Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Software Engineering and Design
Case Study 1 – Task 0

Prof. Dr. Jürgen Vogel - juergen.vogel@bfh.ch

Project: Case Study 1

- Project
  - teams with 4 or 5 members
    - teams will remain fixed until end of semester
  - as a team, you work on one SE project: Case Study 1
    - the project is divided into ~ 11 Tasks
    - each task is a typical SE task (analysis, design, implementation)
  - counts 25% of overall grade
    - each task will be graded
      - based on documents, software and/or presentation
    - totaling to ~ 150 points
- Team rules
  - your teammates depend on your contribution
  - in the coming weeks, you will meet your team regularly during and outside the class hours (depending on project tasks)
  - if you cannot attend a meeting, you are required to (1) tell your team members and (2) find out what you have missed

Project Files

- Project Files (Documents, Presentations, Code, …)
  - manage documents via file sharing platform (GitHub)
    - invite all team members plus vgj1
- Each team is required to keep a small project diary
  - project tasks, status (open, work in progress, done), responsibilities
  - meetings (date, time, purpose, results)
  - use a plain text document format (e.g., diary.txt or diary.html)

Project Team Setup

Team Setup

- 3 teams: 2 teams with 4 and 1 team with 5 members
- each team gets a color code: red, green, or blue
- sign up on the whiteboard with your BFH identifier
- NOW

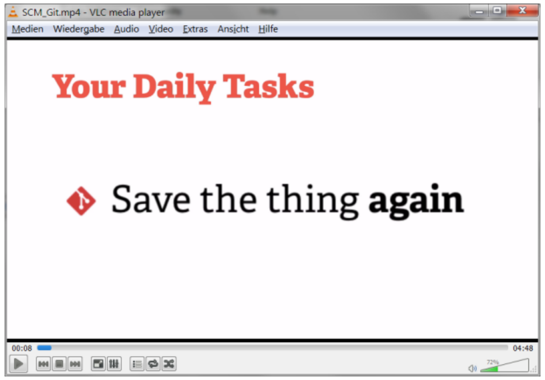
IDE Setup: Java and Eclipse

CS1 Task 0

- install the latest version (7u40) of Java SDK
  - http://www.oracle.com/technetwork/java/javase/downloads/index.html
- install the latest version (4.3) of Eclipse IDE for Java EE Developers
  - http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/keplerr

File Sharing and Versioning (1)

- software development is team work
  - cooperation between developers (architects, programmers, UI designers, …), project managers, and users
- source code (class, data, media, configuration, … files) nowadays is mostly managed in specialized and centralized repositories: version control system
  - also known as "Source Code Management" SCM
  - implements client-server pattern

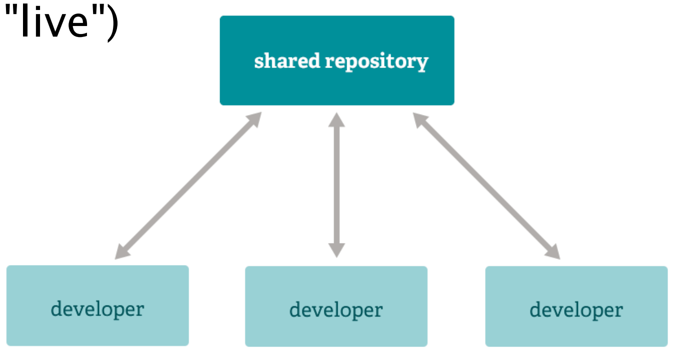

SCM_Git.mp4 - VLC media player
Medien Wiedergabe Audio Video Extras Ansicht Hilfe
00:08
04:48

Your Daily Tasks

Save the thing **again**

- many different version control systems exist
  - e.g., CVS, Subversion, Git, Rational Team Concert, …
  - see http://en.wikipedia.org/wiki/Comparison_of_revision_control_software
- all files in the version control repository get some additional meta data
  - author (of the original file and any update)
  - version incremented with every update
  - version comment (to be provided by the author for each new version)

File Sharing and Versioning (2)



- files are not edited directly in the repository (i.e., "live") but need to be copied to a local directory first
- creating a new file version, an editor therefore must follow a specific workflow

