Technology Background
Development environment, Skeleton and Libraries

Slides by Prof. Dr. Matthias Hölzl
(based on material from Dr. Andreas Schröder)

Outline

Lecture 1

I. Eclipse
II. Git

Lecture 2

IV. Skeleton Overview
V. Libraries Overview
VI. Game Rules and Coding Task

Part I. Eclipse

Part I: Goals

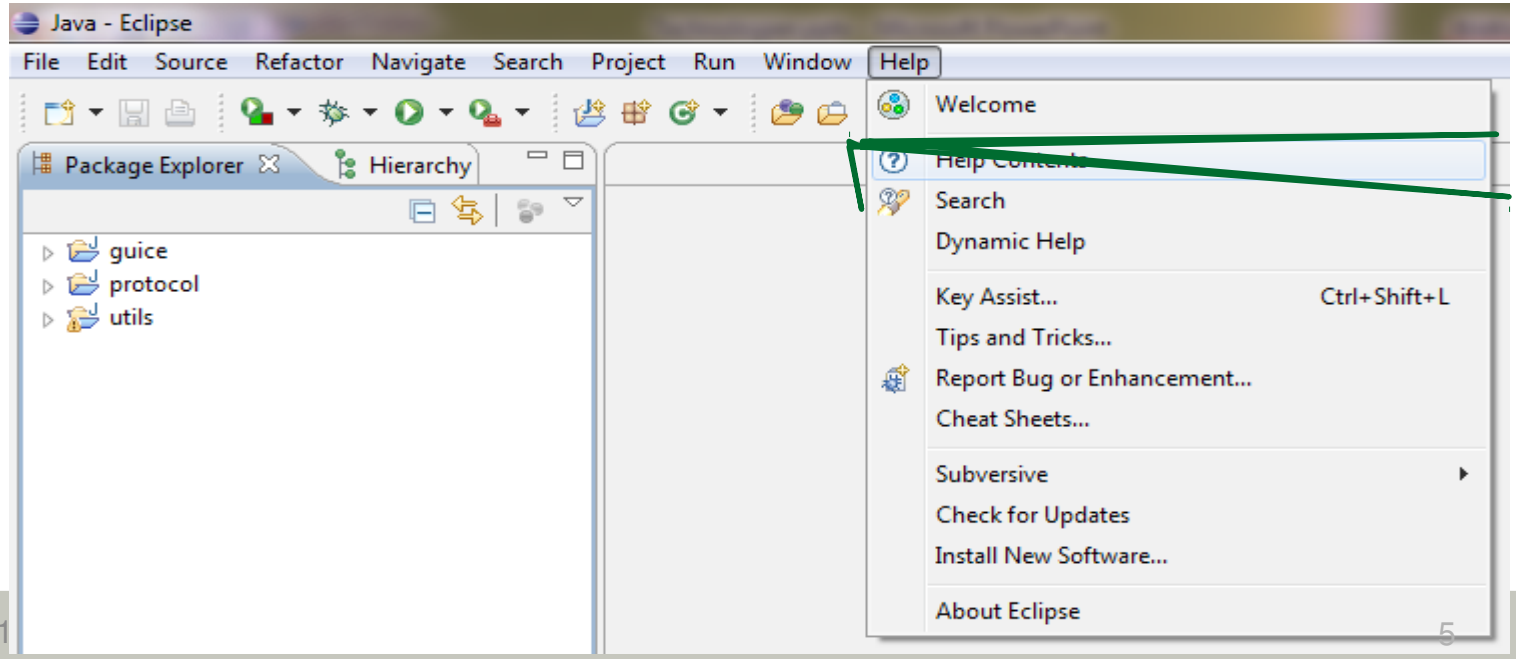
Learning Target

- Recognize the power of Eclipse
- Identify what you did not know yet
- Know where to find tutorials and help
- Being able to set up the Eclipse IDE for the lab

Eclipse

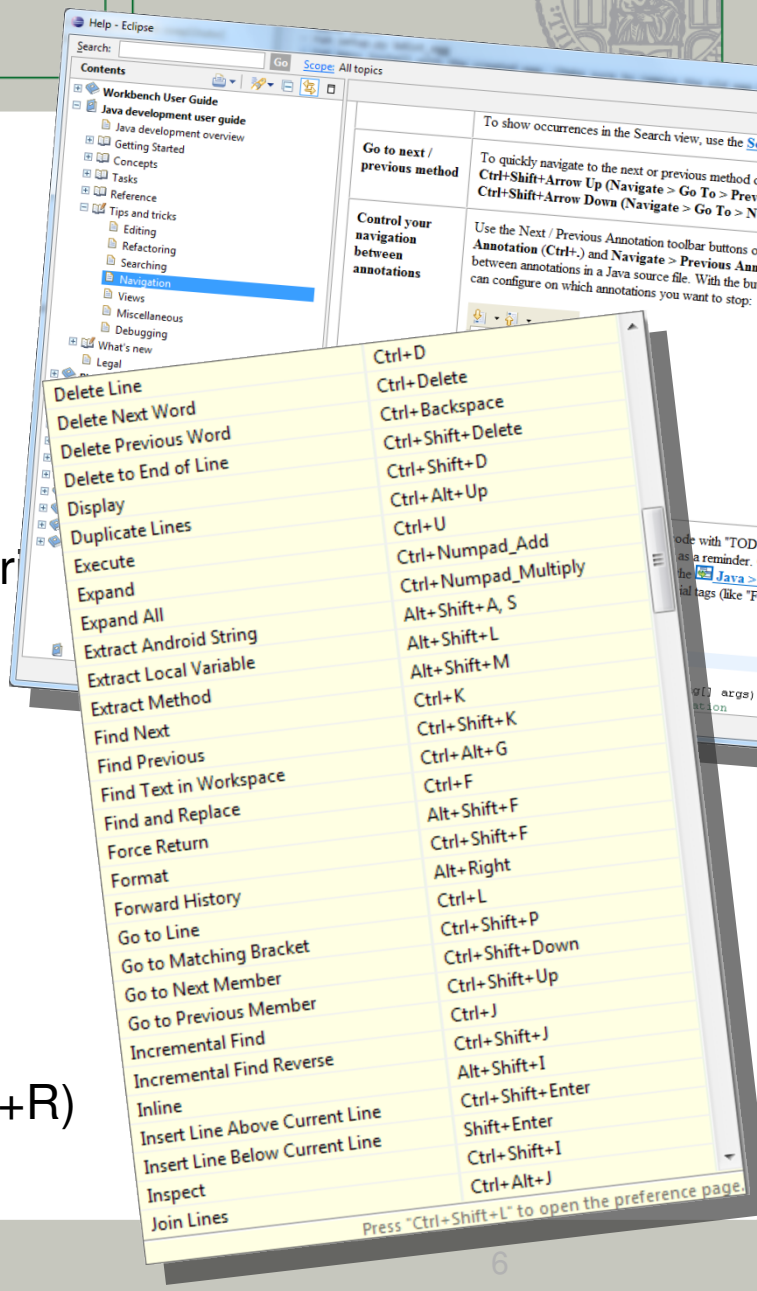
Eclipse is far more than a Java editor

- Code navigation and exploration
- Refactoring
- Background compilation
- Customizable build system
- Extensibile: Git, JUnit, Code Coverage, Web development tools, …



Eclipse

- Recommended **reads**
  - Workbench user guide > Tips and tricks
  - Java development user guide > Tips and tr
- Recommended **shortcuts**
  - Quick Fix (Ctrl+1), Quick Access (Ctrl+3)
  - Open Type / resource (Ctrl+Shift+T / Ctrl+Shift+R)
  - Open declaration / Javadoc (F3 / F2)
  - Quick type hierarchy (Ctrl+T)
  - Quick outline (Ctrl+O)
  - Refactor / Rename (Alt+Shift+T / Alt+Shift+R)
  - … Key bindings overview (Ctrl+Shift+L) J



Setting up Eclipse

To setup your Eclipse, you need to:

1. Download and install eclipse
2. Setup basics: code styles, save actions, file encoding, …
3. Setup Tomcat
4. Setup Git
5. Setup DB
6. Setup Launch configurations
7. Install other plugins/extensions as needed (e.g., EclEmma)

Part II. Git

Git

- Git is a modern **distributed** version control system (VCS)
- Initial release **2005** by Linus Torwalds
- Widely adopted in open source communities: Linux Kernel, Ruby on Rails, Android, Debian, …

- Can best be learned if you **forget everything you know** about how (centralized) version control works!

