

CS 580 Client-Server Programming  
Spring Semester, 2010  
Doc 2 Source Control & Testing  
Jan 26, 2010

Copyright ©, All rights reserved. 2010 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (<http://www.opencontent.org/opl.shtml>) license defines the copyright on this document.

## Testing & Subversion References

JUnit Web site: <http://www.junit.org/>

JUnit JavaDoc, <http://kentbeck.github.com/junit/javadoc/latest/>

Brian Marick's Testing Web Site: <http://www.exampler.com/testing-com/>

Testing for Programmers, Brian Marick, Available at: <http://www.exampler.com/testing-com/writings.html>

Main Mercurial Website, <http://mercurial.selenic.com/>

Mercurial: The Definitive Guide, Bryan O'Sullivan, <http://hgbook.red-bean.com/>

# Source Control

Test your Processes

# Why Use Source Control?

# Common Free Source Control Systems

## CVS

Concurrent Versions System

Command line interface in Unix

Various interfaces in Window

## Subversion

Claims to be a better CVS

Many commands are same as CVS

## Git

Created by Linus Torvald  
Distributed Version control

## Mercurial

Python based  
Distributed version control

# Mercurial

<http://mercurial.selenic.com/>

Runs on:

Mac OS X

Unix

Linux

Windows

Command line interface

GUI interface for windows

Eclipse plugins

Can use locally with no server

# Simple Workflow

Commit changes



code



Commit changes

(creates changeset)



code



Commit changes

(creates changeset)



code



Start repository



# Changesets

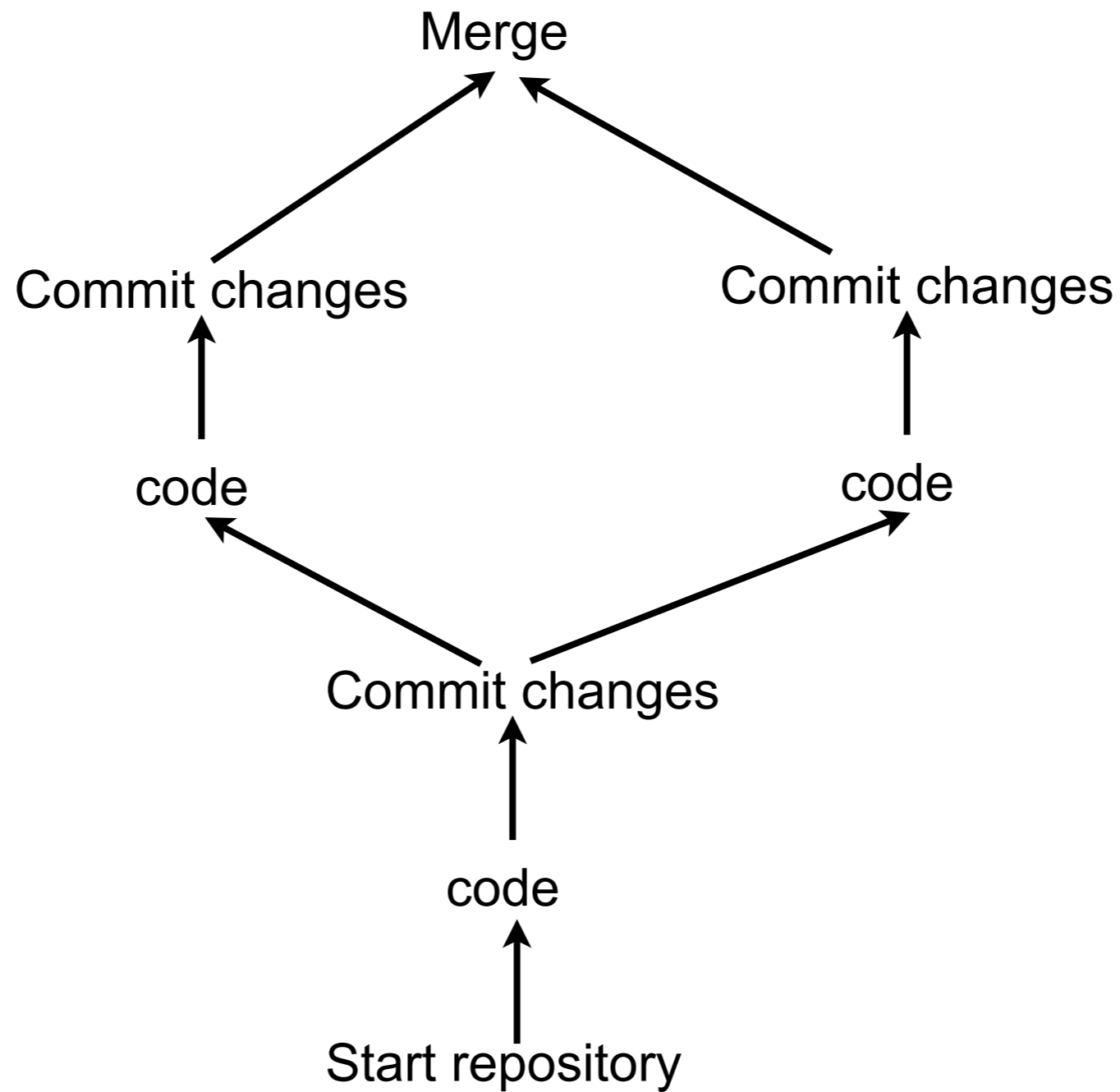
Revision  
Number

Changeset  
identifier

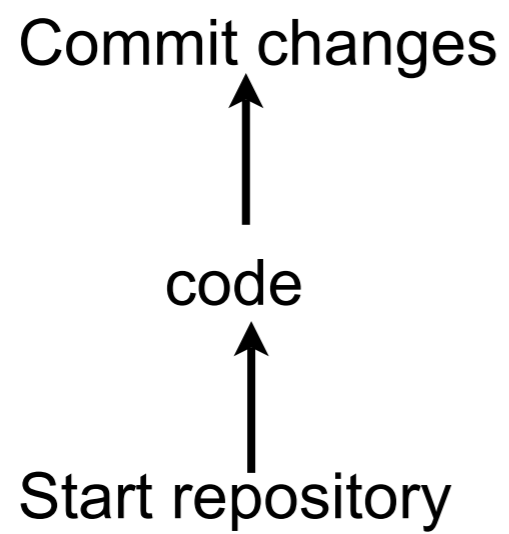
Option text  
Identifier

changeset: 4 : 2278160e78d4  
tag: tip  
user: Bryan O'Sullivan <bos@serpentine.com>  
date: Sat Aug 16 22:16:53 2008 +0200  
summary: Trim comments.

