EGit Tutorial

Jonas Helming

Maximilian Koegel

January 2013

http://eclipsesource.com

Table of Contents

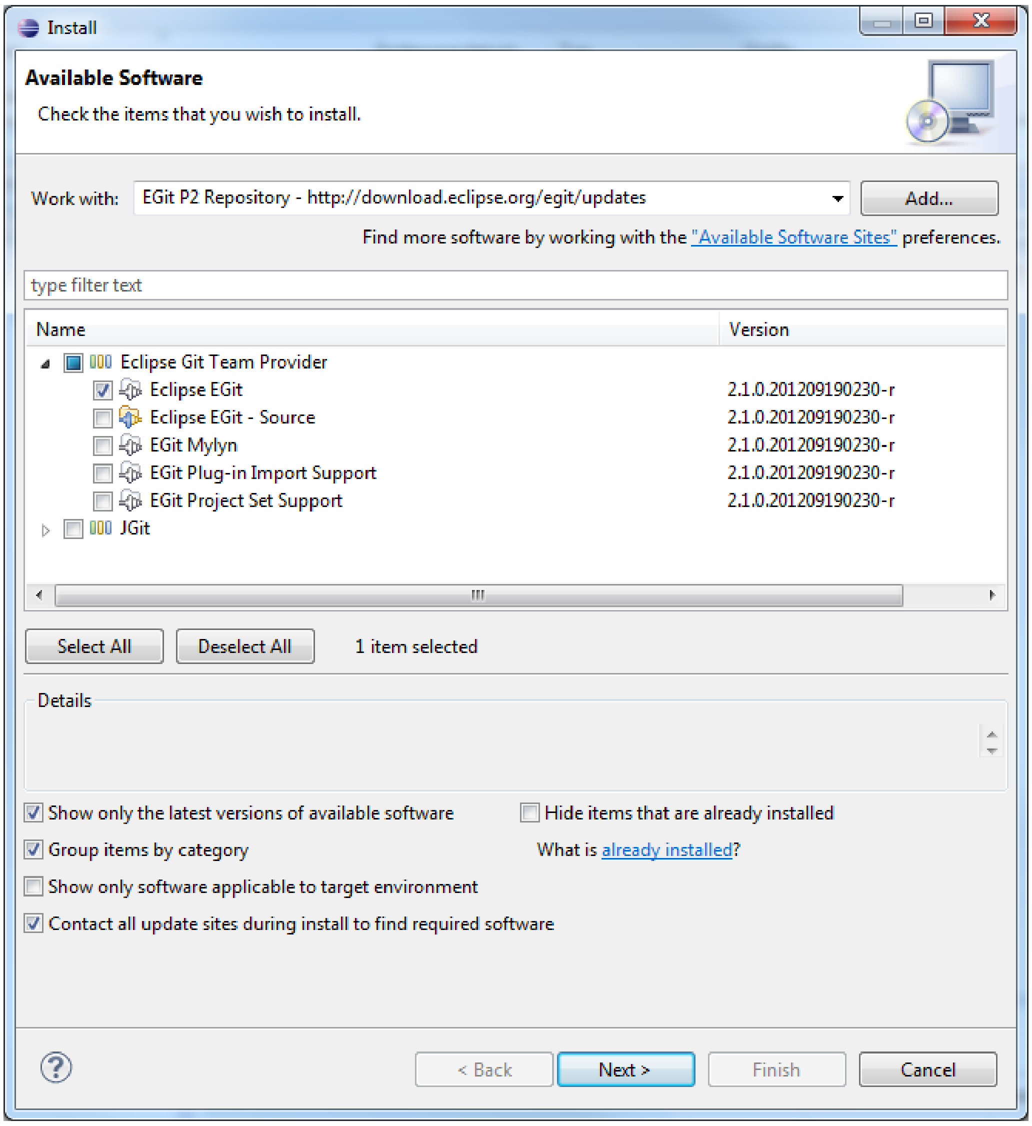
1. EGit Tutorial 2
2. Installing EGit in Eclipse 2
3. EGit Configuration 3
4. Creating Local Repositories 3
5. Commit 5
6. Adding Files 7
7. Reverting Changes 7
8. Revert via Compare 8
9. Revert via Reset 8
10. Cloning Repositories 9
11. Creating Branches 13
12. Merge 14
13. Resolving Conflicts 14
14. Fetch and Pull 15
15. Push 15
16. Synchronize 15
17. History View 16
18. Creating Patches 16
19. Repository View 16
20. Additional Information 17
21. Index 17
22. Reset Types 18
23. Useful pages 18

1. EGit Tutorial

This tutorial guides you through the essential use cases for EGit, in addition to notes on how to install, configure and create your first repositories. The tutorial is targeted at developers with a basic knowledge of Git processes. Please help us to keep this tutorial up-to-date by reporting any issues or questions.

2. Installing EGit in Eclipse

EGit is already included in the Eclipse Juno Release, so you do not need to install it. If you use an older version of Eclipse, open the Eclipse Wizard to install new software *Help => Install New Software*. Insert http://download.eclipse.org/egit/updates after *Work with:* and hit Return. Select Eclipse EGit as a child from Eclipse Team Git Provider. You don't have to install any other plugins. Click *Next* and confirm your selection in the following window by pressing *Next* again. Finally, accept the terms of use and the license agreement and click *Finish* to start the installation. After the installation has finished restart Eclipse.



Work with: EGit P2 Repository - http://download.eclipse.org/egit/updates

Find more software by working with the "Available Software Sites" preferences.

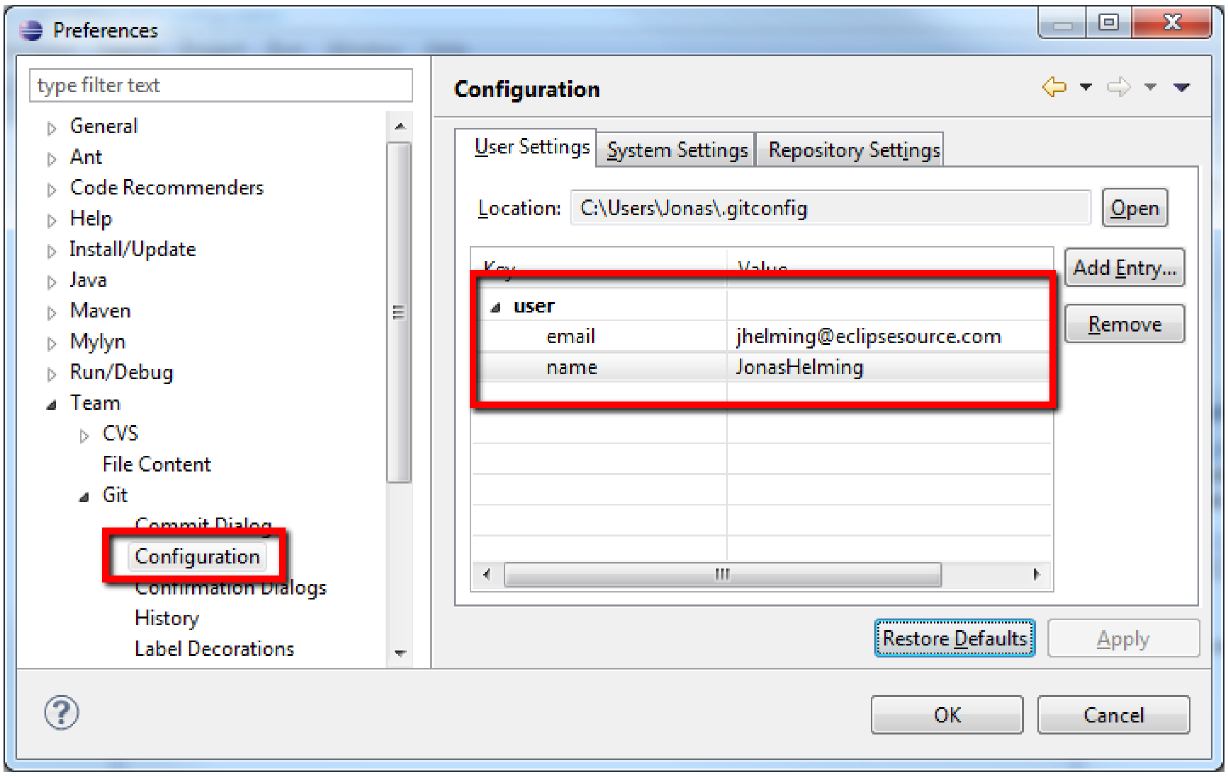
Name	Version
☐ Eclipse Git Team Provider	
✓ Eclipse EGit	2.1.0.201209190230-r
☐ Eclipse EGit - Source	2.1.0.201209190230-r
☐ EGit Mylyn	2.1.0.201209190230-r
☐ EGit Plug-in Import Support	2.1.0.201209190230-r
☐ EGit Project Set Support	2.1.0.201209190230-r
☐ JGit	

1 item selected

✓ Show only the latest versions of available software
☐ Hide items that are already installed
✓ Group items by category
What is already installed?
☐ Show only software applicable to target environment
✓ Contact all update sites during install to find required software

3. EGit Configuration

Every commit in EGit will include the user's name and his email-address. These attributes can be set in the Preferences-window *Window => Preferences*. Navigate to *Team => Git => Configuration* and hit the *New Entry...* Button. Enter *user.name* as *Key* and your name as *Value* and confirm. Repeat this procedure with *user.email* and your email address and click *OK* in the Preferences window. The username and email should be the same you use for your git account, ie. your GitHub account.



