

ISW: Software Engineering

WS 2015/16

Einführung in Versionsverwaltung mit Git

Marcus Seiler

Institute of Computer Science
Chair of Software Engineering
Im Neuenheimer Feld 326
69120 Heidelberg, Germany

<http://se.ifi.uni-heidelberg.de>

marcus.seiler@informatik.uni-heidelberg.de



RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

1. Probleme während der Softwareentwicklung
2. Lösung: Versionsverwaltung
3. Erste Schritte mit Git
4. Fortgeschrittene Technik in Git
5. Installation und Konfiguration von Git
6. Projekt importieren und freigeben
7. Änderungen einpflegen

Probleme während der Entwicklung (1/3)

■ Softwareentwicklung ist nicht linear

- Es wurden Fehler in der Implementierung gemacht und diese möchte man korrigieren
- Getroffene Entscheidungen sind nicht immer die besten

■ Software wird im Team entwickelt

- Viele EntwicklerInnen arbeiten gemeinsam in einem Projekt
- Die EntwicklerInnen wollen wissen, wer etwas im Projekt getan hat (was, wann, warum)
- Der/die einzelneN EntwicklerIn möchte wissen, was er/sie getan hat (wann, warum)

■ Wartung von unterschiedlichen Versionen

- KundenInnen bekommen stabile Version (Release)
- Die interne Entwicklung geht aber weiter
- Bug-Fixes müssen in die stabile Version eingepflegt werden

Probleme während der Entwicklung (2/3)

- **Backups sind notwendig, aber:**
 - man verliert oft den Überblick
 - man weiß nicht, welches Backup von welcher Version erstellt wurde
 - oder in welcher Version der Bug X korrigiert wurde
 - komplette (oder iterative) Backups benötigen viel Zeit und Speicherplatz
- **80/20 Regel: 20% Entwicklung, 80% Debugging**
 - “Ich kann den Bug nicht finden ...” / “Gestern hat es noch funktioniert ...”
- **Bei Wechsel des Arbeitsbereichs muss Projekt kopiert werden**
 - Inkonsistenzen sind vorprogrammiert
- **Kommunikation wird vergessen**
 - “In Klasse X habe ich Y implementiert wegen Z ...”
 - “Der Bug X in Klasse Y ist korrigiert ...”
 - “Stop! Die Version ist fehlerhaft! Ich gebe dir meine ...”

Probleme während der Entwicklung (3/3)

Viele EntwicklerInnen

Viele Klassen / Ressourcen

Kein Überblick



Instandhaltung des
Projekts ist gefährdet

Viele Versionen

Viele Arbeitsbereiche

Wo ist der Unterschied?

Aktuelle Version

```
public void myTestMethod()  
{  
    log("Test method entered.");  
  
    boolean keepOnRunning = true;  
    int i=0;  
  
    while( keepOnRunning )  
    {  
        int r1 = doSomething1();  
        int r2 = doSomething1();  
  
        if( r1+r2 > MAX_THRESHOLD )  
        {  
            log("Threshold exceeded.");  
            log("Iterations: " + i);  
            keepOnRunning = false;  
        }  
        else  
        {  
            printStatus(r1, r2);  
            i++;  
        }  
    }  
}
```

Vorherige Version

```
public void myTestMethod()  
{  
    log("Test method entered.");  
  
    boolean keepOnRunning = true;  
    int i=0;  
  
    while( keepOnRunning )  
    {  
        int r1 = doSomething1();  
        int r2 = doSomething2();  
  
        if( r1+r2 > MAX_THRESHOLD )  
        {  
            log("Threshold exceeded.");  
            log("Iterations: " + i);  
            keepOnRunning = false;  
        }  
        else  
        {  
            printStatus(r1, r2);  
            i++;  
        }  
    }  
}
```

Aktuelle Version

```
public void myTestMethod()  
{  
    log("Test method entered.");  
  
    boolean keepOnRunning = true;  
    int i=0;  
  
    while( keepOnRunning )  
    {  
        int r1 = doSomething1();  
        int r2 = doSomething1();  
  
        if( r1+r2 > MAX_THRESHOLD )  
        {  
            log("Threshold exceeded.");  
            log("Iterations: " + i);  
            keepOnRunning = false;  
        }  
        else  
        {  
            printStatus(r1, r2);  
            i++;  
        }  
    }  
}
```

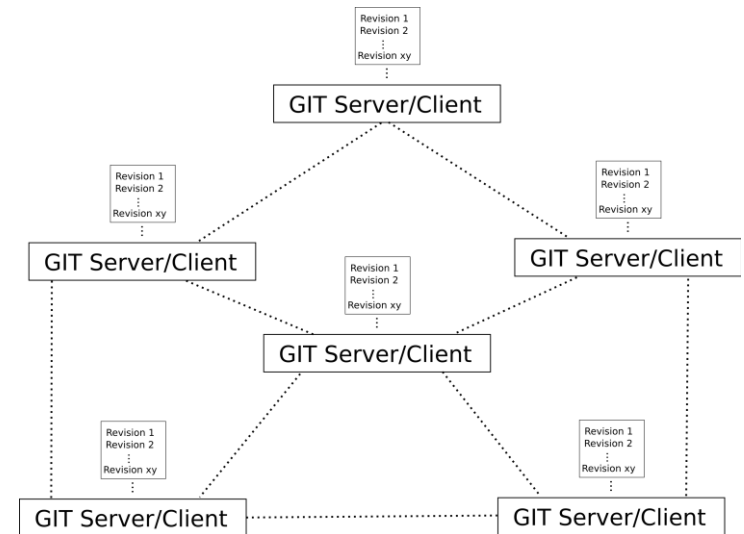
Vorherige Version

```
public void myTestMethod()  
{  
    log("Test method entered.");  
  
    boolean keepOnRunning = true;  
    int i=0;  
  
    while( keepOnRunning )  
    {  
        int r1 = doSomething1();  
        int r2 = doSomething2();  
  
        if( r1+r2 > MAX_THRESHOLD )  
        {  
            log("Threshold exceeded.");  
            log("Iterations: " + i);  
            keepOnRunning = false;  
        }  
        else  
        {  
            printStatus(r1, r2);  
            i++;  
        }  
    }  
}
```

Was kann Versionsverwaltung leisten?

- Sichern der eigenen Arbeit
- Änderungen rückgängig machen
- Mit anderen Personen kollaborativ zusammenarbeiten
 - Mehrere Personen können Dokumente teilen und editieren
- Dokumente sind online verfügbar
 - Wenn das Repository online erreichbar ist, hat man von überall Zugriff auf die Dateien

- Beginn: April 2005 als Ersatz für BitKeeper
- Wird von sehr vielen Projekten verwendet, z.B. Android, KDE, GNOME, LibreOffice, Linux-Kernel, Eclipse
- Dezentralisiertes Client-Server System



- Clients:
 - Command Line (<http://git-scm.com/downloads>)
 - GitEye (<http://www.collab.net/giteyeapp>)
 - EGit (Plugin für Eclipse) (<http://www.eclipse.org/egit/>)

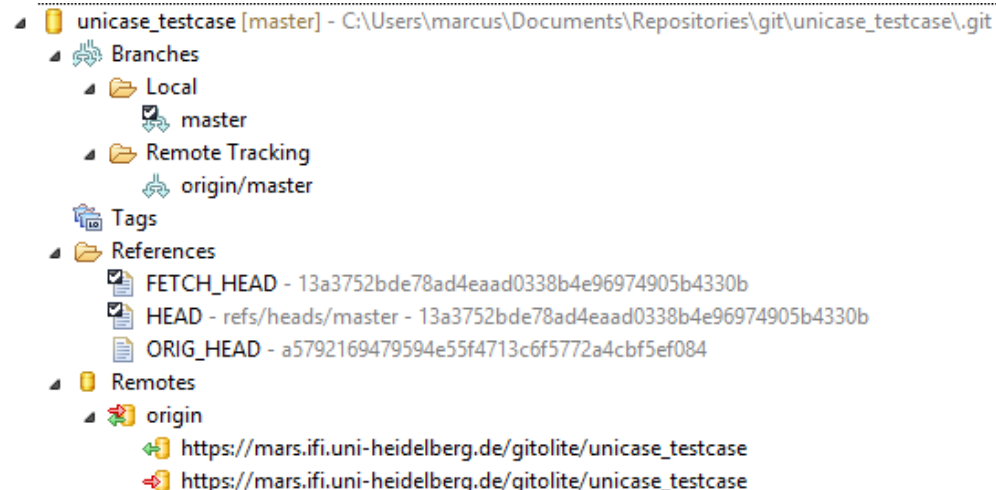
- **(Meist) trotzdem ein zentrales Repository (Remote Repository)**
 - “öffentliches” Repository

- **Alle Unterschiede zwischen zwei oder mehr Versionen sind erkennbar**
 - “Welche Änderungen gibt es in der heutigen Version im Vergleich zur gestrigen Version?”
 - “Was haben andere Entwickler in Klasse X geändert?”