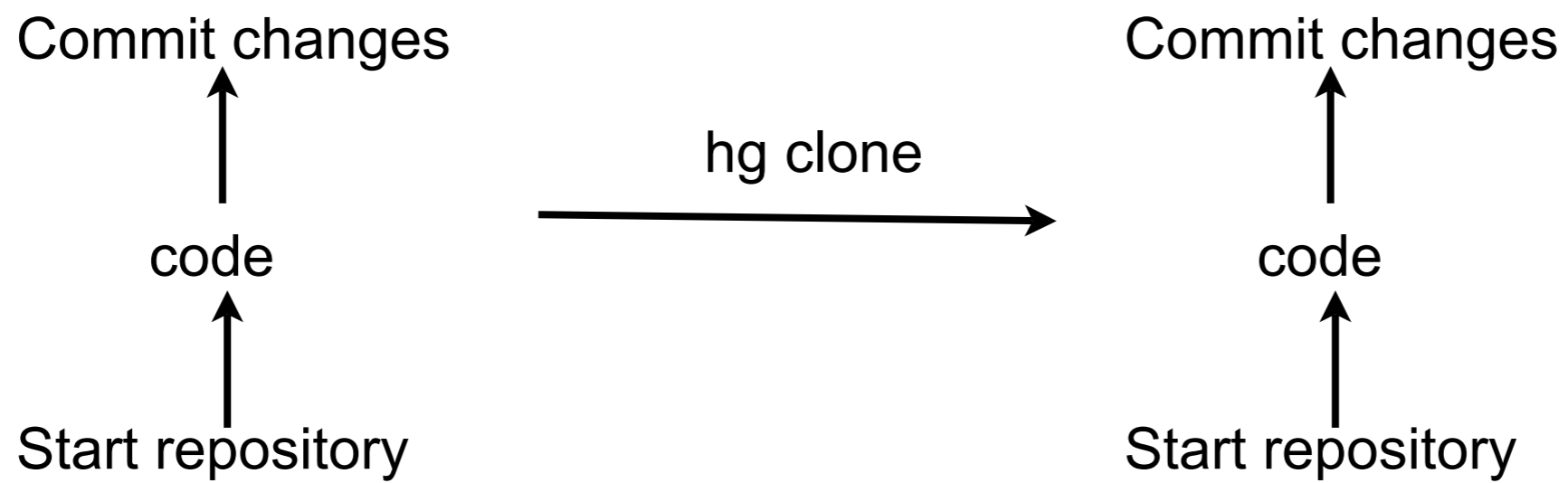
# Branch in one Repository



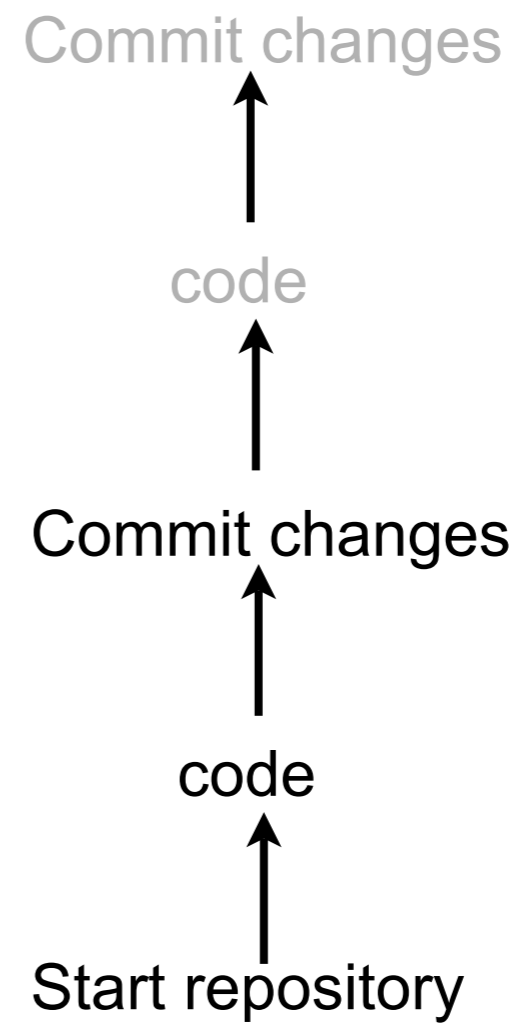
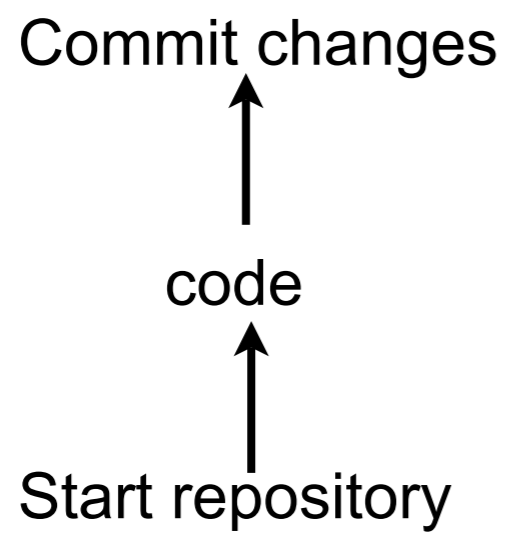
# Standard Mercurial Workflow