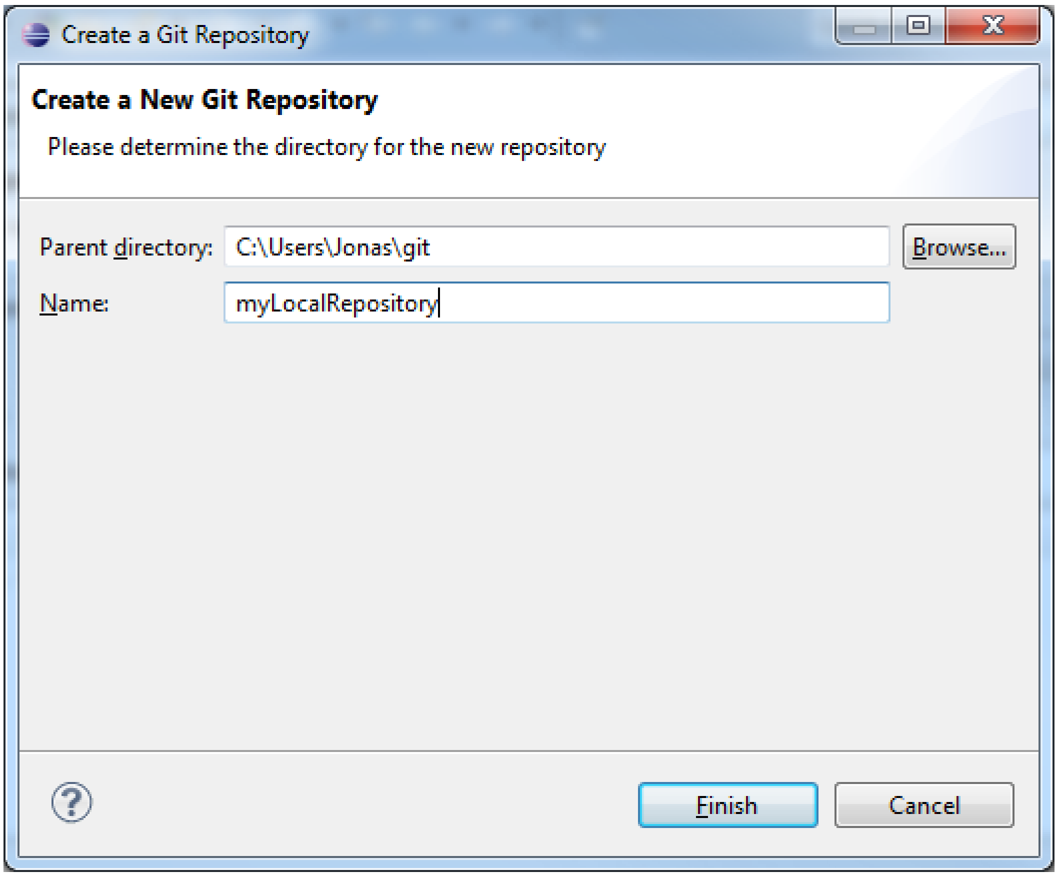
Location: C:\Users\Jonas\.gitconfig

Key	Value
user	
email	jhelming@eclipsesource.com
name	JonasHelming

4. Creating Local Repositories

One major advantage of Git compared to SVN or CVS is that you can easily create local repositories, even before you share them with other people. In this way, you can version your work locally. First, you have to create a project that you want to share via your local repository. For later purposes it would be useful to add some files, e.g. a Java class to your project.

After you have created your project, select the context menu by right clicking it and navigate to *Team => Share Project...* . Select *Git* as the repository type and hit *Next*. In the following window select your project, hit the *Create Repository*-button and click *Finish*. The repository will be assigned automatically.

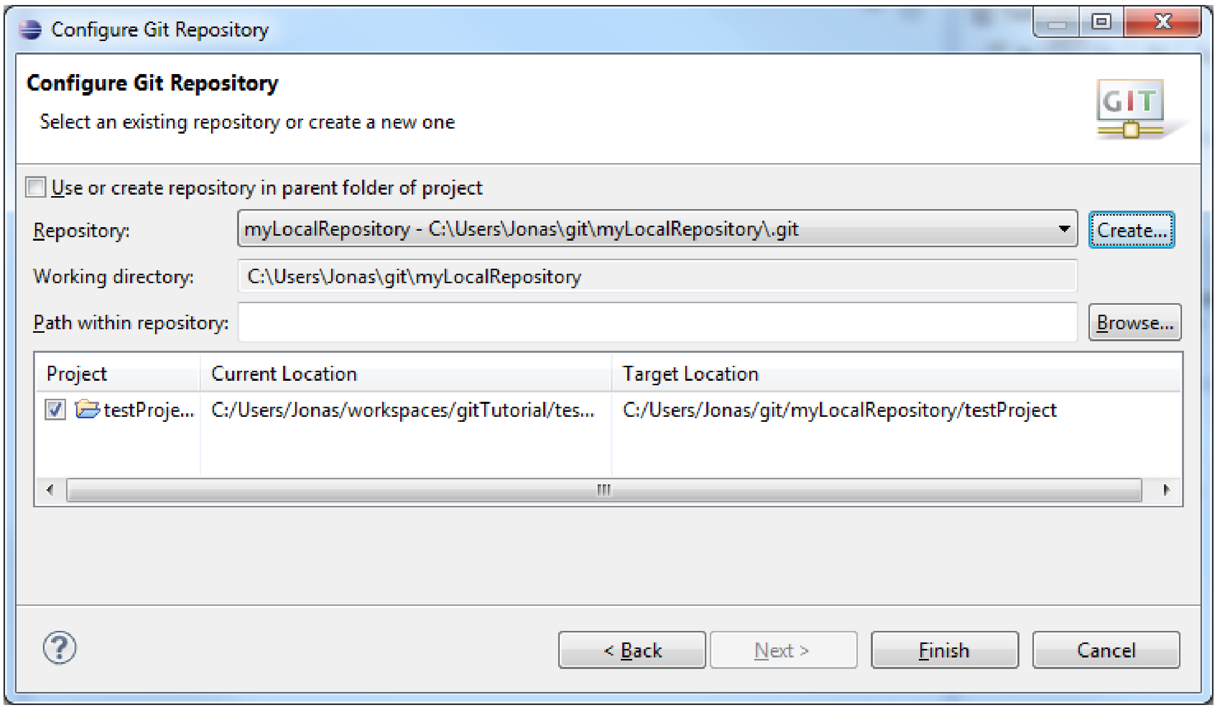

Create a Git Repository

Create a New Git Repository

Please determine the directory for the new repository

Parent directory: C:\Users\Jonas\git

Name: myLocalRepository


Configure Git Repository

Configure Git Repository

Select an existing repository or create a new one

☐ Use or create repository in parent folder of project

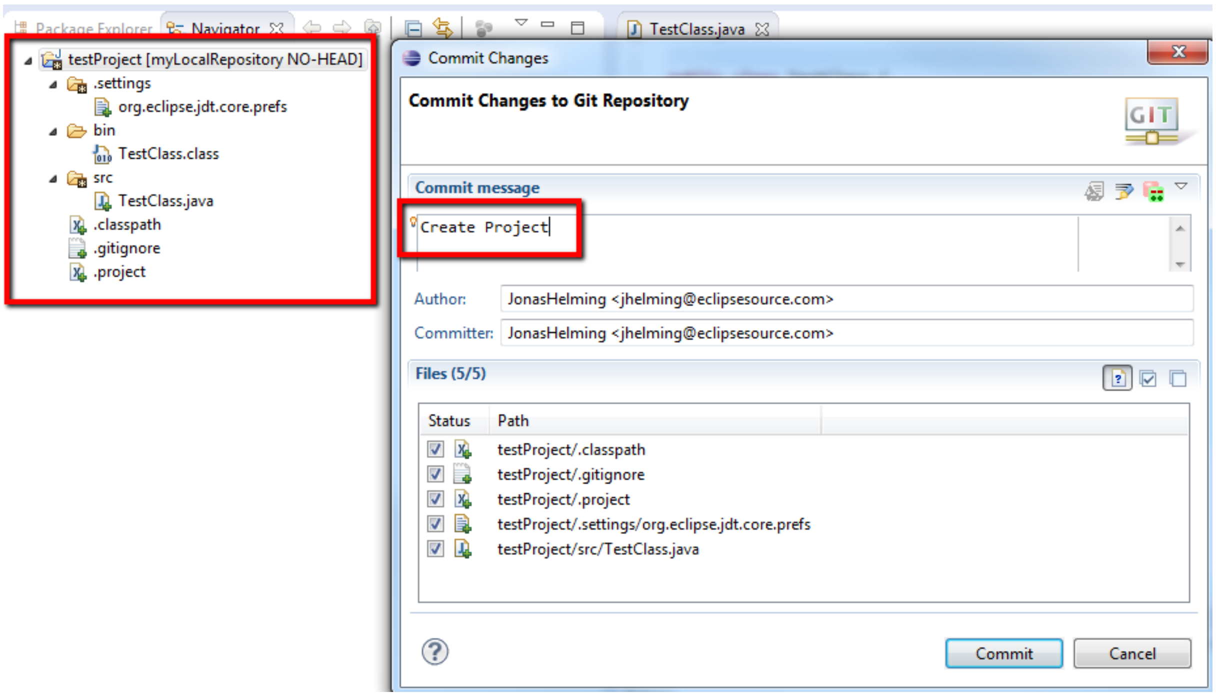
Repository: myLocalRepository - C:\Users\Jonas\git\myLocalRepository\.git