1. check out a certain version of the file(s)
   - in most cases the latest version
2. modify files as necessary
3. check if file has been updated by some other editors in the meantime
   - parallel work may happen anytime
   - if yes, check whether there are conflicting changes
     - e.g., modifying the same paragraph
   - if yes, resolve conflict (i.e., decide on final text)
     - often requires some communication with the editor
     - also known as "merging"
4. once done, check in new version of file(s)
   - together with some meaningful comment
   - also known as "commit"

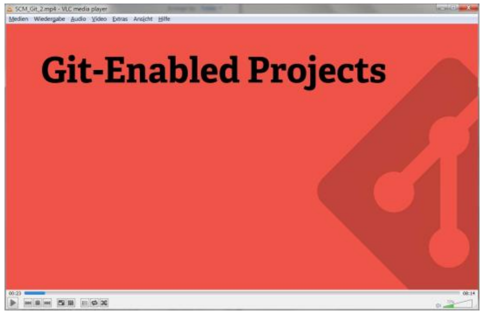
File Sharing and Versioning (3)

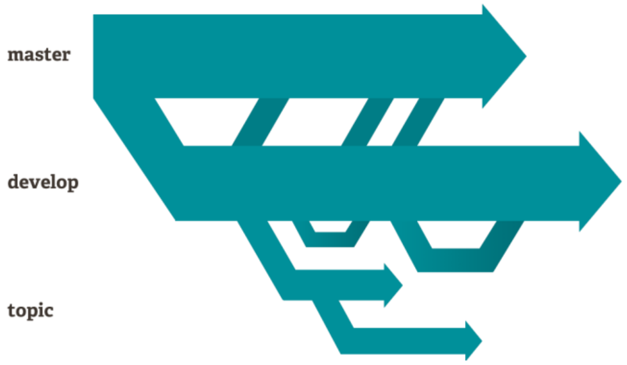
- this workflow may generate many different versions for the same file
  - also known as "history"
- editors may
  - access old versions if required (i.e., old versions are backups)
  - compare different versions of the same file to
    - review the changes of colleagues and how they affect "your" code
    - identify the cause for newly introduced bugs,
    - analyze design decisions,
    - etc.
  - work on different versions in parallel
    - also known as "branches"
    - e.g., work on a new feature in the "task x.y" branch, try out an idea in an "experimental" branch and fix a bug in the "main" branch
- based on a set of files with specific version numbers, an overall version of the entire project can be defined
  - e.g., a release
  - also known as "snapshot", "tag", "label", or "baseline"

Git

Git (http://git-scm.com/)

- is an SCM and has originally been developed for the open source developement of the Linux kernel (by Linus Torvalds)
- is a distributed SCM and keeps a complete local copy of the SCM server repository
  - creating a local repository is known as "cloning"
- focus is support for branching and merging
  - e.g., create a branch per feature and later on decide which feature should go into the next release
  - i.e., ideas may be developed in private and are published once ready (or abandoned quietly)
- Git uses a 2 step commit workflow
  1. preparation ("staging"): notify Git about changed files
  2. commit: add the change to the repository
- is a command line tool





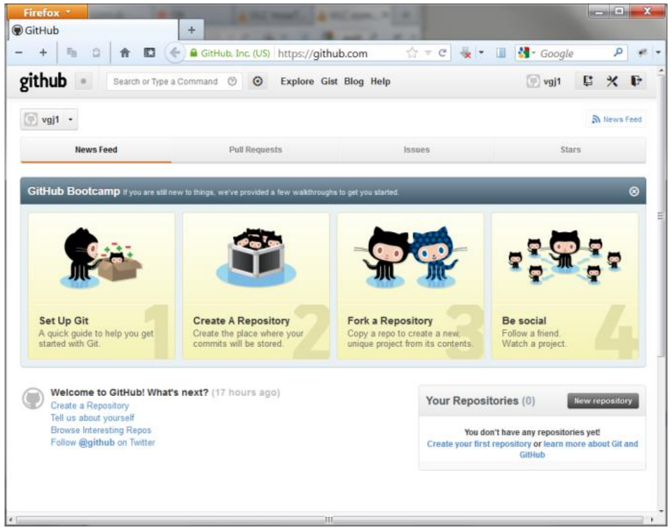
Working with Git: GitHub Server

GitHub (https://github.com/)