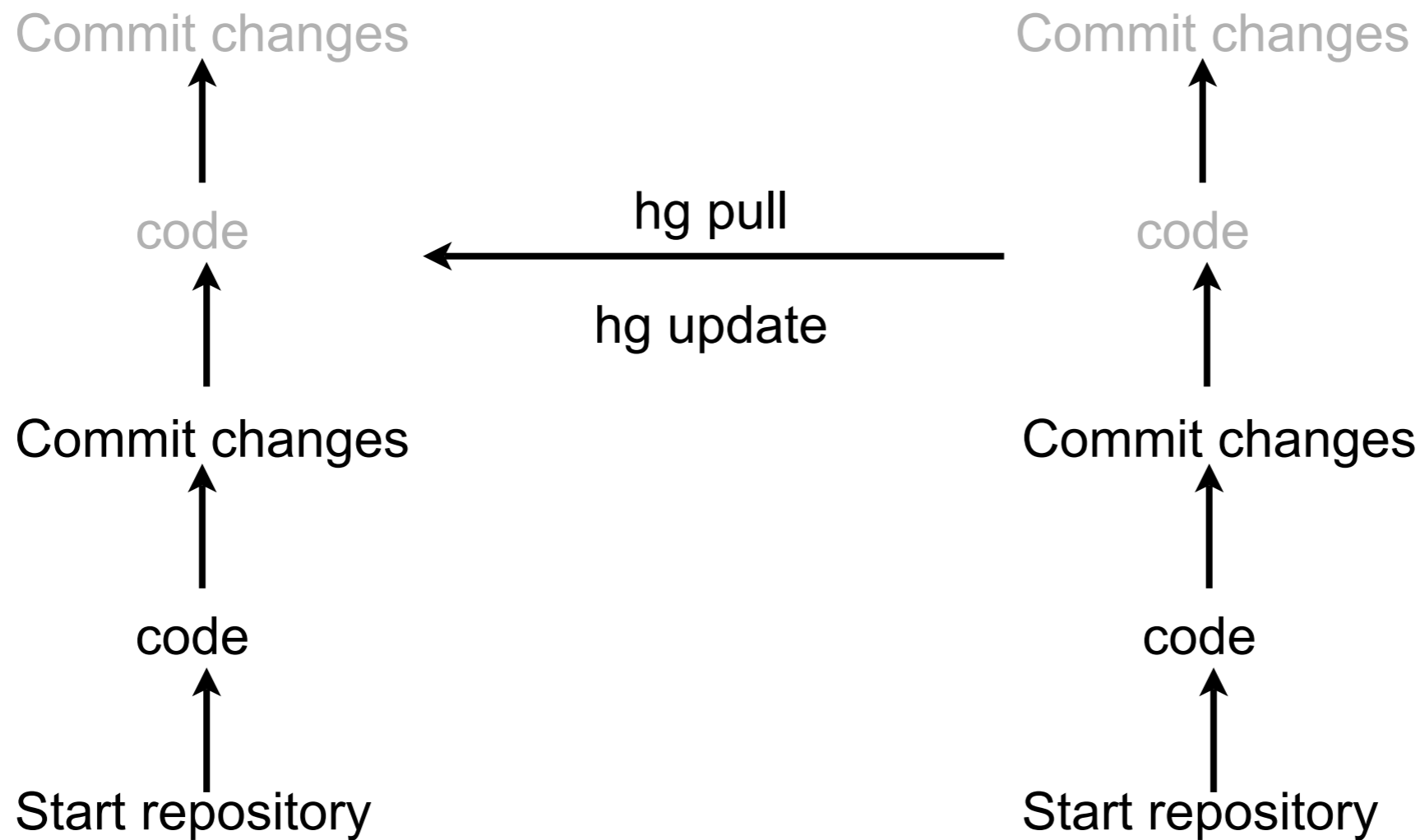
# Standard Mercurial Workflow



# Standard Mercurial Workflow



# Standard Mercurial Workflow



# Basic Source Control Operations

Starting a new project

Adding code to a project

Modifying existing code

Retrieving past versions of code

Handling conflicts in code

Creating code branches

Merging code branches

Cloning repositories

Pulling repositories

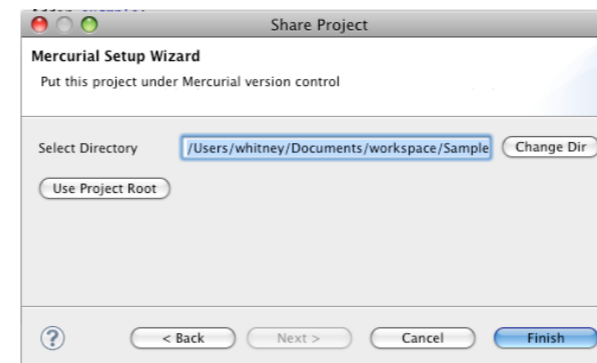
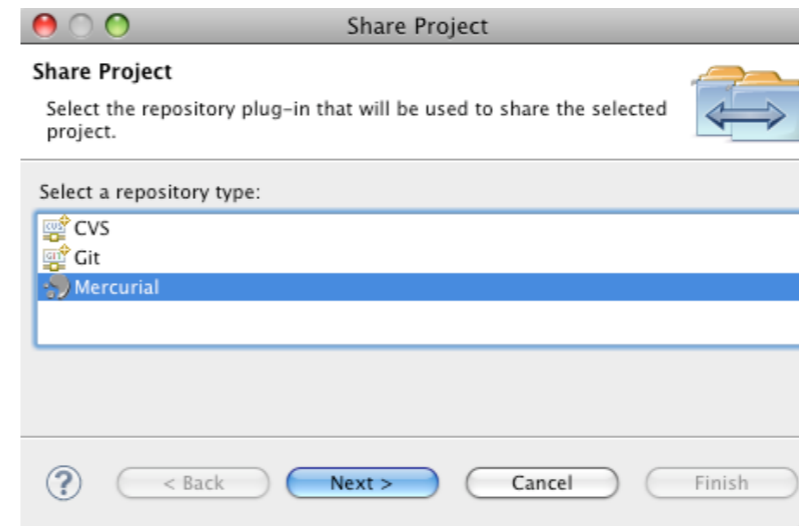
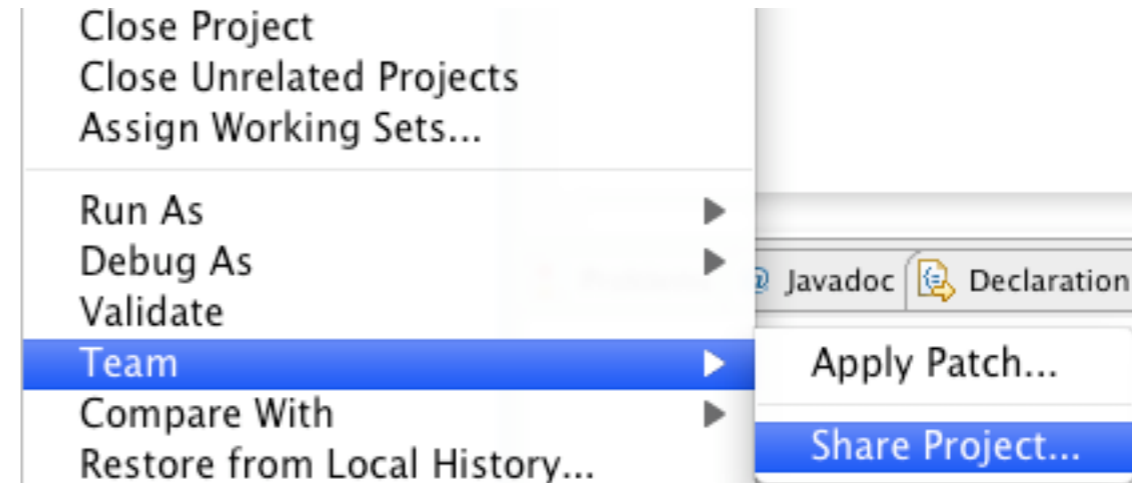
Pushing repositories

# Mercurial Commands



# Creating a Repository

hg init myproject



# Repository & Working Directory

## Repository

In .hg directory

Contains

history of changes

list of files part of project

## Working Directory

Contains

Project files

Project directories

.hg

# Adding Files to Repository

Must tell Repository which files to maintain

`hg add`

Adds all files in the current directory

`hg add filename`

Adds the named file

Just adds files to list of files to maintain

Does not add contents of files to repository

# Committing files to Repository

```
hg commit
```

# Adding/Committing in Eclipse

The screenshot shows the Eclipse IDE interface. On the left, the 'Team' menu is open, displaying various options. The 'Commit...' option is highlighted in blue. In the background, a code editor shows the text 'private int foo;'.

- New
- Go Into
- Open in New Window
- Open Type Hierarchy F4
- Show In ⌘⌘W
- Copy ⌘C
- Copy Qualified Name
- Paste ⌘V
- Delete ⌘X
- Remove from Context ⌘⇧⌘↓
- Build Path
- Source ⌘⌘S
- Refactor ⌘⌘T
- Import...
- Export...
- Refresh F5
- Close Project
- Close Unrelated Projects
- Assign Working Sets...
- Run As
- Debug As
- Validate
- Team

Commit...  
Push...  
Pull...  
Update  
Switch To...  
Apply Patch...  
Import Patch...  
Export Patch...  
Tags...  
Bookmarks...  
Branch...  
Merge...  
Rebase...



The screenshot shows the 'Commit changes to local Mercurial repository' dialog box. It contains a text area for the commit message, a dropdown for selecting an old commit message, a text field for the user to record as the committer (set to 'whitney'), and a table of files to be committed. The table lists files with their status, all of which are 'Untracked'. There are also checkboxes for 'Select/unselect all', 'Show added/removed files', and 'Revert unchecked resources', along with a 'Show Diff' button and 'Cancel' and 'OK' buttons at the bottom.

Commit changes to local Mercurial repository  
Enter a commit message and select the files to commit.