Working directory: C:\Users\Jonas\git\myLocalRepository

Path within repository:

Project	Current Location	Target Location
✓ testProje...	C:/Users/Jonas/workspaces/gitTutorial/tes...	C:/Users/Jonas/git/myLocalRepository/testProject

The newly created repository will be empty, although the project is assigned to it. (Note the changed icons: the project node will have a repository icon, the child nodes will have an icon with a question mark, ignored files, e.g. the bin directory, won't have any icons at all.) Before you can commit the files to your repository, you need to add them. Simply right click the shared project's node and navigate to *Team => Add*. After this operation, the question mark should change to a plus symbol. To set certain folders or files to be ignored by Git, e.g. the bin folder, right click them and select *Navigate => Ignore*. The ignored items will be stored in a file called *gitignore*, which you should add to the repository. The last thing to do is commit the project by right clicking the project node and selecting *Team => Commit...* from the context menu. In the Commit wizard, all files should be selected automatically. Enter a commit message (the first line should be headline-like, as it will appear in the history view) and hit the *Commit* button. If the commit was successful, the plus symbols will have turned into repository icons.



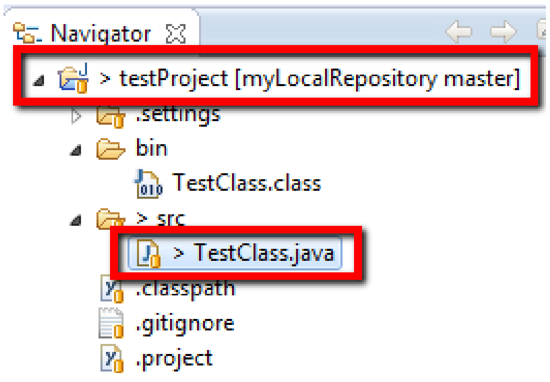
Create Project

Author: JonasHelming <jhelming@eclipsesource.com>
Committer: JonasHelming <jhelming@eclipsesource.com>

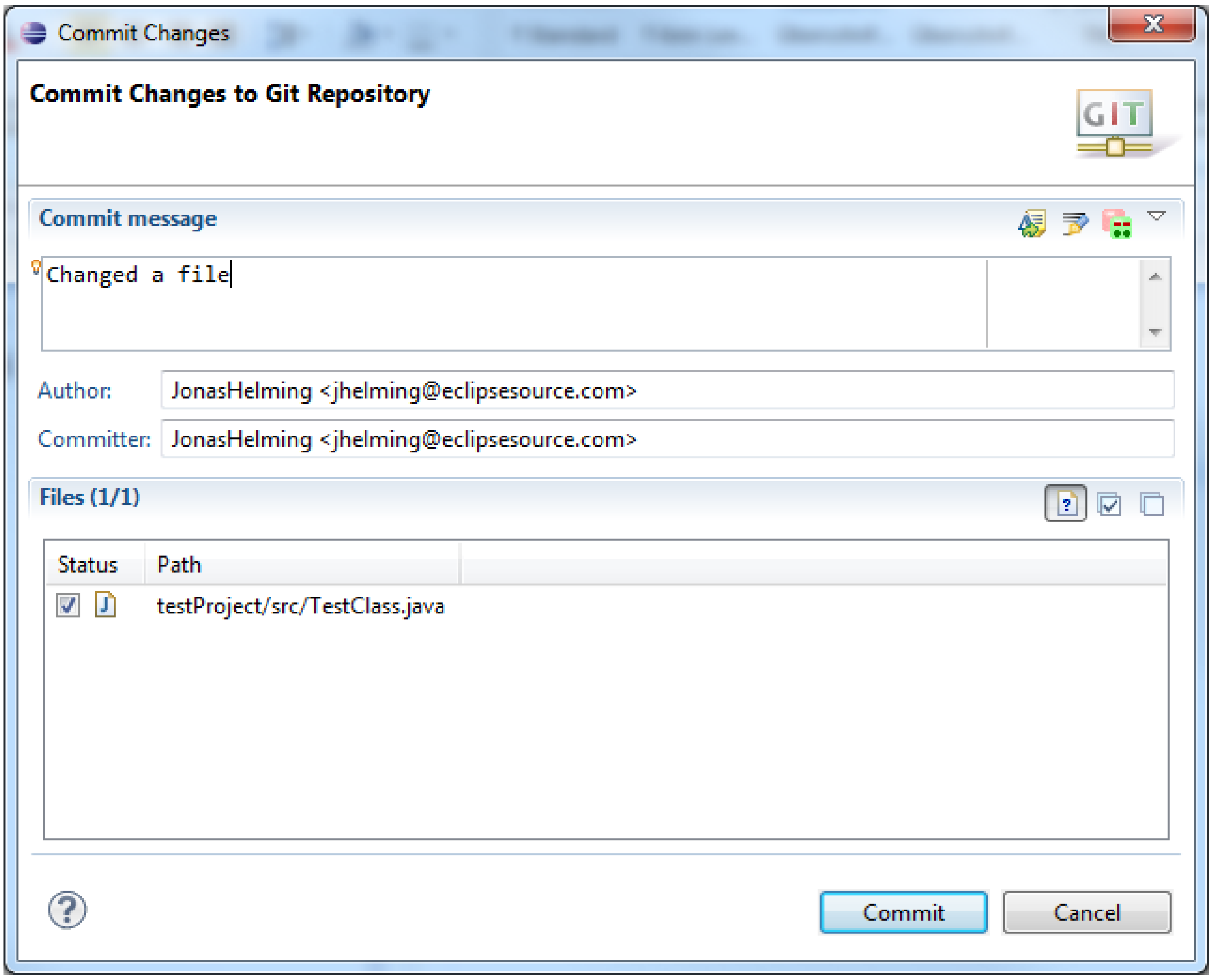
testProject/.classpath
testProject/.gitignore
testProject/.project
testProject/.settings/org.eclipse.jdt.core.prefs
testProject/src/TestClass.java

5. Commit

Now you can start to modify files in your project. To save changes made in your workspace to your repository, you will have to commit them. After changing files in your project, a ">" sign will appear right after the icon, telling you the status of these files is dirty. Any parent folder of this file will be marked as dirty as well.



If you want to commit the changes to your repository, right click the project (or the files you want to commit) and select *Team => Commit...* . This will open a new window, allowing you to select the files you want to commit. Before you can commit the files, you will have to enter a commit message in the upper textbox. After you're done, click *Commit* to commit the selected files to your repository.



Changed a file

Author: JonasHelming <jhelming@eclipsesource.com>
Committer: JonasHelming <jhelming@eclipsesource.com>

Status	Path
✓	testProject/src/TestClass.java

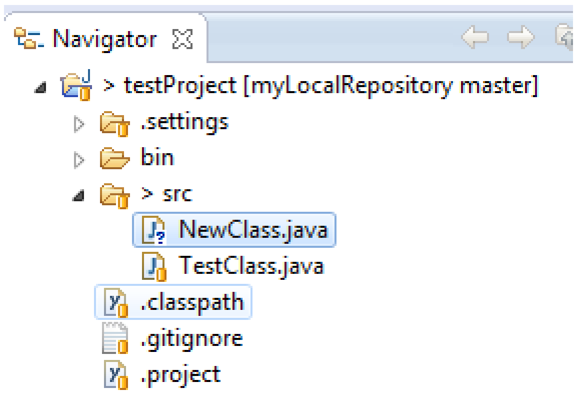
Note that the status of the changed file is *Mod., not staged*. By staging the files before you commit (see the section “Additional Information”), you can change the status to *Modified* (and the dirty sign to a staged icon).

If you later realize that your previous commit was incomplete (e.g. you missed committing a file) or your commit message was wrong, you might want to use *Amend previous commit*. This will merge the current commit and the previous commit into one, so you don't have to perform an extra commit (and maybe cause confusion). However, this should only be used if the previous commit hasn't already been published to a shared repository.

