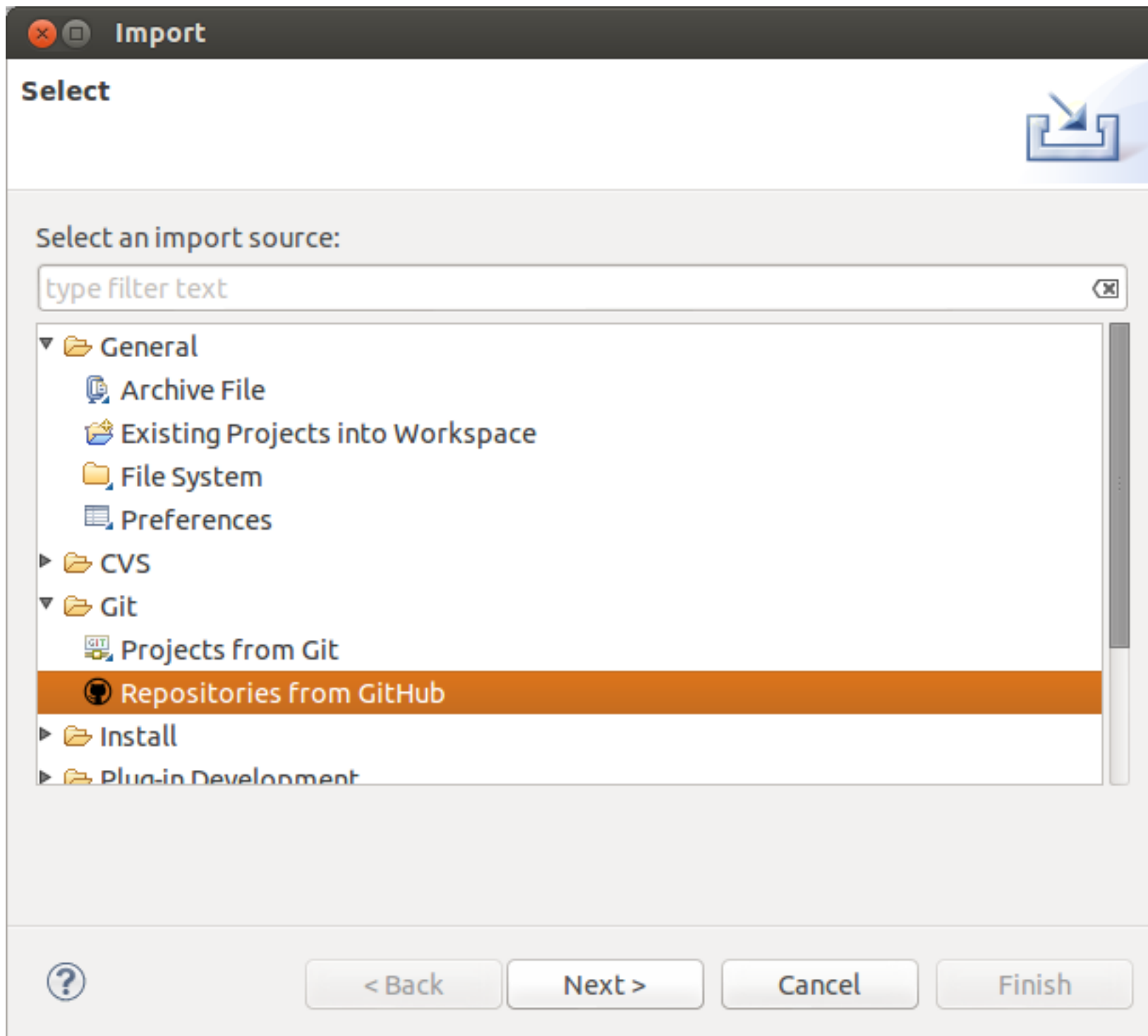
Cancel

Finish

NOTE:Unfortunately the GitHub connect does currently not support that you merge the pull request.

## 25.3. Import projects directly from GitHub

You can also import now directly projects from GitHub repositories.





## 25.4. More infos about the GitHub Mylyn integration

For a detailed description of the Mylyn and EGit integration please see the following webpage.

<http://wiki.eclipse.org/EGit/GitHub/UserGuide>

---

## 26. Writing good commit messages

### 26.1. Importance of Git commit messages

A *commit* adds a new version to the repository. This version is described by a *commit message*.

The `_commit message_` describes the changes recorded in a commit. It should help the user to understand the history of the repository.

A commit message should therefore be descriptive and informative without repeating the code changes.

### 26.2. Guidelines for useful commit messages

A commit message should have a header and a body. The header should be less than 50 with a maximum of 72 characters. The body should wrap its text at 72. The body is separated from the header by an empty line.