Sample Commit

Select old commit message

User to record as committer: whitney

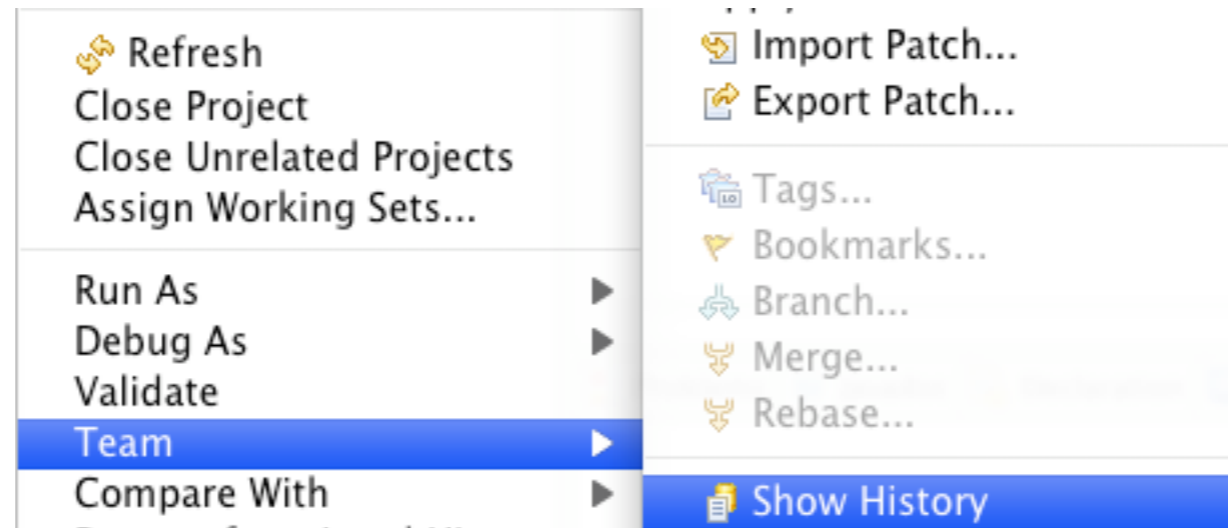
Select Files:

File	Status
<input checked="" type="checkbox"/> .classpath	Untracked
<input checked="" type="checkbox"/> .project	Untracked
<input checked="" type="checkbox"/> .settings/org.eclipse.jdt.core.prefs	Untracked
<input checked="" type="checkbox"/> src/edu/sdsu/cs/cs580/Example.java	Untracked

Select/unselect all  
 Show added/removed files  
Show Diff  
 Revert unchecked resources

Cancel OK

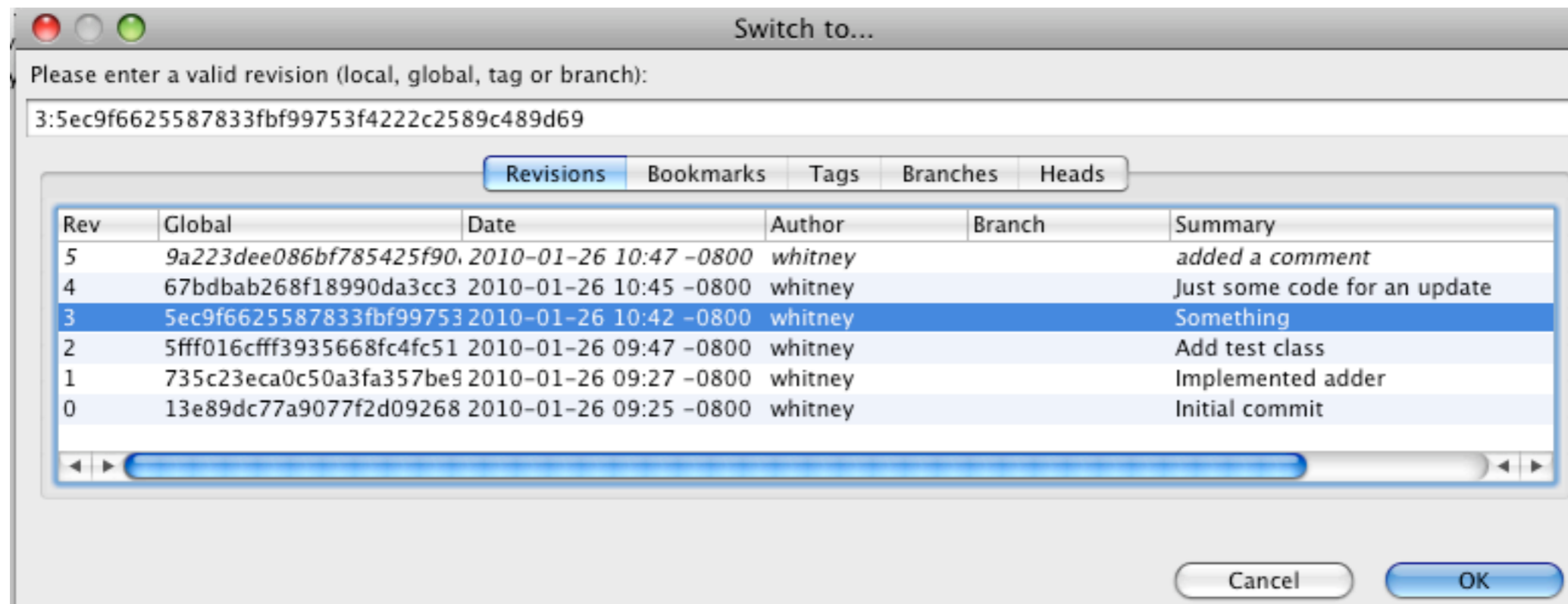
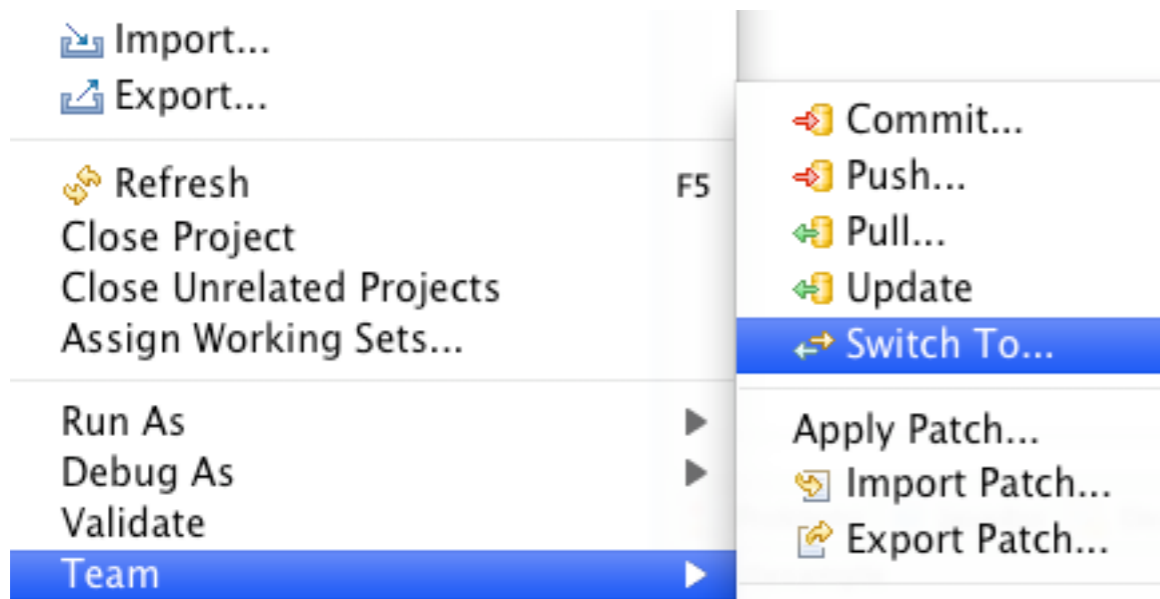
# hg log