- hosts a publicly accessible Git server
- usage is free for open source project

CS1 Task 0: GitHub Setup

- each team member creates an account
  - use your bfh login name (e.g., vgj1)
- for each team, create a single repository
  - project: ch.bfh.btx8081.w2013.{team color}
  - enter description
  - check "public"
- invite your team members and vgj1 to your repository
  - go to Settings -> Collaborators -> Add...
- remember HTTPS URI to your repository
  - e.g., https://github.com/vgj1/ch.bfh.btx8081.w2013.test.git



GitHub Bootcamp If you are still new to things, we've provided a few walkthroughs to get you started.

Set Up Git
A quick guide to help you get started with Git.

Create A Repository
Create the place where your commits will be stored.

Fork a Repository
Copy a repo to create a new, unique project from its contents.

Be social
Follow a friend.
Watch a project.

Welcome to GitHub! What's next? (17 hours ago)
Create a Repository
Tell us about yourself
Browse Interesting Repos
Follow @github on Twitter

Your Repositories (0)

You don't have any repositories yet!
Create your first repository or learn more about Git and GitHub

Working with Git: EGit

EGit (http://www.eclipse.org/egit/)

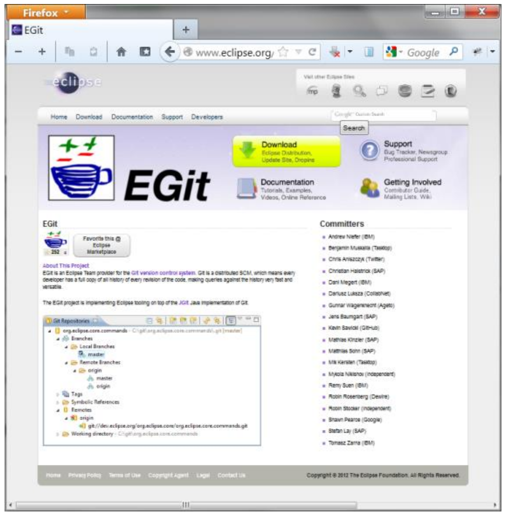
- is an Eclipse plugin for Git
- based on the JGit Java implementation of Git
- pre-installed with your Eclipse version

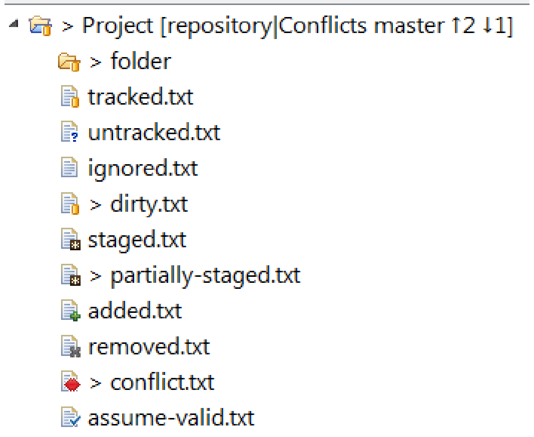
CS1 Task 0: EGit Configuration

1. basic configuration
   - open Eclipse Preferences -> Team -> Git
   - set "default repository" (e.g., `C:\develop\git\`)
   - set "ignore whitespace changes"
   - set "store credentials in secure store"
2. get familiar with the "label decorations" in the Eclipse project explorer

Tutorials

- http://www.vogella.com/articles/EGit/article.html
- http://unicase.blogspot.ch/2011/01/egit-tutorial-for-beginners.html




```
> Project [repository|Conflicts master ↑2 ↓1]
    > folder
    tracked.txt
    untracked.txt
    ignored.txt
    > dirty.txt
    staged.txt
    > partially-staged.txt
    added.txt
    removed.txt
    > conflict.txt
    assume-valid.txt
```

CS1 Task 0: Setup Project with EGit and GitHub (1)