- **Erlaubt Änderungen (eigene oder fremde) zu**
 - untersuchen, übernehmen, ablehnen, diskutieren

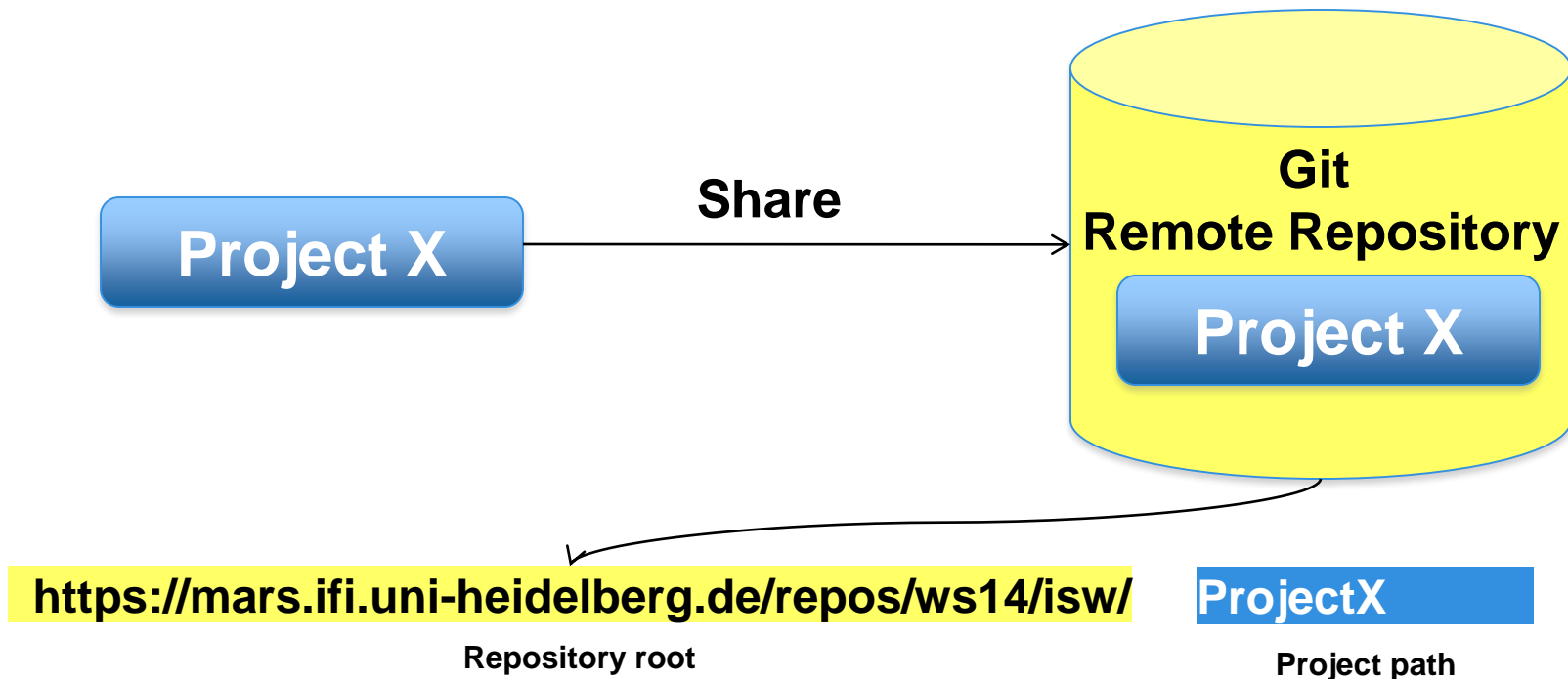
- **Vorheriger Zustand kann wiederhergestellt werden**
 - Komplett oder nur teilweise
- **Versionen können durch Tags gekennzeichnet werden**
 - Benutzung für Releases, Meilensteine oder einfache Backups
- **Meta-Informationen**
 - Wer hat Änderungen gemacht? (wann, was, warum, wo)

- **Share:** Ein Projekt unter Versionsmanagement stellen
- **Clone:** Kopiert ein bestehendes Git-Repository
- **Add:** Dateien zum lokalen Repository hinzufügen
- **Commit:** Änderungen im lokalen Repository veröffentlichen
- **Push:** Lokale Änderungen im Remote Repository veröffentlichen
- **Pull:** Änderungen aus dem Remote Repository übernehmen
- Repository Layout:



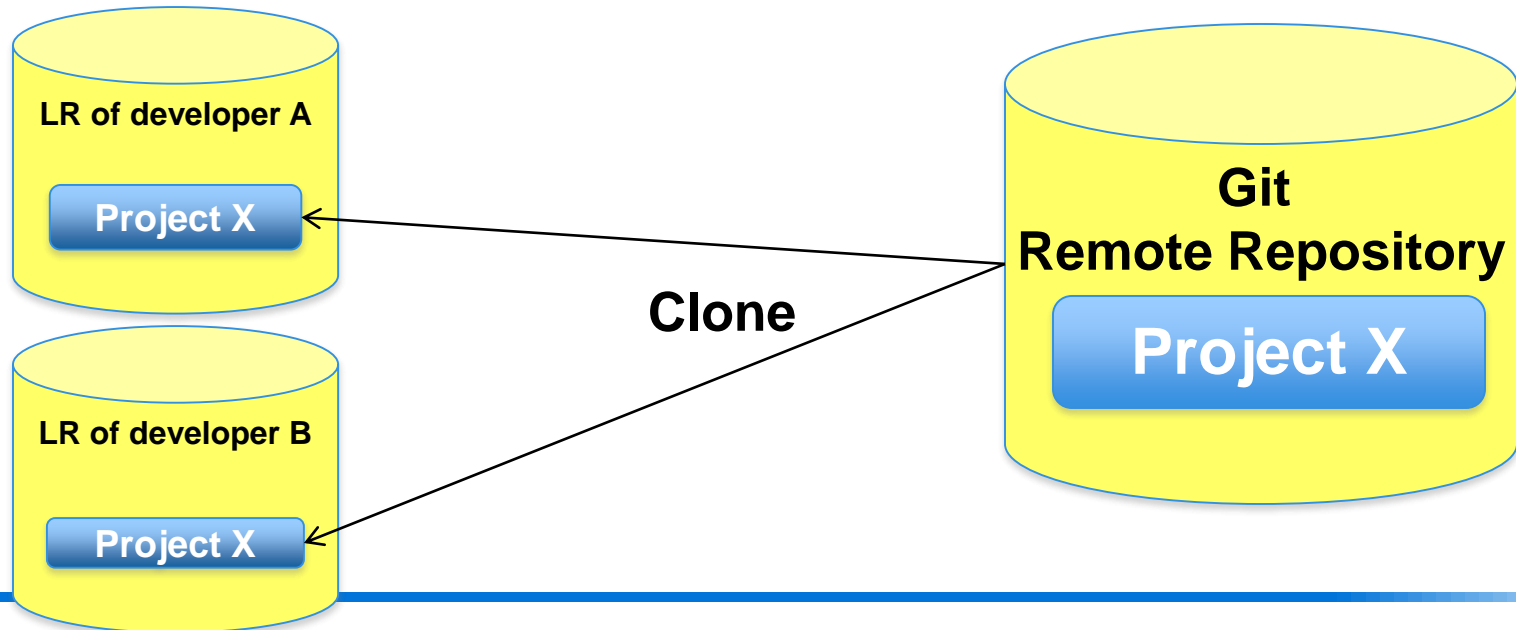
Share: Ein Projekt unter Versionsmanagement stellen

- Das Projekt ist vorher nur lokal vorhanden
- Durch “Share” wird es zum Repository hinzugefügt
- Nun ist es für alle (authorisierten) Entwickler verfügbar



Clone: Initialer Download des Repository

- Entwickler “clonen” das Projekt Repository
- Erhalten eine lokale Kopie (local Repository = LR)
- Nun können Sie mit ihrer Arbeit am Projekt beginnen
- “Clonen” passiert nur einmal zu Beginn → Danach nur noch “Pull”



Add + Commit: Änderungen im LR speichern

- Nach “Clone” ändert EntwicklerIn das Projekt lokal
- Änderungen müssen mit “Add” hinzugefügt werden
- Durch Commit werden die Änderungen im LR veröffentlicht
- Jeder Commit hat ein “Commit Comment”, welcher beschreibt, was für Änderungen gemacht wurden
- Ein “Commit Comment” muss folgende Informationen enthalten:
 - Was wurde genau geändert?
 - ggf. Bezug zu Issue im Issue Tracking System
- Vor einem Commit muss **immer**:
 - die eigene Änderungen überprüft werden
 - Sichergestellt werden, dass die Software kompilierbar ist (sonst wird das gesamte Team behindert)

Revisionsnummer (Commit-ID)




- Automatische generierte eindeutige Identifikationsnummer
- Wird pro Commit erzeugt
- Bezieht sich auf alle Dateien innerhalb des Commits

Repository: unicastestcase

Message	Author	Date	Id
<code>master</code> <code>origin/master</code> <code>FETCH_HEAD</code> <code>HEAD</code> Applied git ignore on binary output folders.	Marcus Seiler <marcus.seiler@informatik.uni-heidelberg.de>	2 weeks ago	13a3752b...
<code>ORIG_HEAD</code> Initial import unicastestcase plugin.	Marcus Seiler <marcus.seiler@informatik.uni-heidelberg.de>	3 months ago	a5792169...