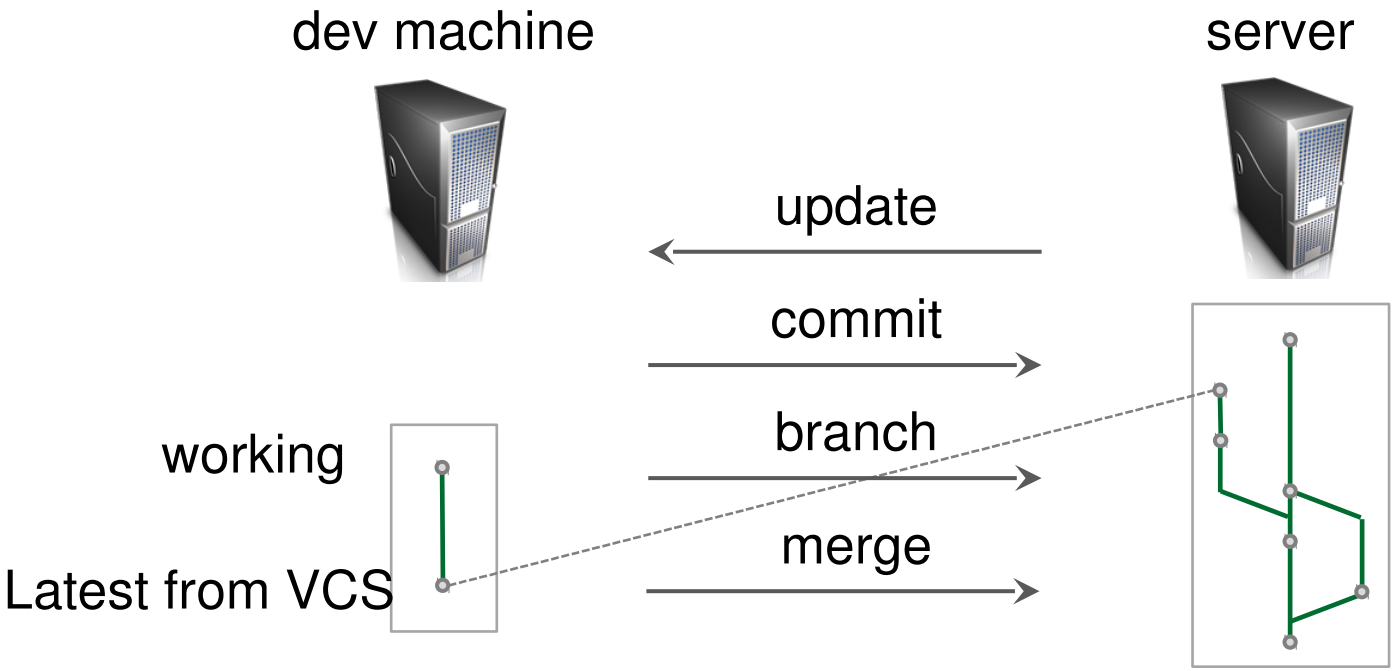
Fundamentals of version control

- Repository – a database containing files under version control and the history of these files.
- Working Copy – a local copy of files from the repository. May be modified, and may not represent the most recent repository revision.
- Revision – the state of a file (CVS), of a branch (Git), or of the whole repository (SVN) as committed to the version control system.
- Change Set – a set of modifications to files under version control.
- Commit – the act of writing a change set from the working copy to the repository.
- Update – the act of fetching changes that have been performed on the repository since the last update and applying them to the working copy.

Fundamentals of version control

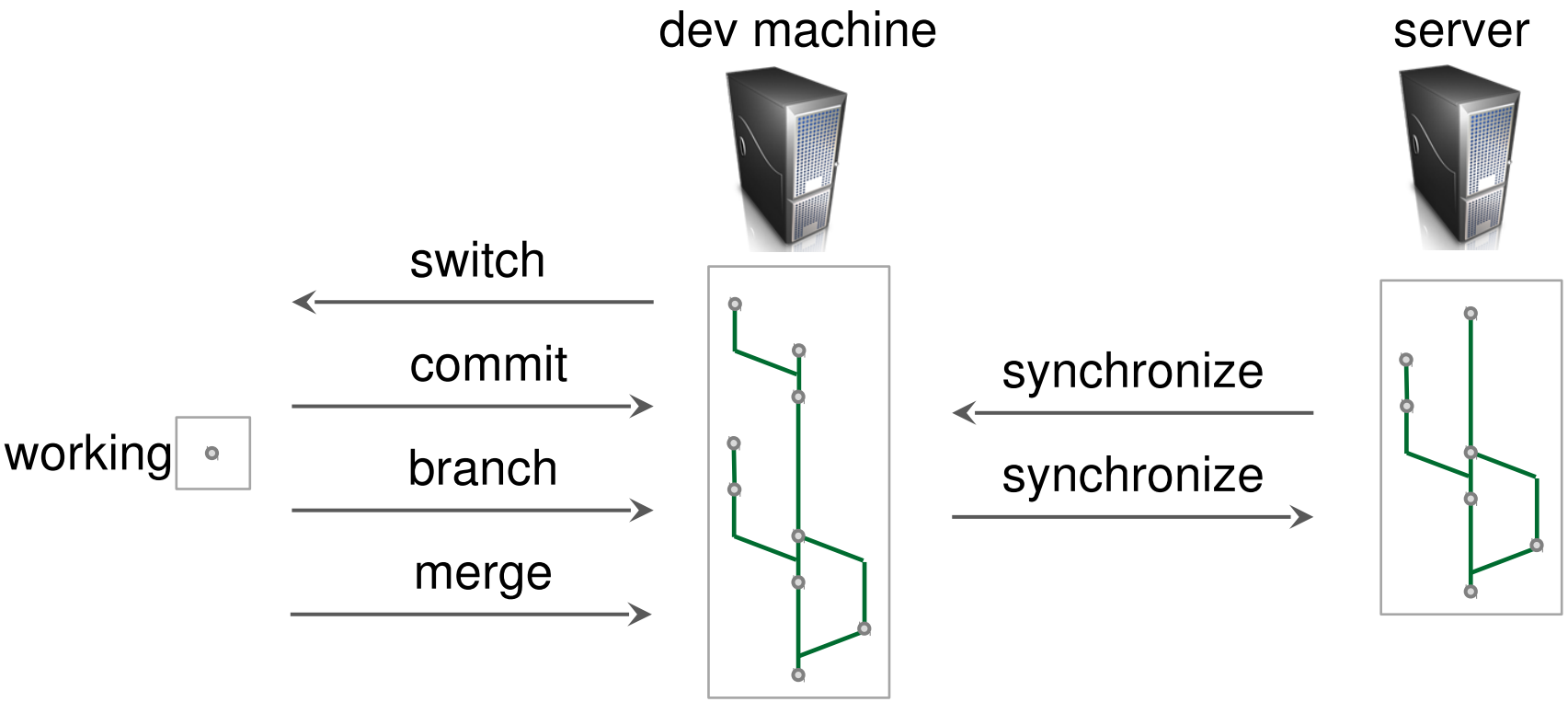
- Branch – a set of files under version control that evolve independently of other file sets. Often defines an own line of development of a product.
- Tag – a human-readable link to a specific revision. Is often used to mark the source code of released versions (e.g. tag v_2_0_3).
- Trunk/Master – the branch denoting the main line of development of a product.
- Merge – the act of reconciling change sets from parallel branches.
- Switch – the act of changing the working copy from a branch to another.
- Conflict – occurs when a file was changed concurrently, and the VCS cannot reconcile the changes automatically. Conflicts must be **resolved** manually.

