CORSO DI LAUREA IN INFORMATICA

Tecnologie Software per il Web

ECLIPSE AND GIT

a.a. 2020-2021

Version control system

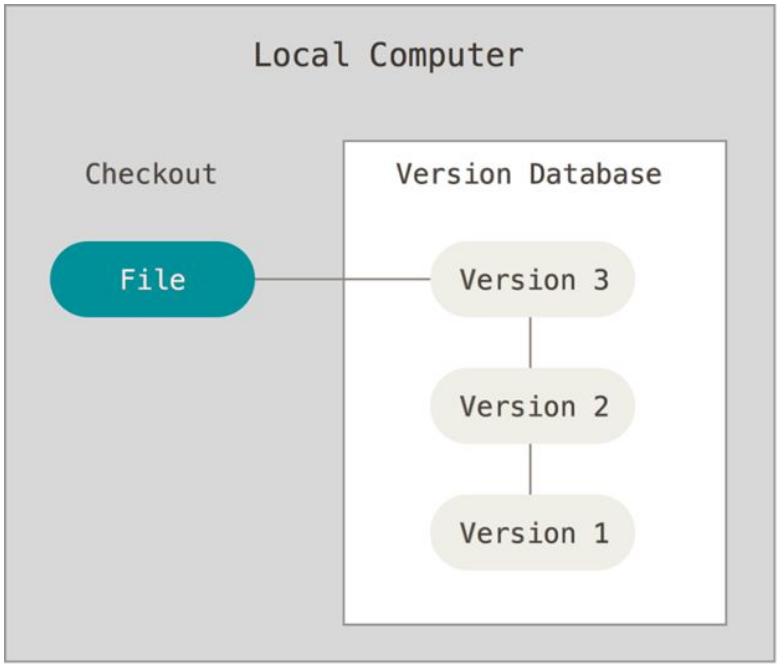
- A system that records changes to a file or set of files over time so that you can recall specific versions later
- Aka VCS
- Usually used to control the versions of a software project
- Allows:
  - revert selected files back to a previous state
  - revert the entire project back to a previous state
  - compare changes over time
  - see who last modified something that might be causing a problem
  - …

Version-control methods

- Local
- Centralized
- Distributed

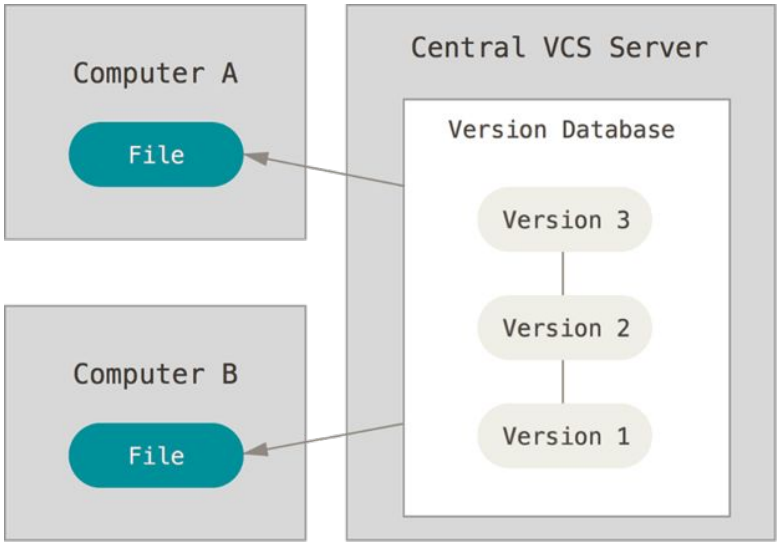
Local VCSs

- A local database of versions keeping track of file changes
- Example: RCS
- Main limitations:
  - Not suitable for a collaborative environment
  - Single point of failure represented by the database



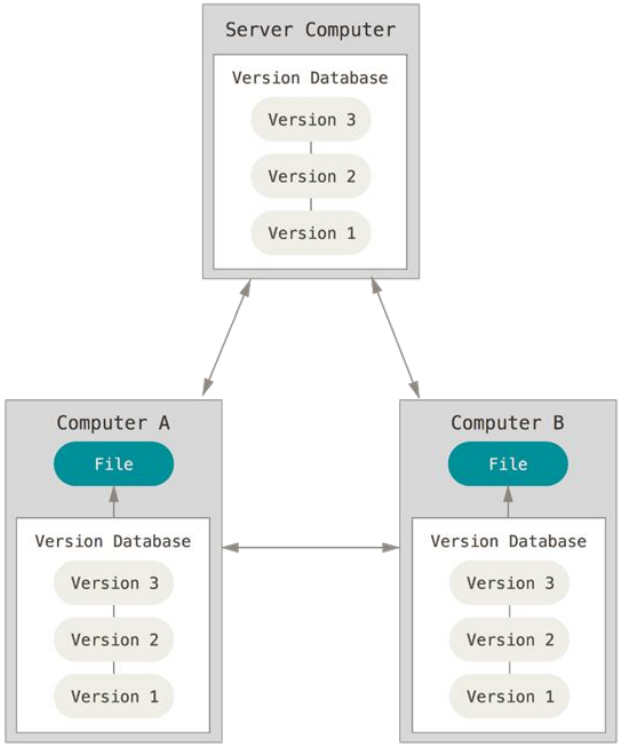
Centralized VCSs

- A central server containing the database of versions
- Clients check out files from the server
- Examples: Subversion (SVN), CVS
- Main limitations:
  - Single point of failure represented by the database
  - Frequent downloads and uploads



Distributed VCSs

- Clients don't just check out files from the server; they fully mirror the database of versions
  - Basically clients clone a remote database of versions locally
- Examples: Git, mercurial

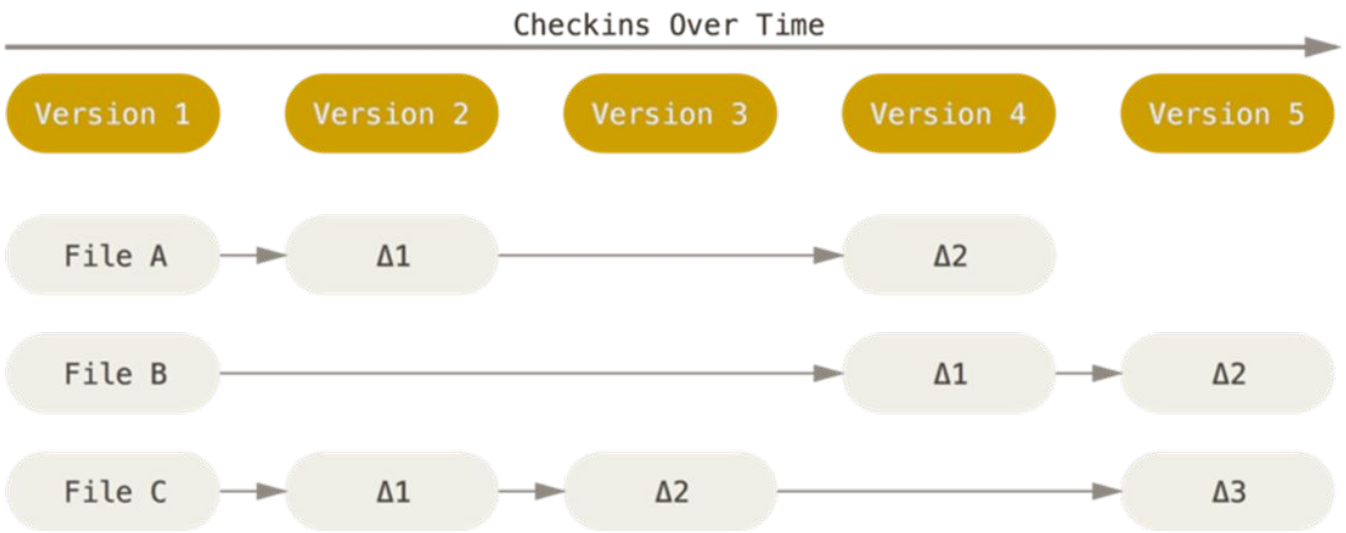


Git

- Git is a distributed version control system
- Developed by Linus Torvalds for managing the Linux Kernel evolution
- https://git-scm.com/

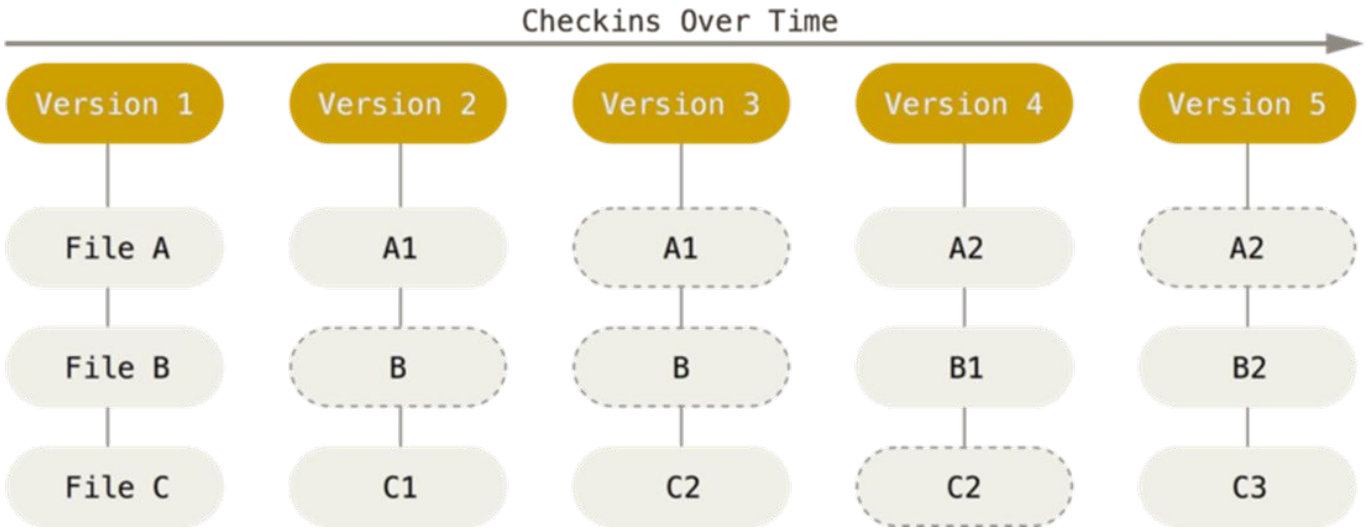
Git vs. SVN

- SVN is a delta-based VCS – i.e., it stores the first version of each file and then the changes made to each file (i.e., deltas) over time



Git vs. SVN

- Git stores snapshots of a miniature filesystem
- Every time a project (under version control) is modified, Git takes a picture of such a filesystem and stores a reference to that snapshot
  - If a file has not changed, Git doesn't store the file again, just a link to the previous identical file



Git support for Eclipse

- Via the Eclipse IDE you can perform Git commands like staging, commit, merge, rebase, pull and push

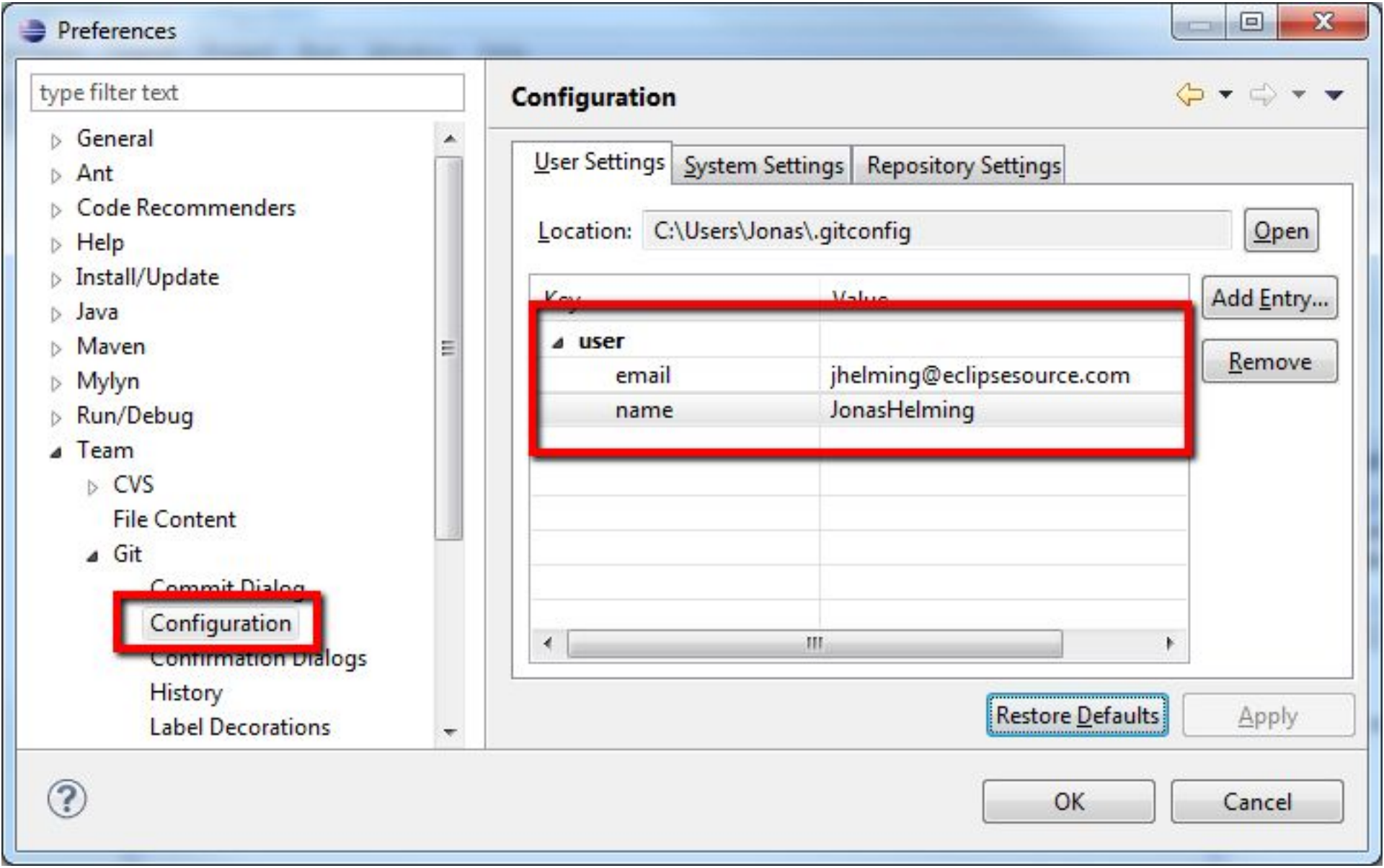
- EGit is the Git integration for the Eclipse IDE

Installing EGit in Eclipse

- Most Eclipse IDE distributions from Eclipse.org already contain support for Git
  - In this case no additional installation is required
- If the Git tooling is not available, you can install it via the Eclipse Marketplace
  - Select the **Help Marketplace…** menu entry. Search “egit” and install it.