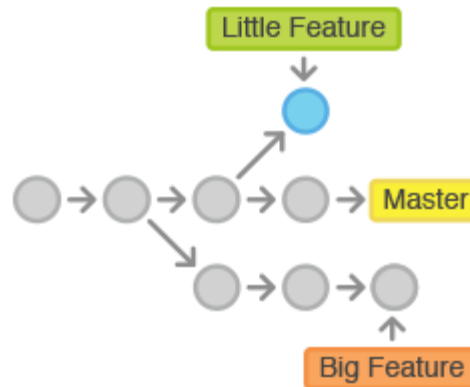
commit 13a3752bde78ad4eaad0338b4e96974905b4330b
Author: Marcus Seiler <marcus.seiler@informatik.uni-heidelberg.de> 2014-09-30 09:44:22
Committer: Marcus Seiler <marcus.seiler@informatik.uni-heidelberg.de> 2014-09-30 09:44:22
Parent: a5792169479594e55f4713c6f5772a4cbf5ef084 (Initial import unicastestcase plugin.)
Branches: origin/master, master

Applied git ignore on binary output folders.

 org.unicastestcase.edit/.gitignore
 org.unicastestcase.ui/.gitignore
 org.unicastestcase/.gitignore

- Push: veröffentlicht mindestens einen Commit in dem RR
- Pull: Der aktuelle Stand im RR wird in die lokale Kopie (LR) übernommen
- Nach längerer Pause IMMER Projekt aktualisieren BEVOR mit der Arbeit fortgefahren wird!
- Änderungen werden angezeigt und können mit “Add” übernommen werden

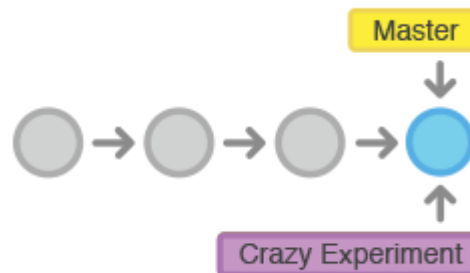
- Neue Funktion hinzufügen oder einen Fehler beheben – egal, wie groß oder klein –, neuen Branch anlegen
- Abkapseln von Änderungen
 - Sicherstellen, dass der instabile Code aus diesem Branch nicht zur offiziellen Codebasis hinzugefügt wird



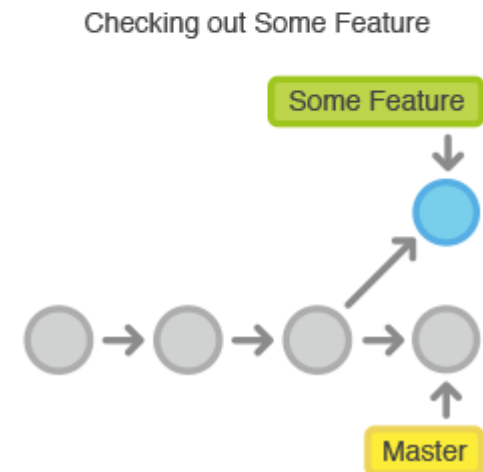
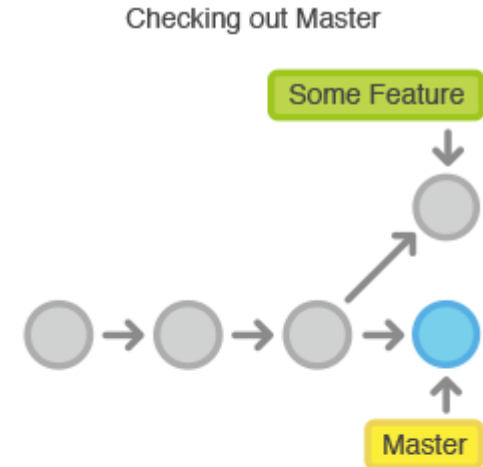
- Branch erstellen



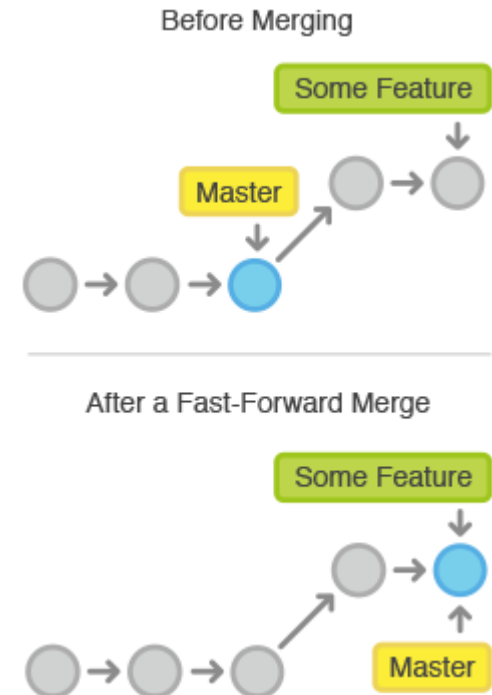
- “git branch crazy-experiment”



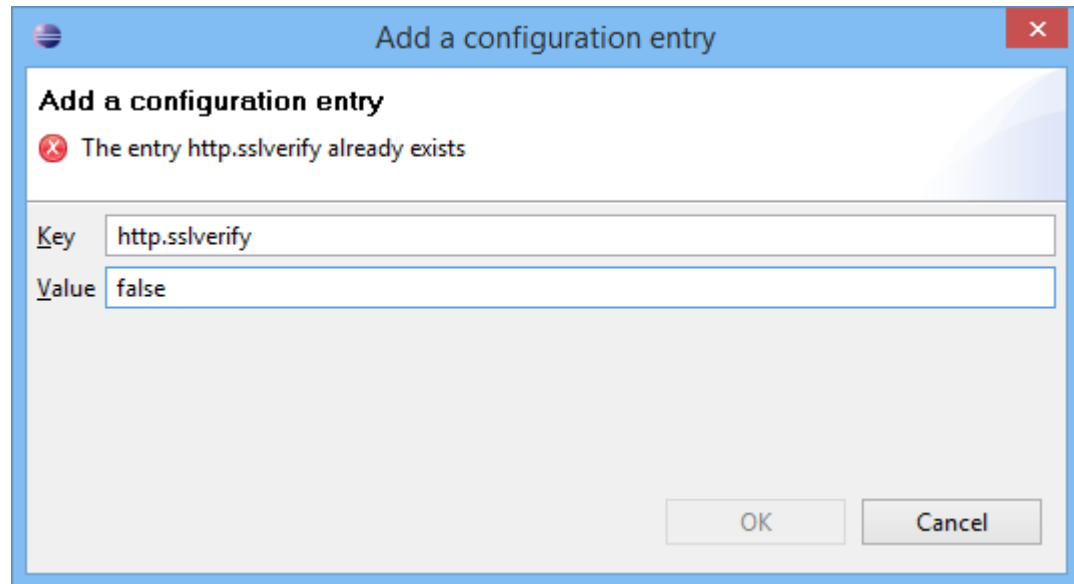
- Nach dem Erstellen eines Branches muss dieser ausgecheckt werden



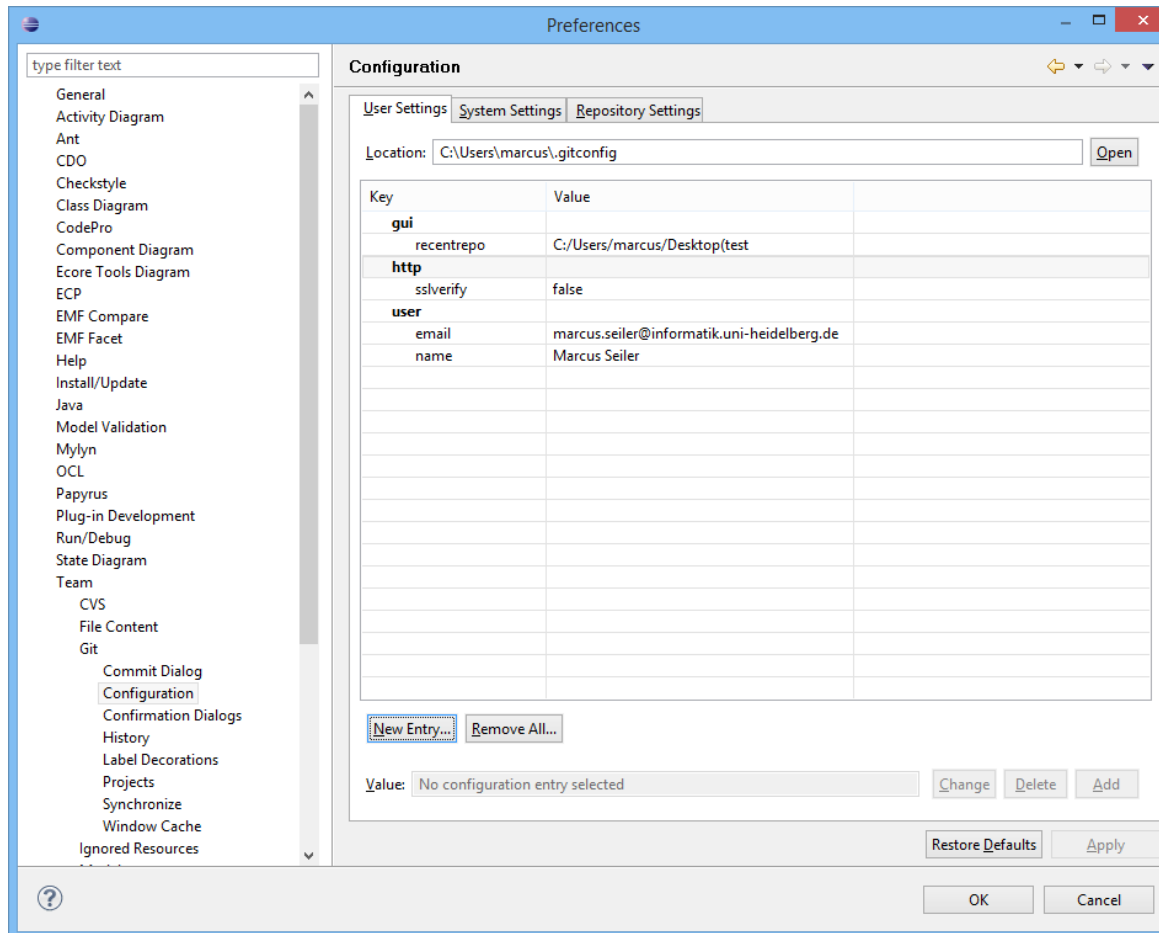
- Nach Abschluss der Entwicklung muss der Branch mit dem Hauptzweig zusammengeführt werden
- Mit dem Befehl „git merge“ lassen sich unabhängige Entwicklungszweige, die mit „git branch“ erstellt wurden, wieder in einen einzelnen Branch zusammenführen.