Centralized VCS



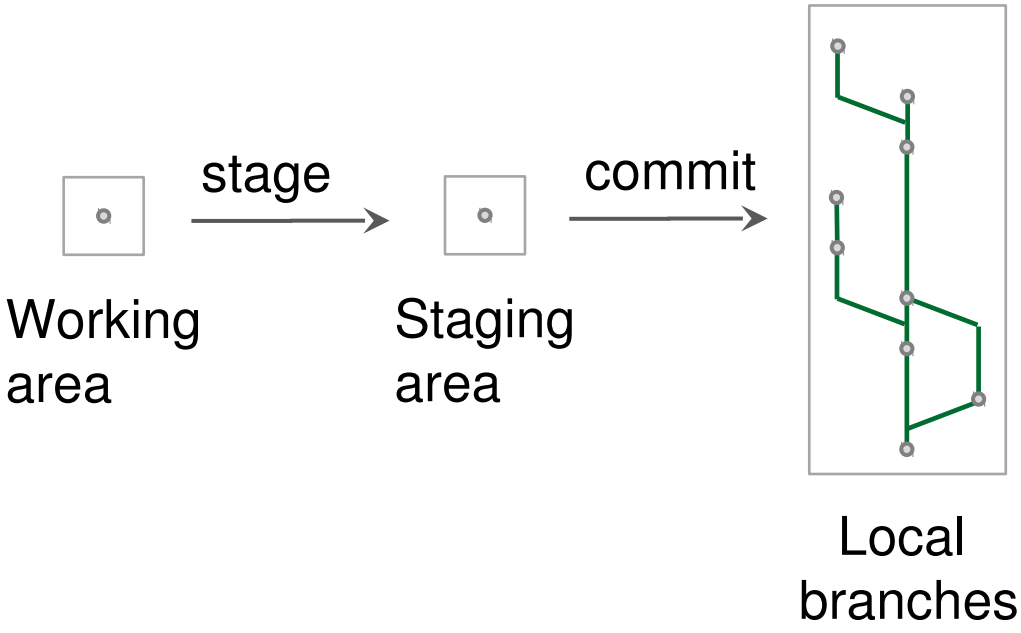
- Cannot work **without connectivity**
- Needs **server** to branch and merge
- Cannot save experimental features **locally**

Decentralized VCS



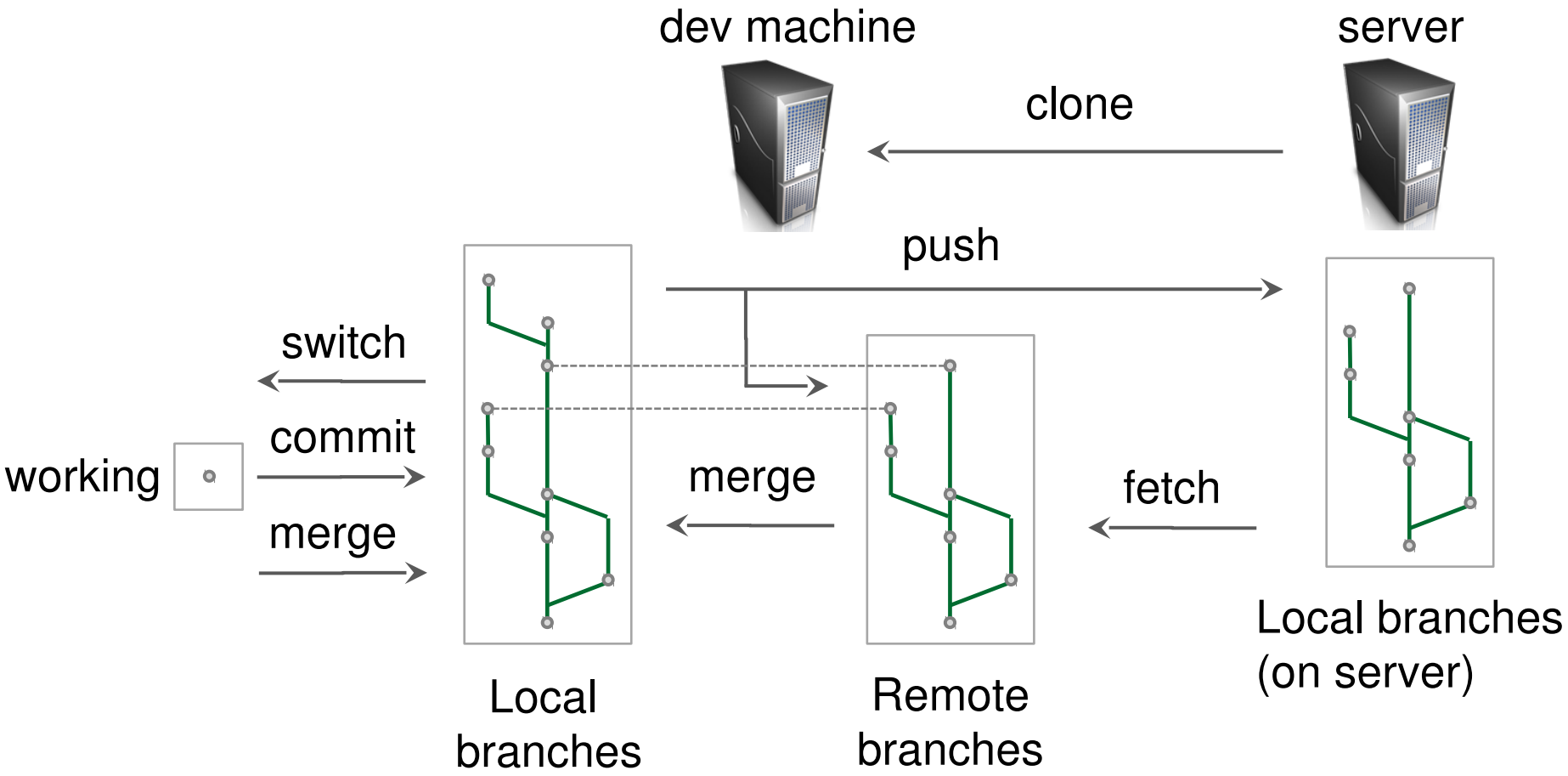
- Works **without connectivity**
- Can branch and merge against **local VCS**
- Needs **synchronization** among multiple VCS

Git Staging



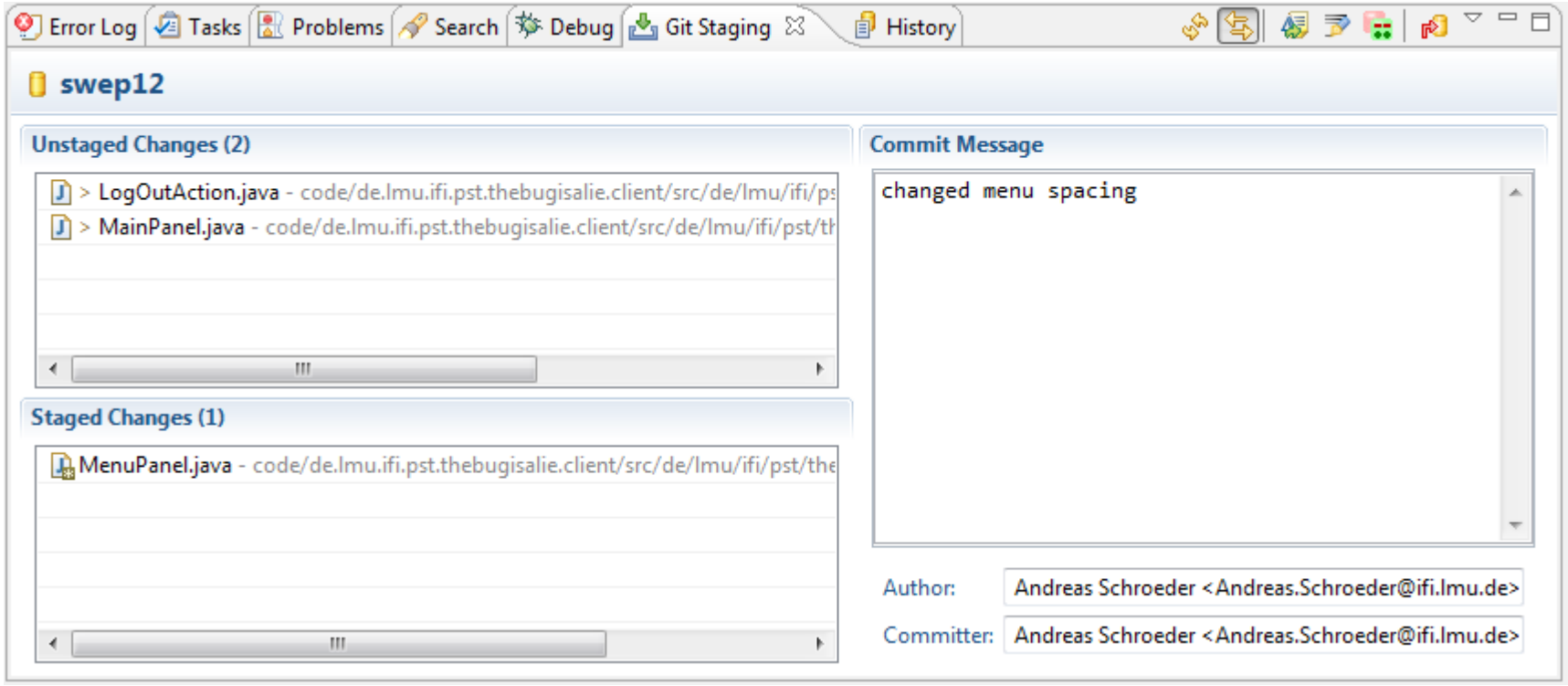
- Git allows to select changes for commit
- “Staging area” lies between working area and local branches