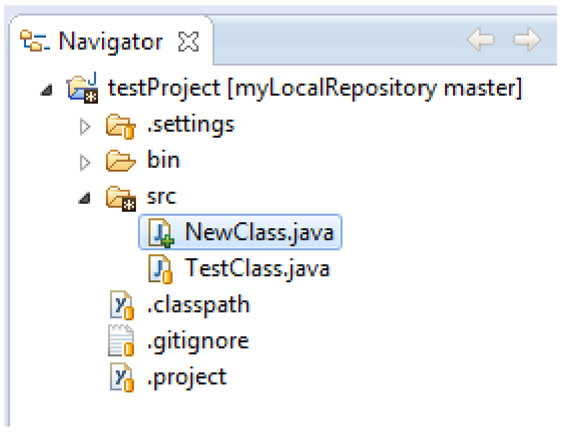
Another option is *Show untracked files*. By checking this checkbox, new files you created but did not add yet, will be available for you to select in the Commit window.

6. Adding Files

To add a new file to the repository, you will have to create it in your shared project first. The new file will, again, appear with a question mark.



Right click it and navigate to *Team* => *Add*. The question mark will turn into a plus symbol and the file will be tracked by Git, but it is not yet committed. All of the file's parent folders should now have a symbol that looks like an asterisk indicating that it is ‘staged’ (more on that later). In the next commit, the file will be added to the repository and the plus symbol will turn into a repository icon. The repository icons of all the file's parents (packages/project...) will turn into staged icons. EGit also allows selecting untracked files to be added in the commit dialog if you turn on the option “Show untracked files”. In this case, they will be added and committed at the same time.

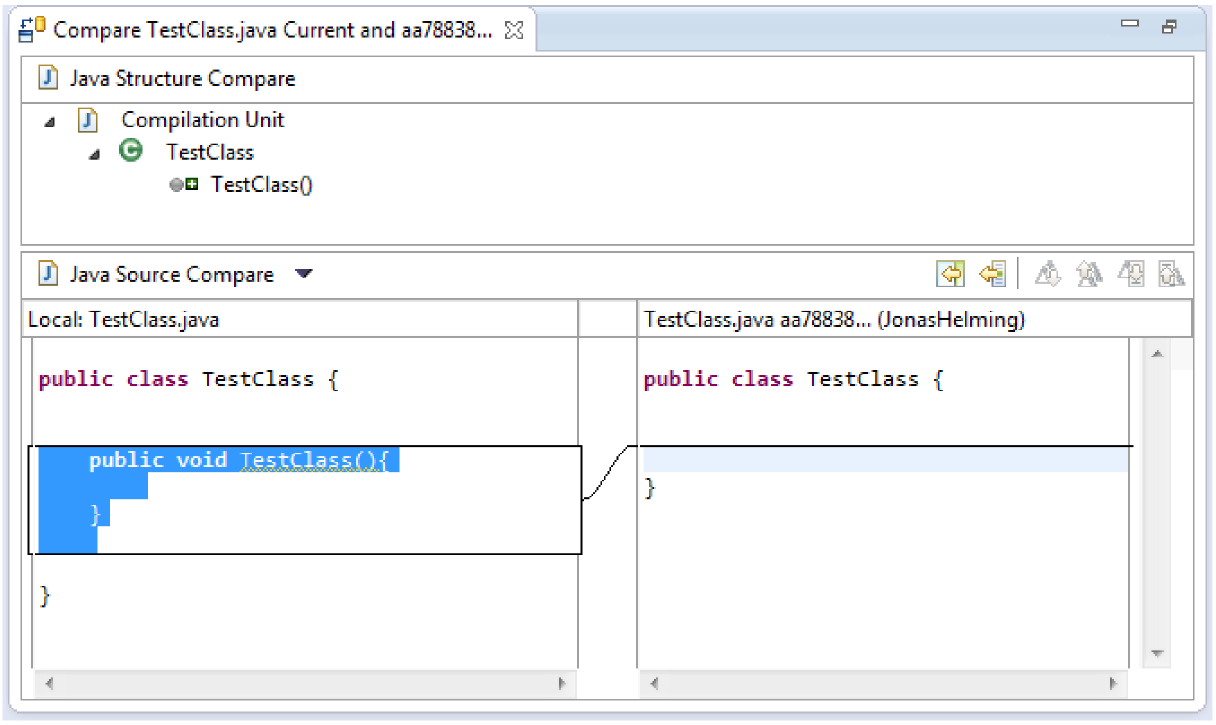


7. Reverting Changes

If you want to revert any changes, there are two options. You can compare each file you want to revert with the HEAD revision (or the index, more on that later) and undo some or all changes done. Second, you can hard-reset your project, causing any changes in the working directory to be reverted.

8. Revert via Compare

Right click the file you want to revert and select *Compare With => HEAD Revision*. This will open a comparison with the HEAD Revision, highlighting any changes done. If you want to completely revert your file, hit the *Copy All Non-Conflicting Changes from Right to Left-* button in the Java Source Compare toolbar. If you only want to revert several lines, select each line individually and hit the *Copy Current Change from Right to Left* button (in the toolbar) for each line. To complete the Revert operation, you will have to save either the comparison or your local copy of the file.

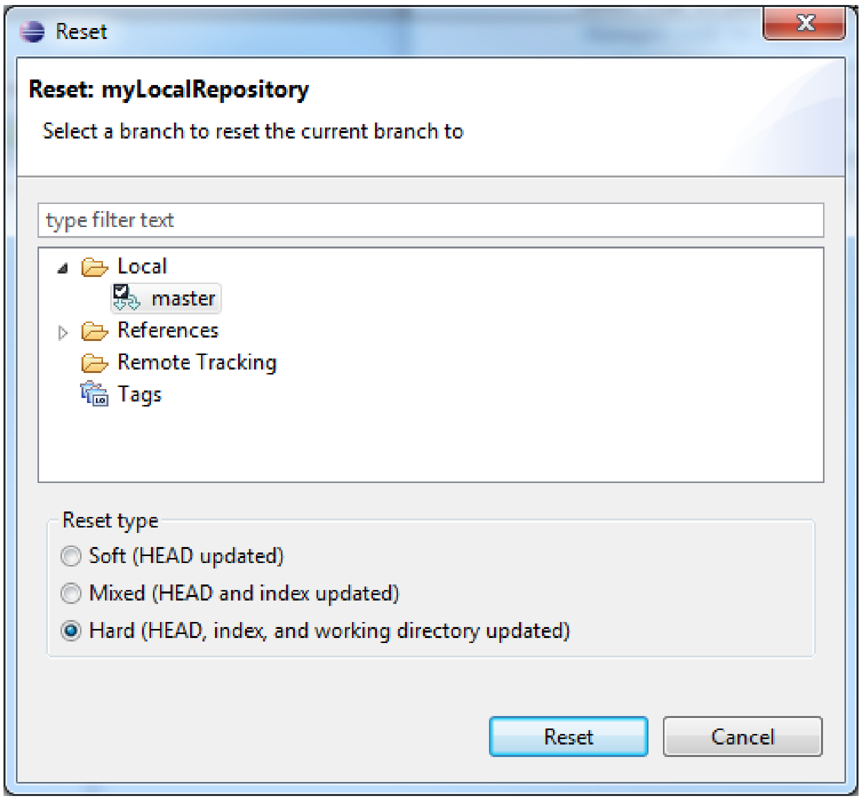


```
public class TestClass {

    public void TestClass(){
    }

}
```

```
public class TestClass {

}
```

9. Revert via Reset

To reset all changes made to your project, right click the project node and navigate to *Team => Reset...* . Select the branch you want to reset to (if you haven't created any other branches, there will be just one) and choose *Hard* as a reset type. By confirming this operation, all changes will be reset to this branch's last commit, including all changes done in the workspace (and index, more on that later). Be careful with this option as all changes in your workspace will be lost.


Reset

Reset: myLocalRepository

Select a branch to reset the current branch to

type filter text

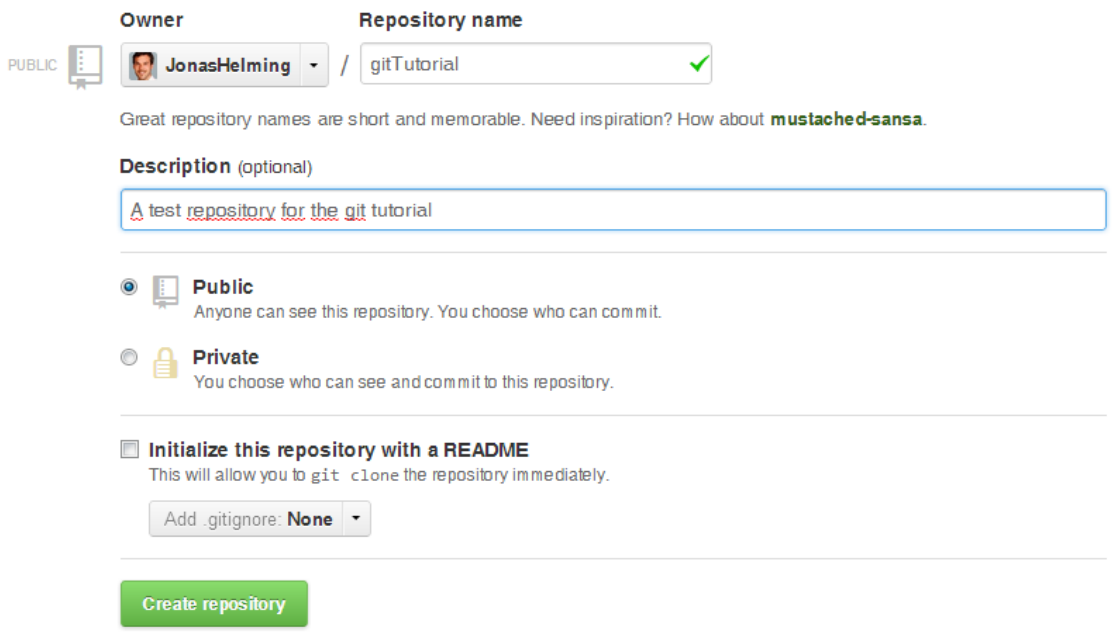
- Local
  - master
- References
- Remote Tracking
- Tags

Reset type

- ( ) Soft (HEAD updated)
- ( ) Mixed (HEAD and index updated)
- [✓ Hard (HEAD, index, and working directory updated)]

10. Cloning Repositories

Note:
For this and some of the following sections (especially Fetch/Push), you might want to use https://github.com to create your own remote repository. Public repositories are free at GitHub and performing the actions might help you gain some insights.