This ensures that the commit message is displayed well on the command line or in graphical tools.

The body describes the reason why the change was made. The changes in the file can be reviewed with the help of Git.

The commit message should be in present tense, e.g., "Adds better error handling" instead of "Added better error handling".

The last paragraph can also contain *metadata* as key-value pairs. This data is also referred to as the *commit message footer*.

This metadata can be used to trigger a certain behavior. For example, the *Gerrit* code review system uses the *Change-Id* key followed by a *change-id*. This *changed id* is used to identify to which review the message belongs.

The *commit message footer* can also have e.g., 'Signed-off-by'. Or it may link to a bug tracking system, e.g., 'Bug: 1234'.

## 26.3. Example message

The following can serve as an example for a commit message.

```
Short summary (less than 50 characters)
```

```
Detailed explanation, if required, line break at around 72 characters  
more stuff to describe...
```

```
Fixes: bug #8009
```

```
Change-Id: I26b5f96ccb7b2293dc9b7a5cba0760294afba9fd
```

## 26.4. Good and bad example for a Git history

The following listing shows the output of the `git log --oneline` command of a Git repository with bad commit messages. The first value in each line is the shortened SHA-1, the second the commit message. This history is not useful.

```
21a8456 update  
29f4219 update  
016c696 update  
29bc541 update  
740a130 initial commit
```

The next listing shows the history of another Git repository in which better commit messages have been used. This history already gives a good overview about the activities.

```
7455823 Adds search and filter to the model editor tree
9a84a8a Missing DynamicMenuContribution in child selector
952e014 Fixes spelling error in Toolbar/Add child
71eeea9 Adds option to import model elements from legacy RCP
123672c New Application wizard is missing dependencies
97cdb9a Creates an id for handlers
```

The above example also adds the corresponding bug number to the commit message. Some teams (like the Eclipse platform team) use this approach, others prefer to add the bug number to the commit messages.

## 27. Exercise: Working with a local Git repository in Eclipse

### 27.1. Target: Using Eclipse Git for a local repository

The following section explains how to create a local Git repository for one project with

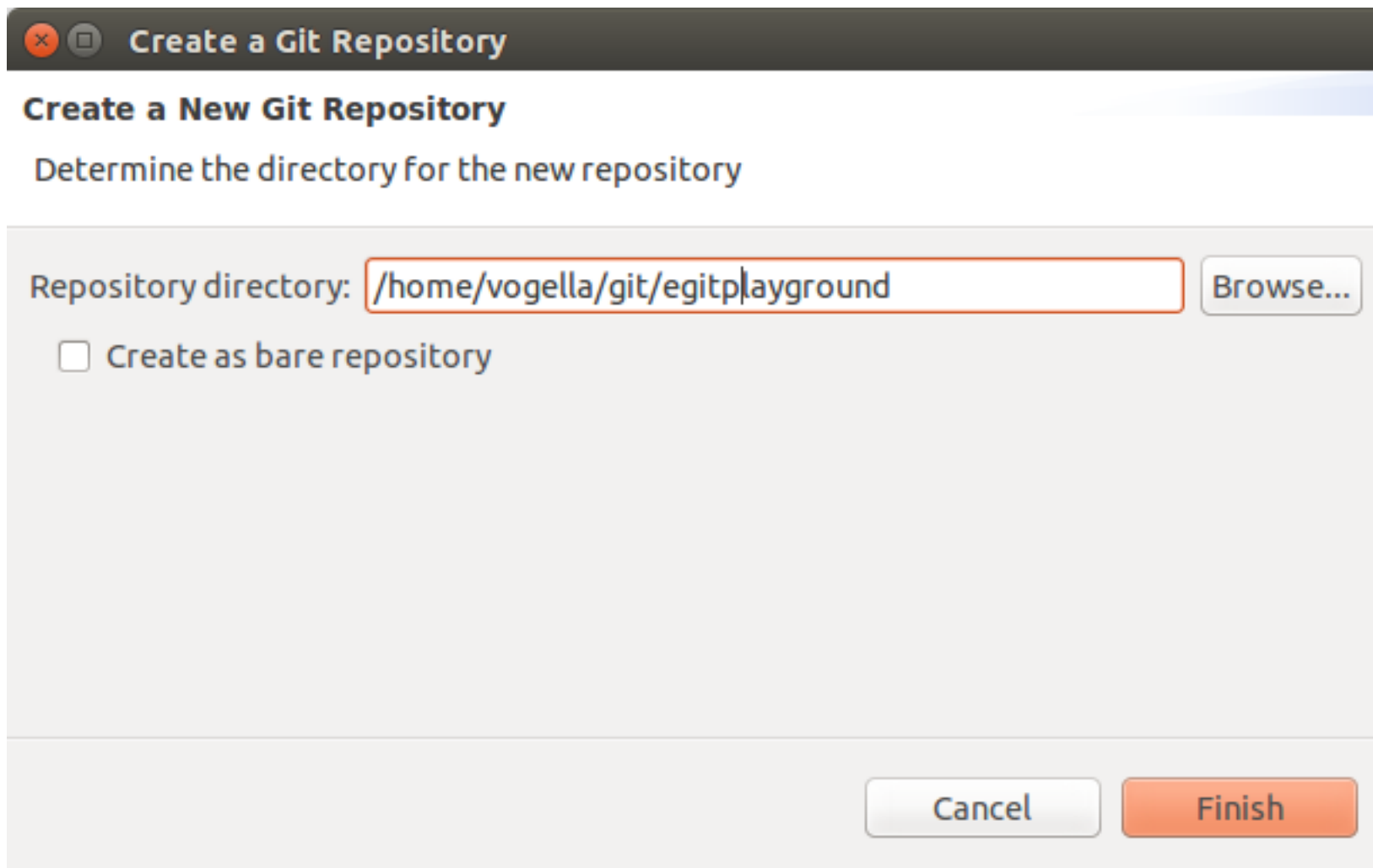
Eclipse. This allows you to keep track of your changes in the project. It also allows you to revert to another state at a later point in time.