Git



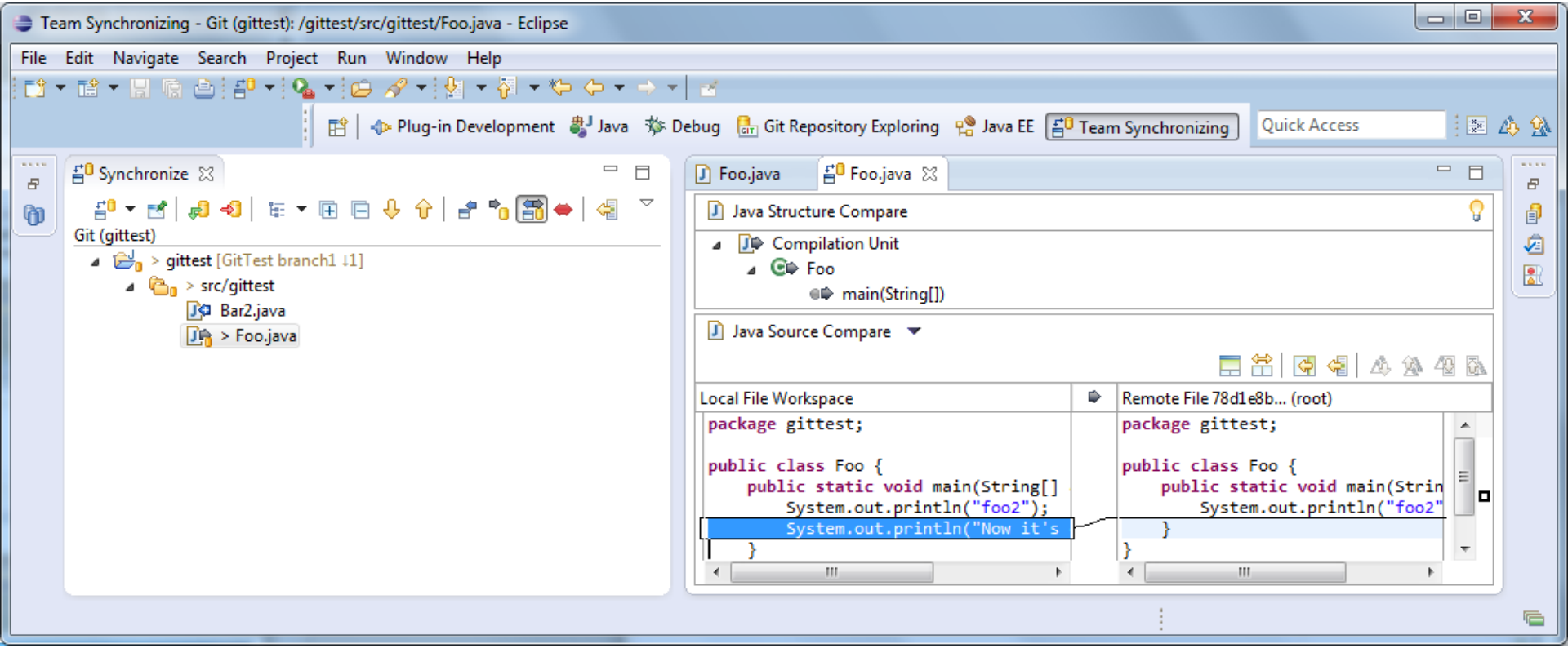
pull = fetch & merge

eGit Staging View



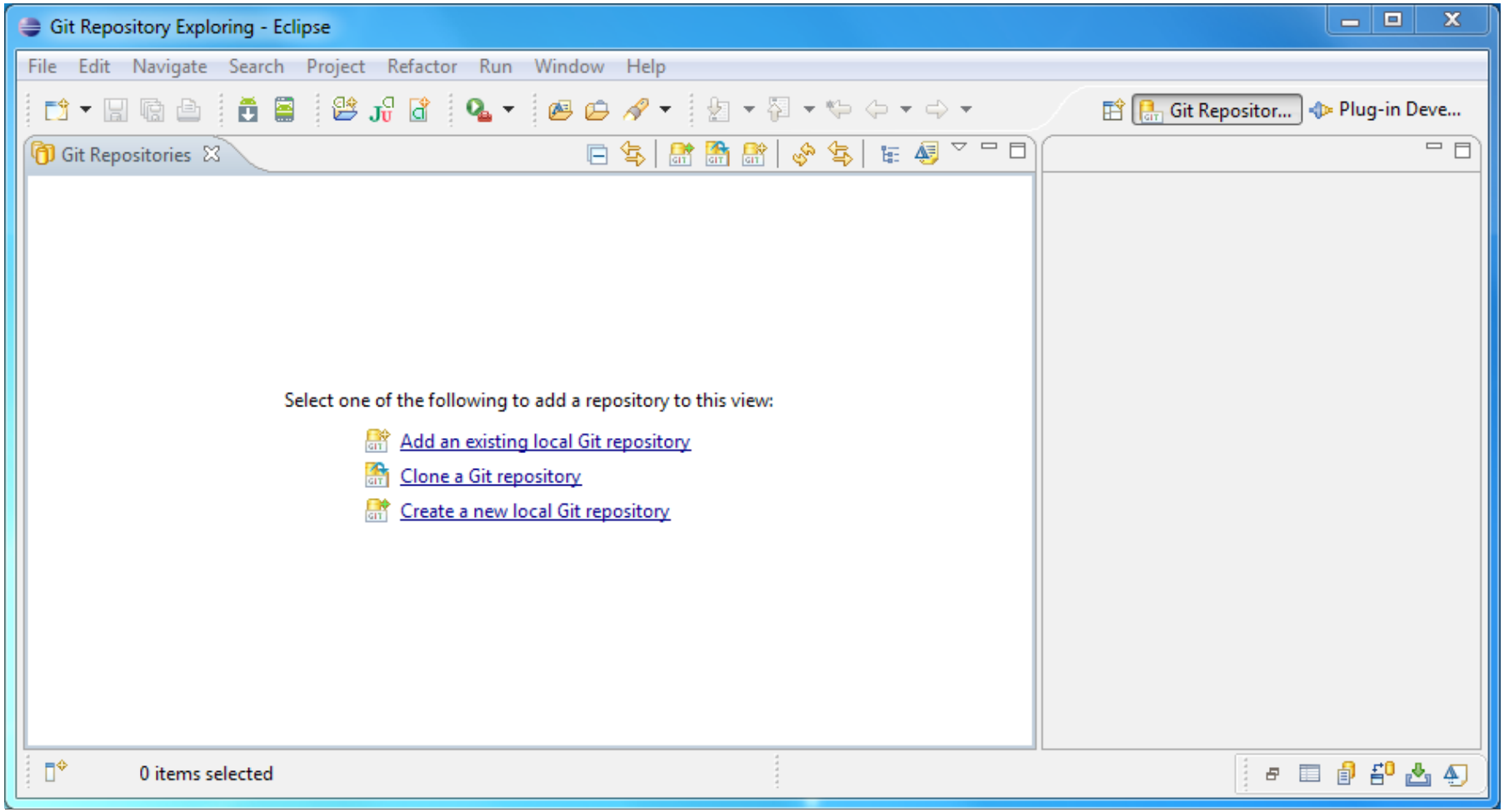
- Helpful tool for creating commits
- Faster than Menu > Team > Add