- EGit ist ein Plugin für die Eclipse IDE (bereits enthalten)
 - Stellt die Funktionalität von Git in der Eclipse IDE bereit
- **WICHTIG:** Nach dem Starten von Eclipse
 - Window -> Preferences -> Team -> Git -> Configuration
 - Neuer Eintrag

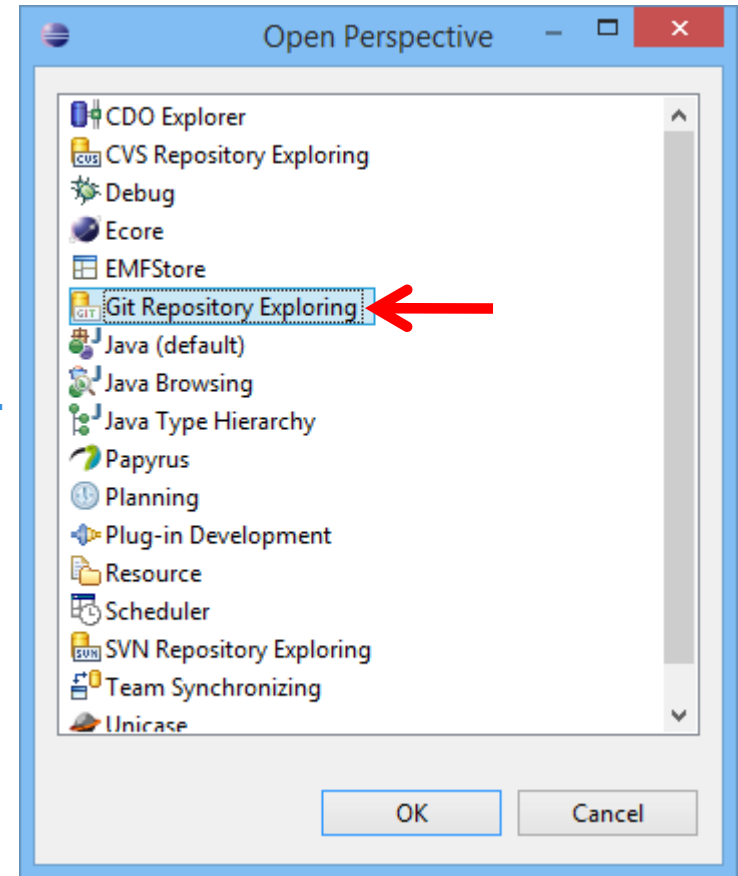


- Unter User noch: Name und Email ergänzen

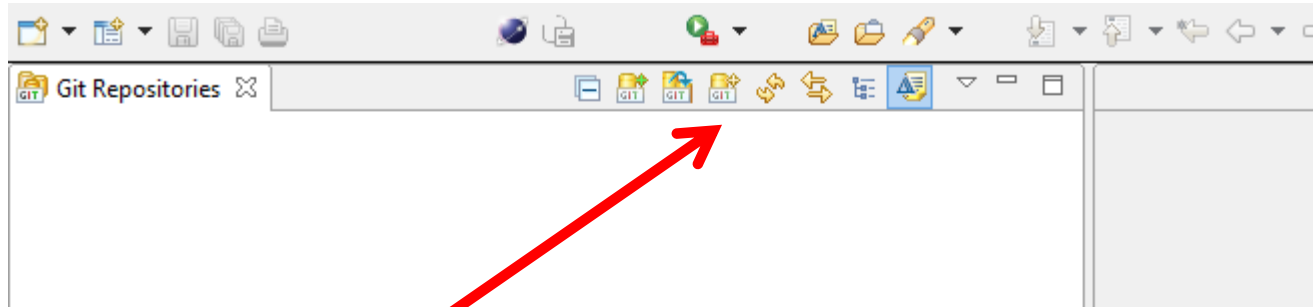


Öffnen der Perspektive “Git Repositories”

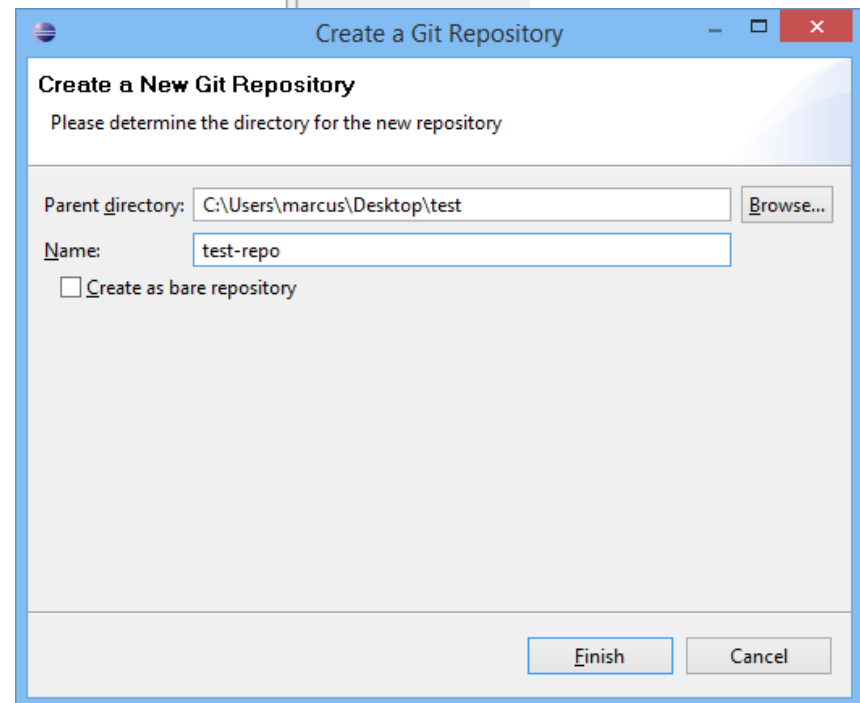
Eclipse → Window → Open Perspective → Other ...