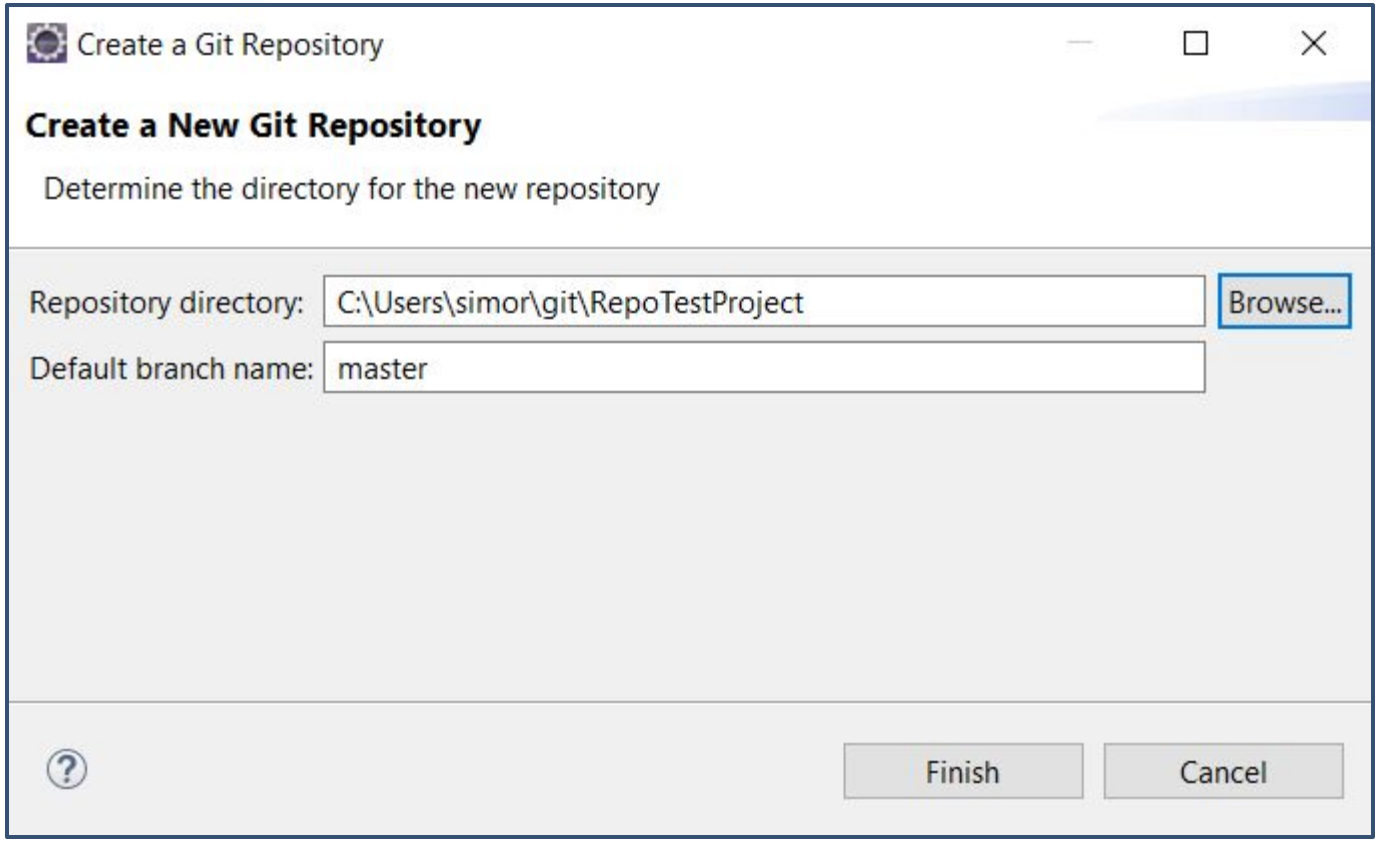
EGit Configuration

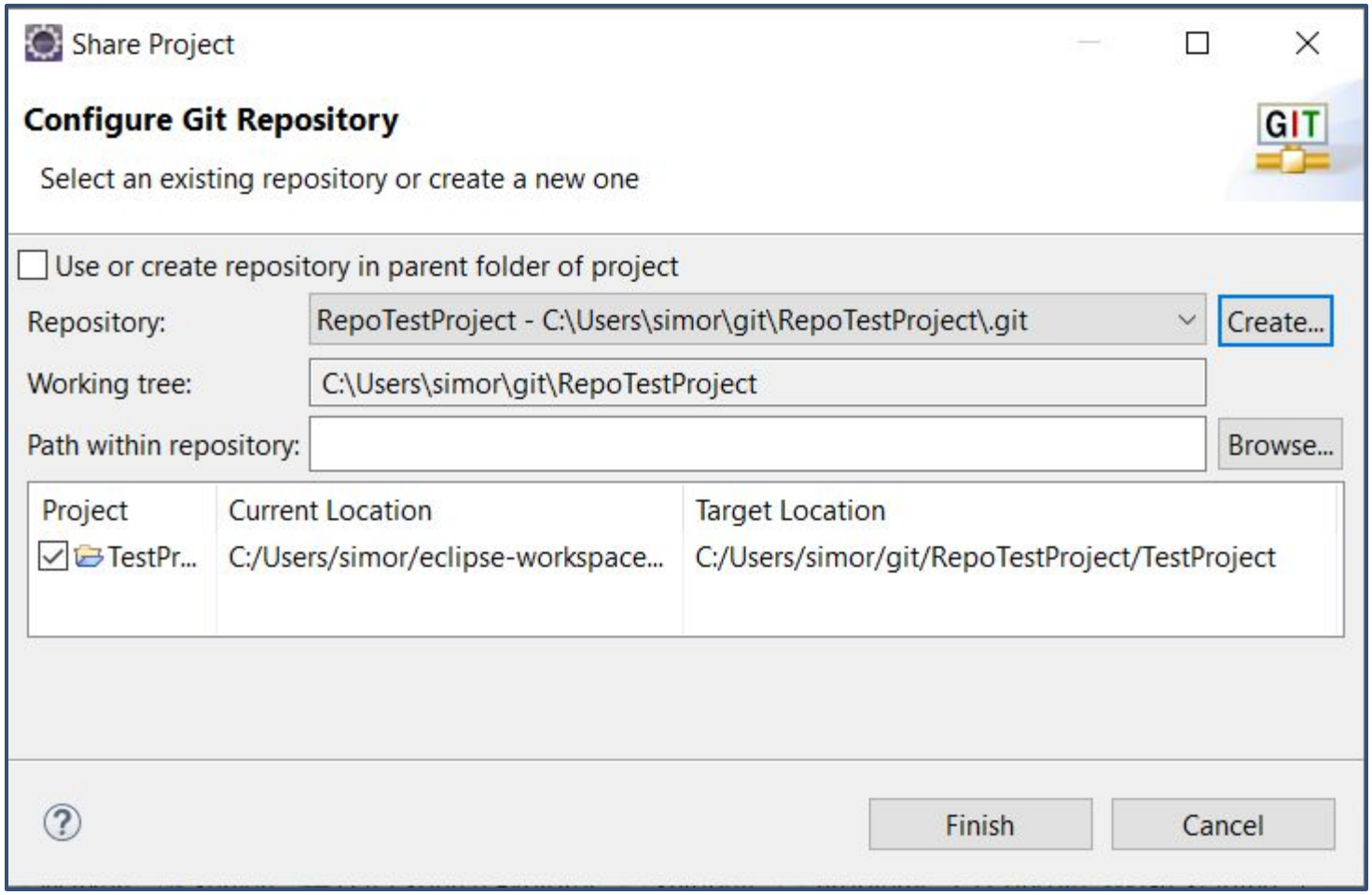
- **Every commit in EGit will include the user's name and his email-address**
- These attributes can be set in the Preferences-window ***Window □ Preferences***
  - Navigate to **(Version Control)** ***Team □ Git □ Configuration*** and hit the *New Entry…* Button
  - Enter ***user.name*** as *Key* and 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 (e.g, GitHub or Bitbucket) account**


Preferences
type filter text
General
Ant
Code Recommenders
Help
Install/Update
Java
Maven
Mylyn
Run/Debug
Team
CVS
File Content
Git
Commit Dialog
Configuration
Confirmation Dialogs
History
Label Decorations
Configuration
User Settings
System Settings
Repository Settings
Location:
C:\Users\Jonas\.gitconfig
Open
Key
Value
user
email
jhelming@eclipsesource.com
name
JonasHelming
Add Entry...
Remove
Restore Defaults
Apply
OK
Cancel

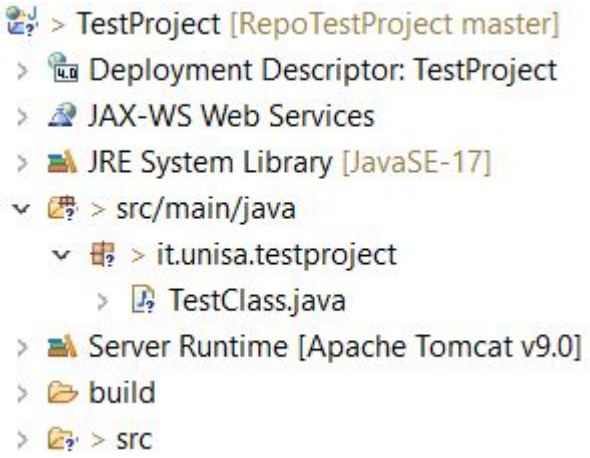
Creating Local Repositories

- 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 in Eclipse 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…***
  - In the following window select your project, hit the ***Create Repository*** button and click *Finish*


Create a Git Repository
Create a New Git Repository
Determine the directory for the new repository
Repository directory:
C:\Users\simor\git\RepoTestProject
Browse...
Default branch name:
master
Finish
Cancel


Share Project
Configure Git Repository
Select an existing repository or create a new one
GIT
Use or create repository in parent folder of project
Repository:
RepoTestProject - C:\Users\simor\git\RepoTestProject\.git
Create...
Working tree:
C:\Users\simor\git\RepoTestProject
Path within repository:
Browse...
Project
Current Location
Target Location
TestPr...
C:/Users/simor/eclipse-workspace...
C:/Users/simor/git/RepoTestProject/TestProject
Finish
Cancel

- The newly created repository will be empty, although the project is assigned to it
  - **Note the changed icons:** the project node and some child nodes will have an icon with a question mark
- 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 to Index***
  - After this operation, the question mark should change to an asterisk symbol
  - To set certain folders or files to be ignored by Git, e.g., the bin folder, right click them and select ***Team □ 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



Markers | Servers | Data Source Explorer | Snippets | Problems | Console | Git Staging ×

Filter files

> RepoTestProject [master]

Unstaged Changes (0)

Staged Changes (11)

- .classpath - TestProject
- .gitignore - TestProject
- .jsdtscope - TestProject/.settings
- .project - TestProject

Commit Message

Unborn branch: this commit will create the branch 'master'.