Synchronize Workspace View

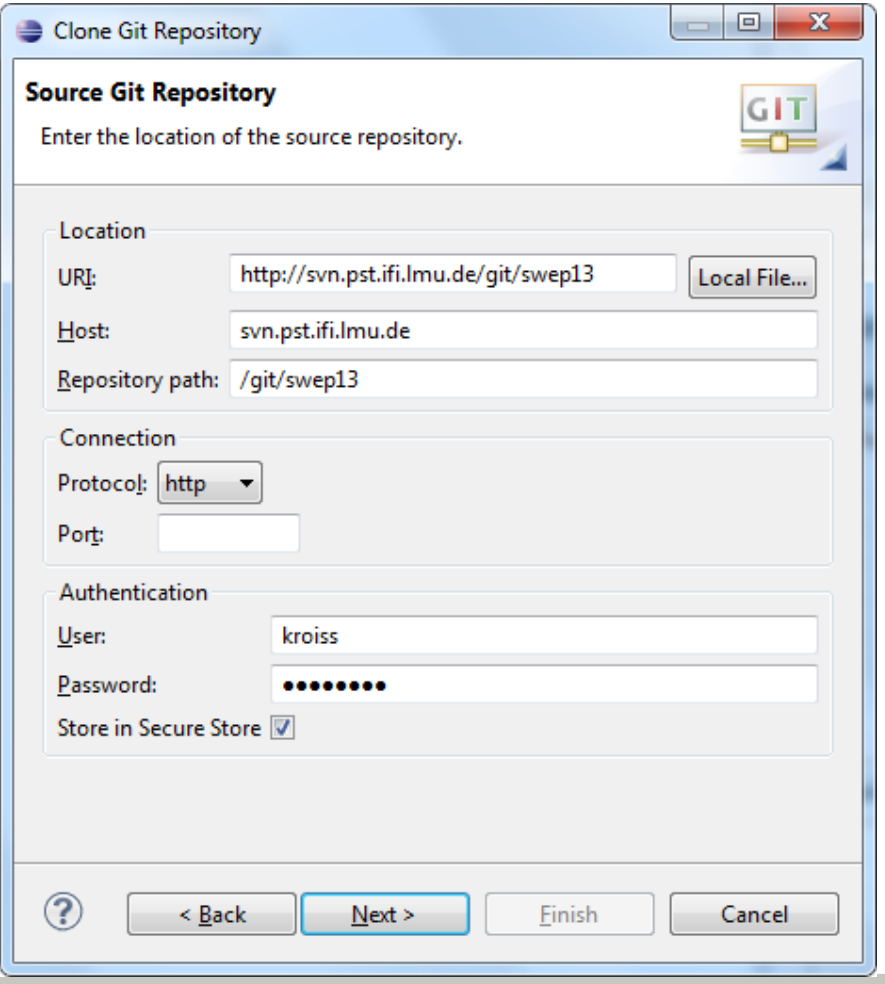


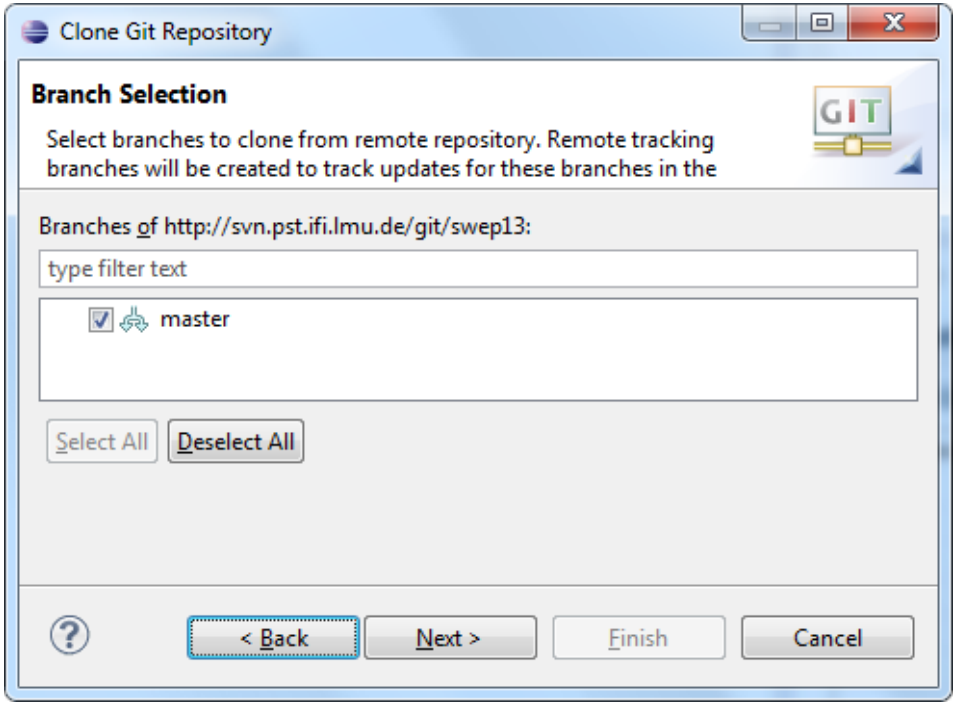
- Menu Team Synchronize Workspace (or change to Team Synchronizing perspective)