Phase 1: Do on 1 team member's machine

1. in Eclipse, prepare your "master" project
   - create a new Java project with name `ch.bfh.btx8081.w2013.{team color}`
   - create an additional source folder `doc`
   - create your project diary file `diary.html` in `doc`
2. create a local Git repository
   - right click on the project -> Team -> Share Project -> Git -> next
   - create a new repository (e.g., `C:\develop\git\`)
     - same name as project
   - select project -> Finish
3. initial commit
   - right click on the project -> Team -> Commit
   - commit message "project setup"
   - files: select your project diary from `doc` folder
   - commit

CS1 Task 0: Setup Project with EGit and GitHub (2)

4. connect project to your GitHub repository
   - open "Git Repository Exploring" perspective
   - select your project `ch.bfh.bti7081.s2013.{team color}`
   - right-click "Remotes" and choose "Create Remote"
   - for "Remote name" leave "origin", select "push" and click "OK"
   - click "Change"
   - enter your project's GitHub repository URI from earlier
   - add user name and password and press "Finish"
   - click "save and push"
   - you should now see your updated project at GitHub

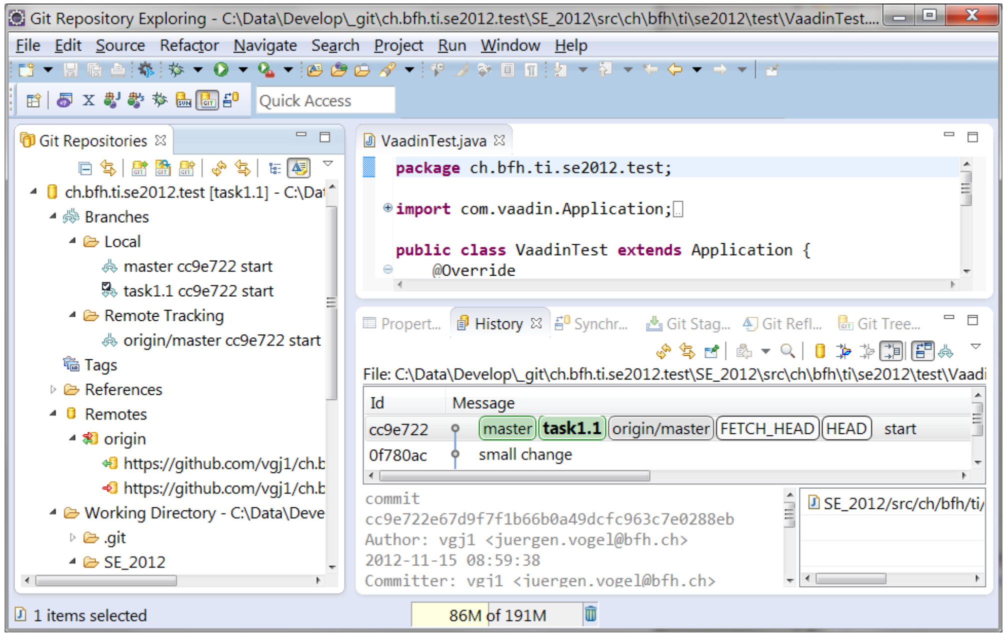
CS1 Task 0: Setup Project with EGit and GitHub (3)

Phase 2: Do on OTHER team members' machines

1. check out project from GitHub ("clone")
   - in Eclipse, open File -> Import
   - select "Git" -> "Projects from Git" -> "URI"
   - enter GitHub repository URI (e.g. `https://github.com/vgj1/project.git`)
   - add user name and password and click "Next"
   - select branch "master"
   - select destination, e.g., `c:\develop\git\ch.bfh.ti.se2012.red`, and click "Next"
   - select "import existing project" click "Next", select project and click "Finish"

Summary EGit/GitHub Setup

- For each managed resource (file), we have
  - a working file in the eclipse workspace
  - a managed copy in 2 connected Git repositories
    - local machine (= local file system)
    - remote machine (= GitHub), called "origin"
    - open "Git Repository Exploring" perspective to see both