Create Project

Author: sromano87 <simo.romano@outlook.it>

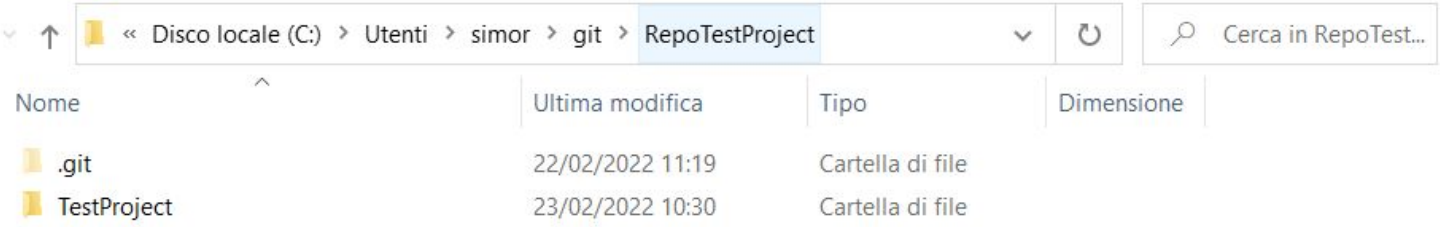
Committer: sromano87 <simo.romano@outlook.it>

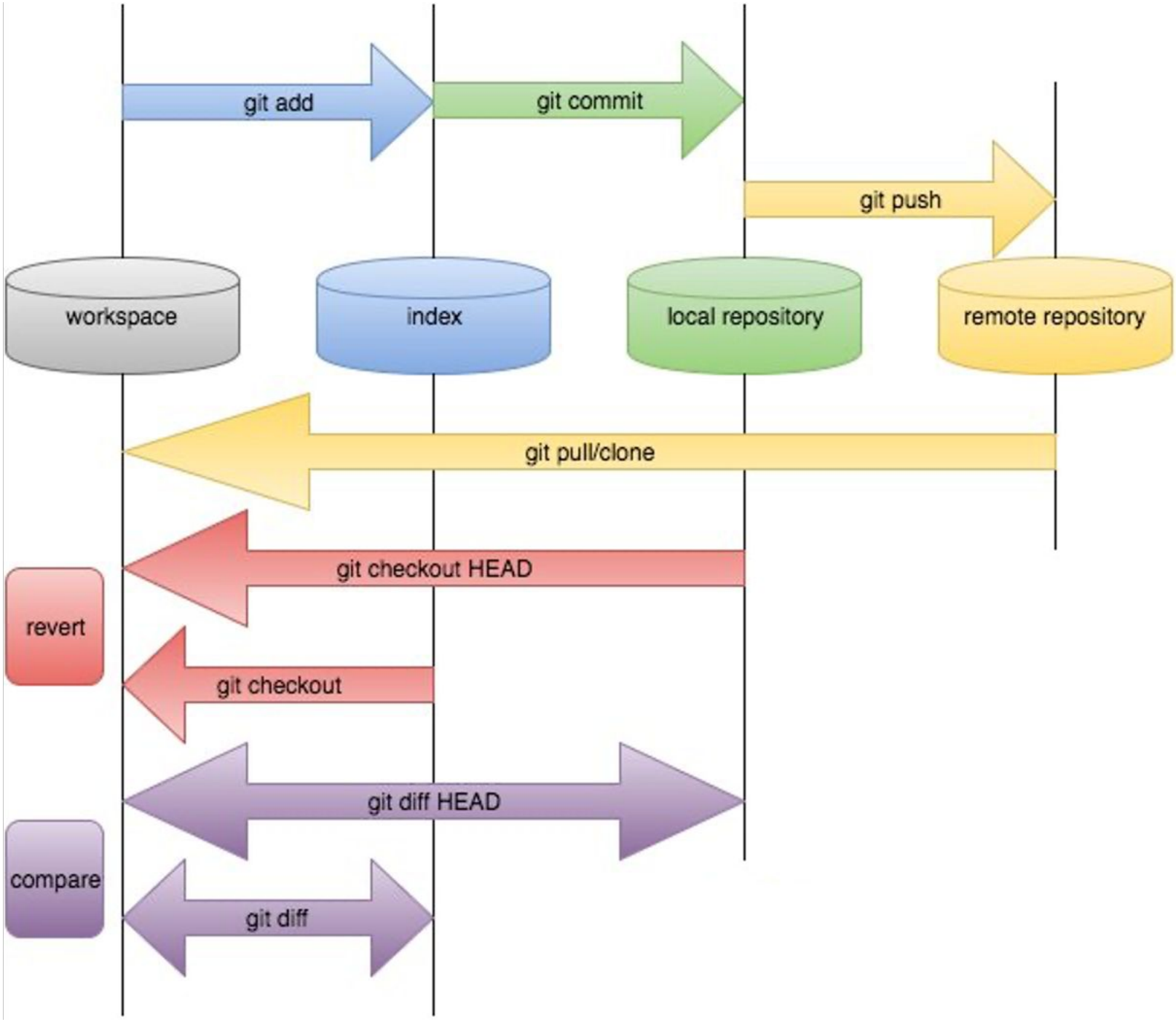
Commit and Push... | Commit

- TestProject [RepoTestProject master]
  - Deployment Descriptor: TestProject
  - JAX-WS Web Services
  - JRE System Library [JavaSE-17]
  - src/main/java
    - it.unisa.testproject
      - TestClass.java
  - Server Runtime [Apache Tomcat v9.0]
  - build
  - src

How a Git project looks like

- Git directory: a directory named *.git* containing, among other things, the local database of versions (i.e., the local repository)
- Working directory: the workspace for your project -- it contains the Git directory

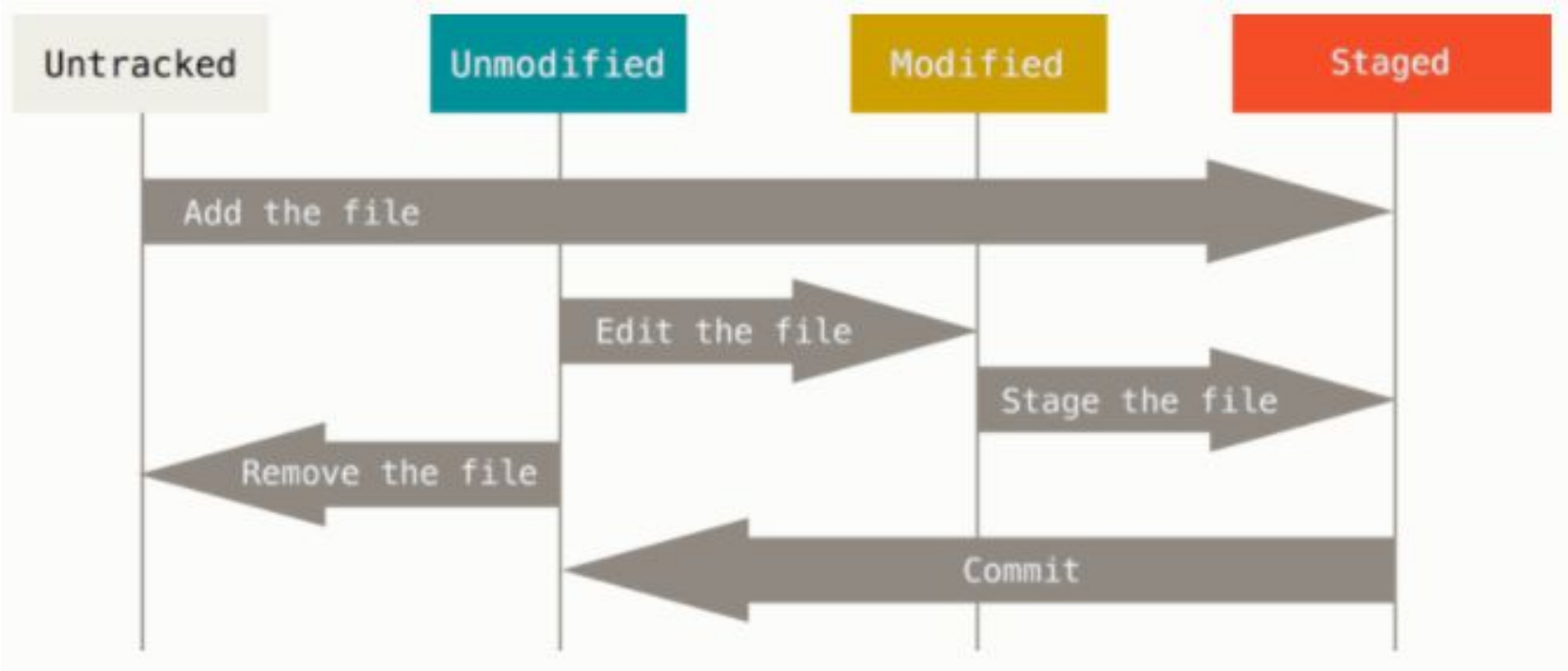



git add
git commit
git push
workspace
index
local repository
remote repository
git pull/clone
git checkout HEAD
revert
git checkout
git diff HEAD
compare
git diff

Icon Decorations/Signs

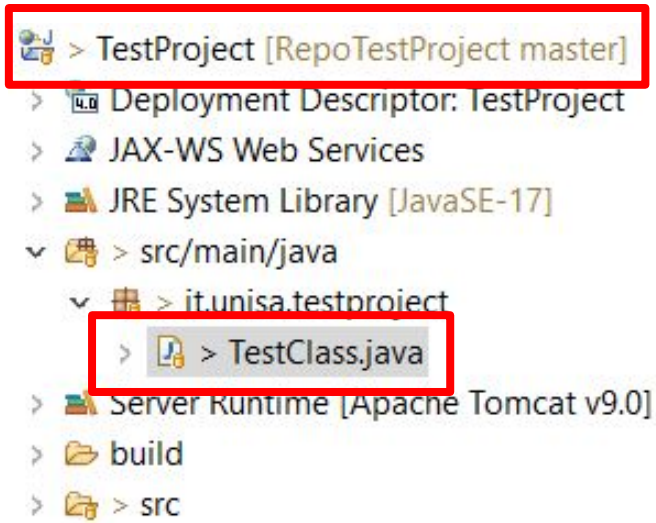
	ignored: The repository treats these files as if they were non-existent (e.g. the bin-directory by default). Add a .gitignore file or *Team => Ignore* to ignore a file.
	untracked: Any file known, but not yet recorded. To track a file, add it or select the *Show untracked files*-option in the commit-wizard 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.

Recording changes



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

Project Explorer

- Servers
- > TestProject [RepoTestProject master]
  - Deployment Descriptor: TestProject
  - JAX-WS Web Services
  - JRE System Library [JavaSE-17]
  - > src/main/java
    - > it.unisa.testproject
      - > TestClass.java
  - Server Runtime [Apache Tomcat v9.0]
  - build
  - > src

TestClass.java

```
1  package it.unisa.testproject;
2
3  public class TestClass {
4
5      public static void main (String[] args) {
6          |
7      }
8
9  }
10
```

Change

Markers | Servers | Data Source Explorer | Snippets | Problems | Console | Git Staging | Filter files

> RepoTestProject [master]

Unstaged Changes (1)

> TestClass.java - TestProject/src/main/java/it/unisa/testproject

Staged Changes (0)