## 27.2. Create a new Git repository via Eclipse

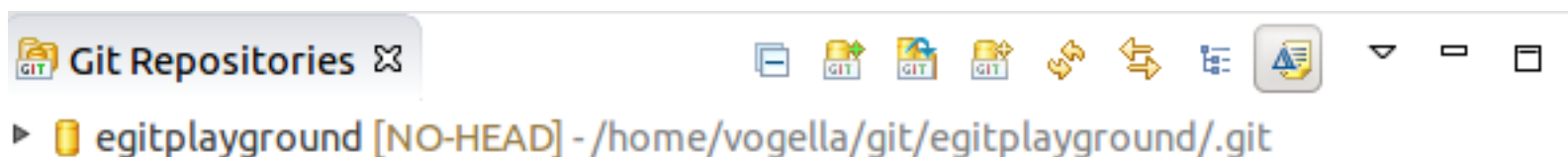
Open the *Git Repositories* view and select the *Create a new Git repository and add it to this view* option.



Select a new directory outside of your workspace. By convention this directory is a subdirectory in the *git* folder of the users home directory.



If you press the Finish button this dialog creates the directory and adds a reference to the new Git repository to the *Git Repositories* view.



## 27.3. Create .gitignore file

You want to configure Git to ignore the generated bin folder with the class files. Create for this purpose a *.gitignore* file in your Git repository with the following content.



Unfortunately Eclipse Git does not allow to create a file directly in the repository. You have to do this step outside of the Eclipse IDE, either via the command line or via your system project explorer.



Recent versions of MS Windows decided to prevent you from renaming a file in the file explorer without using a file extension. Create a file in *Notepad* or *Editor* (new name for Notepad) and select Save-As. Ensure you have removed the .txt extension.

```
bin
```

All files and directories which apply to the pattern described in this file will be ignored by Git. In this example, all files in the *bin* are ignored.



You can also configure Eclipse to automatically ignore derived resources,



e.g., class files via the Window ▢ Preferences ▢ Team ▢ Git ▢ Projects ▢  
Automatically ignore derived resources .. setting.