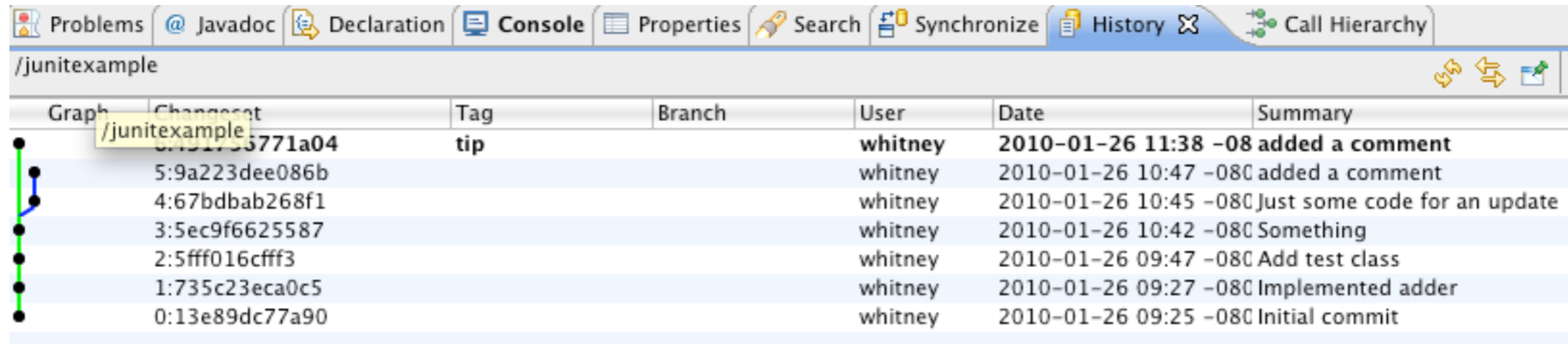
/junitexample

Graph	Changeset	Tag	Branch	User	Date	Summary
●	5:9a223dee086b	tip		whitney	2010-01-26 10:47 -08	added a comment
●	4:67bdbab268f1			whitney	2010-01-26 10:45 -08	Just some code for an update
●	3:5ec9f6625587			whitney	2010-01-26 10:42 -08	Something
●	2:5fff016cff3			whitney	2010-01-26 09:47 -08	Add test class
●	1:735c23eca0c5			whitney	2010-01-26 09:27 -08	Implemented adder
●	0:13e89dc77a90			whitney	2010-01-26 09:25 -08	Initial commit

# Reverting to different Version



# Creating Branches

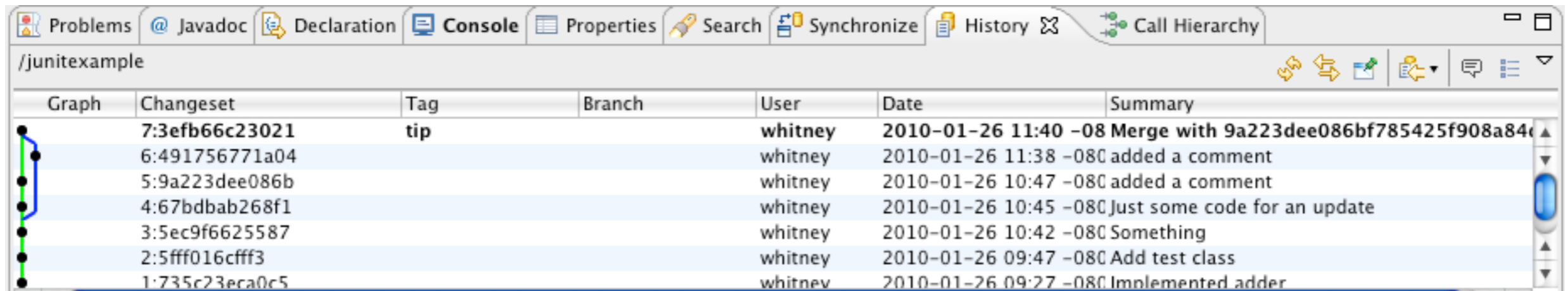


Graph	Commit	Tag	Branch	User	Date	Summary
	5:751753771a04	tip		whitney	2010-01-26 11:38 -08	added a comment
	5:9a223dee086b			whitney	2010-01-26 10:47 -08	added a comment
	4:67bdbab268f1			whitney	2010-01-26 10:45 -08	Just some code for an update
	3:5ec9f6625587			whitney	2010-01-26 10:42 -08	Something
	2:5fff016cfff3			whitney	2010-01-26 09:47 -08	Add test class
	1:735c23eca0c5			whitney	2010-01-26 09:27 -08	Implemented adder
	0:13e89dc77a90			whitney	2010-01-26 09:25 -08	Initial commit