PUBLIC

Owner JonasHelming / **Repository name** gitTutorial

Great repository names are short and memorable. Need inspiration? How about **mustached-sansa**.

Description (optional)

A test repository for the git tutorial

[✓ **Public**]
Anyone can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

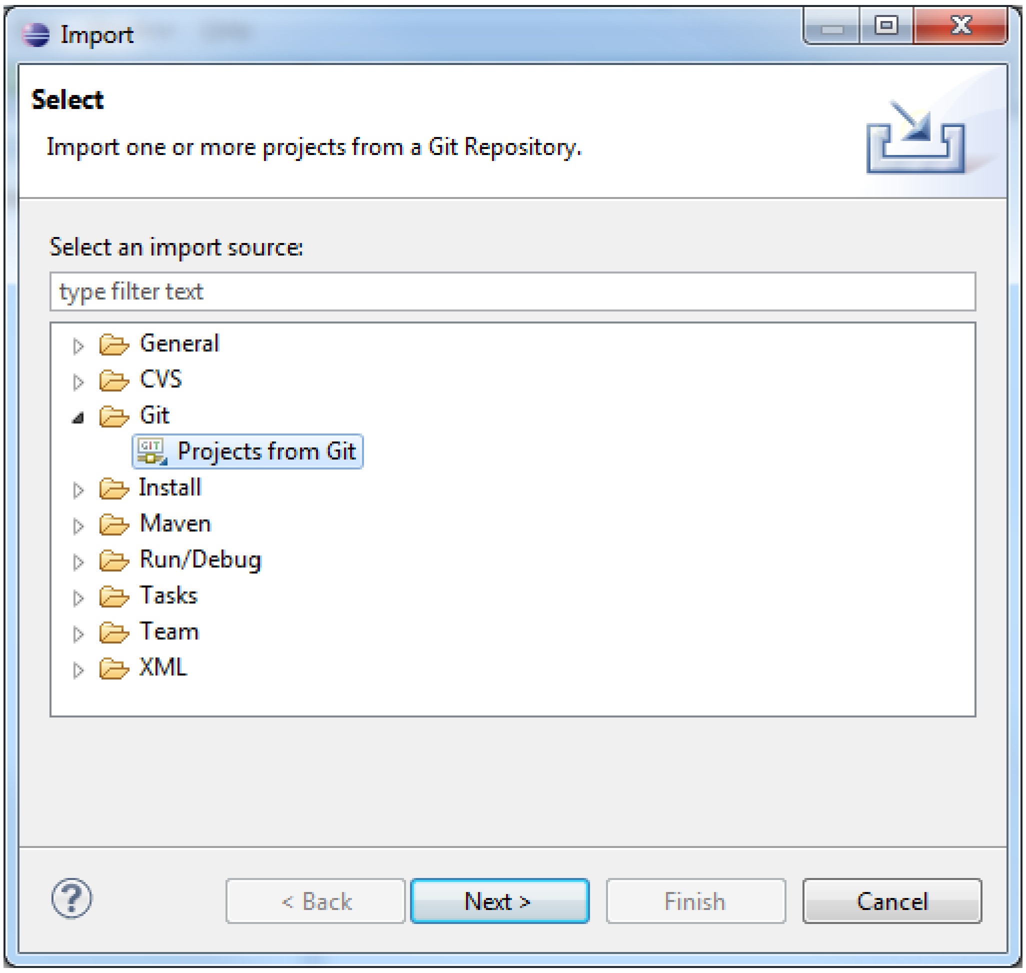
Initialize this repository with a README
This will allow you to `git clone` the repository immediately.

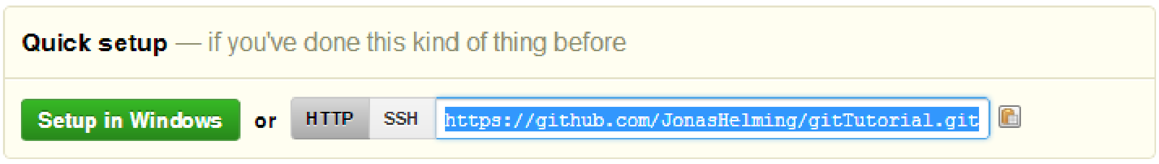
Add .gitignore: **None**

In order to checkout a remote project, you will have to clone its repository first. Open the Eclipse Import wizard (e.g. *File => Import*), select *Git => Projects from Git* and click *Next*. Select "URI" and click next. Now you will have to enter the repository's location and connection data. Entering the URI will automatically fill some fields. Complete any other required fields and hit *Next*. If you use GitHub, you can copy the URI from the web page.

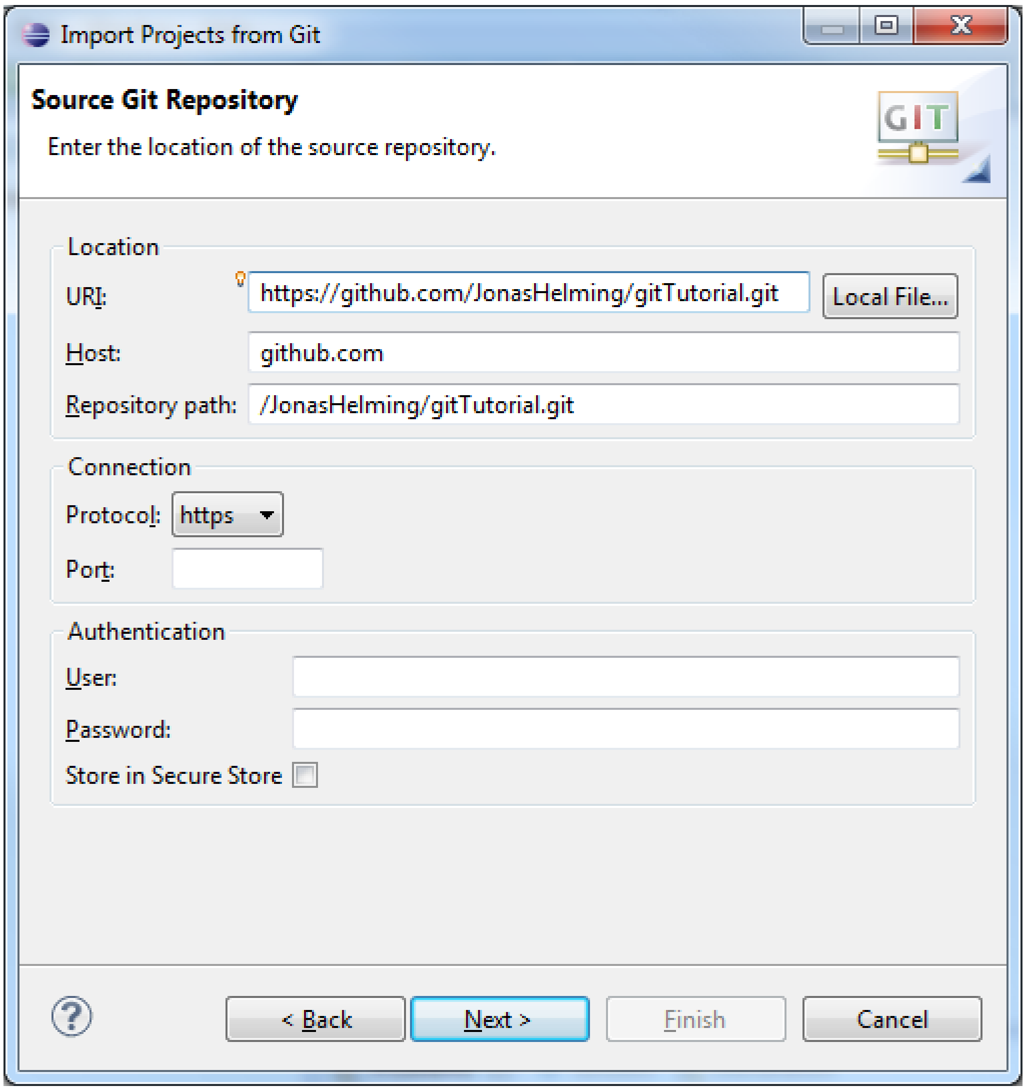
Select all branches you wish to clone and hit *Next* again.