## 27.4. Creating an Eclipse project

Create a new Java project called `com.vogella.git.first` in Eclipse. Create the `com.vogella.git.first` package and the following class.

```
package com.vogella.git.first;

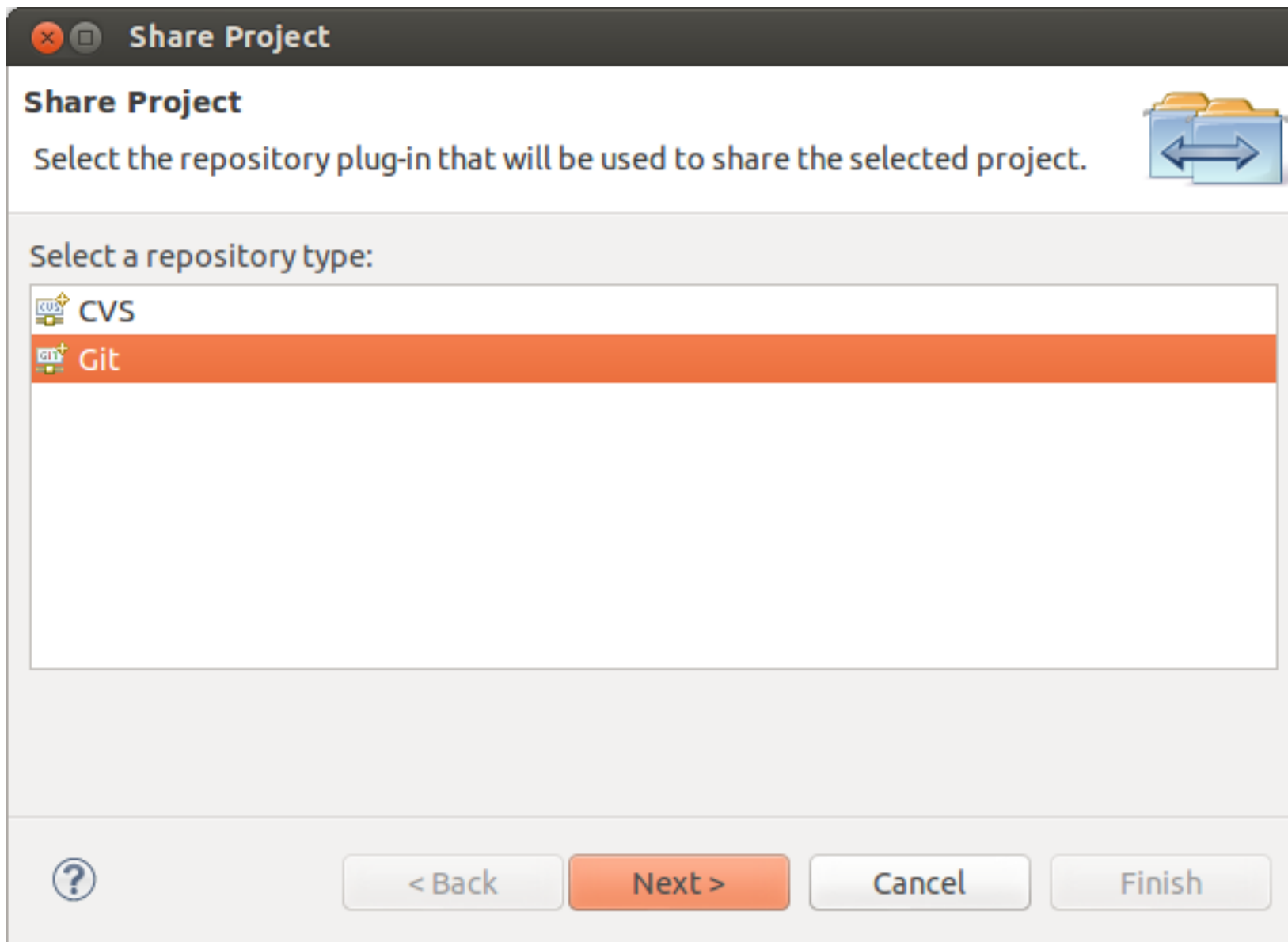
public class GitTest {
    public static void main(String[] args) {
        System.out.println("Git is fun");
    }
}
```

## 27.5. Put project under version control

To put your new project under version control with Git, right-click on your project, select Team ▢ Share Project ▢ Git.

Depending on your installation you may have to select that you want to use Git as a version control system.





On the next dialog select your existing Git repository from the drop-down list and press the Finish button.

## Configure Git Repository

### Configure Git Repository

Select an existing repository or create a new one



☐ Use or create repository in parent folder of project

Repository:

egitplayground - /home/vogella/git/egitplayground/.git

Create...

Working directory:

/home/vogella/git/egitplayground

Path within repository:

Browse...

Project	Current Location	Target Location
<input checked="" type="checkbox"/> com.vog	/home/vogella/workspace/test/com	/home/vogella/git/egitplayground/com.vogella.git.fir