Git Repository Exploring - C:\Data\Develop_git\ch.bfh.ti.se2012.test\SE_2012\src\ch\bfh\ti\se2012\test\VaadinTest....
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Git Repositories
VaadinTest.java
Propert... History Synchr... Git Stag... Git Refl... Git Tree...
1 items selected
86M of 191M

```
ch.bfh.ti.se2012.test [task1.1] - C:\Da
  Branches
    Local
      master cc9e722 start
      task1.1 cc9e722 start
    Remote Tracking
      origin/master cc9e722 start
  Tags
  References
  Remotes
    origin
      https://github.com/vgj1/ch.b
      https://github.com/vgj1/ch.b
  Working Directory - C:\Data\Deve
    .git
    SE_2012
```

```java
package ch.bfh.ti.se2012.test;

import com.vaadin.Application;

public class VaadinTest extends Application {
    @Override
```

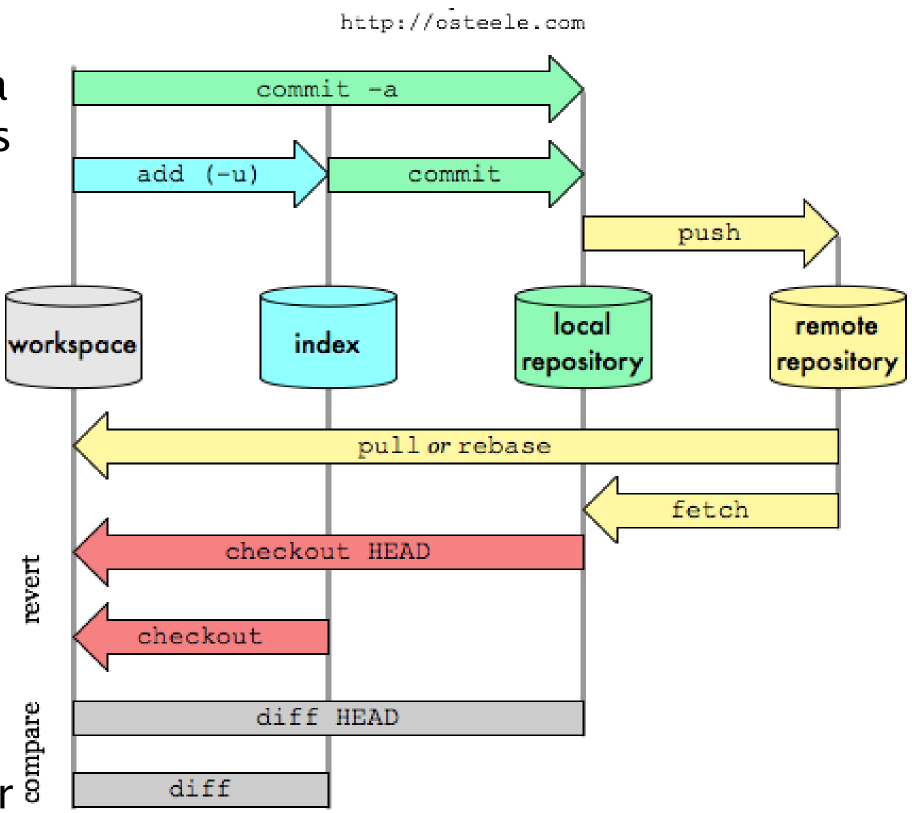
File: C:\Data\Develop_git\ch.bfh.ti.se2012.test\SE_2012\src\ch\bfh\ti\se2012\test\Vaadi

Id	Message
cc9e722	master task1.1 origin/master FETCH_HEAD HEAD start
0f780ac	small change

```
commit
cc9e722e67d9f7f1b66b0a49dcfc963c7e0288eb
Author: vgj1 <juergen.vogel@bfh.ch>
2012-11-15 08:59:38
Committer: vgj1 <juergen.vogel@bfh.ch>
```

SE_2012/src/ch/bfh/ti/

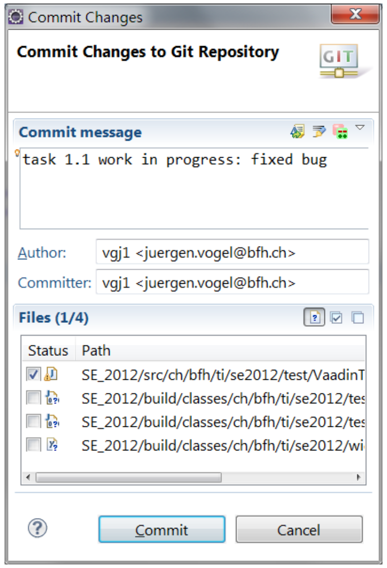
Git Commands Overview



Git commands for syncing resources (in a certain branch) between these 3 locations