Hit the *Clone...* button to open another wizard for cloning Git repositories.





https://github.com/JonasHelming/gitTutorial.git


Import Projects from Git

Source Git Repository

Enter the location of the source repository.

Location

URI: https://github.com/JonasHelming/gitTutorial.git

Host: github.com

Repository path: /JonasHelming/gitTutorial.git

Connection

Protocol: https

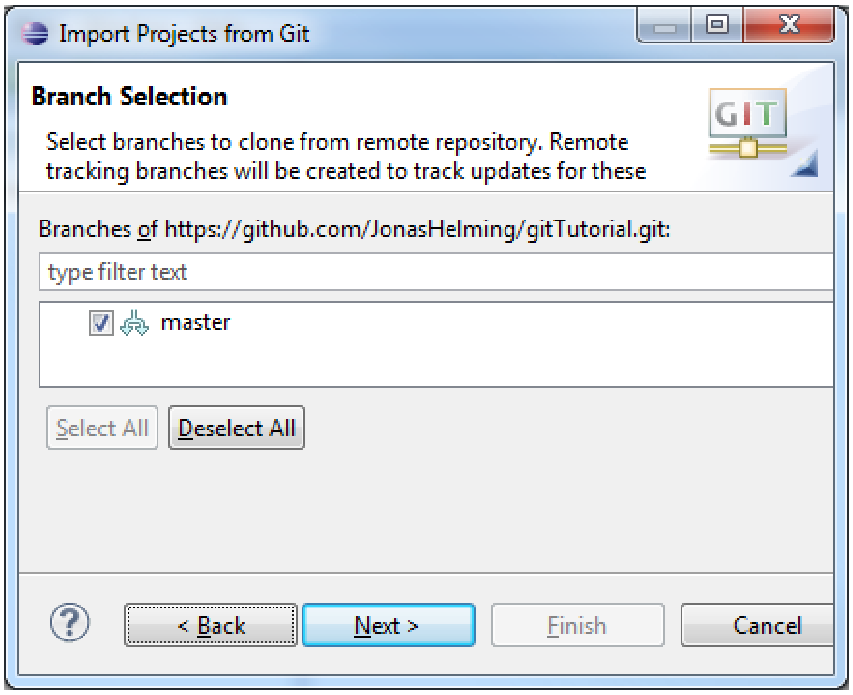
Port:

Authentication

User:

Password:

Store in Secure Store ☐


Import Projects from Git

Branch Selection

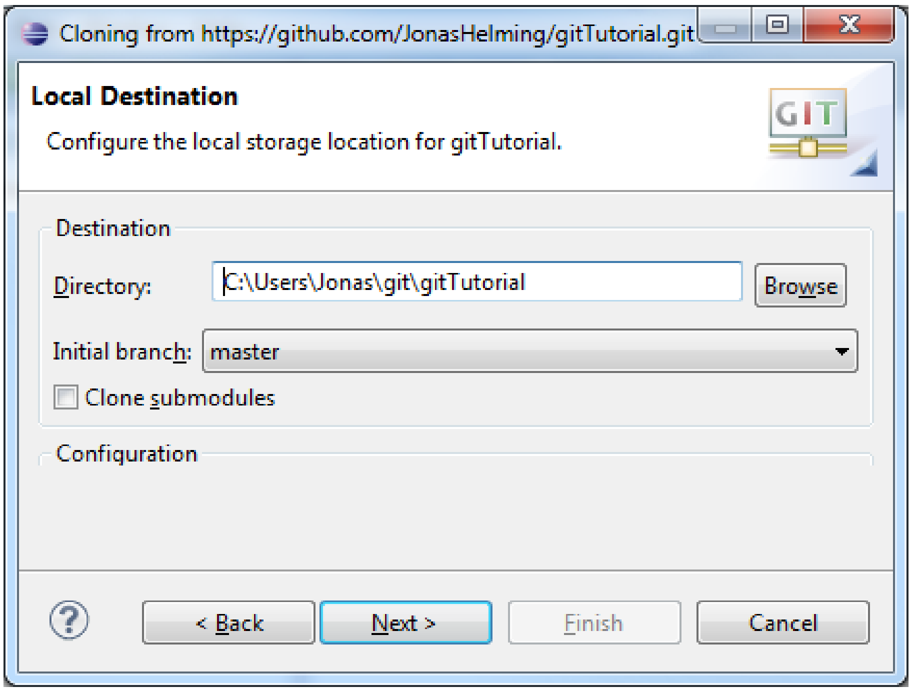
Select branches to clone from remote repository. Remote tracking branches will be created to track updates for these

Branches of https://github.com/JonasHelming/gitTutorial.git:

type filter text

☑ master

Choose a local directory to save this repository in.



Cloning from https://github.com/JonasHelming/gitTutorial.git

Local Destination

Configure the local storage location for gitTutorial.

Destination

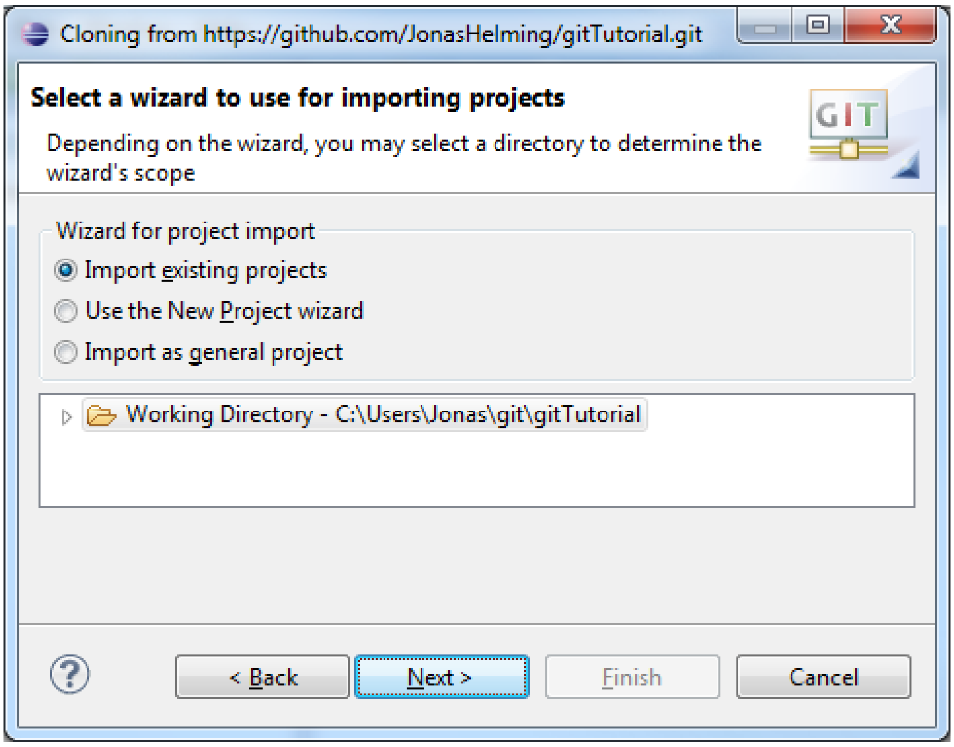
Directory: C:\Users\Jonas\git\gitTutorial

Initial branch: master

Clone submodules

Configuration

To import the projects, select the cloned repository and hit *Next*. Select *Import Existing Projects* and hit *Next*. Please note that there needs to be existing projects in your repository, and if you use your own repository it might be empty. In the following window, select all projects you want to import and click *Finish*. The projects should now appear in the Navigator/Package Explorer. (Note the repository symbol in the icons indicating that the projects are already shared.)