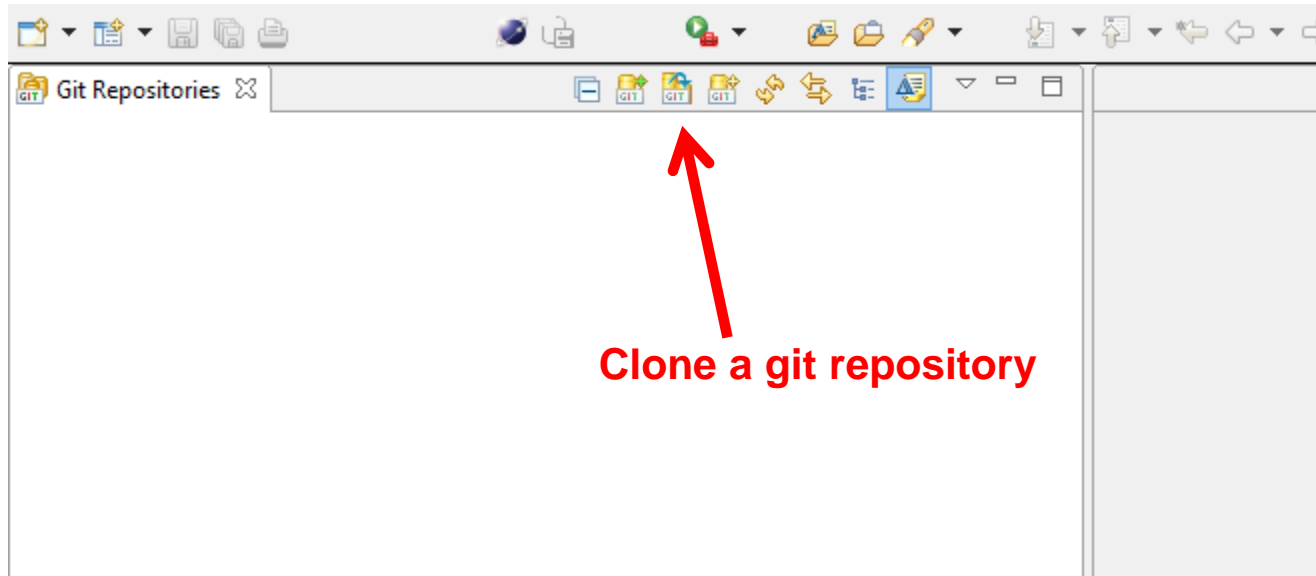


Lokales Repository erstellen



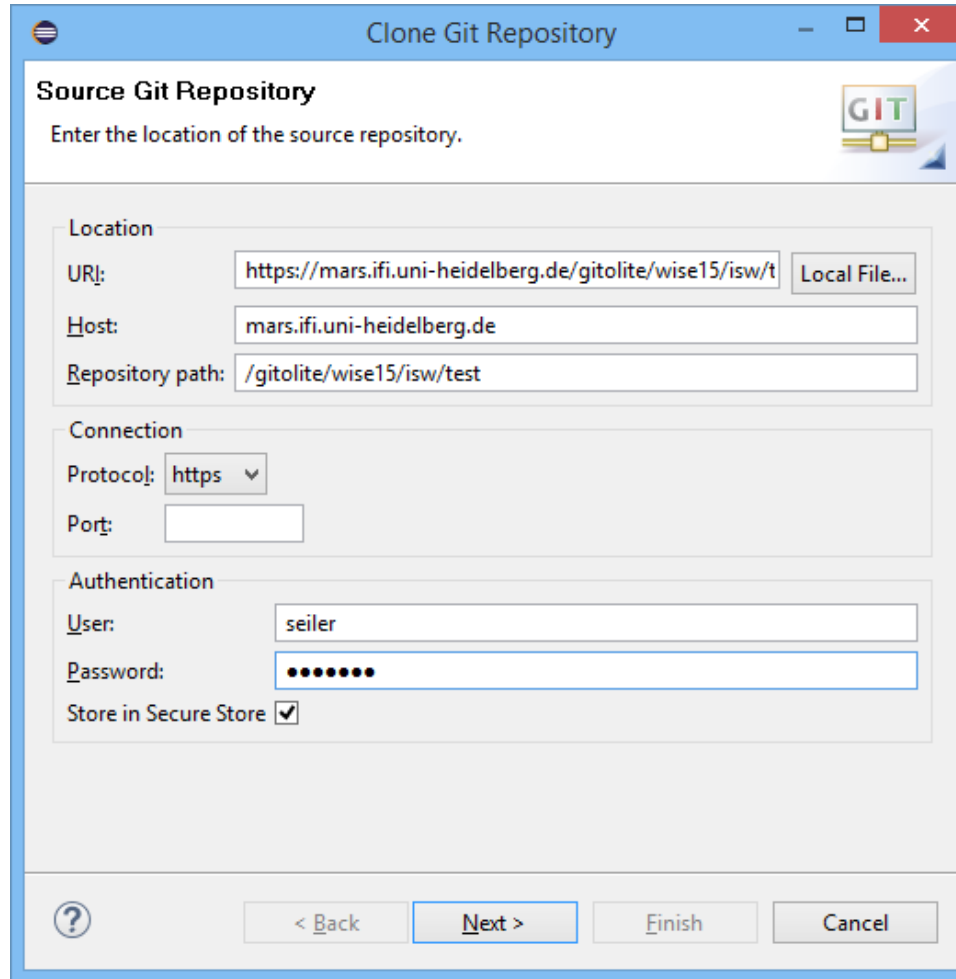
Create a new Git Repository





Repository clonen

Repository URL und Zugangsdaten eingeben

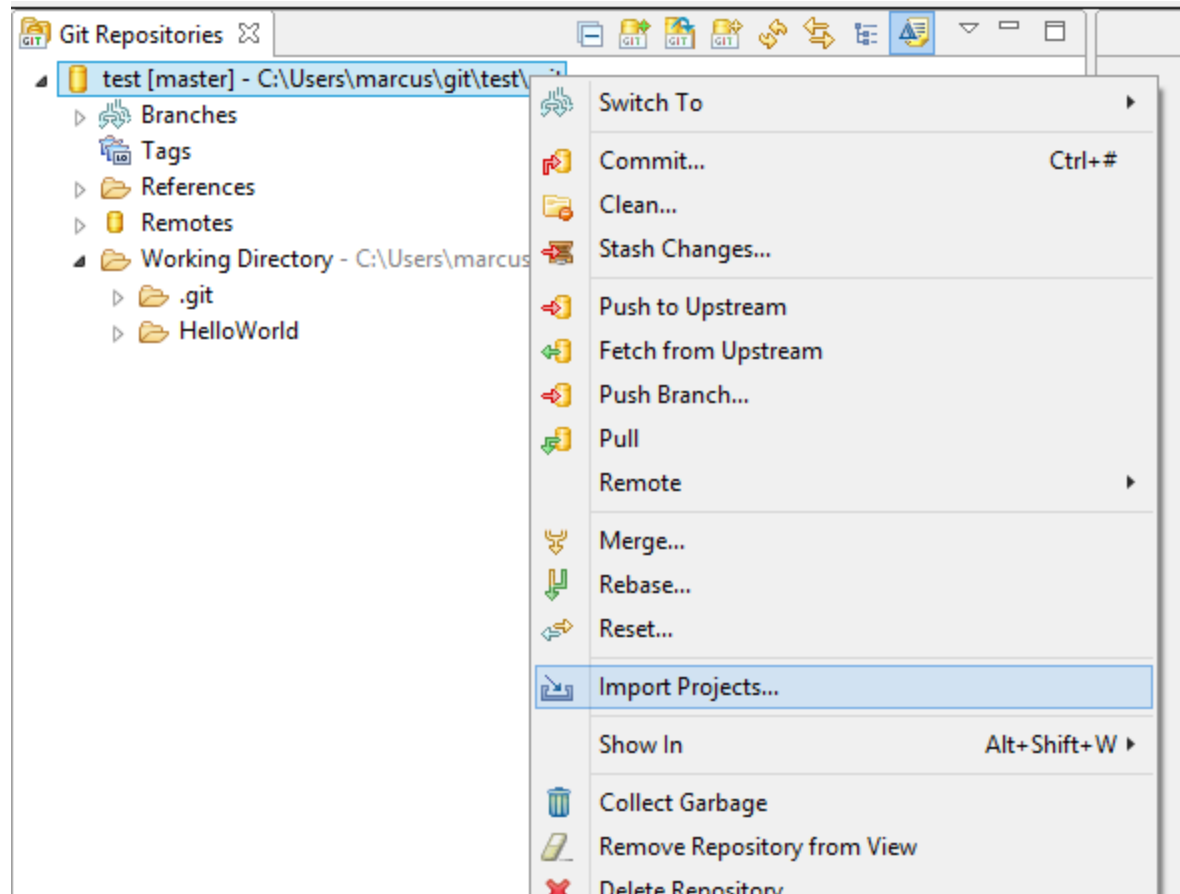


The image shows a 'Clone Git Repository' dialog box with the following fields and options:

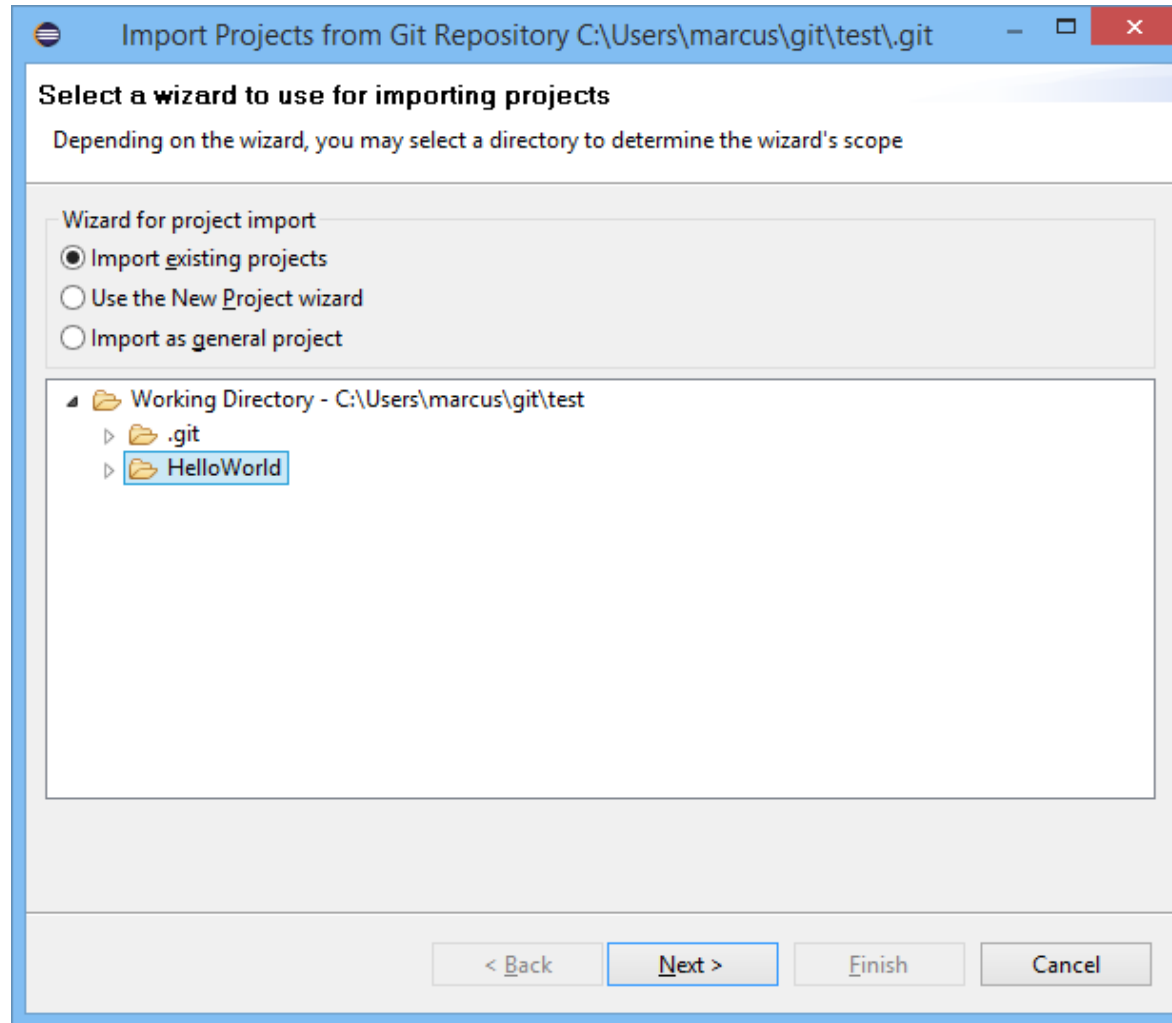
- Source Git Repository**: Enter the location of the source repository.
- Location**:
 - URI:**
 - Host:**
 - Repository path:**
- Connection**:
 - Protocol:** (dropdown)
 - Port:**
- Authentication**:
 - User:**
 - Password:**
 - Store in Secure Store** ☒

At the bottom, there are navigation buttons: . A help icon (?) is also present.