Commit Message

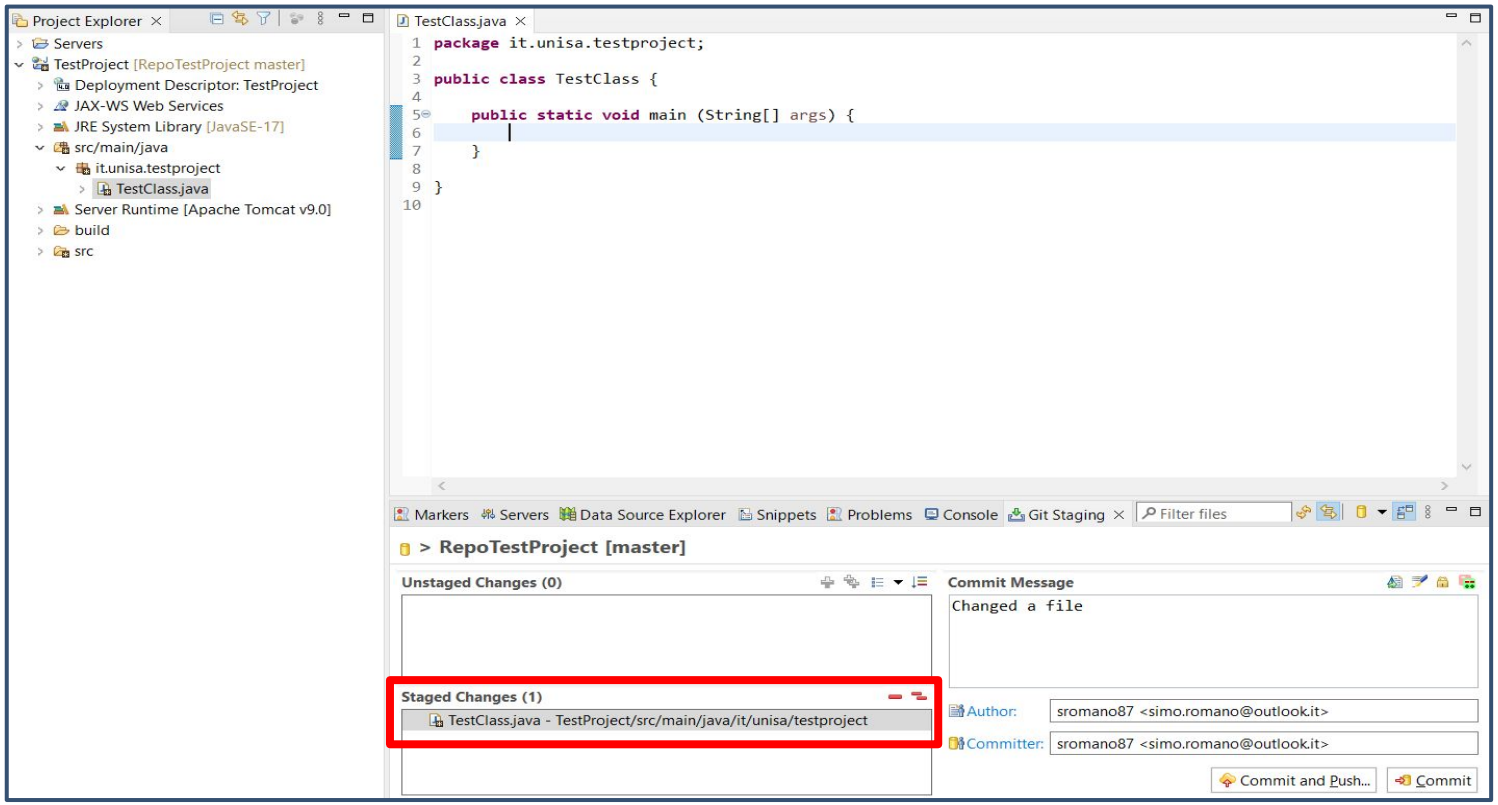
Changed a file

Author: sromano87 <simo.romano@outlook.it>

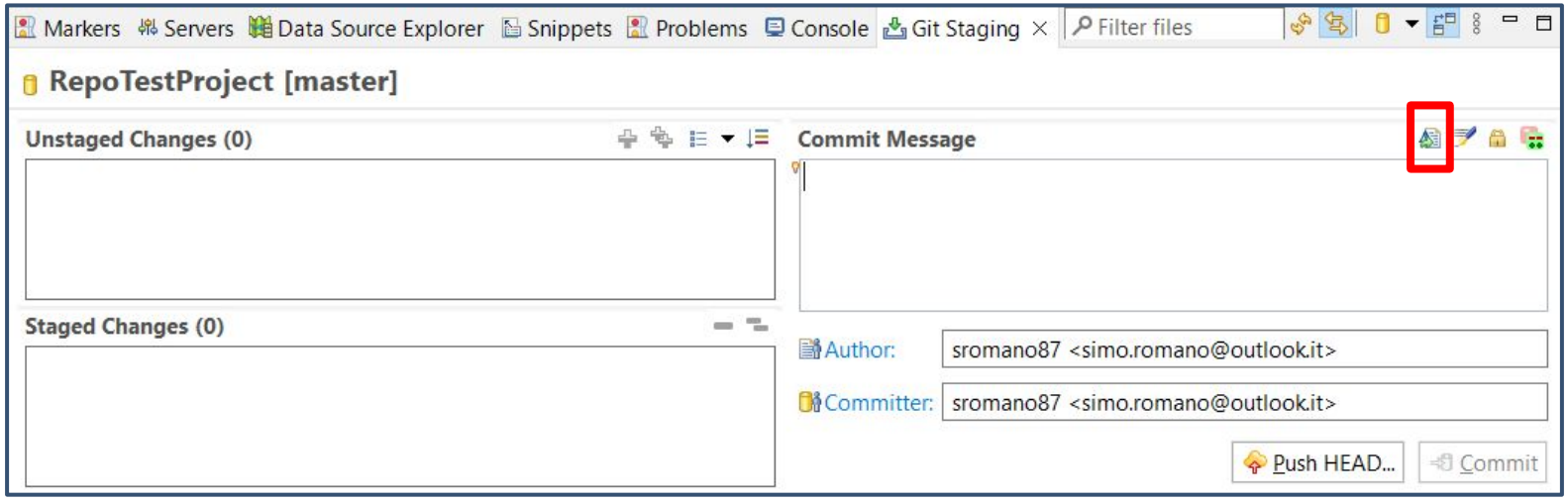
Committer: sromano87 <simo.romano@outlook.it>

Push HEAD... | Commit

- Note that the status of the changed file is *modified, not staged*
  - By staging the files before you commit, you can change the status to *modified* (and the dirty sign to a staged icon)

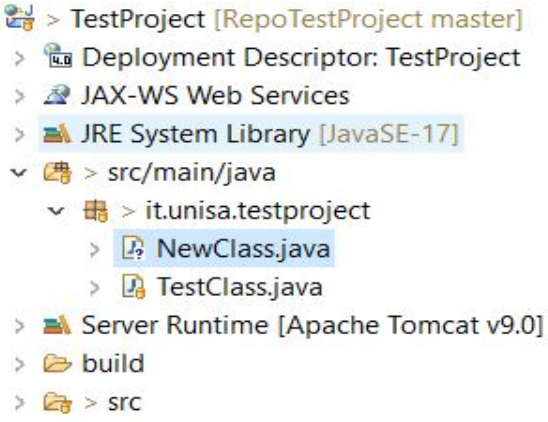


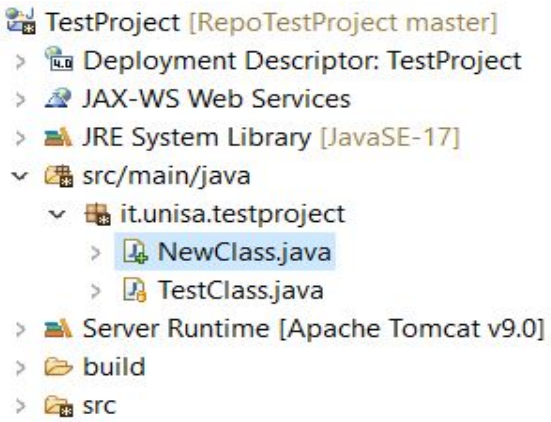
- If you later realize that your previous commit was incomplete (e.g., you forgot to commit a file) or your commit message was wrong, you might want to use ***Amend previous commit***
  - This will merge the current commit and the previous commit into one, so you don't have to perform an extra commit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



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 to Index***
  - 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'





- In the next commit, the file will be added to the repository and the plus symbol will turn into a repository icon

Reverting Changes

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

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

TestClass.java | NewClass.java | Compare TestClass.java Current and f8888a9

Java Structure Compare

- Compilation Unit
  - TestClass
    - TestClass()
    - main(String [])

Java Source Compare

Copy All Non-Conflicting Changes from Right to Left

Local: TestClass.java

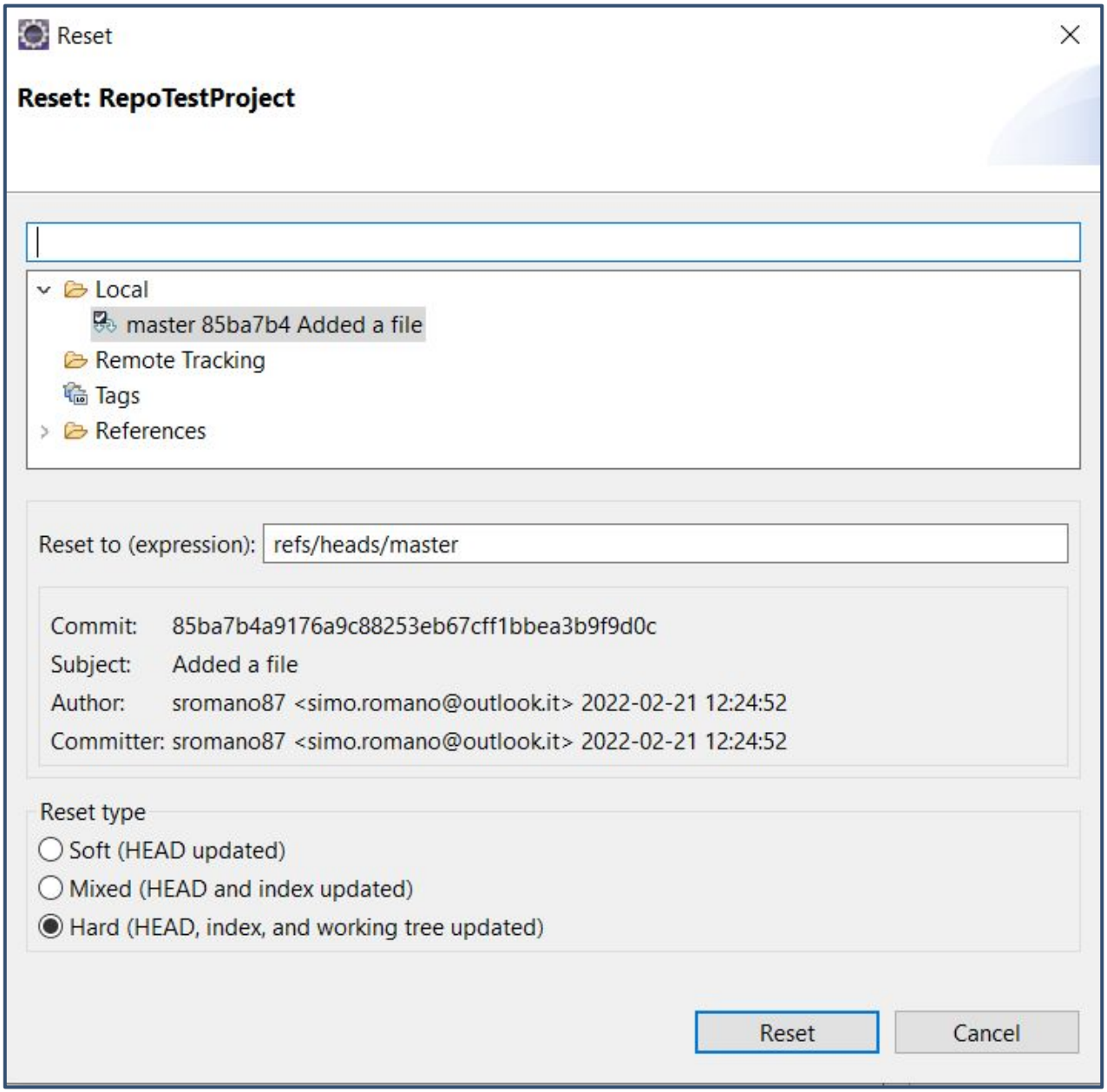
```
1 package it.unisa.testproject;
2
3 public class TestClass {
4
5     public static void main (String[] args) {
6     }
7
8     public TestClass() {
9     }
10
11 }
12
```

TestClass.java f8888a9 (sromano87)

```
1 package it.unisa.testproject;
2
3 public class TestClass {
4
5     public static void main (String[] args) {
6
7     }
8
9 }
10
```

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 in the section "index")
  - Be careful with this option as all changes in your workspace will be lost


Reset
Reset: RepoTestProject
Local
master 85ba7b4 Added a file
Remote Tracking
Tags
References
Reset to (expression): refs/heads/master
Commit: 85ba7b4a9176a9c88253eb67cff1bbea3b9f9d0c
Subject: Added a file
Author: sromano87 <simo.romano@outlook.it> 2022-02-21 12:24:52
Committer: sromano87 <simo.romano@outlook.it> 2022-02-21 12:24:52
Reset type
Soft (HEAD updated)
Mixed (HEAD and index updated)
Hard (HEAD, index, and working tree updated)
Reset
Cancel

Publishing Repositories

- Right click the project node and navigate to ***Team Push branch (master)***
- Set the destination Git Repository information
  - URI: ***https://simone_romano@bitbucket.org/simone_romano/repotestproject.git***
  - Protocol: ***https***
- Set the authentication information
- Click the Preview button (two times)
- Finally, click on the Push button

Push Branch master

Destination Git Repository

Enter the location of the destination repository.

Remote name: origin

Location

URI: https://bitbucket.org/simone_romano/repotestproject.git Local Folder...

Host: bitbucket.org

Repository path: /simone_romano/repotestproject.git

Connection

Protocol: https

Port:

Authentication

User: simo.romano@outlook.it

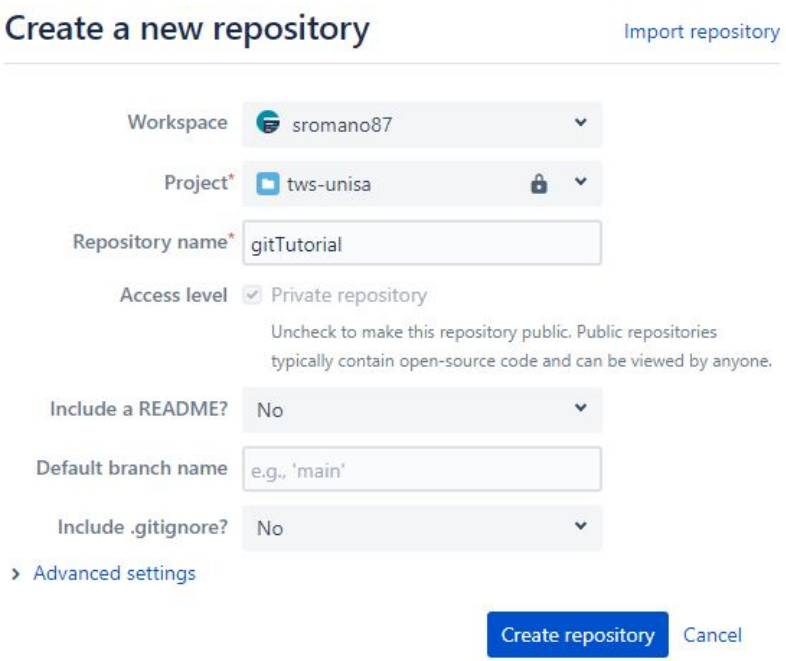
Password: ••••••••••••••

☑ Store in Secure Store

< Back | Preview > | Push | Cancel

Cloning Repositories

- For this and some of the following sections (especially Fetch/Push), you might want to use https://bitbucket.org/ to create your own remote repository
  - Public repositories are free at BitBucket, as well as private repositories with up to 5 users



Let's put some bits in your bucket

HTTPS git clone https://simone_romano@bitbucket.org/simone_romano/gittutori

Get started quickly

Creating a README or a .gitignore is a quick and easy way to get something into your repository.