Cloning from https://github.com/JonasHelming/gitTutorial.git

Select a wizard to use for importing projects

Depending on the wizard, you may select a directory to determine the wizard's scope

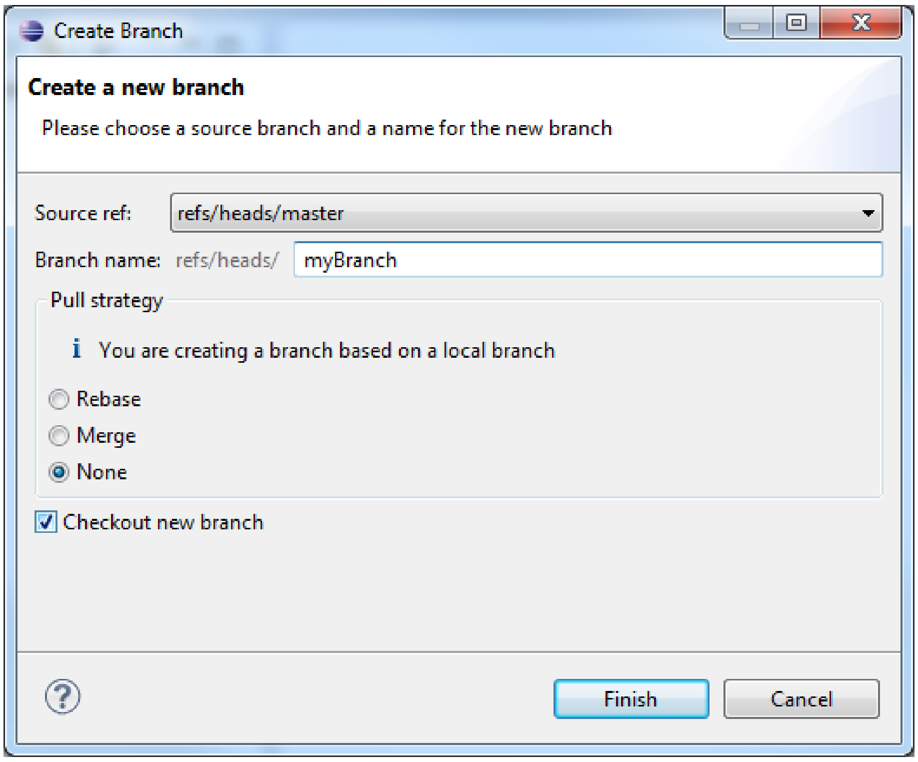
Wizard for project import

- [✓ Import existing projects]
- Use the New Project wizard
- Import as general project

Working Directory - C:\Users\Jonas\git\gitTutorial

11. Creating Branches

To create a new branch in your repository, right click a shared project and navigate to *Team => Switch to => New Branch...* from the context menu. Select the branch you want to create a new branch from, hit *New branch* and enter a name for the new branch.



Source ref: refs/heads/master

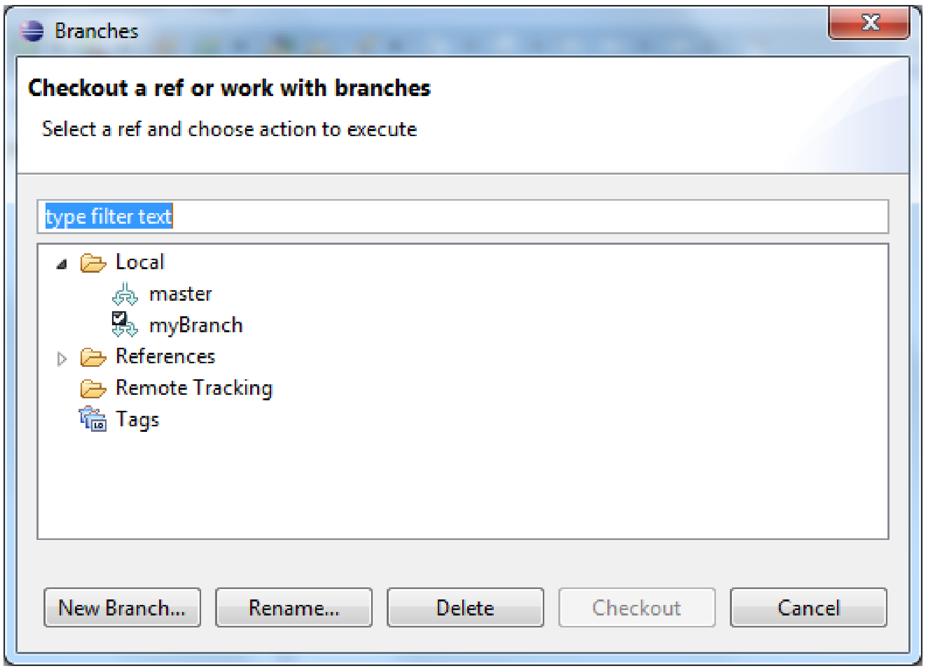
Branch name: refs/heads/ myBranch

You are creating a branch based on a local branch

Rebase [✓ None] Merge

[✓] Checkout new branch

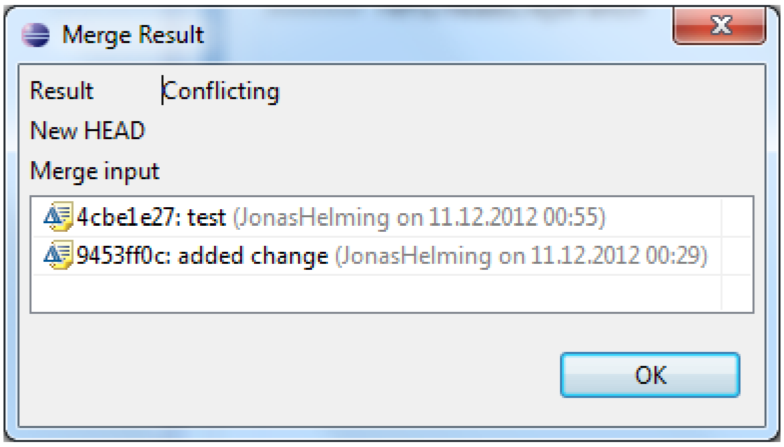
The new branch should appear in the branch selection window. If you would like to checkout the newly created branch, select it and click *Checkout*.



12. Merge

To merge one branch into another, you will have to checkout the branch you want to merge with. Right click the project node and navigate to *Team => Merge...* . Select any branch (other than the checked out branch) and hit *Merge*.

The merge will execute and a window will pop-up with the results. The possible results are *Already-up-to-date*, *Fast-forward*, *Merged*, *Conflicting*, *Failed*. A *Conflicting* result (see image below) will leave the merge process incomplete. You will have to resolve the conflicts (see next section). A *Failed* result may occur when there are already conflicting changes in the working directory.

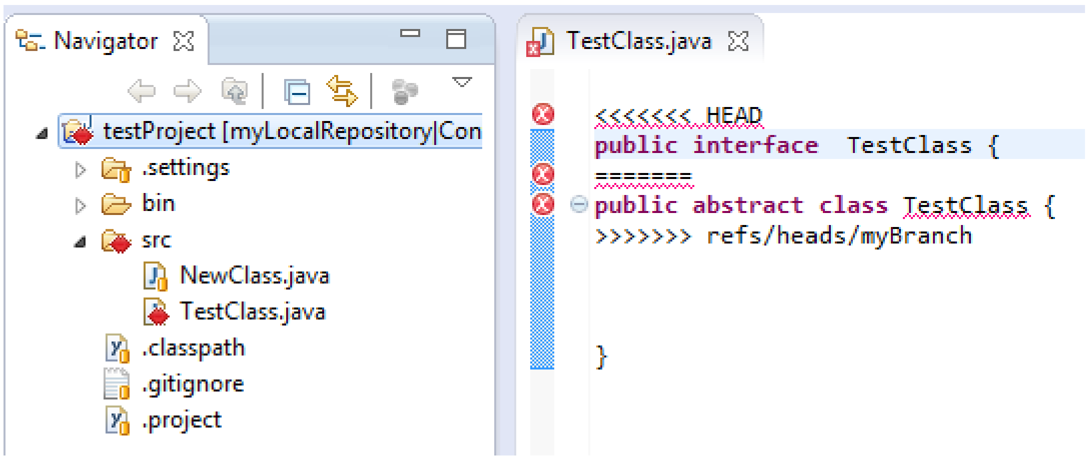

Merge Result

Result	Conflicting
New HEAD	
Merge input	

4cbe1e27: test (JonasHelming on 11.12.2012 00:55)
9453ff0c: added change (JonasHelming on 11.12.2012 00:29)

13. Resolving Conflicts

If your merge resulted in conflicts (note the red symbols on the file icons), you will have to resolve these manually. Open the conflicting files and scroll to the conflicting changes marked with “<<<<<<<”



```
<<<<<<< HEAD
public interface  TestClass {
=======
public abstract class TestClass {
>>>>>>> refs/heads/myBranch


}
```

After you are finished the manual part of the merge, you will have to tell Git that the conflicts are resolved. To do so, *Add* the files and *Commit* to complete your merge.

14. Fetch and Pull

When cloning remote repositories, git creates copies of the branches as local branches and as remote branches. A *Fetch* operation will update the remote branches only. To update your local branches as well, you will have to perform a *Merge* operation after fetching. The operation *Pull* combines *Fetch* and *Merge*. To perform a *Fetch*, select *Team => Fetch From...* from the project's context menu. Enter the repository you want to fetch branches from. (If you cloned this repository, the remote branch will be selected as default.) In the following window you will have to select what you want to fetch. As default, all branches are selected. The result of the *Fetch*-operation will be shown in a final confirmation window. Follow the same steps to apply a *Pull*.