- (additional index used by Git to monitor changes to the local workspace)
- `add` (stage): marks the current state of a workspace resource for the next commit
- `commit`: updates the local repository with all staged resources from the workspace
- `push`: updates the resources in the remote repository with their branch version of the local repository (only for non-conflicting changes or merged resources)
- `fetch`: updates the resources in the local repository with their branch version in the remote repository
- `merge`: merges all changes (starting from the common root) from the named version of the local repository into the current branch of the workspace and marks any conflicting changes directly in the concerned files
- `pull`: executes a fetch and merge from the remote repository

Working with EGit (1)



task 1.1 work in progress: fixed bug

Author: vgj1 <juergen.vogel@bfh.ch>

Committer: vgj1 <juergen.vogel@bfh.ch>

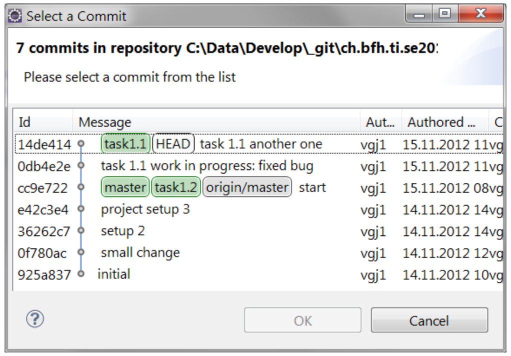
Files (1/4)

Status	Path
✓	SE_2012/src/ch/bfh/ti/se2012/test/VaadinT
	SE_2012/build/classes/ch/bfh/ti/se2012/tes
	SE_2012/build/classes/ch/bfh/ti/se2012/tes
	SE_2012/build/classes/ch/bfh/ti/se2012/wi

1. **working** (developing) and committing changes **locally**
   - now we are ready to work (=update local files)
   - intermediate changes (=not yet to be delivered to the rest of the team) may be committed anytime to the local Git repository
   - to commit, right click on project and select Team -> Commit
     - select which file updates belong to this commit
       - NEVER commit any files from the build directory!
     - always put a meaningful commit, e.g., "added new tasks to diary.html"
     - deselect the "push" option to keep your changes local
2. **reviewing the history**
   - the history gives an overview of all changes
   - right click on project, select Team -> Show in History
     - you can see all commits, comments, the author, the concerned files
   - right click on a certain file, select Team -> Show in History opens file view

Working with EGit (2)

3. **reverting local changes**
   - local changes can be reverted to an earlier version (=undo)
   - for single files (or entire project): right click on file (or project), select Replace with -> Commit (or Branch) and select the correct version



7 commits in repository C:\Data\Develop_git\ch.bfh.ti.se20

Please select a commit from the list

Id	Message	Aut...	Authored ...	C
14de414	task1.1 HEAD task 1.1 another one	vgj1	15.11.2012 11	vg
0db4e2e	task 1.1 work in progress: fixed bug	vgj1	15.11.2012 11	vg
cc9e722	master task1.2 origin/master start	vgj1	15.11.2012 08	vg
e42c3e4	project setup 3	vgj1	14.11.2012 14	vg
36262c7	setup 2	vgj1	14.11.2012 14	vg
0f780ac	small change	vgj1	14.11.2012 12	vg
925a837	initial	vgj1	14.11.2012 10	vg

Working with EGit (3)