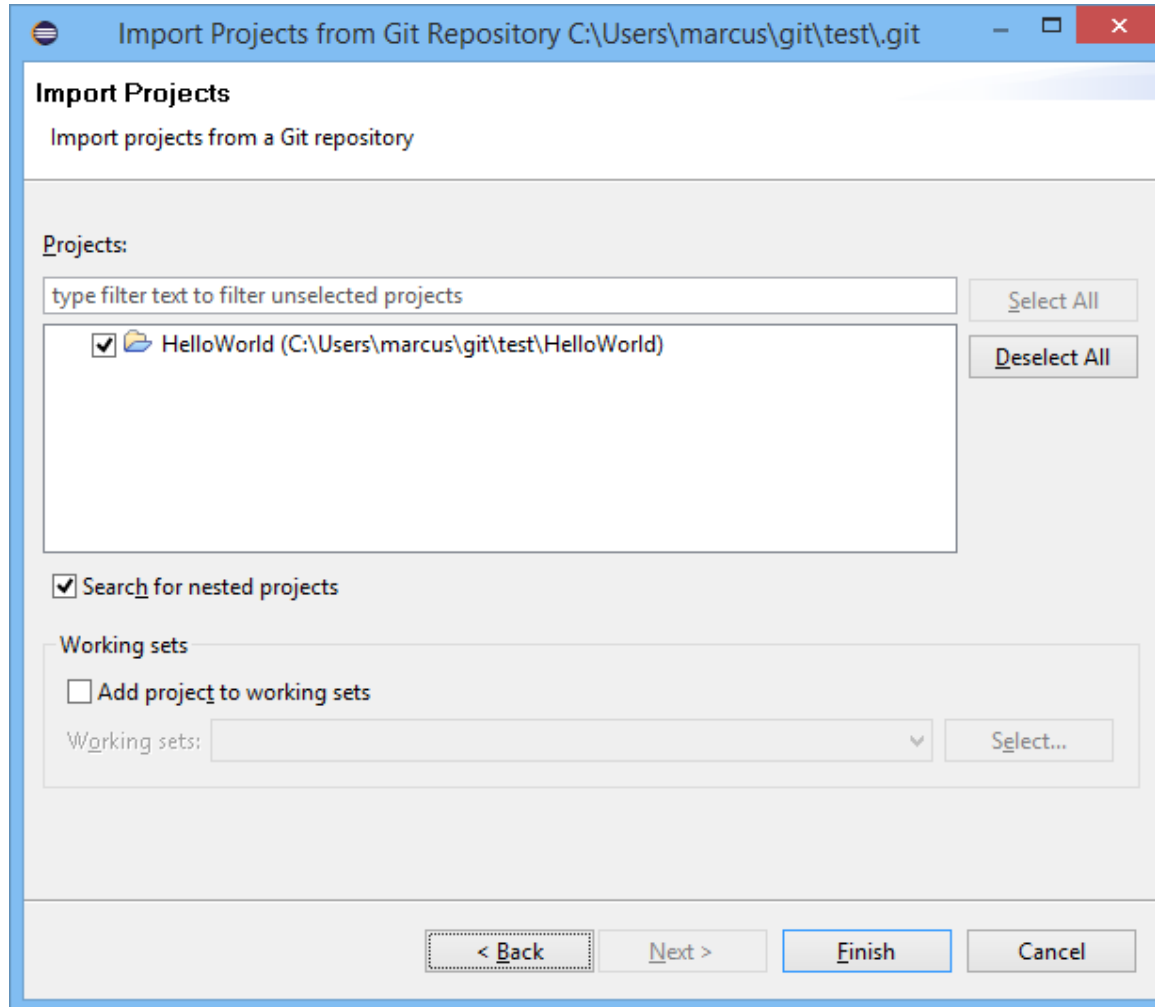
Projekt importieren (1/3)



Projekt importieren (2/3)



Projekt importieren (3/3)



Import Projects from Git Repository C:\Users\marcus\git\test\git

Import Projects

Import projects from a Git repository

Projects:

type filter text to filter unselected projects

☒ HelloWorld (C:\Users\marcus\git\test\HelloWorld)