# Merging - without conflicts

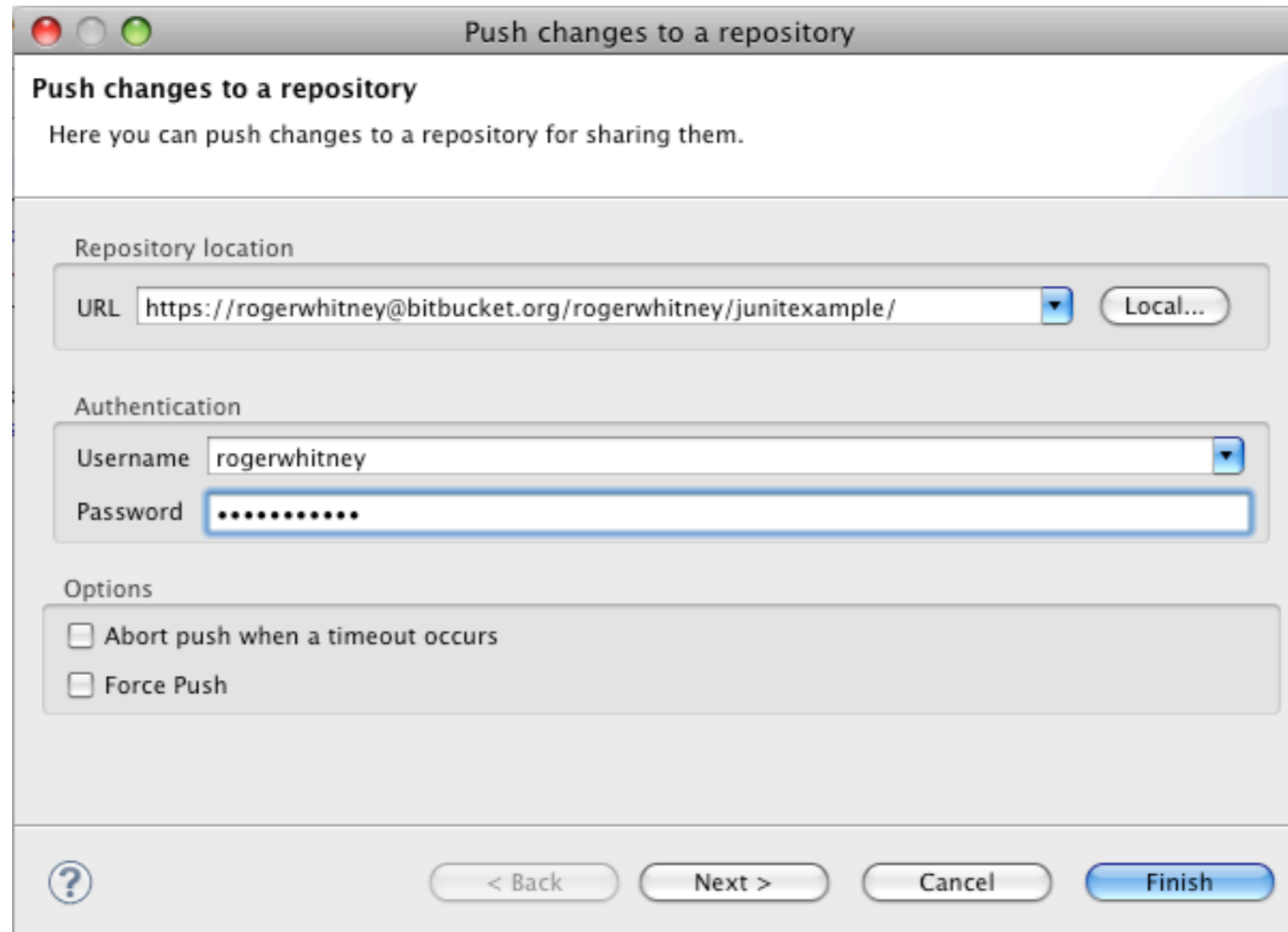
use the merge item in the team menu  
and commit



Graph	Changeset	Tag	Branch	User	Date	Summary
	7:3efb66c23021	tip		whitney	2010-01-26 11:40 -08	Merge with 9a223dee086bf785425f908a84c
	6:491756771a04			whitney	2010-01-26 11:38 -08	added a comment
	5:9a223dee086b			whitney	2010-01-26 10:47 -08	added a comment
	4:67bdbab268f1			whitney	2010-01-26 10:45 -08	Just some code for an update
	3:5ec9f6625587			whitney	2010-01-26 10:42 -08	Something
	2:5fff016cfff3			whitney	2010-01-26 09:47 -08	Add test class
	1:735c23eca0c5			whitnev	2010-01-26 09:27 -08	Implemented adder

# Uploading to BitBucket

After creating a project in BitBucket  
Use the Push item in the team menu



The screenshot shows a dialog box titled "Push changes to a repository". The dialog has a title bar with standard window controls (red, yellow, green buttons) and the text "Push changes to a repository". Below the title bar, the main content area is titled "Push changes to a repository" and contains the text "Here you can push changes to a repository for sharing them." The dialog is divided into three sections: "Repository location", "Authentication", and "Options".

**Repository location**

URL

**Authentication**

Username

Password

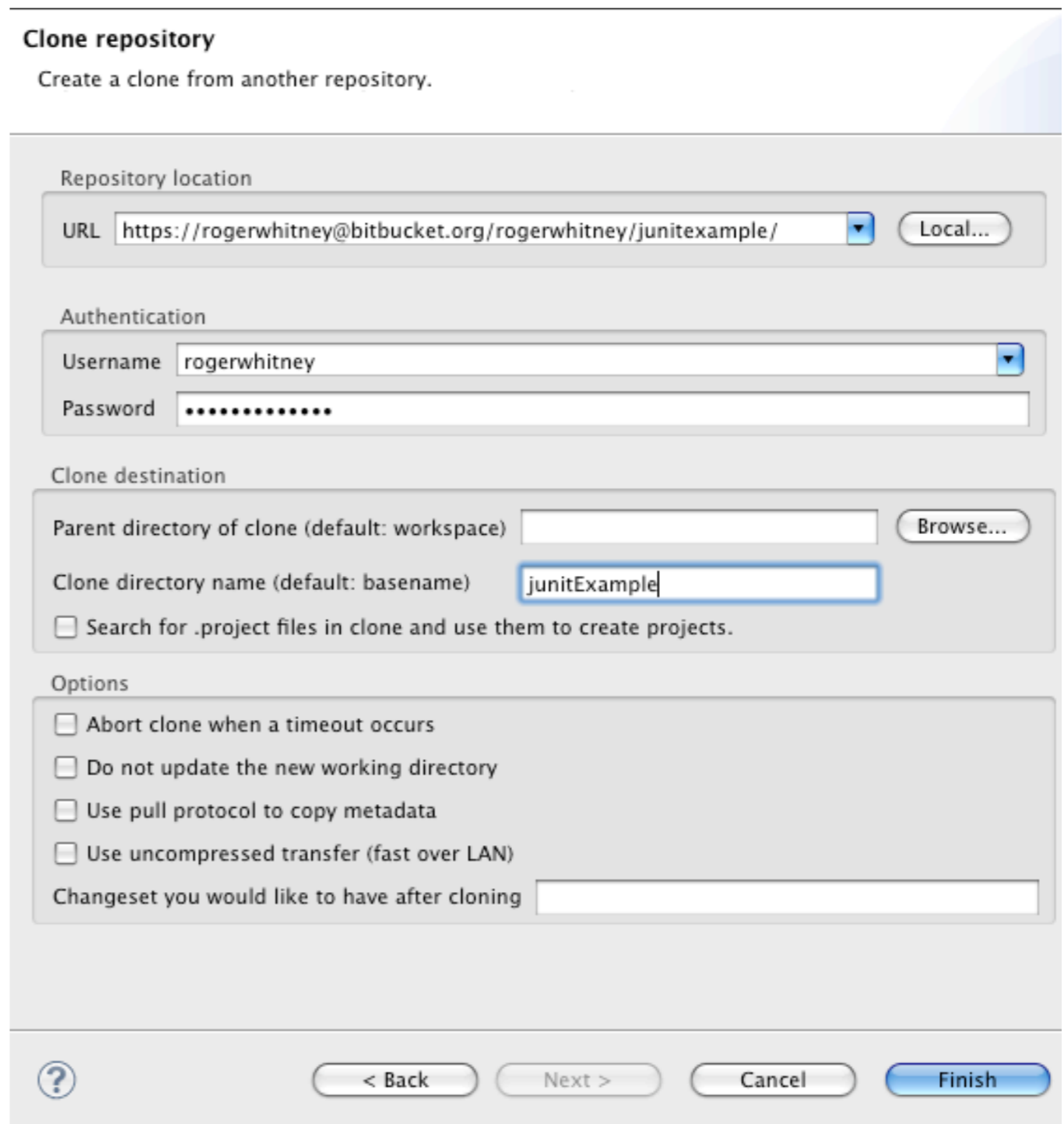
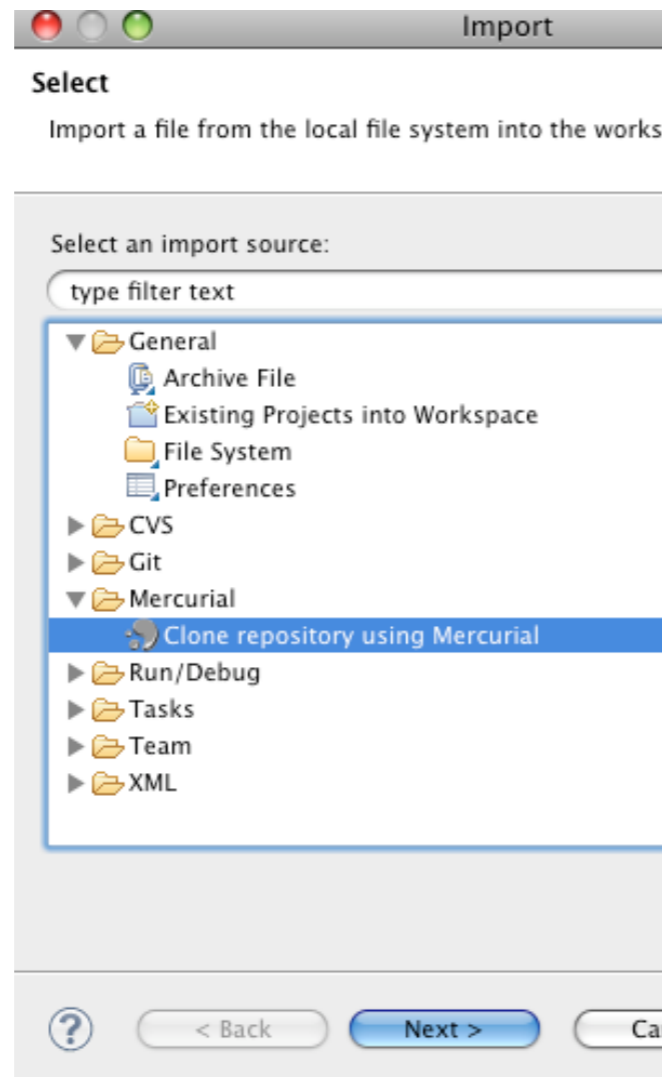
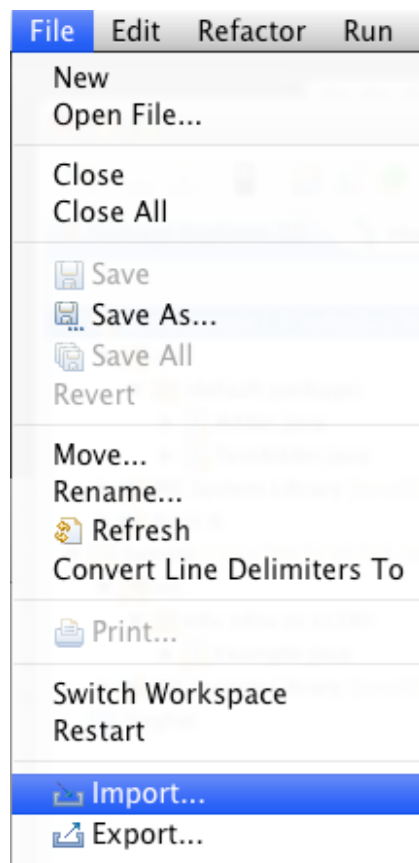
**Options**