eGit from the start

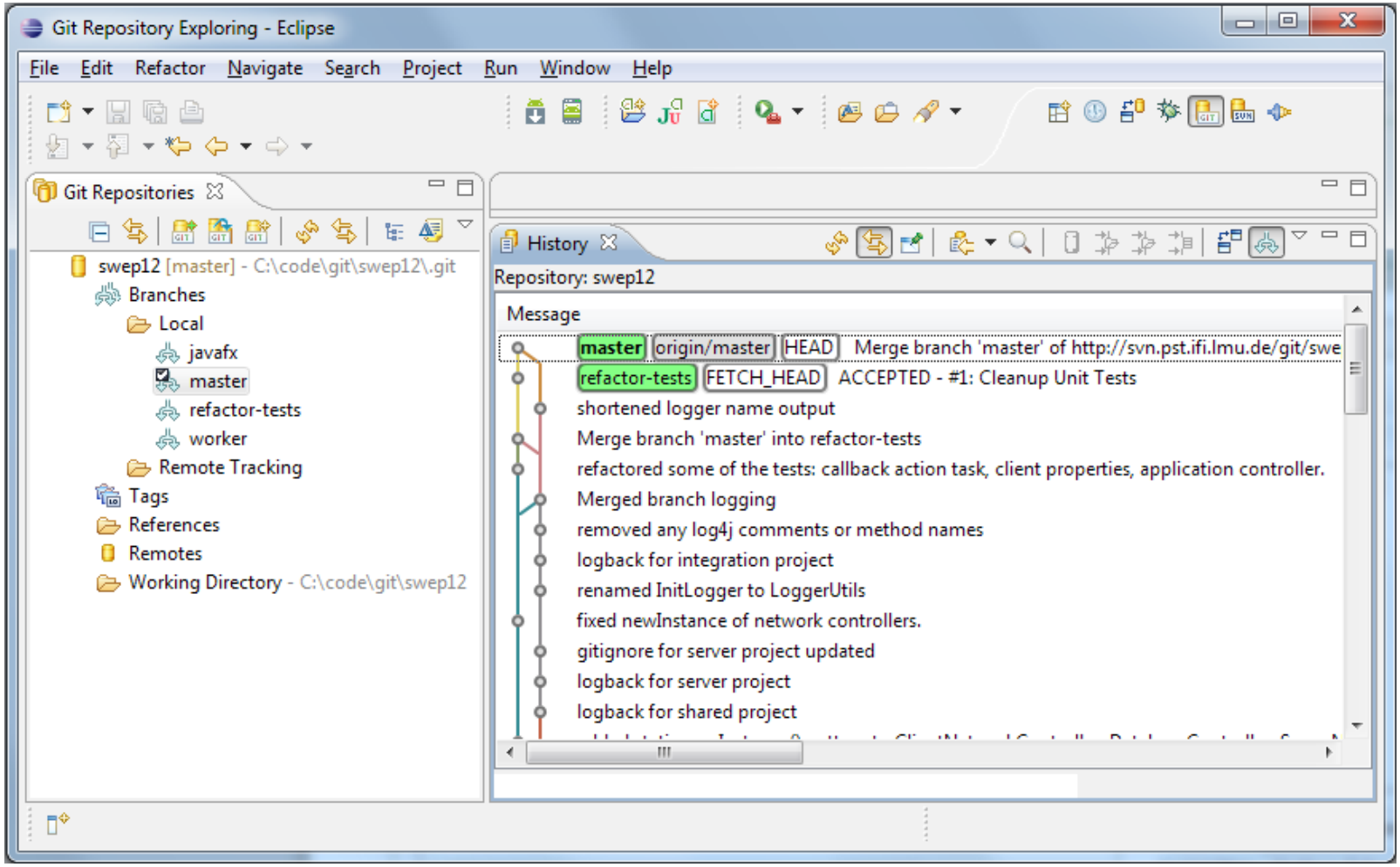


eGit: cloning

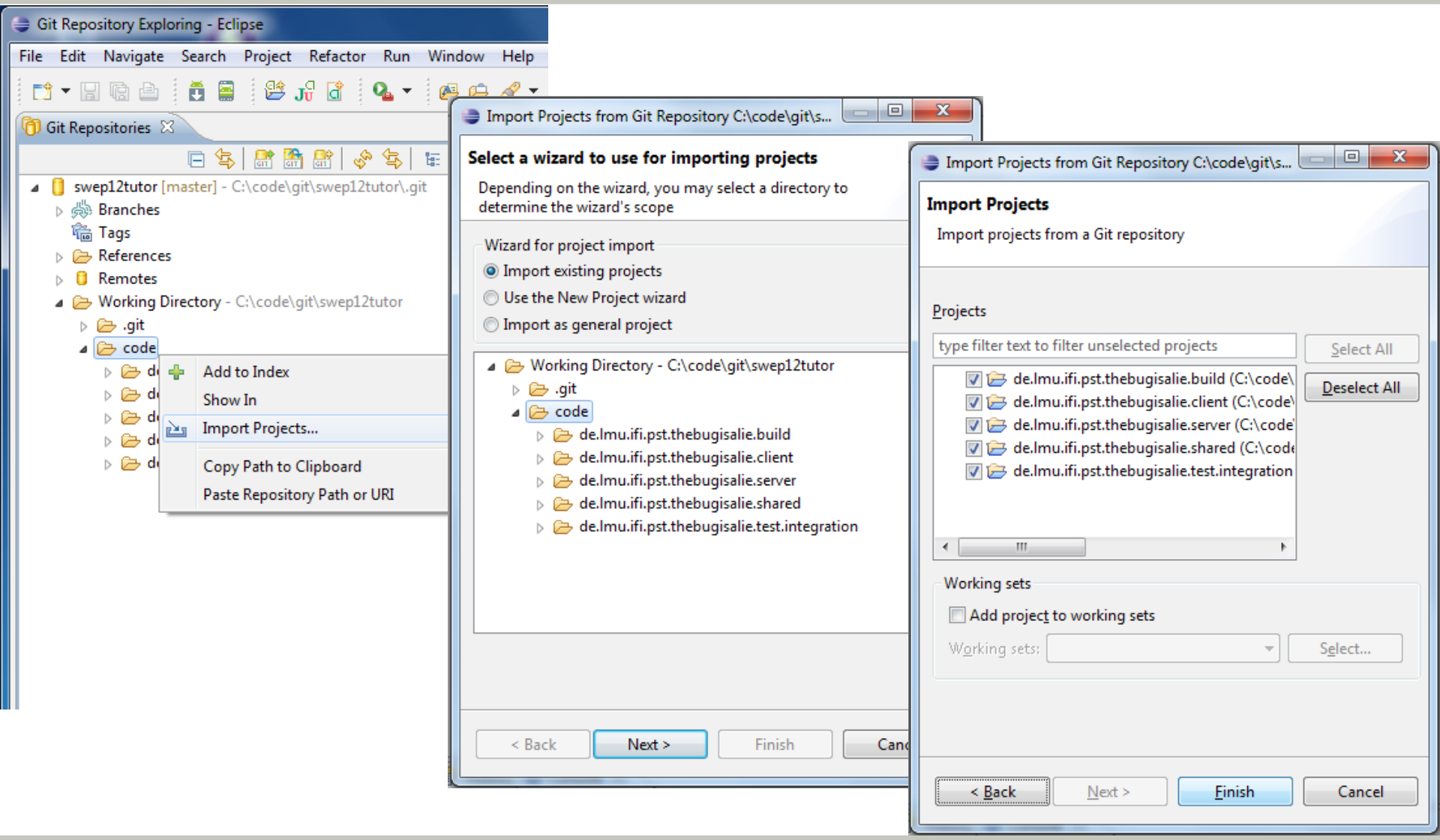




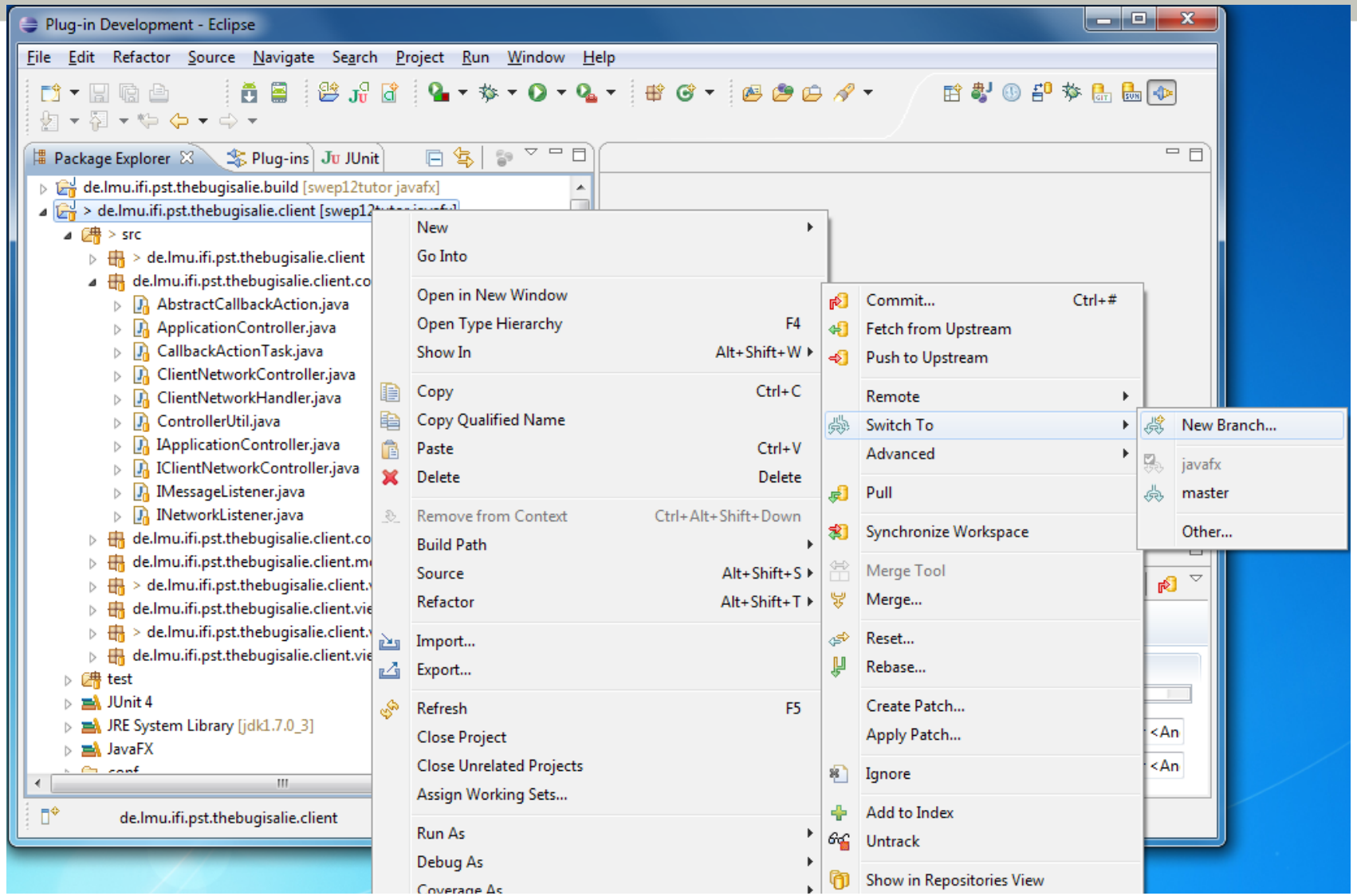
eGit: clone results



eGit: Import Projects

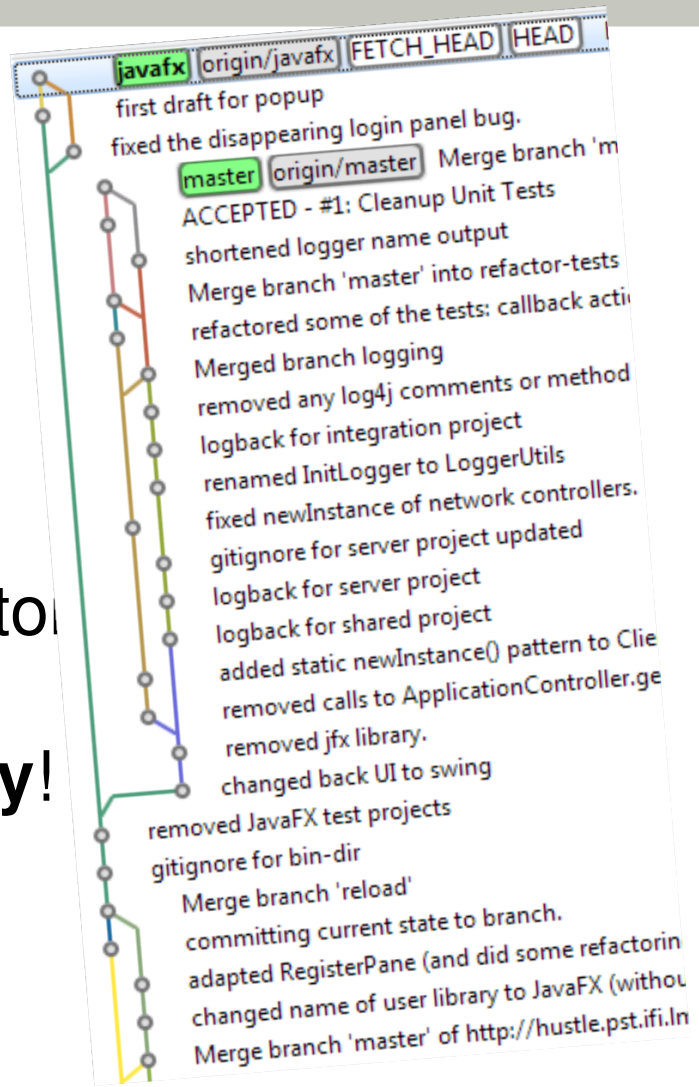


Creating a new branch

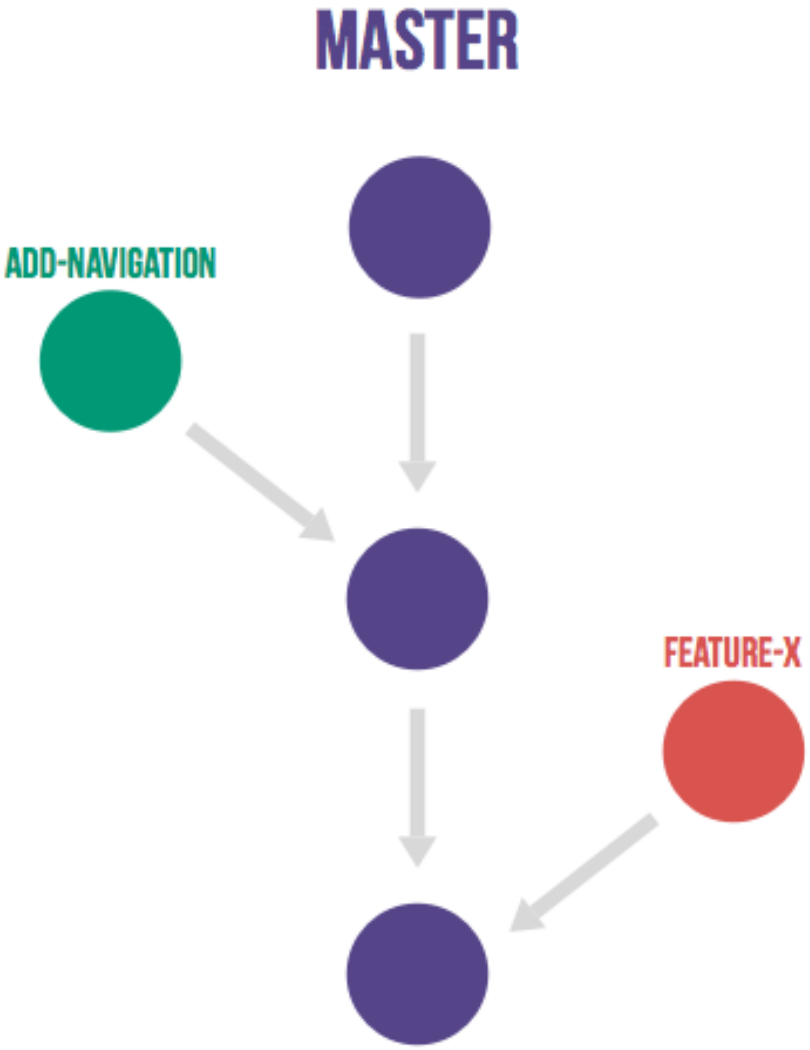


The art of branching

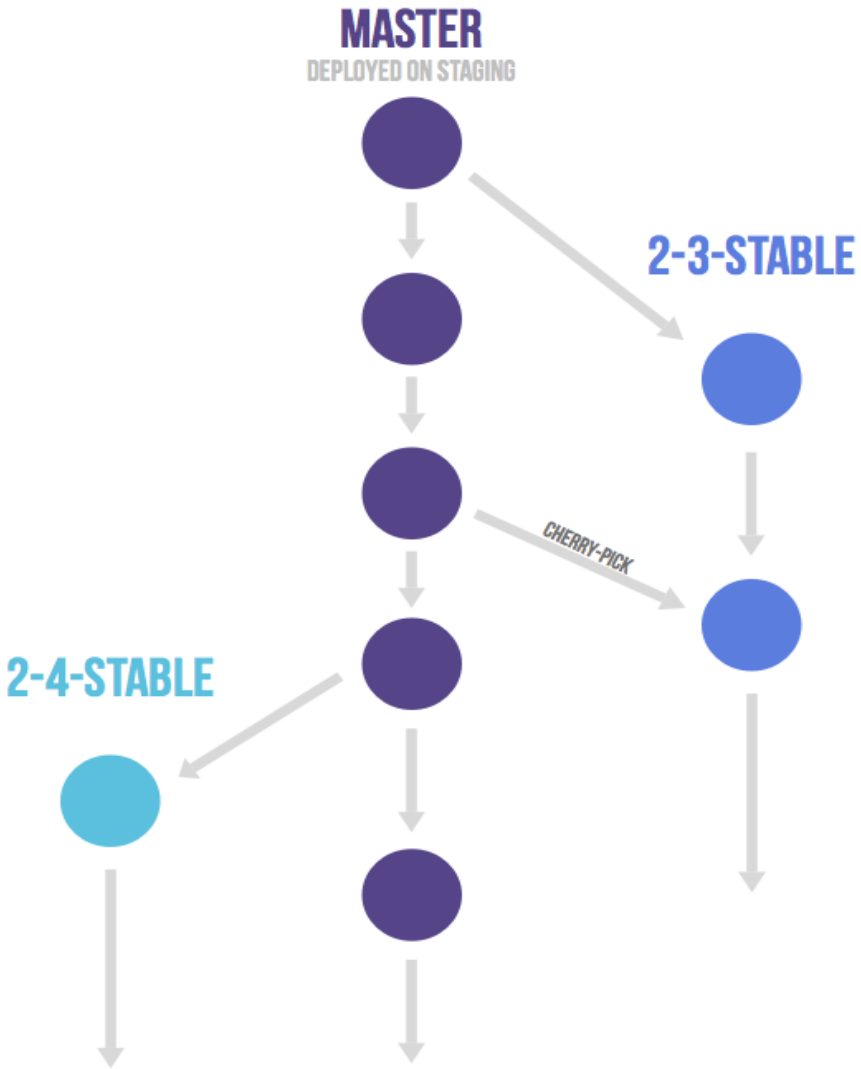
- When working on a user story
  - Create a **branch** for your story (= feature branch)
  - Work on the branch
  - **Merge** the branch into master
- Don't disconnect from the repositor (= team): fetch & merge master changes to your branch **regularly**!