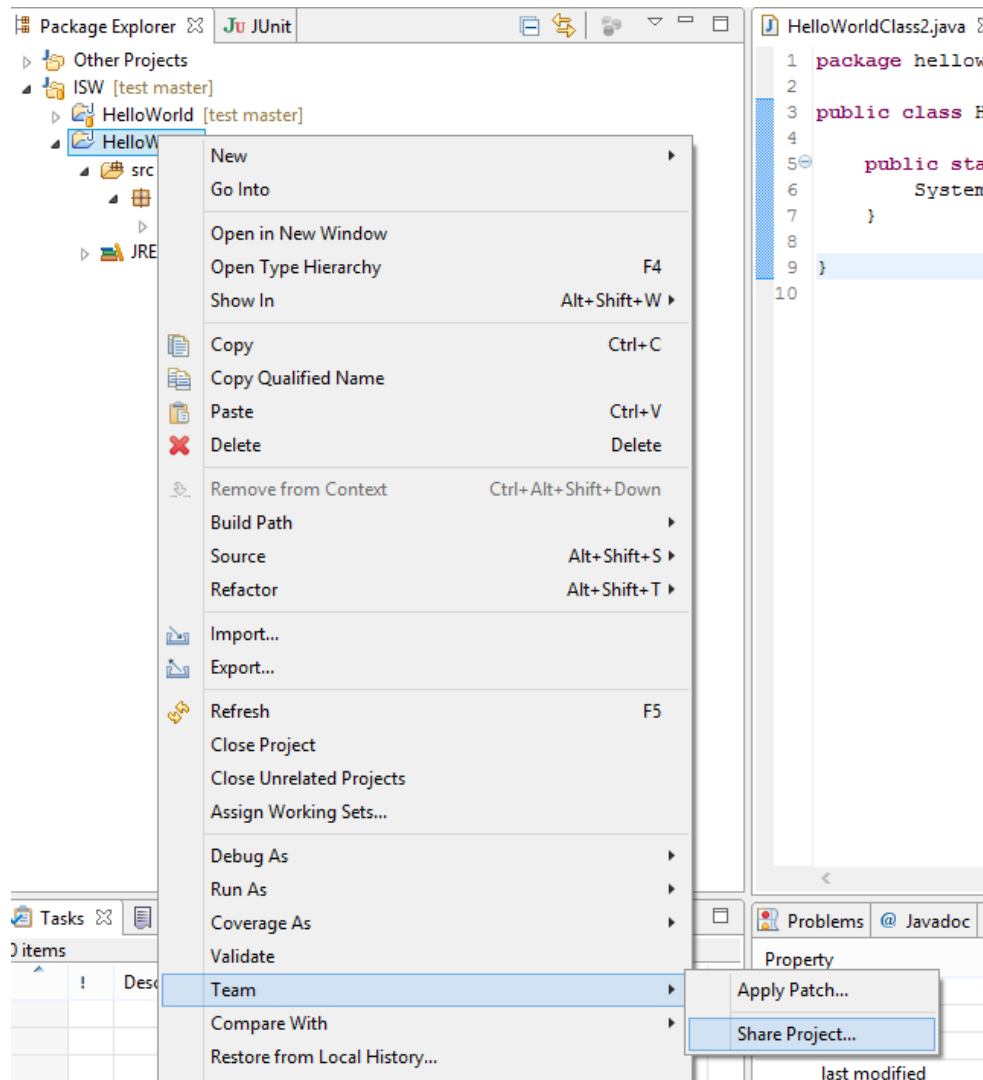
☒ Search for nested projects

Working sets

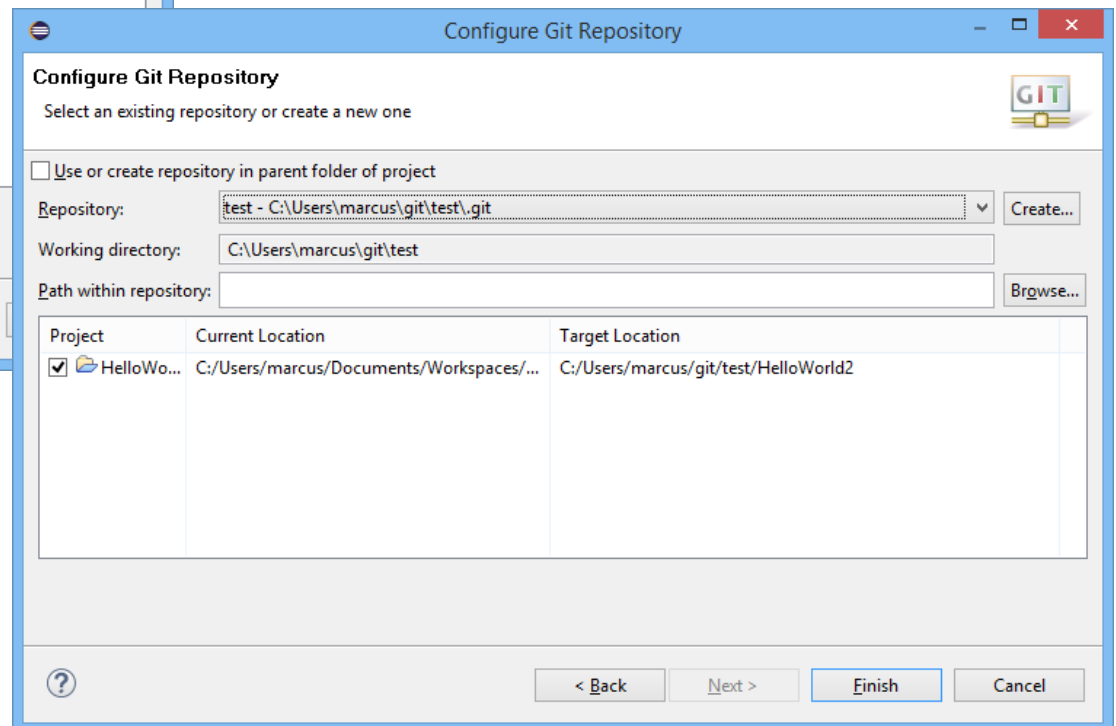
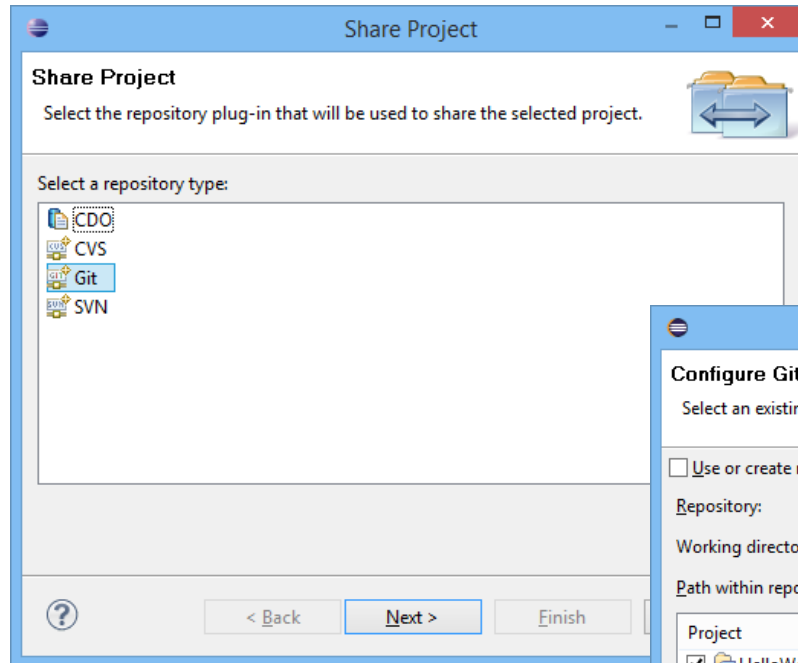
☐ Add project to working sets

Working sets:

Projekt teilen (1/5)



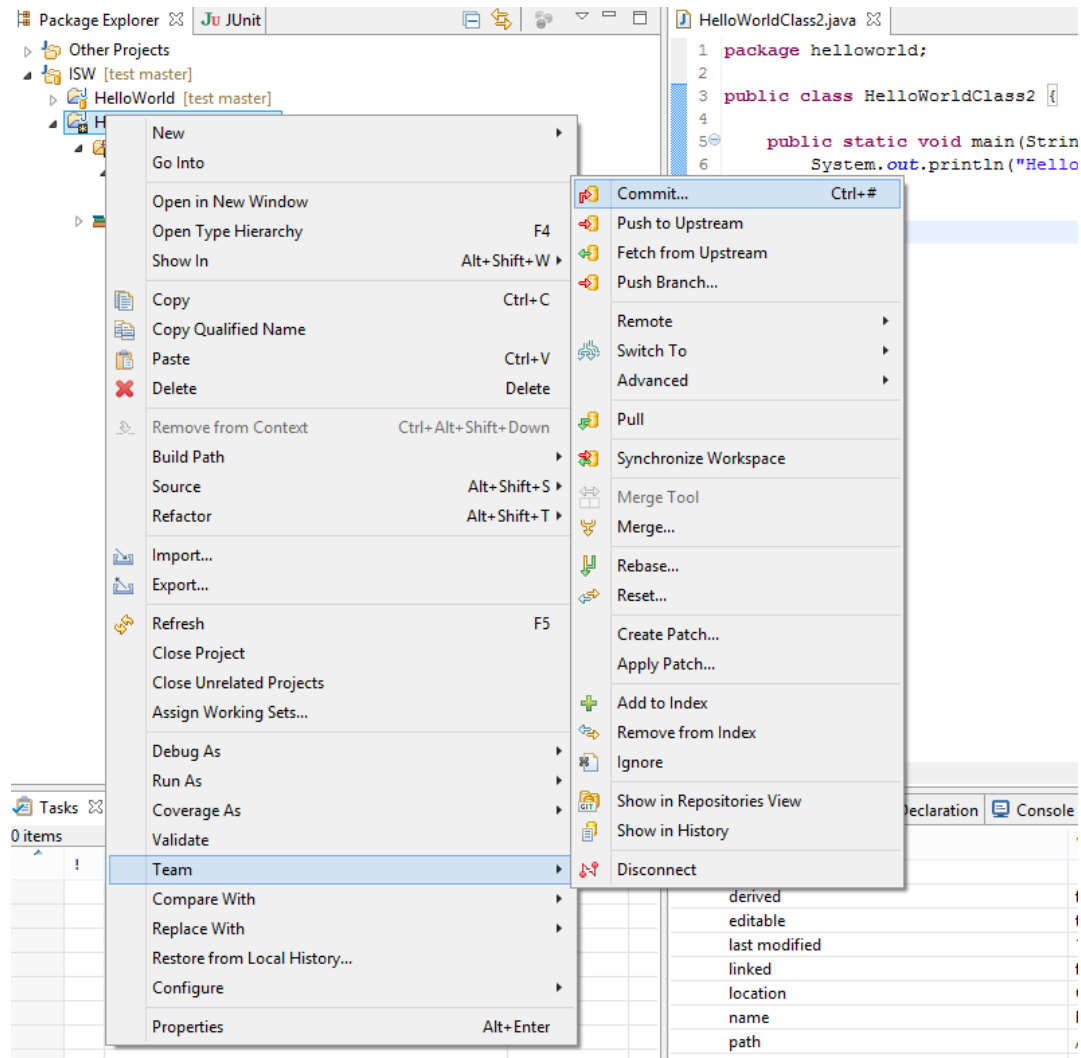
Projekt teilen (2/5)

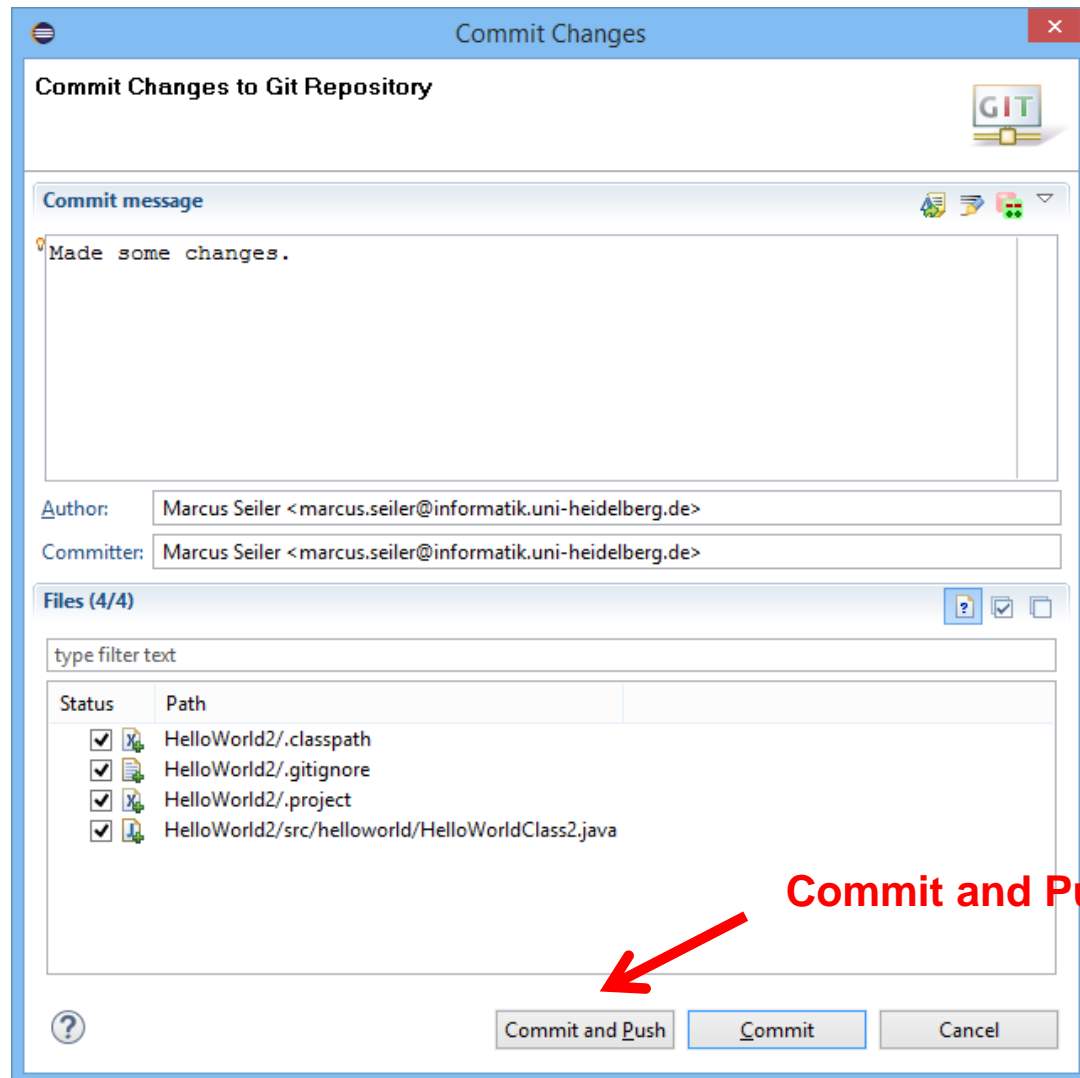


The screenshot shows an IDE interface with a project named 'ISW [test master]' and a file named 'HelloWorld2'. A context menu is open over the file, displaying various actions. The 'Team' option is highlighted, indicating the next step in the workflow. The background code shows a Java class 'HelloWorldClass2' with a 'main' method that prints 'Hello World'.