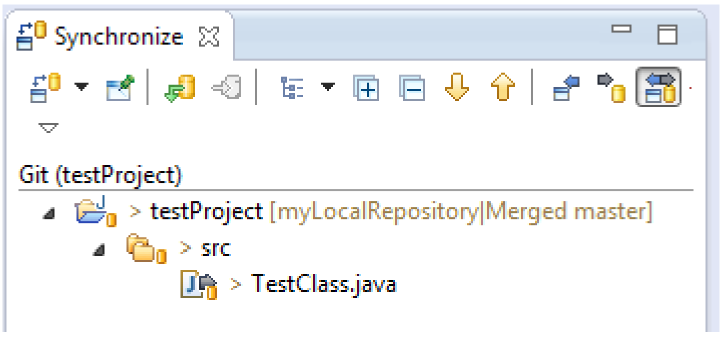
15. Push

Local changes made to your local branches can be pushed to remote repositories causing a merge from your branches into the branches of the remote repository (X pulls from Y is the same as Y pushes to X). The *Push*-wizard is pretty much the same as the *Fetch*-wizard. First, right click the project node and navigate to *Team=> Push...* . Enter the repository you want to push your branches to (the default for this will be the same as the *Fetch*-default if you didn't configure a *Push*-default) and hit *Next*. Choose the branches you want to push or click *Add all branches spec* if you want to push all branches. You can also select branches you want to delete from the remote repository. If you are done hit *Finish*. A final window will show the results of the *Push*.

16. Synchronize

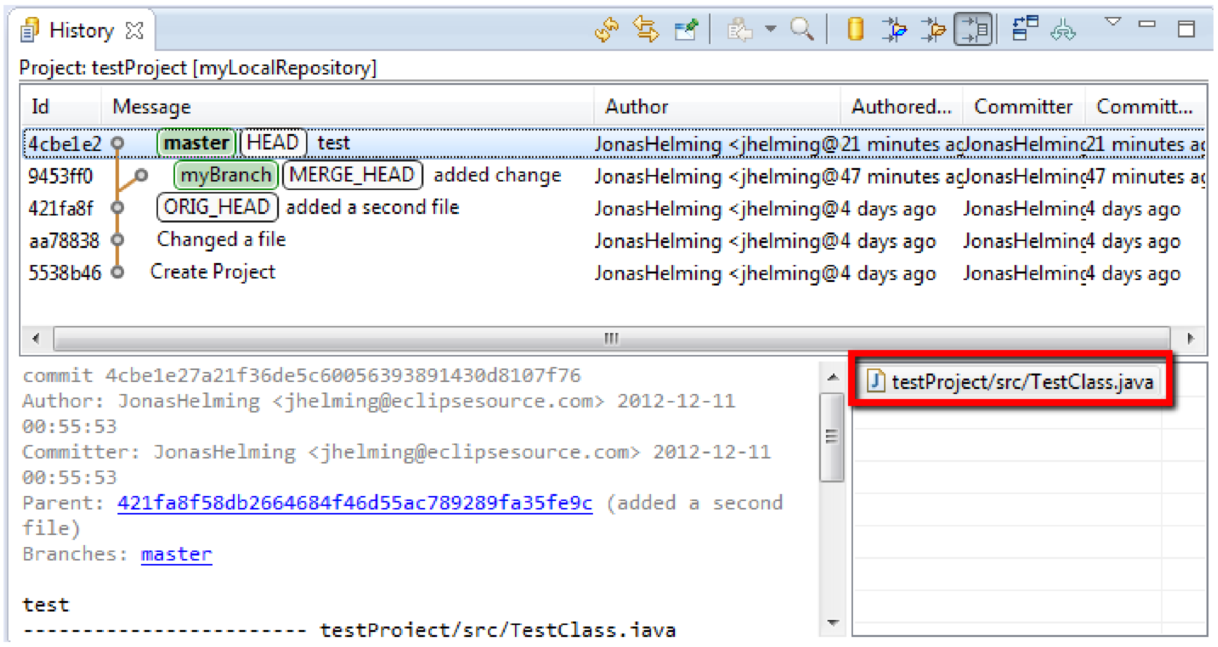
Comparisons between your workspace and the local repository or between the current branch and others and are done via the *Synchronize*-operation. If you right click Team => Sychronize Workspace, your local workspace will be compared with the current branch showing uncommitted changes. If you select *Team => Advanced => Synchronize...* . , you can select other branches to compare your current branch with. In this case you can also include local uncommitted changes.

To compare the branches you may want to switch to the Synchronizing perspective, where you can get a more detailed view of several changes. Here is an example of a *Synchronize* operation in the Synchronizing perspective:



17. History View

To show any shared file's history, right click it and select *Team => Show in History*. This will open the History View, giving an overview of the commits and allowing you to perform several actions (compare, creating branches/tags, reset...). Every commit you select comes up with a revision comment and revision details. The revision comment (bottom left corner) includes parents, children, commit message and changes whereas the revision details (bottom right corner) name the changed files and the actions performed upon them (A=ADD, M=MODIFY, D=DELETE). Selecting a file in the revision details will scroll the revision comment to the changes to that file.



Project: testProject [myLocalRepository]

Id	Message	Author	Authored...	Committer	Committ...
4cbe1e2	master HEAD test	JonasHelming <jhelming@	21 minutes ag	JonasHelming	21 minutes ag
9453ff0	myBranch MERGE_HEAD added change	JonasHelming <jhelming@	47 minutes ag	JonasHelming	47 minutes ag
421fa8f	ORIG_HEAD added a second file	JonasHelming <jhelming@	4 days ago	JonasHelming	4 days ago
aa78838	Changed a file	JonasHelming <jhelming@	4 days ago	JonasHelming	4 days ago
5538b46	Create Project	JonasHelming <jhelming@	4 days ago	JonasHelming	4 days ago

```
commit 4cbe1e27a21f36de5c60056393891430d8107f76
Author: JonasHelming <jhelming@eclipsesource.com> 2012-12-11
00:55:53
Committer: JonasHelming <jhelming@eclipsesource.com> 2012-12-11
00:55:53
Parent: 421fa8f58db2664684f46d55ac789289fa35fe9c (added a second
file)
Branches: master

test
------------------------ testProject/src/TestClass.java
```

testProject/src/TestClass.java

18. Creating Patches

Any patch can only include one commit that is from a parent to its child (if a child has just one parent, you cannot merge so you will need to patch). To create a patch you have to open the History View first (see previous section). Right click a commit you want to create a patch for (this must be a child with exactly one parent) and select *Create Patch...*.

Select either *Clipboard* or *File* and hit *Next* and click *Finish*. The resulting patch can be applied to the parent commit via *Team => Apply Patch...*.

19. Repository View

The repository view is useful when working with branches/tags and executing operations on them, as well as handling remote repositories and getting an overview of all your repositories. To open this view, select *Team => Show in Repositories View* from any file's context menu.

20. Additional Information

Icon Decorations/Signs

📄	*ignored*: The repository treats these files as if they were non-existent (e.g. the bin-directory by default). Add a .gitignore file or *Team => Ignore* to ignore a file.
📄?	*untracked*: Any file known, but not yet recorded. To track a file, add it or select the *Show untracked files*-option in the commit-wizard and commit it directly.
📄	*tracked*: Any file known to and recorded by the repository.
📄+	*added*: Any file known to the repository, but not yet committed. Perform a *Commit* to change this file's status to tracked.
📄✖	*removed*: Any file that should be removed from the repository. For this icon to appear *Team => Untrack* has to be performed. By deleting the file from the workspace, the file will disappear (and therefore no icon will appear). However, it will still be removed from the repository with the next commit.
📄 >	*dirty*: Any tracked file with changes that have not yet been added to the index.
📄	*staged*: Any tracked file with changes that are already included in the index.
📄 >	*partially-staged:* Any tracked file with changes, where some changes are already included in the index, and others that are not yet added.
📄 >	*conflicted*: Any file where the merge result caused a conflict. Resolve the conflicts and perform an *Add* operation to change this file's status.
📄✓	*assume-valid*: Any modifications won't be checked by Git. This option can be activated via *Team => Assume unchanged*. However, it can only be turned off via the command line. Performing a *Reset* operation resets this status as well.

21. Index

The index, sometimes referred to as staging area, is an area between the working directory and the repository. Any change made to any file will change this file's status to *dirty* (see above). Any *dirty* file can be added to the index with an *Add* operation. The file's status changes to *staged*. You can compare files to the index and reset the index without resetting the workspace. In the original Git, files had to be added to the index before performing a *Commit* operation. This is not necessary in EGit, as *Team => Commit* allows you to commit unstaged changes.

22. Reset Types

You can reset your current branch to any other branch, tag or commit you want. Right click any commit in the History View and select *Reset*. There are three options available:

Soft:

The current branch's tip will point to this branch/tag/commit. Changes in the index and working directory, however, won't be reset.

Mixed:

Same as a soft reset, only that the current index will be replaced by the selected branch/tag/commit's index. The working directory stays unchanged.

Hard:

All changes will be reverted to the selected branch/tag/commit. Uncommitted changes will be lost, therefore this operation has to be confirmed.

23. Useful pages

- To create your own remote repositories and perform operations on them, you might want to register at https://github.com. As long as your repository is public, github is free.
- A tutorial with more information on certain options and actions: http://wiki.eclipse.org/EGit/User_Guide

For more information, contact us:

Maximilian Koegel and Jonas Helming

EclipseSource Munich leads

Email: e4@eclipsesource.com

http://eclipsesource.com/munich