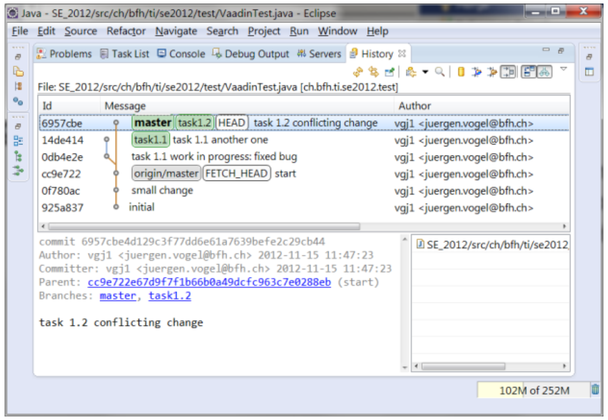
4. **publishing local changes**

- if our local branch is ready to be delivered to the team, commit it to GitHub
  - before TEST whether everything works
    - do not deliver untested code to your team members: nothing is worse than checking out buggy code and having to figure out what is wrong
- BUT: each team member may update the shared global repository at any time and there may be conflicting changes, e.g., changes to the same method

a) **update** local copy of the master branch
  - switch to "Git Repository Exploring" perspective
  - right click on the Git repository, select "Fetch from upstream" and click ok

b) **merge** and resolve conflicts
  - switch to the master branch (right click on project, Team -> Switch)
  - right click on project, Team -> Merge, select branch to merge with option "commit", click "Merge"
  - if there is no conflict, you just get a status report
  - else you need to resolve the conflict (next slide)

c) **commit** all changes via Team -> Push to Upstream


Java - SE_2012/src/ch/bfh/ti/se2012/test/VaadinTest.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Task List Console Debug Output Servers History

File: SE_2012/src/ch/bfh/ti/se2012/test/VaadinTest.java [ch.bfh.ti.se2012.test]

Id	Message	Author
6957cbe	master task1.2 HEAD task 1.2 conflicting change	vgj1 <juergen.vogel@bfh.ch>
14de414	task1.1 task 1.1 another one	vgj1 <juergen.vogel@bfh.ch>
0db4e2e	task 1.1 work in progress: fixed bug	vgj1 <juergen.vogel@bfh.ch>
cc9e722	origin/master FETCH_HEAD start	vgj1 <juergen.vogel@bfh.ch>
0f780ac	small change	vgj1 <juergen.vogel@bfh.ch>
925a837	initial	vgj1 <juergen.vogel@bfh.ch>

```
commit 6957cbe4d129c3f77dd6e61a7639befe2c29cb44
Author: vgj1 <juergen.vogel@bfh.ch> 2012-11-15 11:47:23
Committer: vgj1 <juergen.vogel@bfh.ch> 2012-11-15 11:47:23
Parent: cc9e722e67d9f7f1b66b0a49dcfc963c7e0288eb (start)
Branches: master, task1.2

task 1.2 conflicting change
```

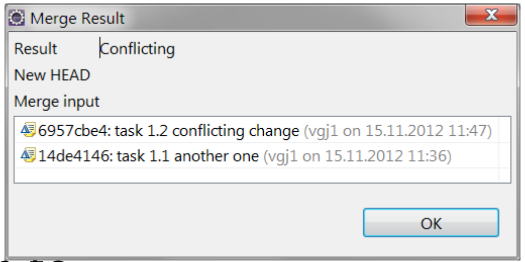
SE_2012/src/ch/bfh/ti/se2012

102M of 252M

Working with EGit (4)

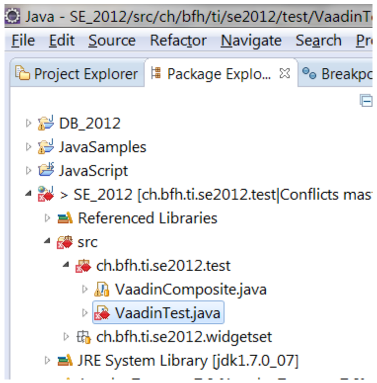
5. **resolving conflicts**

- a conflict means: another developer has changed code in the same area as you
- switch to the master branch (right click on project, Team -> Switch)
- right click on project, Team -> Merge, select the branch to merge with option "commit", click "Merge"
  - if there is a conflict, you get a conflict report
- during merge, Git has modified the source code of all files with conflicts
  - files with conflicts are marked in project explorer
  - inside the file, a special notation shows both versions
- now you need to manually fix all these areas
  - open Merge Tool for some support (see next slide)
  - talk to your teammate to figure out the correct code
- when everything is working (test!), commit all changes
  - right click on file, Team -> Add to Index to stage the file
  - commit / push