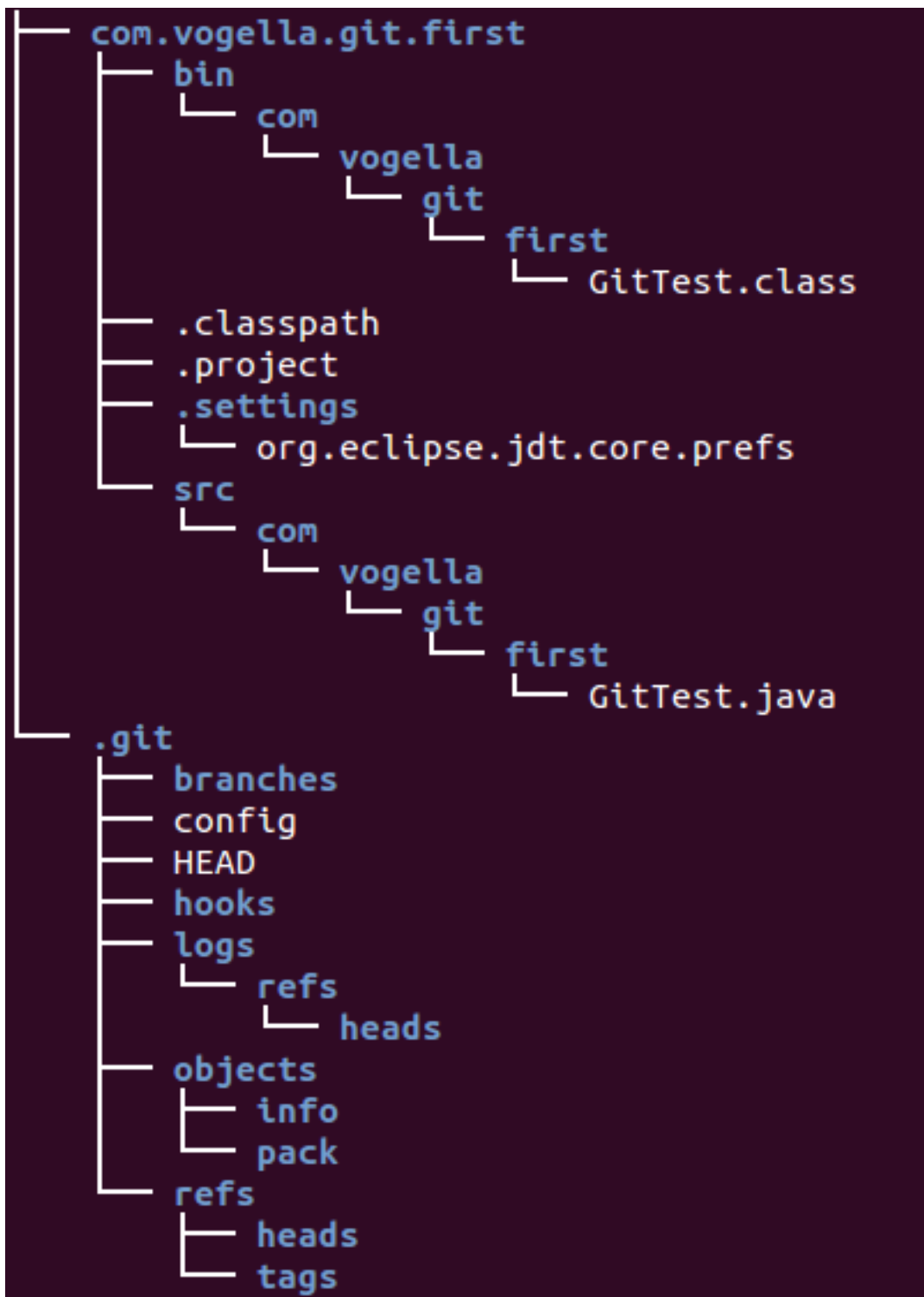
< Back

Next >

Cancel

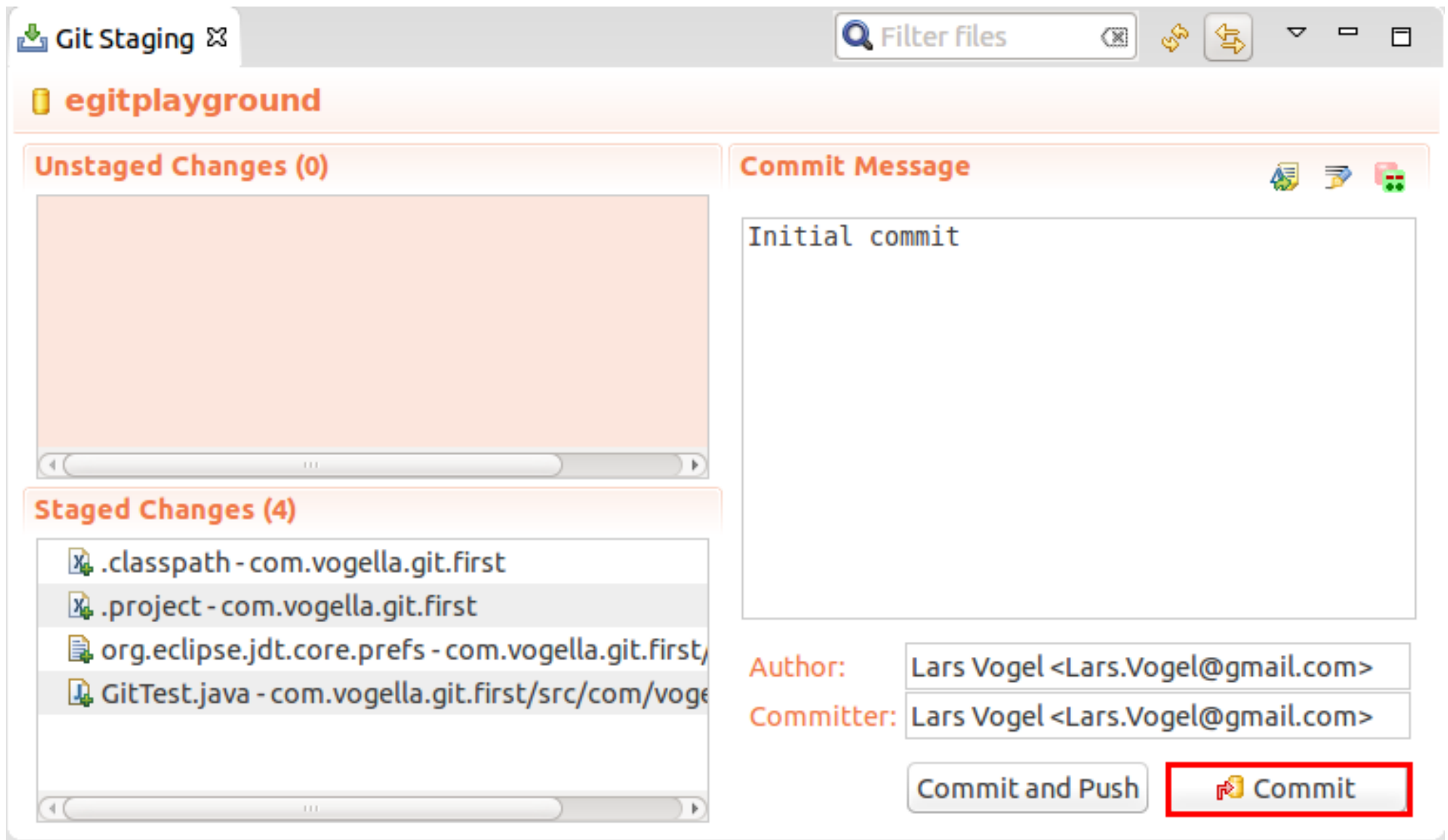
Finish

This moves the project to your Git repository. The following screenshot shows the generated directory structure. The `.git` directory contains the Git repository, the other directories contain the files of the working tree.



## 27.6. Using the Git Staging view for the initial commit

Open the *Git Staging* view, if it is not yet open via 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.



## 27.7. Using the Git Staging view for committing changes

Change the `System.out.println` message in your `GitTest` class.