Create a README Create a .gitignore

Get your local Git repository on Bitbucket

Step 1: Switch to your repository's directory

```
cd /path/to/your/repo
```

Step 2: Connect your existing repository to Bitbucket

```
git remote add origin https://simone_romano@bitbucket.org/simone_romano/gittutorial.git
git push -u origin master
```

Need more information? Learn more

- 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 "Clone 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 BitBucket, you can copy the URI from the web page
- Select all **branches** (if any) you wish to clone and hit *Next* again
- Choose a local directory to save this repository in

Import Projects from Git

Source Git Repository

Enter the location of the source repository.

Location

URI: https://simone_romano@bitbucket.org/simone_romano/gittutorial.git [Local Folder...] [Local Bundle File...]

Host: bitbucket.org

Repository path: /simone_romano/gittutorial.git

Connection

Protocol: https

Port:

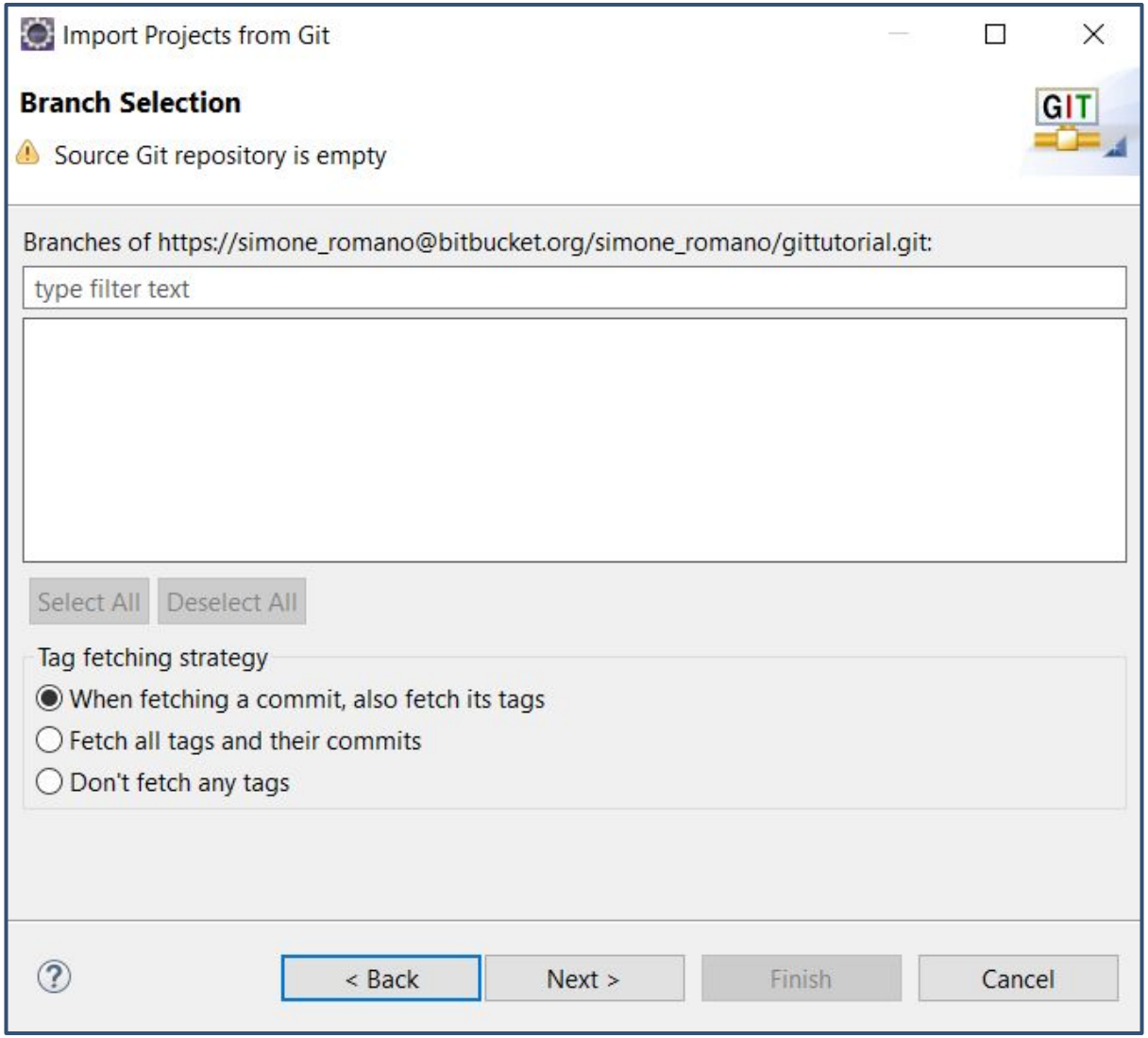
Authentication

User: simo.romano@outlook.it

Password: ••••••••••••••

☐ Store in Secure Store

[?] [< Back] [Next >] [Finish] [Cancel]


Import Projects from Git
Branch Selection
Source Git repository is empty
GIT
Branches of https://simone_romano@bitbucket.org/simone_romano/gittutorial.git:
type filter text
Select All
Deselect All
Tag fetching strategy
When fetching a commit, also fetch its tags
Fetch all tags and their commits
Don't fetch any tags
< Back
Next >
Finish
Cancel

Import Projects from Git

Local Destination

Configure the local storage location for gittutorial.

Destination

Directory: C:\Users\simor\git\RepoGitTutorial\gittutorial Browse

Initial branch:

☐ Clone submodules

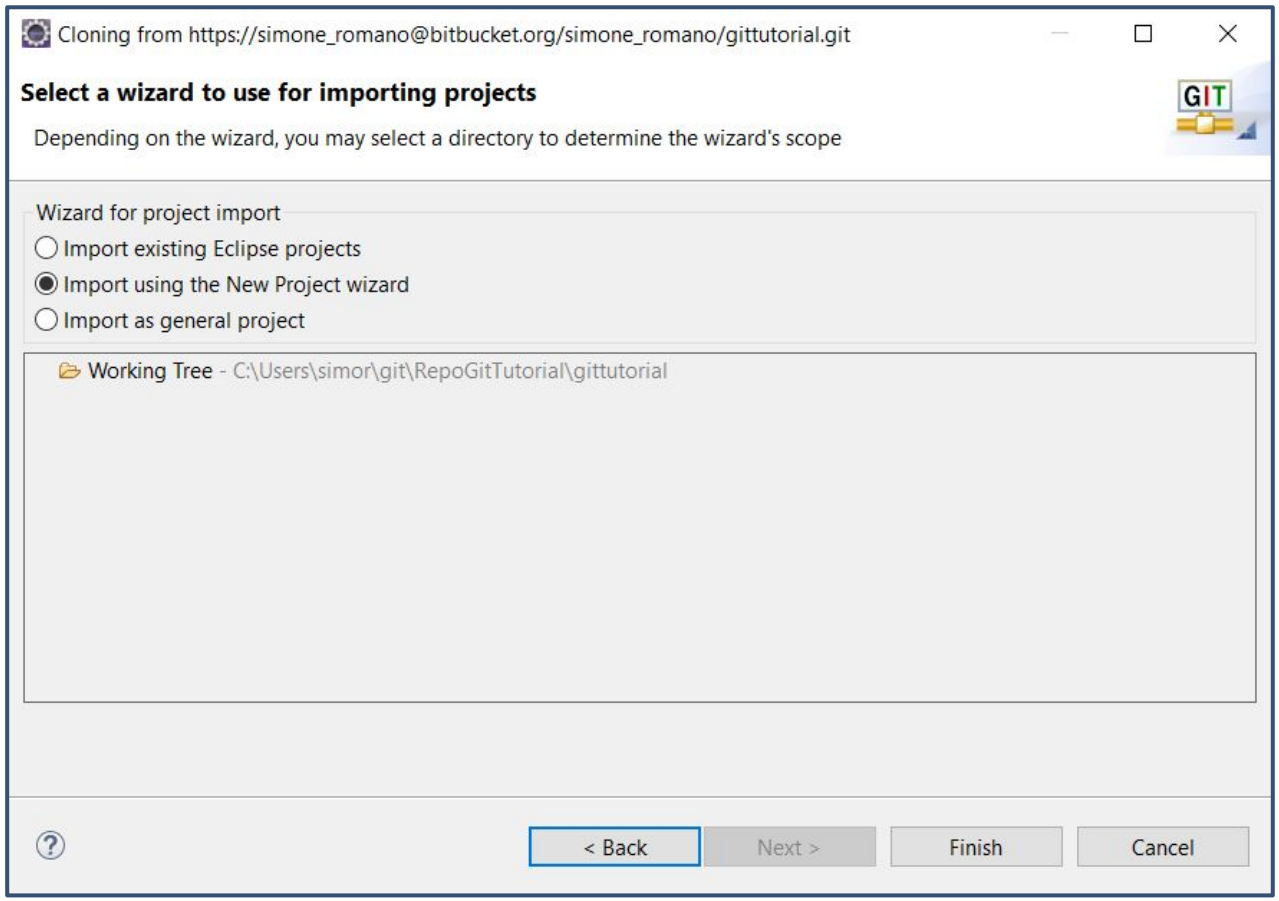
Configuration

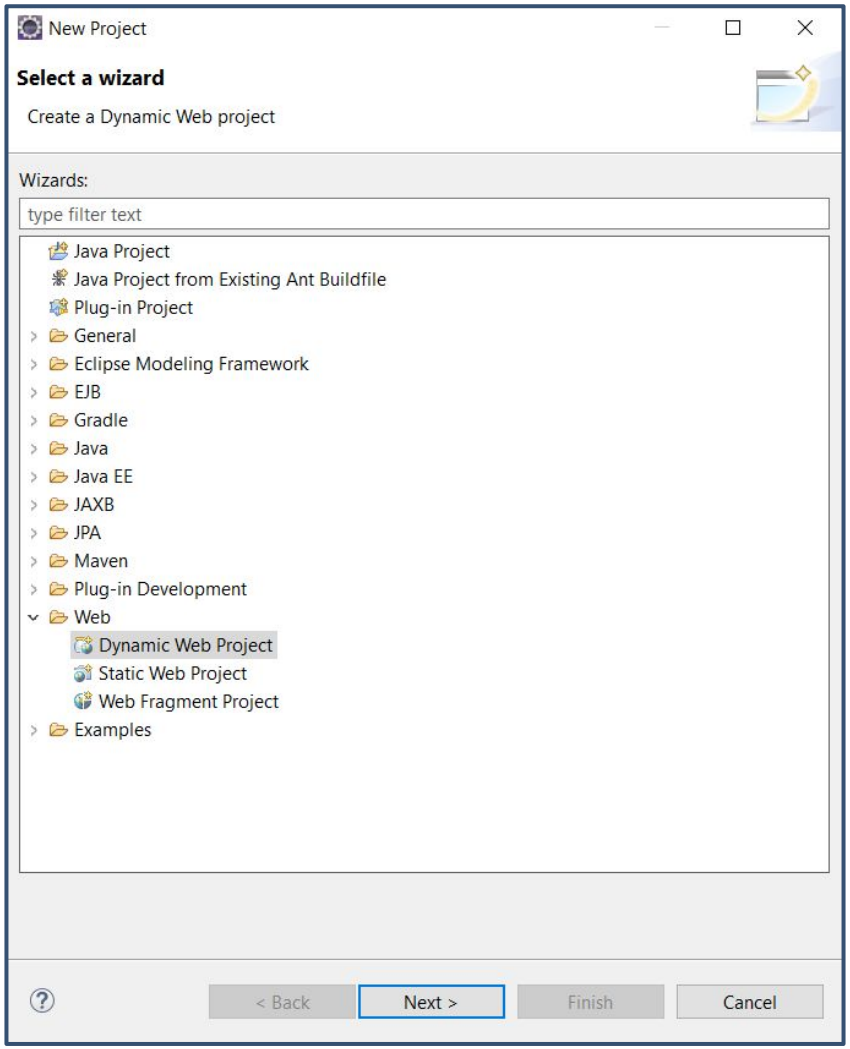
Remote name: origin

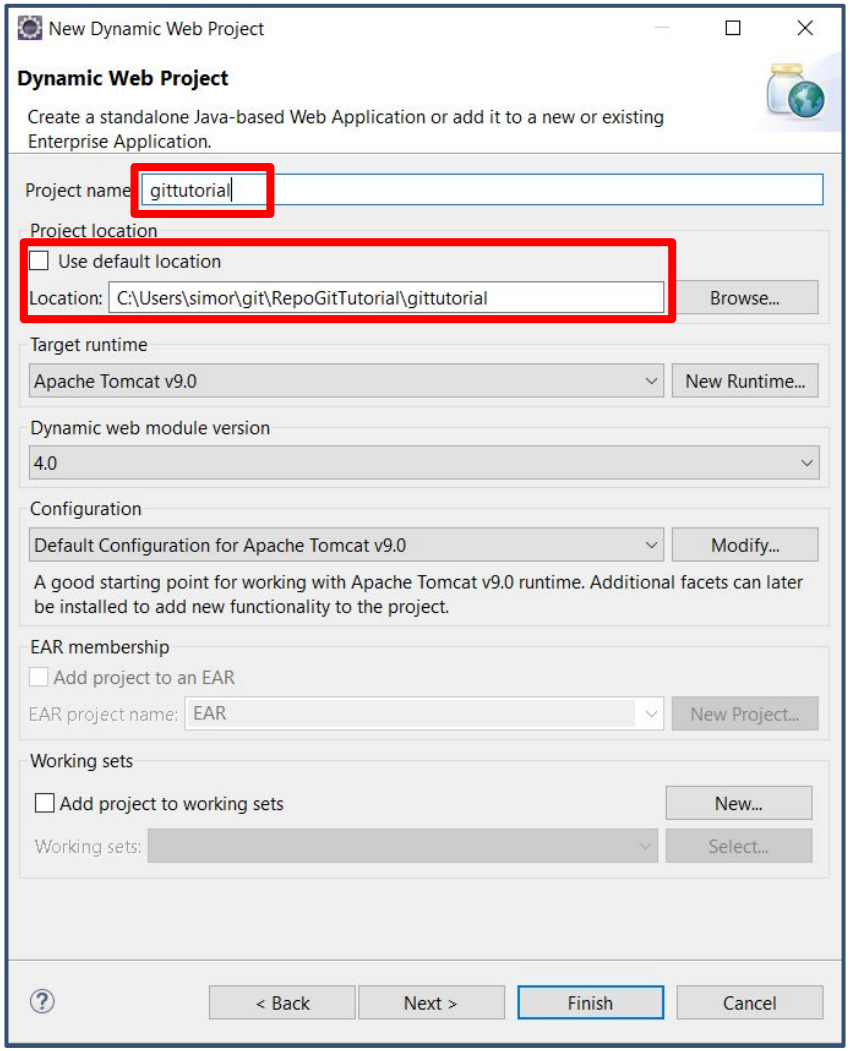
< Back | Next > | Finish | Cancel

- Select the wizard for the project you will import into Eclipse and hit Next
  - ***Import Existing Eclipse Projects*** if you will import an Eclipse Project
  - Or ***Import using the New Project wizard*** if you will init a project (for example a dynamic web project)
- In the following windows, provide the information required from the selected wizard
- The project should now appear in the Navigator/Package Explorer

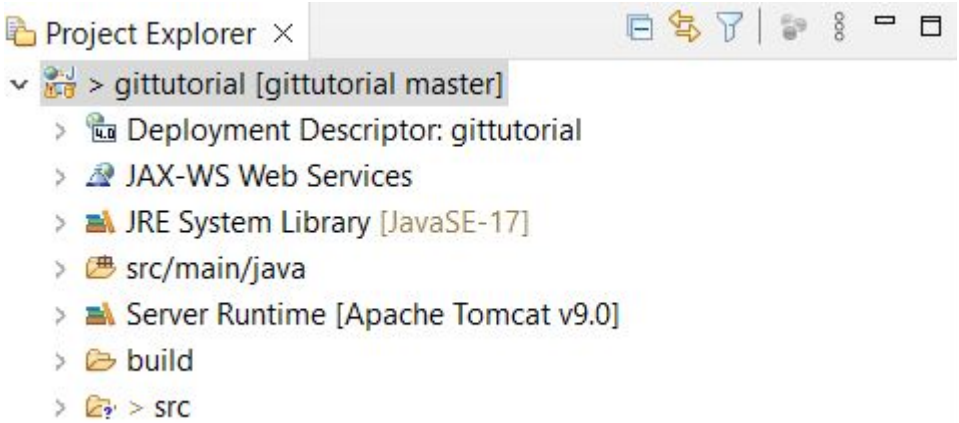
- If you already have an Eclipse project, choose ***Import Existing projects***
    - If not, ***Import using the New Project wizard***




New Project
Select a wizard
Create a Dynamic Web project
Wizards:
type filter text
Java Project
Java Project from Existing Ant Buildfile
Plug-in Project
General
Eclipse Modeling Framework
EJB
Gradle
Java
Java EE
JAXB
JPA
Maven
Plug-in Development
Web
Dynamic Web Project
Static Web Project
Web Fragment Project
Examples
< Back
Next >
Finish
Cancel

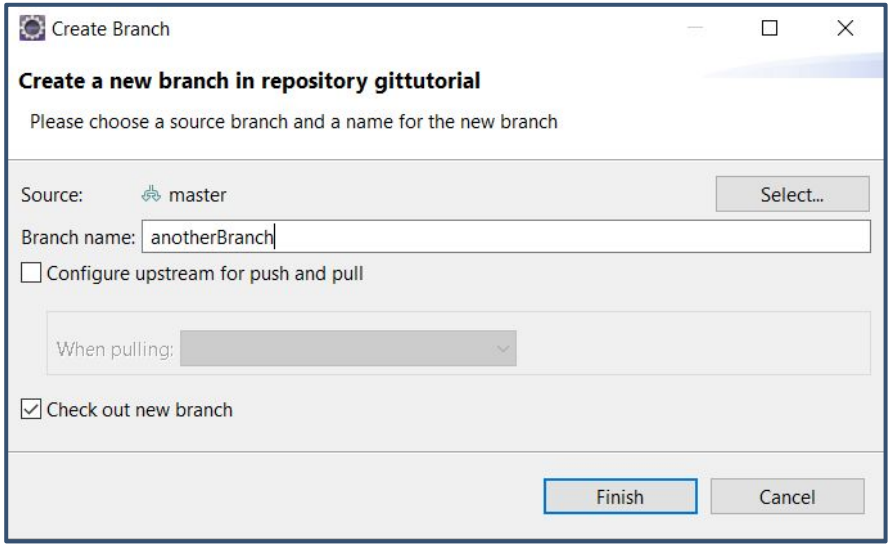

New Dynamic Web Project
Dynamic Web Project
Create a standalone Java-based Web Application or add it to a new or existing Enterprise Application.
Project name: gittutorial
Project location
Use default location
Location: C:\Users\simor\git\RepoGitTutorial\gittutorial
Browse...
Target runtime
Apache Tomcat v9.0
New Runtime...
Dynamic web module version
4.0
Configuration
Default Configuration for Apache Tomcat v9.0
Modify...
A good starting point for working with Apache Tomcat v9.0 runtime. Additional facets can later be installed to add new functionality to the project.
EAR membership
Add project to an EAR
EAR project name: EAR
New Project...
Working sets
Add project to working sets
New...
Working sets:
Select...
< Back
Next >
Finish
Cancel

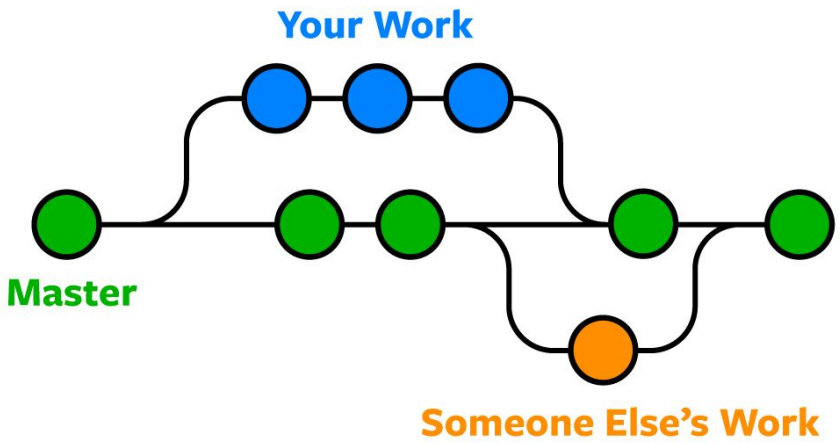
- Note that there are untracked files
  - You will have to add these files to the index and then commit them

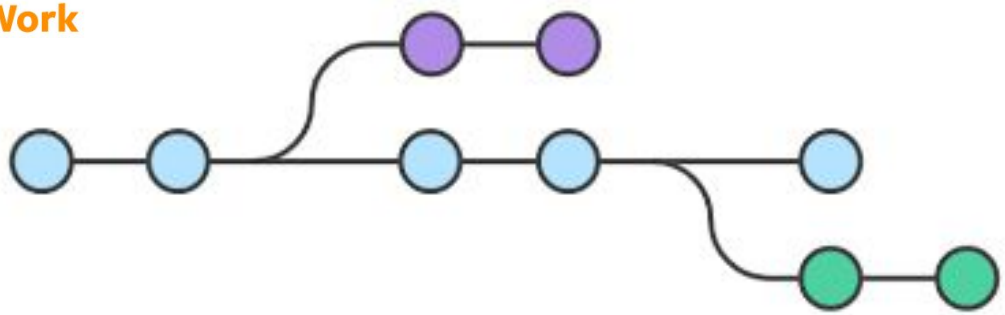


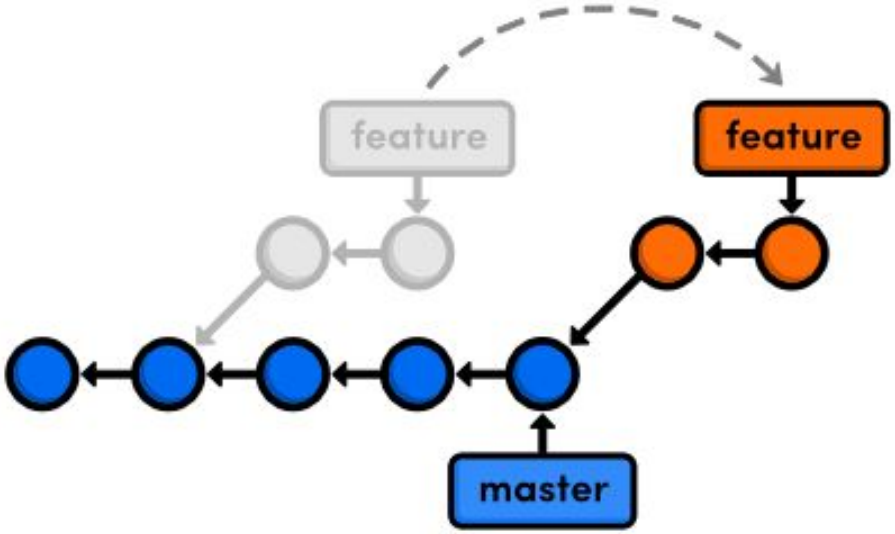
Creating Branches

- 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
  - Click ***Select…*** to choose the branch you want to create a new branch from
  - Name the new branch
  - Ensure to select ***Check out new branch***, if you want to check out the newly created branch