- Read the Git **tutorials**



GitLab Flow: Feature Branches



GitLab Flow: Release Branches



Commit messages

- What does the following history tell you?

```
ea42b79 There were some bugs in the code!!!! Didn't know
2db0f12 fixed two build-breaking issues: the ant task co
147709f Tweaks to some files
22b25e0
7f96f57 polishing
59a2ed6 update
```

- And the following?

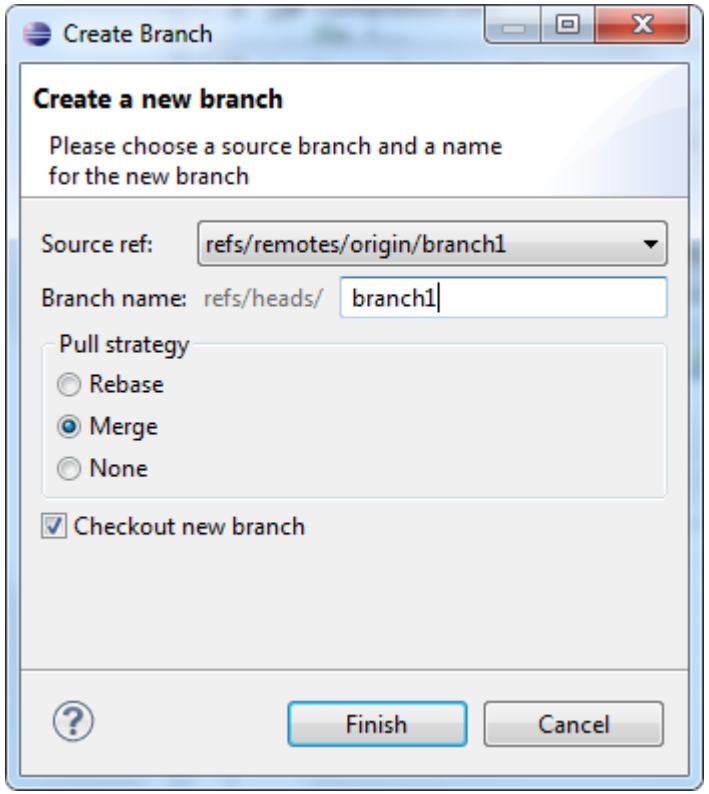
```
5ba3db6 Fix failing CompositePropertySourceTests
84564a0 Rework @PropertySource early parsing logic
e142fd1 Add tests for ImportSelector meta-data
887815f Update docbook dependency and generate epub
ac8326d Polish mockito usage
```