Abort push when a timeout occurs

Force Push

At the bottom of the dialog, there is a help icon (question mark in a circle) on the left, and four buttons: "< Back", "Next >", "Cancel", and "Finish". The "Finish" button is highlighted in blue.

# Downloading From Bitbucket



# Unit Testing

# Testing

## Johnson's Law

If it is not tested it does not work

The more time between coding and testing

More effort is needed to write tests

More effort is needed to find bugs

Fewer bugs are found

Time is wasted working with buggy code

Development time increases

Quality decreases

# Unit Testing

Tests individual code segments

Automated tests

# What wrong with:

Using print statements

Writing driver program in main

Writing small sample programs to run code

Running program and testing it be using it

We have a QA Team, so why should I write tests?



# When to Write Tests

First write the tests

Then write the code to be tested

Writing tests first saves time

- Makes you clear of the interface & functionality of the code

- Removes temptation to skip tests

# What to Test

Everything that could possibly break

Test values

- Inside valid range

- Outside valid range

- On the boundary between valid/invalid

GUIs are very hard to test

- Keep GUI layer very thin

- Unit test program behind the GUI, not the GUI

# Common Things Programs Handle Incorrectly

Adapted with permission from “A Short Catalog of Test Ideas” by Brian Marick,  
<http://www.testing.com/writings.html>

## Strings

Empty String

## Collections

Empty Collection

Collection with one element

Collection with duplicate elements

Collections with maximum possible size

## Numbers

Zero

The smallest number

Just below the smallest number

The largest number

Just above the largest number

# XUnit

Free frameworks for Unit testing

SUnit originally written by Kent Beck 1994

JUnit written by Kent Beck & Erich Gamma

Available at: <http://www.junit.org/>

Ports to many languages at:

<http://www.xprogramming.com/software.htm>

# XUnit Versions

3.x

Old version

Works with a versions of Java

4.x

Current version 4.8.1

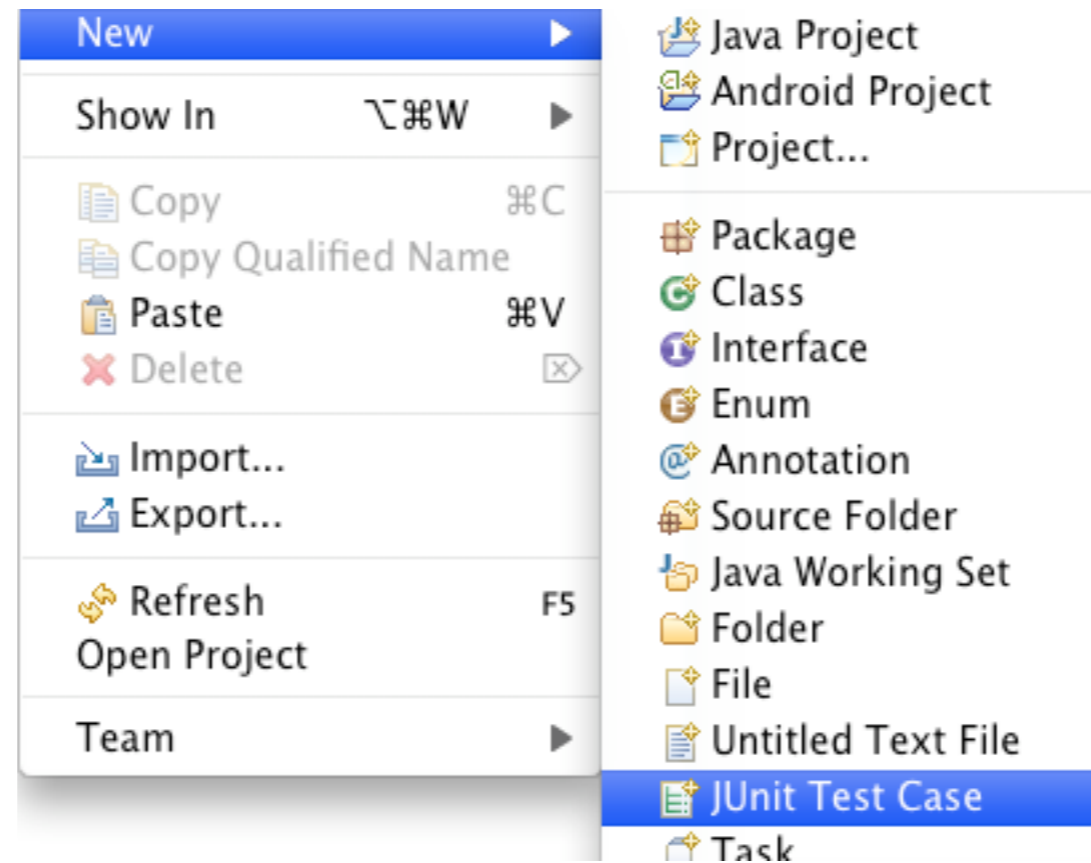
Uses Annotations

Requires Java 5 or later

# Simple Class to Test

```
public class Adder {  
    private int base;  
    public Adder(int value) {  
        base = value;  
    }  
  
    public int add(int amount) {  
        return base + amount;  
    }  
}
```

# Creating Test Case in Eclipse



# Creating Test Case in Eclipse

**JUnit Test Case**  
⚠ The use of the default package is discouraged.

New JUnit 3 test  New JUnit 4 test

Source folder:

Package:

Name:

Superclass:

Which method stubs would you like to create?

setUpBeforeClass()  tearDownAfterClass()  
 setUp()  tearDown()  
 constructor

Do you want to add comments? (Configure templates and default value [here](#))  
 Generate comments

Class under test:

Fill in dialog window &  
create the test cases



# Test Class

```
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;
import org.junit.Test;
```

```
public class TestAdder {
```

```
    @Test
```

```
    public void testAdd() {
```

```
        Adder example = new Adder(3);
```

```
        assertEquals(4, example.add(1));
```

```
    }
```

```
    @Test
```

```
    public void testAddFail() {
```

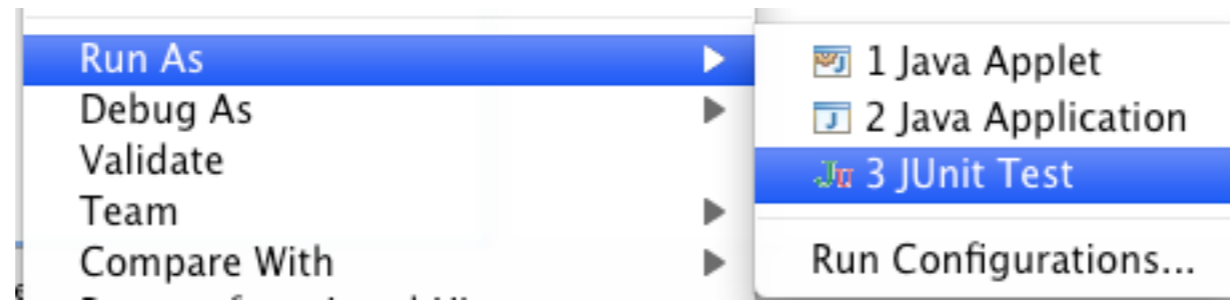
```
        Adder example = new Adder(3);
```

```
        assertTrue(3 == example.add(1));
```

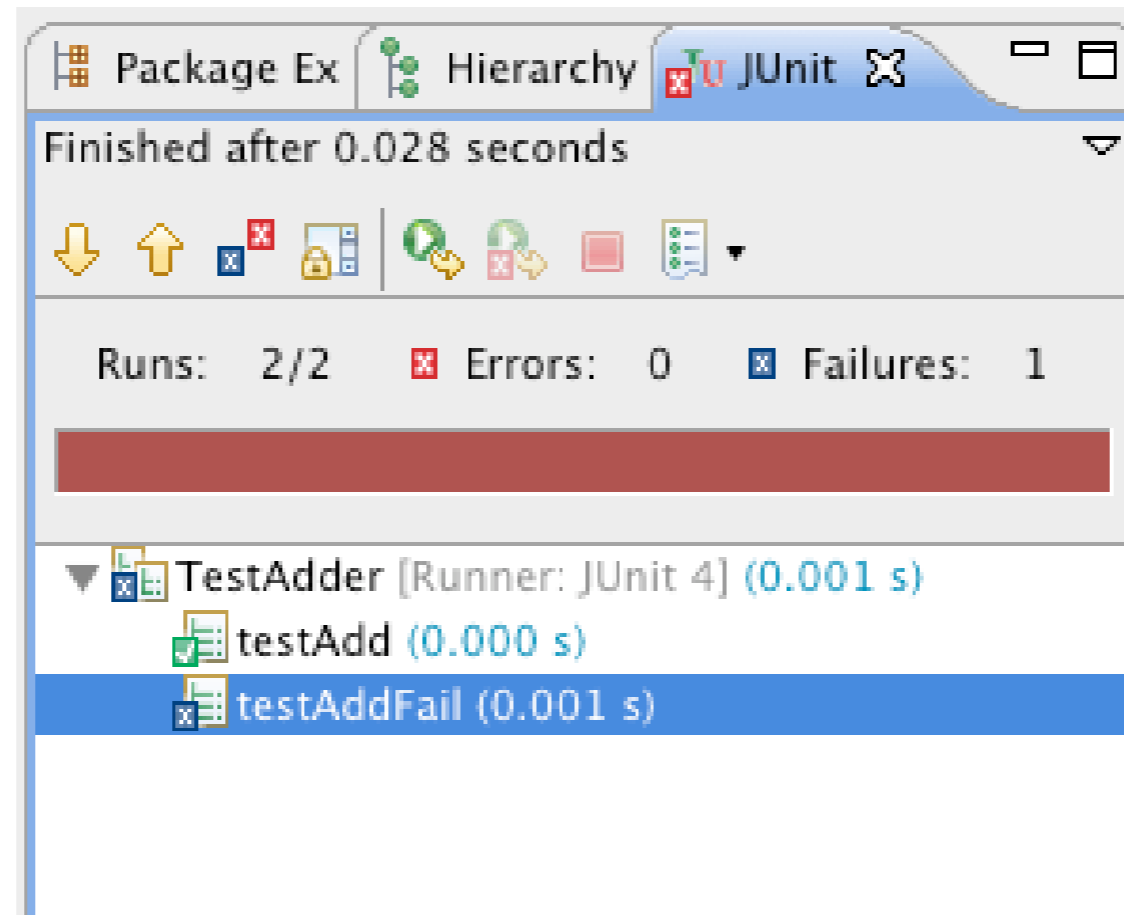
```
    }
```

```
}
```

# Running the Tests



# The result



# Assert Methods

assertArrayEquals()  
assertTrue()  
assertFalse()  
assertEquals()  
assertNotEquals()  
assertSame()  
assertNotSame()  
assertNull()  
assertNotNull()  
fail()

# Annotations

After

AfterClass

Before

BeforeClass

Ignore

Rule

Test

# Using Before

```
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;
```

```
import org.junit.Before;
import org.junit.Test;
```

```
public class TestAdder {
    Adder example;

    @Before
    public void setupExample() {
        example = new Adder(3);
    }

    @Test
    public void testAdd() {
        assertEquals(4, example.add(1));
    }
}
```