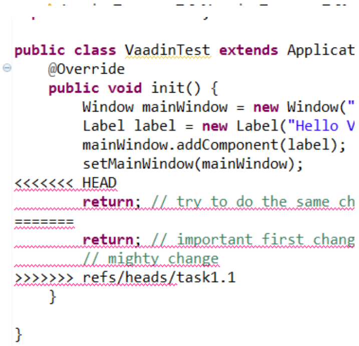
Merge Result

Result: Conflicting
New HEAD
Merge input
6957cbe4: task 1.2 conflicting change (vgj1 on 15.11.2012 11:47)
14de4146: task 1.1 another one (vgj1 on 15.11.2012 11:36)



Java - SE_2012/src/ch/bfh/ti/se2012/test/VaadinT

```
DB_2012
JavaSamples
JavaScript
> SE_2012 [ch.bfh.ti.se2012.test|Conflicts mas
  Referenced Libraries
  src
    ch.bfh.ti.se2012.test
      VaadinComposite.java
      VaadinTest.java
    ch.bfh.ti.se2012.widgetset
  JRE System Library [jdk1.7.0_07]
```



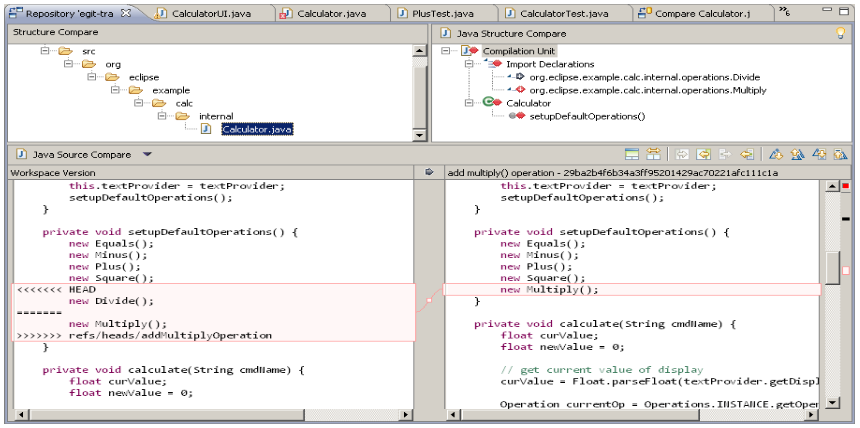
```
public class VaadinTest extends Applicat
    @Override
    public void init() {
        Window mainWindow = new Window("
        Label label = new Label("Hello V
        mainWindow.addComponent(label);
        setMainWindow(mainWindow);
<<<<<<< HEAD
        return; // try to do the same ch
=======
        return; // important first chang
        // mighty change
>>>>>>> refs/heads/task1.1
    }

}
```

Working with EGit (5)

5. **resolving conflicts – merge tool**

- right click on project, Team -> Merge Tool, select "workspace version"
- clicking on a file with a conflict opens the comparison view
  - here you can view your local version and the remote version side by side
  - and copy changes between them
  - or directly edit the code
- modify your code until all conflicts are resolved
- right click on file, Team -> Add to Index to stage the modified resource
- right click on project, Team -> Commit to push your changes to GitHub



add multiply() operation - 29ba2b4f6b34a3ff95201429ac70221afc111c1a

Workspace Version:

```
        this.textProvider = textProvider;
        setupDefaultOperations();
    }

    private void setupDefaultOperations() {
        new Equals();
        new Minus();
        new Plus();
        new Square();
<<<<<<< HEAD
        new Divide();
=======
        new Multiply();
>>>>>>> refs/heads/addMultiplyOperation
    }

    private void calculate(String cmdName) {
        float curValue;
        float newValue = 0;
```

add multiply() operation - 29ba2b4f6b34a3ff95201429ac70221afc111c1a:

```
        this.textProvider = textProvider;
        setupDefaultOperations();
    }

    private void setupDefaultOperations() {
        new Equals();
        new Minus();
        new Plus();
        new Square();
        new Multiply();
    }

    private void calculate(String cmdName) {
        float curValue;
        float newValue = 0;

        // get current value of display
        curValue = Float.parseFloat(textProvider.getDispl

        Operation currentOp = Operations.INSTANCE.getOper
```