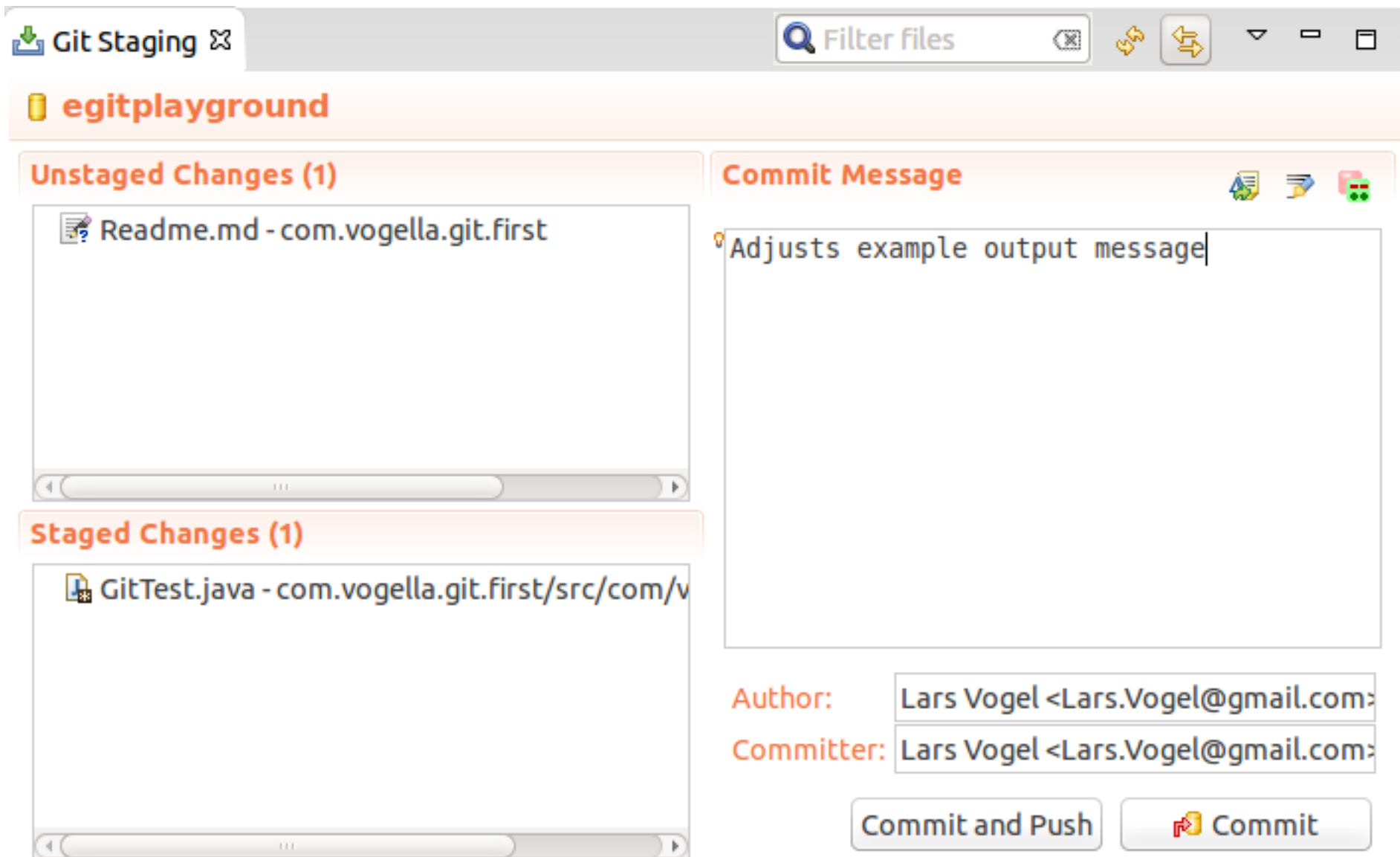
```
package com.vogella.git.first;

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

Create a new file called `Readme.md`.

Commit the changes of the `GitTest` class but do not add and commit the `Readme.md` file to the Git repository.

In the Git Staging view drag only the `GitTest` class into the *Staged Changes* area, write a meaningful commit message and press the commit button.



This change is now also stored in your local Git\_repository. The `Readme.md` file is neither staged nor committed to the Git repository.

## 27.8. Commit more files



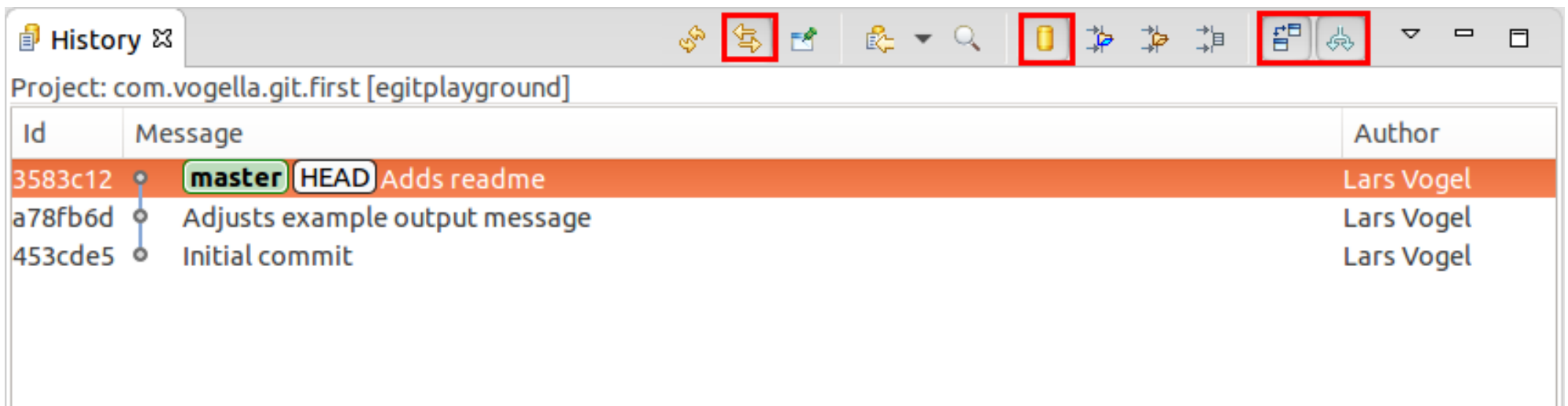
Commit the `Readme.md` file. By now you should know that you have to stage the file and commit it.

## 27.9. Review your commit history via the History view

Open the *History* view to browse the commit history of your repository. Review which files were included in your individual commits.

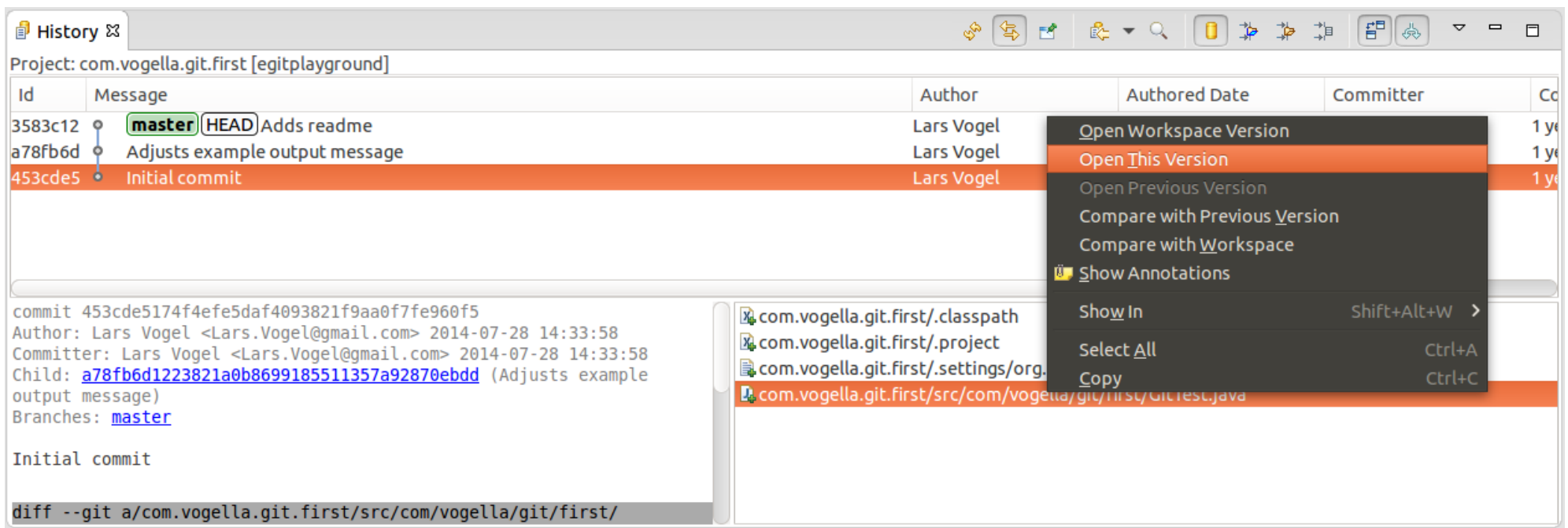
In the *History* view click all toggle buttons as shown in the screenshot

- Link with Editor and Selection
- Show all changes in repository containing the selected resources
- Compare Mode
- Show all Branches and Tags



## 27.10. Open an older version with the current version of a file via the History view

Open the version for the first commit of `GitTest` via the *History* view.



## 27.11. Add more projects to your Git repository

You can of course have more than one Eclipse project in your Git repository. To validate that, create two more Java projects called `com.vogella.egit.multi.java1` and `com.vogella.egit.multi.java2`. Create at least one Java class in each project.

Afterwards select both projects, right-click on them and select **Team** → **Share Project** → **Git**. If asked by the Eclipse IDE, select that you want to use Git.

Select your Git repository you created in this exercise and add both projects to this repository. Press the **Finish** button.