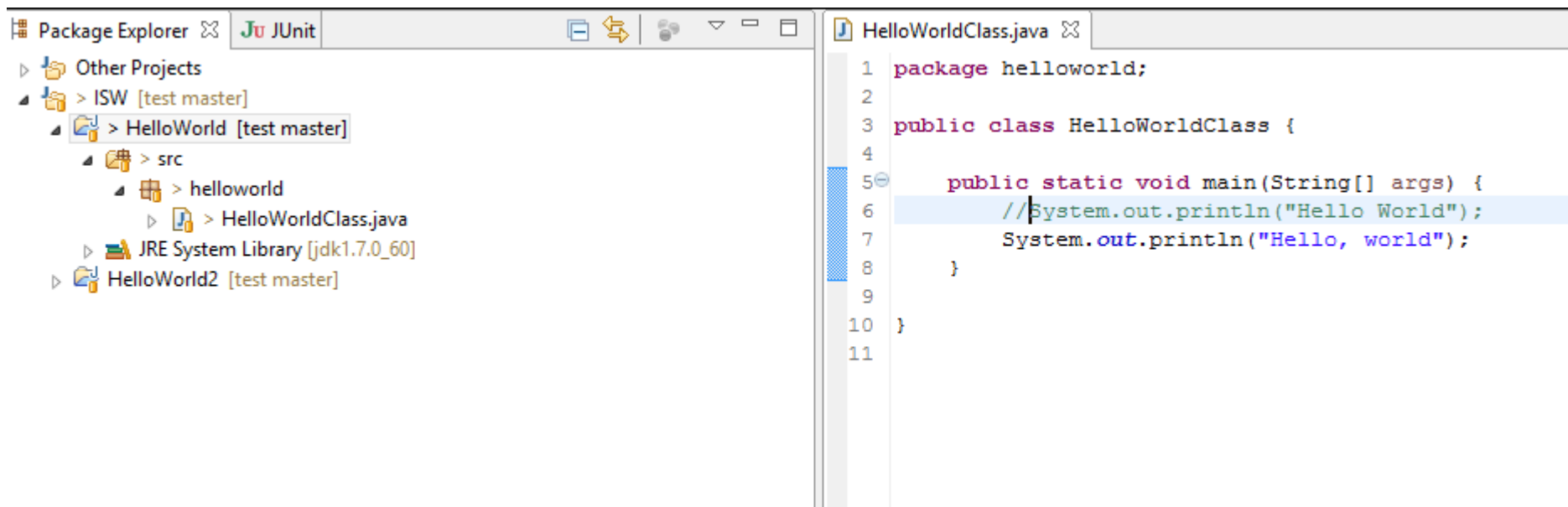
```

2
3 public class HelloWorldClass2 {
4
5     public static void main(String[] args) {
6         System.out.println("Hello World");
    }
  
```

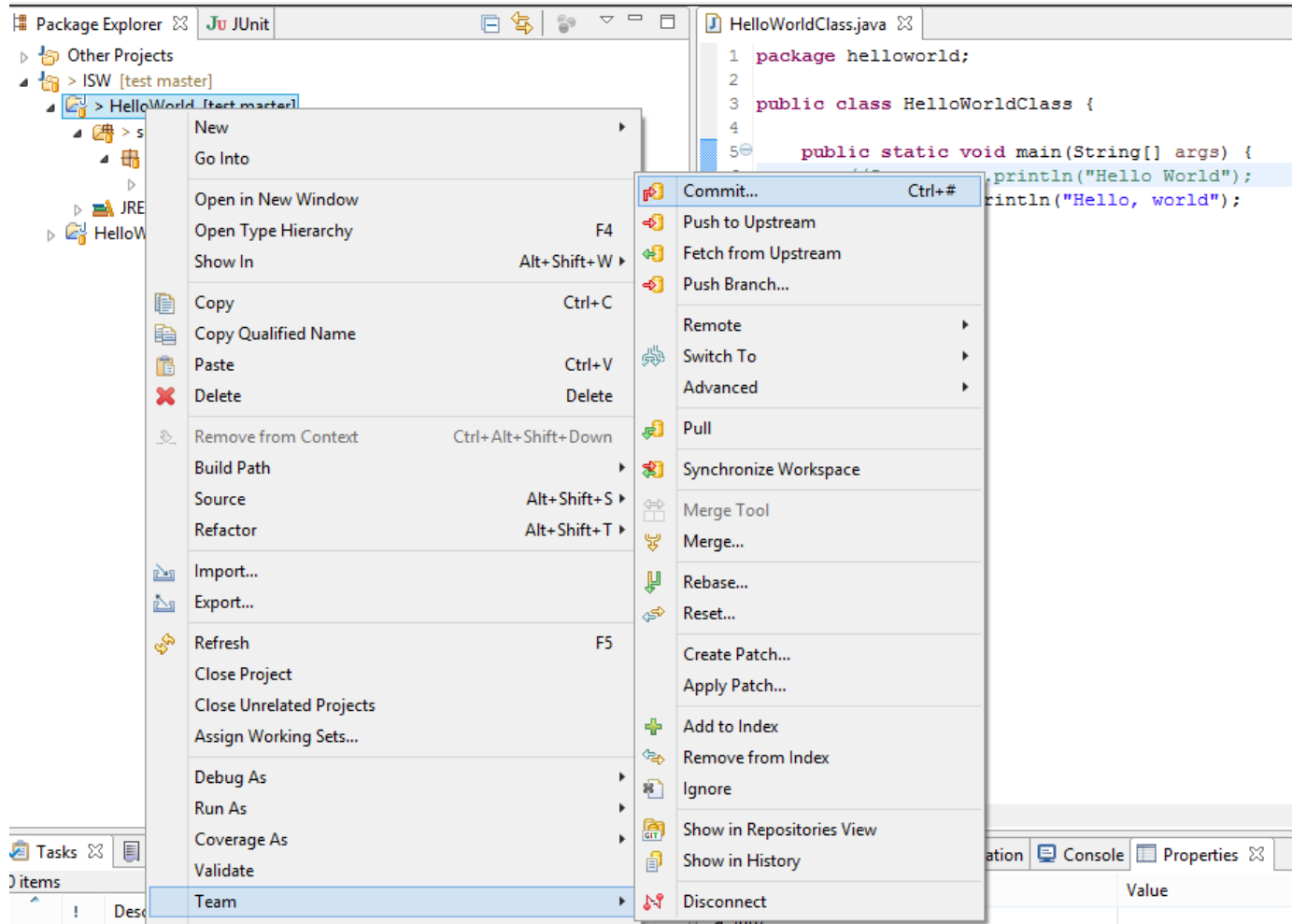




Änderungen einpflegen (1/3)



Änderungen einpflegen (2/3)



Änderungen einpflegen (3/3)

Commit Changes

Commit Changes to Git Repository

Commit message


Changed the output string.


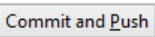
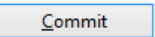
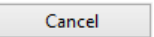
Author: Marcus Seiler <marcus.seiler@informatik.uni-heidelberg.de>

Committer: Marcus Seiler <marcus.seiler@informatik.uni-heidelberg.de>

Files (1/1)

type filter text

Status	Path
<input checked="" type="checkbox"/> 	HelloWorld/src/helloworld/HelloWorldClass.java

Commit and Push

Push Results: test - origin



Pushed to test - origin

master: master [d99ddc0..07f2bad] (1)

d99ddc0: Changed the output string. (Marcus Seiler on 17.10.2014 10:27)

Message Details

Repository <https://mars.ifi.uni-heidelberg.de/gitolite/wise15/isw/test>

Name

`git add`
`git clone`
`git commit`
`git push`
`git add -u`
`git diff`
`git help`
`git log`
`git merge`
`git revert`
`Git reset`
`git status`
`git pull`

Purpose

Add files and/or directories to version control.
Get a fresh working copy of a remote repository.
Commit changes in the local repository.
Update the remote repository.
Delete files and/or directories from version control.
Shows changes for directories/files in a unified diff format.
Get help (in general, or for a particular command).
Show history of recent changes.
Merge two different versions of a file into one.
Undo pushed changes (i.e., resynchronize with remote repository).
Undo committed local changes.
Show the status of files and directories in the working copy.
Get changes from the remote repository into local repository.

- Git Documentation <http://git-scm.com/docs/gittutorial>
- Atlassian Git-Anleitungen
<https://www.atlassian.com/de/git/tutorial/remote-repositories>
- EclipseSource EGit Tutorial
<http://eclipsesource.com/blogs/tutorials/egit-tutorial/>

Marcus Seiler

Institute of Computer Science
Chair of Software Engineering
Im Neuenheimer Feld 326
69120 Heidelberg, Germany

<http://se.ifi.uni-heidelberg.de>

marcus.seiler@informatik.uni-heidelberg.de



RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG