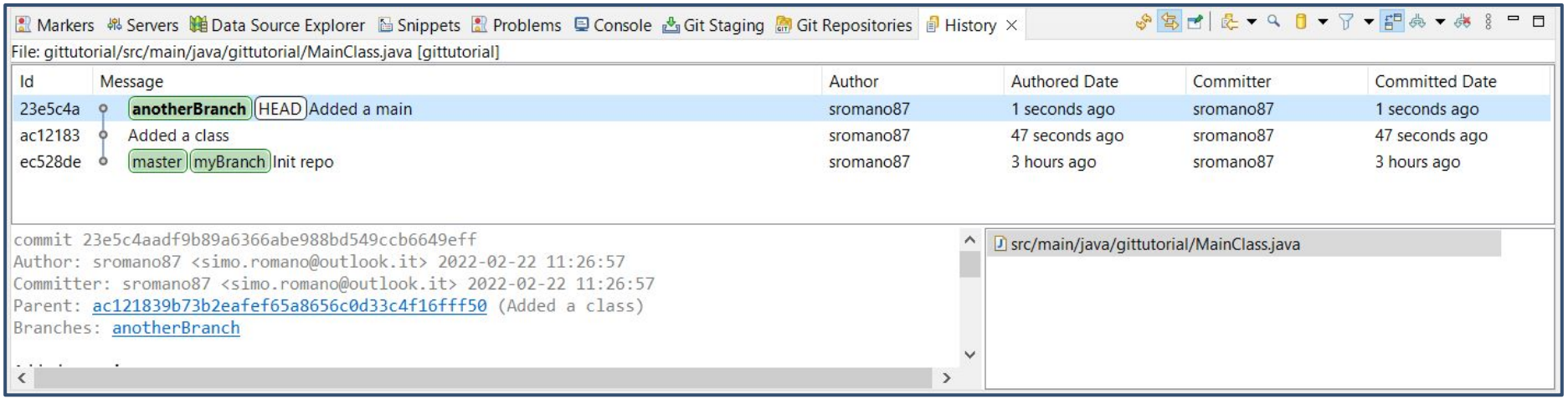



Your Work
Master
Someone Else's Work



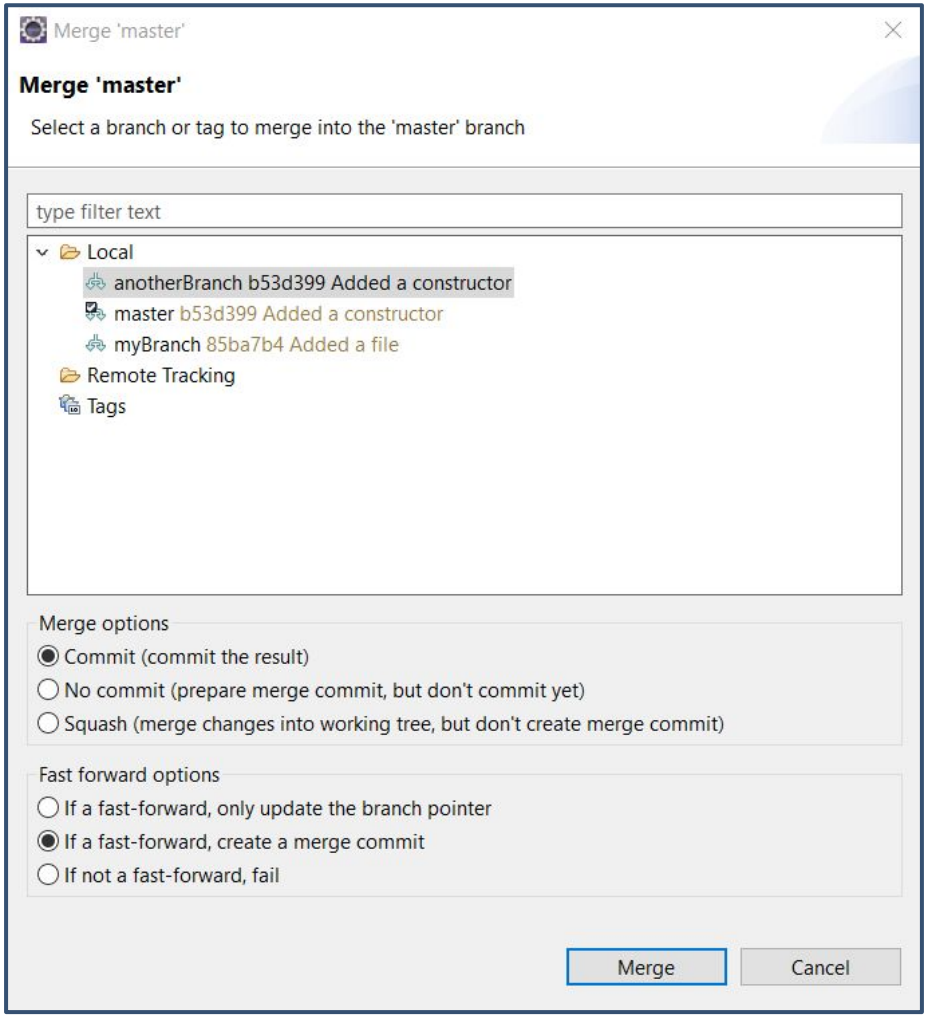

feature
feature
master

- The branch you are currently working on is reported within square brackets in the project node
- To see the commits in the current branch
    - Right click the project and navigate to ***Team → Show in History***



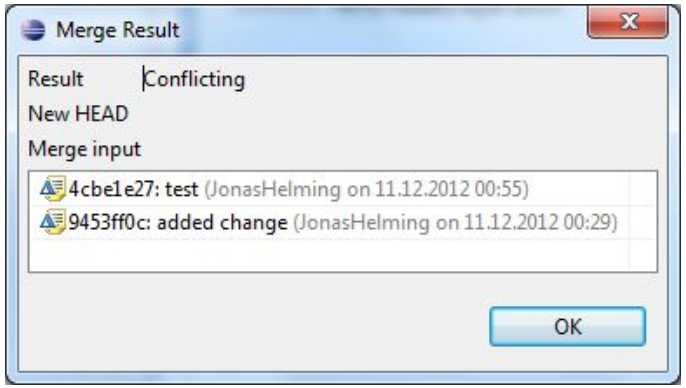
Merge

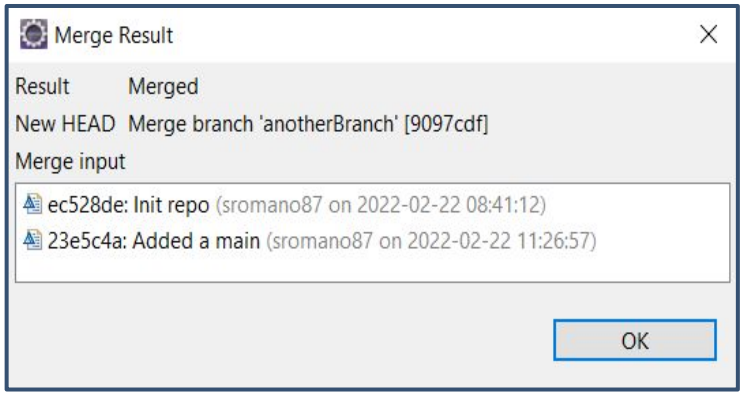
- To merge one branch into another, you will have to check out the branch you want to merge with
  - Right click the project node and navigate to ***Team → Switch to***, then select the branch you want to check out from -- skip this step if you want to merge the current branch
  - Right click the project node and navigate to ***Team → Merge…***
  - Select any branch (other than the checked out branch), select ***commit*** as ***merge option*** and ***create a commit merge*** as ***fast forward option***, and hit ***Merge***


Merge 'master'
Merge 'master'
Select a branch or tag to merge into the 'master' branch
type filter text
Local
anotherBranch b53d399 Added a constructor
master b53d399 Added a constructor
myBranch 85ba7b4 Added a file
Remote Tracking
Tags
Merge options
Commit (commit the result)
No commit (prepare merge commit, but don't commit yet)
Squash (merge changes into working tree, 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
Merge
Cancel

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* (see the image in the right bottom corner), *Conflicting* (see the image in the left bottom corner), *Failed*
  - A *Conflicting* result will leave the merge process incomplete
    - You will have to resolve the conflicts
  - A *Failed* result may occur when there are already conflicting changes in the working directory





Markers | Servers | Data Source Explorer | Snippets | Problems | Console | Git Staging | Git Repositories | History

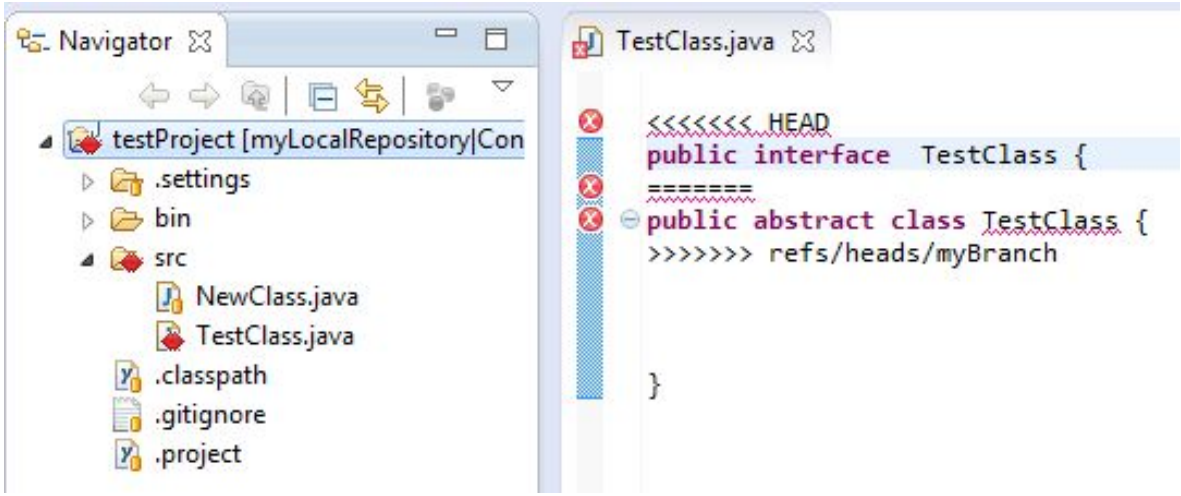
Project: gittutorial [gittutorial]

Id	Message	Author	Authored Date	Committer	Committed Date
9097cdf	master HEAD Merge branch 'anotherBranch'	sromano87	1 seconds ago	sromano87	1 seconds ago
23e5c4a	anotherBranch Added a main	sromano87	2 minutes ago	sromano87	2 minutes ago
ac12183	Added a class	sromano87	3 minutes ago	sromano87	3 minutes ago
ec528de	myBranch Init repo	sromano87	3 hours ago	sromano87	3 hours ago

```
commit 9097cdf6c4a383f6369b3e90a2d2b62e7c6520de
Author: sromano87 <simo.romano@outlook.it> 2022-02-22 11:29:04
Committer: sromano87 <simo.romano@outlook.it> 2022-02-22 11:29:04
Parent: ec528de2a41204735a3ff2dbe4e135678fa2ab1b (Init repo)
Parent: 23e5c4aadf9b89a6366abe988bd549ccb6649eff (Added a main)
Branches: master
```

Resolving Conflicts

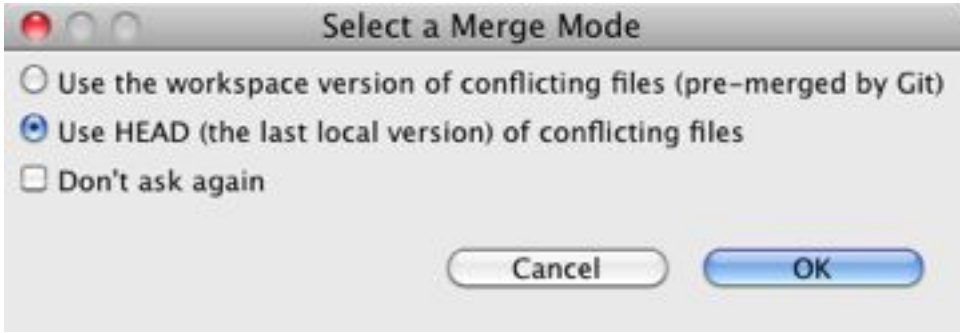
- If your merge resulted in conflicts (note the red symbols on the file icons), you will have to resolve these manually
- Open the conflicting files and scroll to the conflicting changes marked with "<<<<<<<"

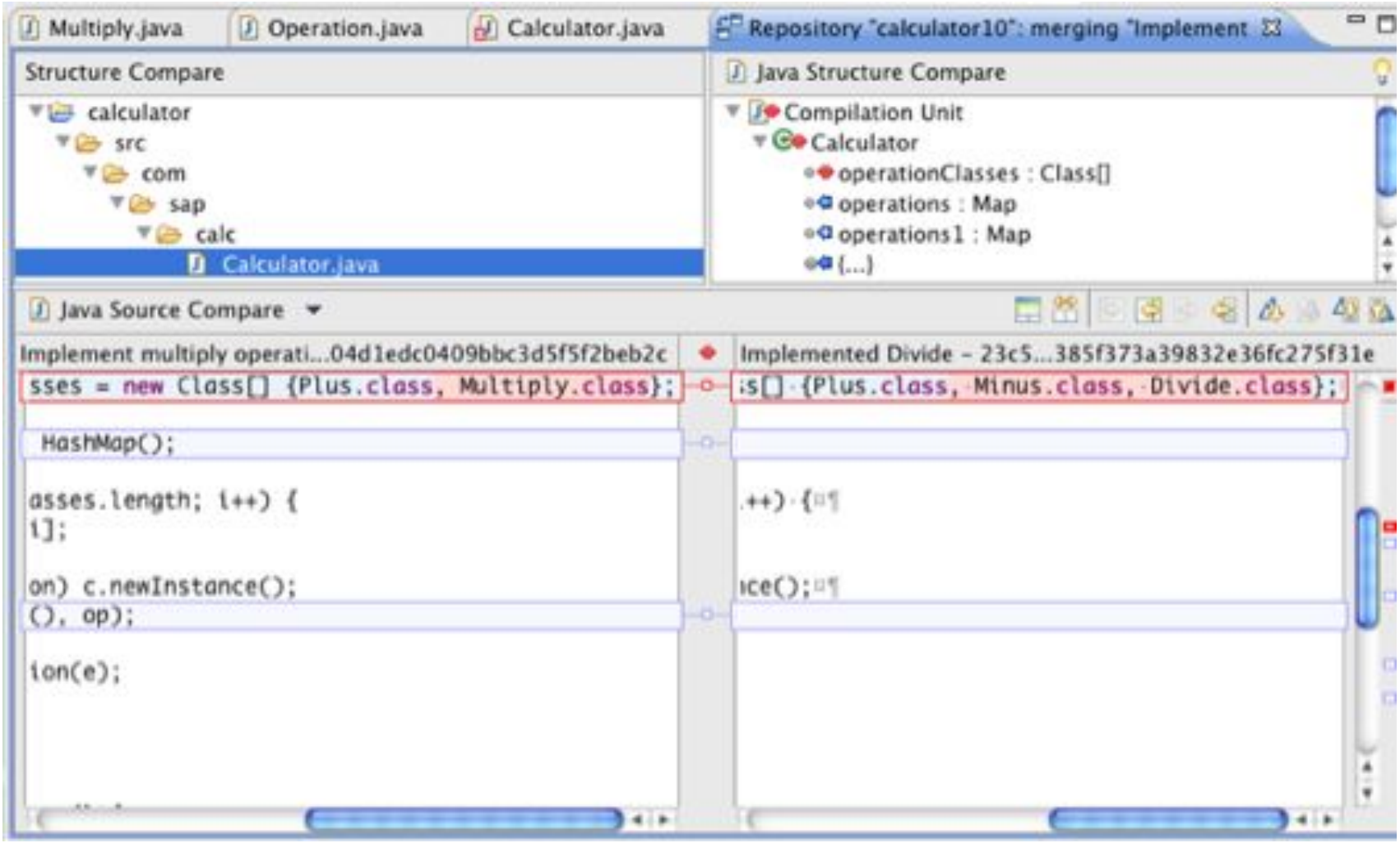


- After you are finished the manual part of the merge, you will have to tell Git that the conflicts are resolved
  - To do so, ***Add*** the files and ***Commit*** to complete your merge

Use Merge tool

- Select the top level resource showing the red conflict label decorator
  - Click **Team □ Merge Tool**
  - Select the merge mode *Use HEAD (the last local version) of conflicting files* and click *OK*



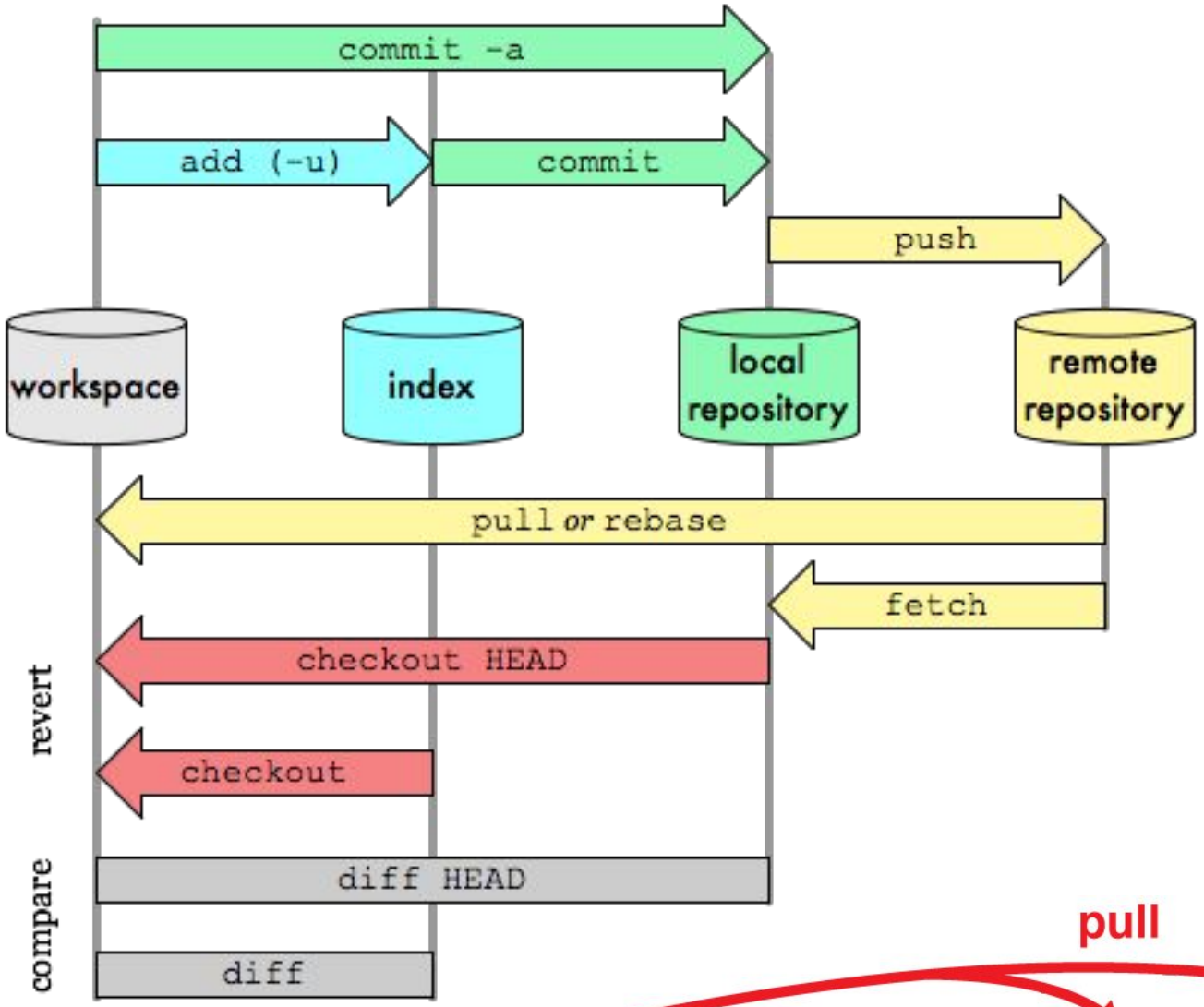

Multiply.java
Operation.java
Calculator.java
Repository "calculator10": merging "Implement
Structure Compare
calculator
src
com
sap
calc
Calculator.java
Java Structure Compare
Compilation Unit
Calculator
operationClasses : Class[]
operations : Map
operations1 : Map
{...}
Java Source Compare
Implement multiply operati...04d1edc0409bbc3d5f5f2beb2c
Implemented Divide - 23c5...385f373a39832e36fc275f31e
sses = new Class[] {Plus.class, Multiply.class};
HashMap();
asses.length; i++) {
i];
on) c.newInstance();
(), op);
ion(e);
s[] {Plus.class, Minus.class, Divide.class};
++) {
ice();

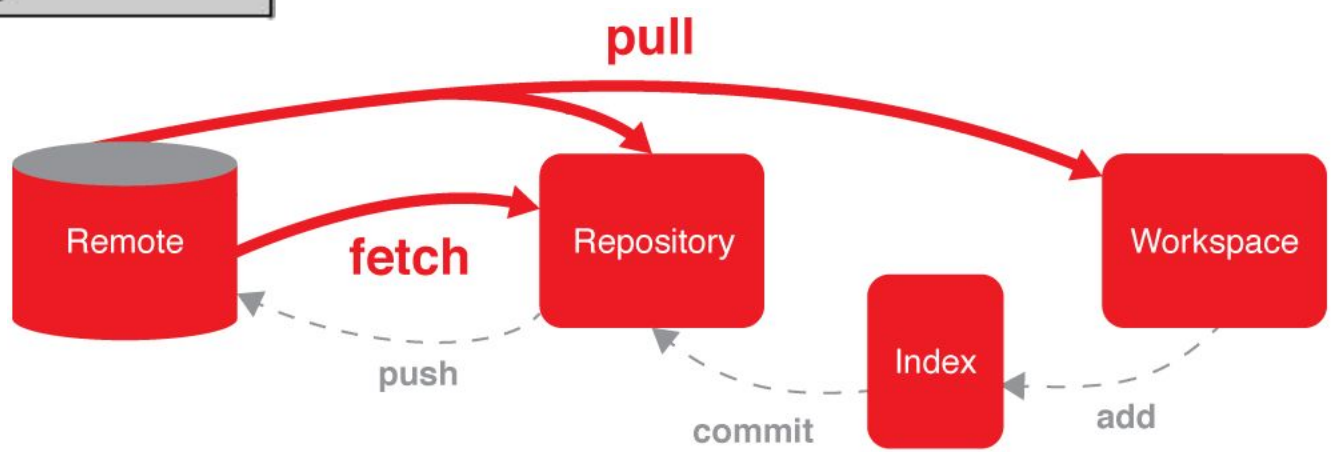
Fetch and Pull

- 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 → Remote → 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*

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
- The ***Push*** wizard is pretty much the same as the *Fetch* wizard
  - First, right click the project node and navigate to ***Team □ Remote □ 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*

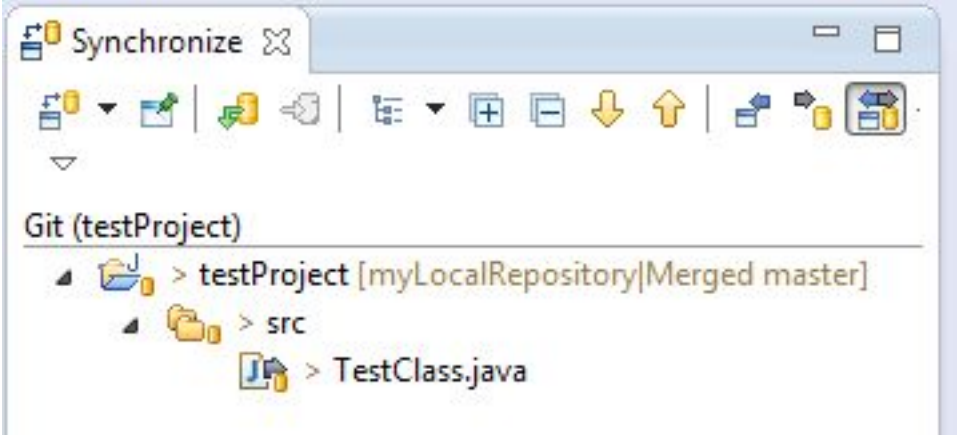

commit -a
add (-u)
commit
push
workspace
index
local repository
remote repository
pull or rebase
fetch
checkout HEAD
checkout
revert
diff HEAD
diff
compare


pull
Remote
fetch
Repository
Workspace
Index
push
commit
add

Synchronize

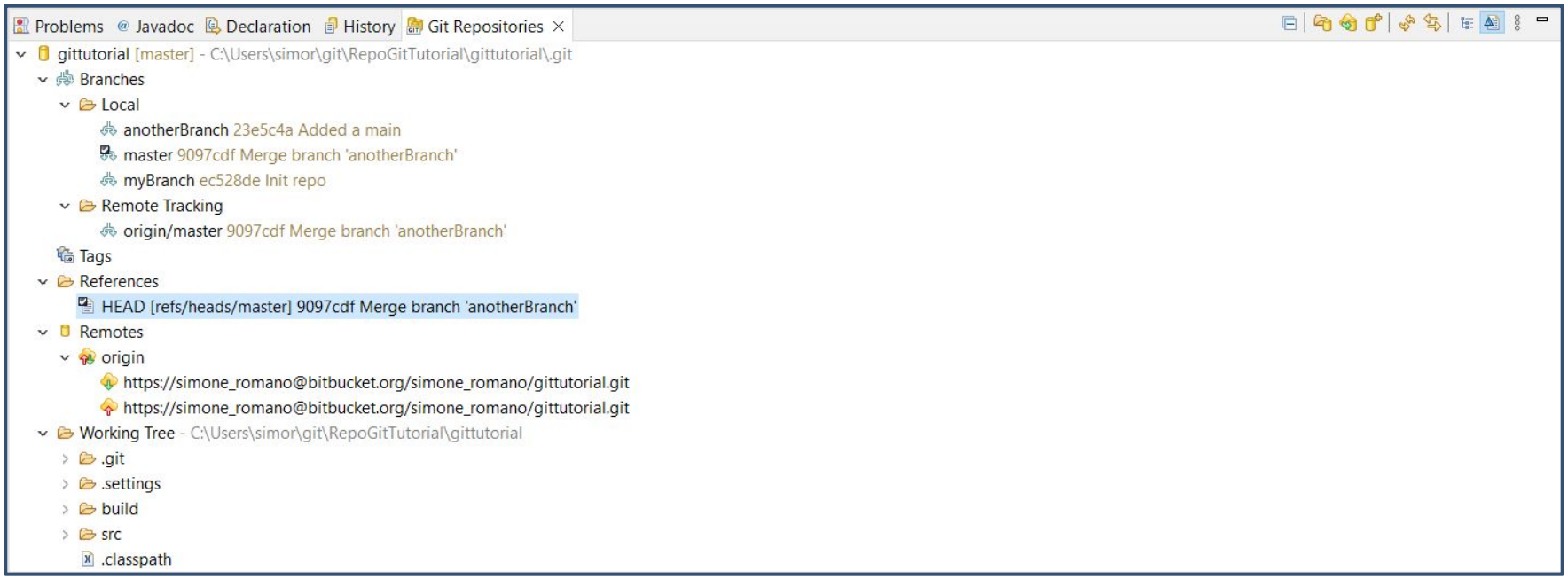
- Comparisons between your workspace and the local repository or between the current branch and others and are done via the ***Synchronize*** operation
  - If you right click **Team □ Synchronize Workspace**, your local workspace will be compared with the current branch showing uncommitted changes
  - 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
- This is an example of a *Synchronize* operation in the Synchronizing perspective:



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



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*
- Any *dirty* file can be added to the index with an *Add* to Index 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

Useful pages

- To create your own remote repositories and perform operations on them, you might want to register at https://bitbucket.org
- A tutorial with more information on certain options and actions: https://wiki.eclipse.org/EGit/User_Guide