## 27.12. Validate the project move and commit changes

Afterwards validate that the projects have been moved. You can check your workspace directory and your Git repository directory via a file explorer. You see that the projects have been moved from their original location to the Git repository.

The changes have not yet been committed. Now commit all files in the two projects to your Git repository.

---

## 28. Contributing to EGit - Getting the source code

EGit is self-hosted on `git://git.eclipse.org`.

See [EGit contributor guide](#) for a description how to work with the EGit and JGit source.

---

## 29. About this website

Support free content



Questions and discussion



Tutorial & code license



Get the source code



## 30. Eclipse Git Resources

[Eclipse Git User Guide](#)

[Contributor guide for the Eclipse Git project](#)

[Different update sites for Eclipse Git](#)

### 30.1. vogella GmbH training and consulting support

#### TRAINING

The vogella company provides comprehensive [training and education services](#) from experts in the areas of Eclipse RCP, Android, Git, Java, Gradle and Spring. We offer both public and inhouse training.

#### SERVICE & SUPPORT

The vogella company offers [expert consulting](#) services, development support and coaching. Our customers range from Fortune 100 corporations to individual developers.

Whichever course you decide to take, you are guaranteed to experience what many before you refer to as [“The best IT class I have ever attended”](#).



## Appendix A: Copyright and License

Copyright © 2012-2017 vogella GmbH. Free use of the software examples is granted under the terms of the EPL License. This tutorial is published under the [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Germany](#) license.

See [Licence](#).

Version 4.2

Last updated 2017-07-11 08:11:57 +02:00