The art of writing commit messages

- Good commit messages are important
  - Separate subject from body with a blank line
  - Limit the subject line to 50 characters
  - Capitalize the subject line
  - Do not end the subject line with a period
  - Use the imperative mood in the subject line
  - Wrap the body at 72 characters
  - Use the body to explain *what* and *why* vs. *How*
- See http://chris.beams.io/posts/git-commit/ for more

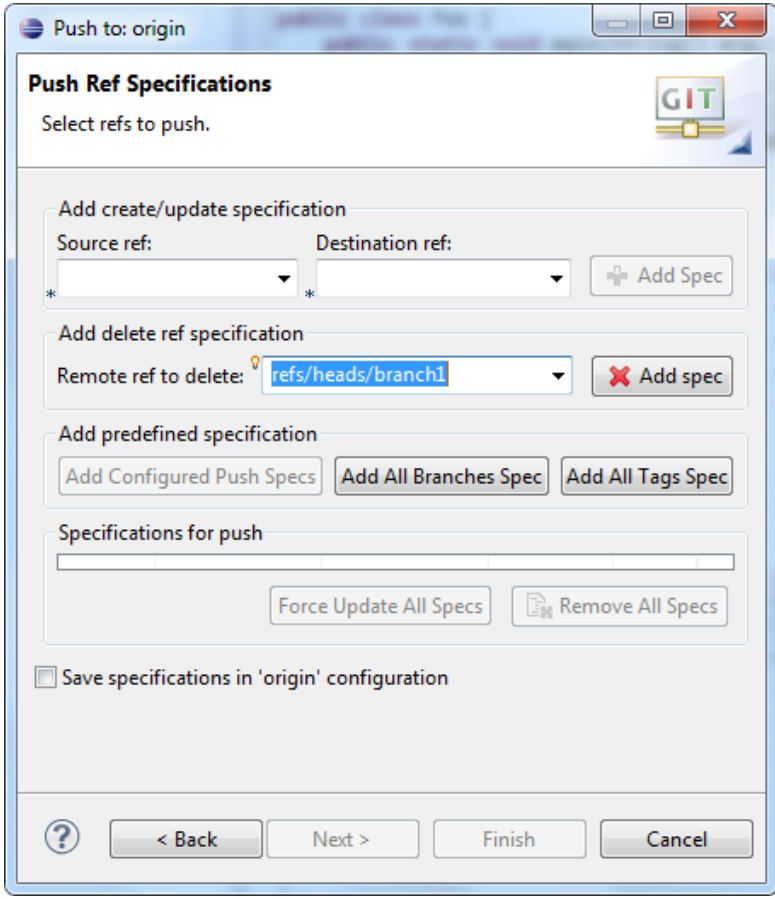
Tracking remote branches

- Instead of checking out a remote branch directly, it's better to create a local branch that **tracks** the remote branch.
- After that, you can pull and push directly to/from the original remote branch.
- If you push a branch that you created locally, the local branch starts tracking the remote copy.



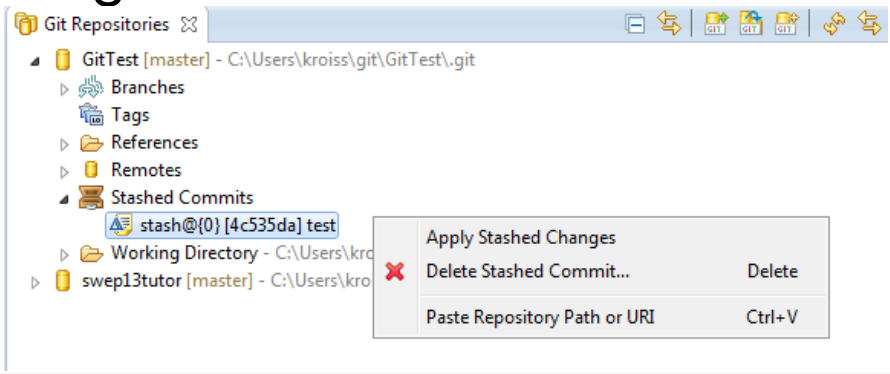
Deleting remote branches

- Deleting the remote reference doesn't do anything on the server.
- To remove a remote branch you have to push "nothing" to the branch:
  `git push origin --delete branch`
  or with the push dialog in eGit (see picture).
- **Deleting remote branches is most fun when others are tracking it J**



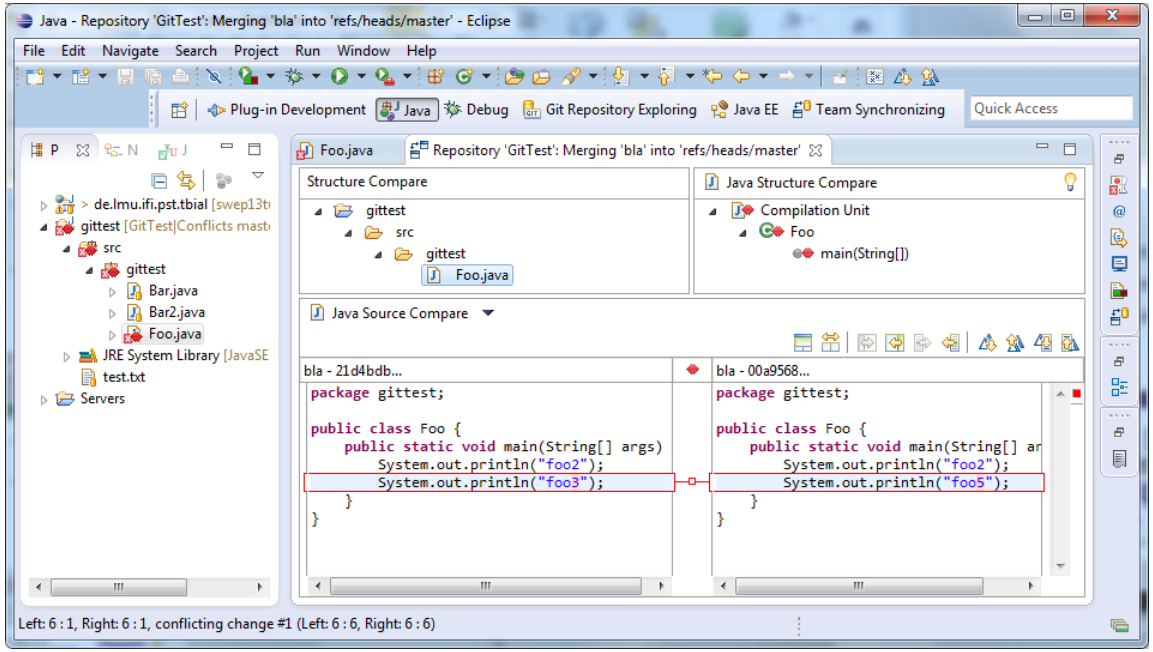
Switching between branches

- Git allows to switch between branches at any time
  - eGit: Menu Team Switch To
- If you have uncommitted changes, you have to either commit them first or **stash your changes**
  - eGit: Menu in Git Repository View Stash Changes
  - Those changes get “stored" and you can switch branches
  - You can restore stashed changes in the Git Repository View



Resolving conflicts

- Merging branches may lead to conflicts. When this happens you end up in a "merging-state" where you have to resolve the conflicts.
- **Resolve conflicts**
  - Menu Team Merge Tool
  - Or/and edit manually
- **Then:**
  - Add to index
  - Commit



Other (e)Git hints

- Undoing changes:
  - Use Menu Replace (HEAD|Commit|…) to replace files/folders with previous versions from the repository
- If you're used to SVN: **don't forget to push**
  - Commit only writes to your local repository. Use "commit and push" in Commit dialog or push explicitly.
- Resolving conflicts without editing (e.g. for binary files)
  - `git checkout --ours <path>`
  - `git checkout --theirs <path>`

Other (e)Git hints

- Use own repository (e.g. GitHub/GitLab) to experiment!
- Install full Git distribution (www.git-scm.com) and (optionally) GUI like TortoiseGit, etc.
- Have a look at the git command line

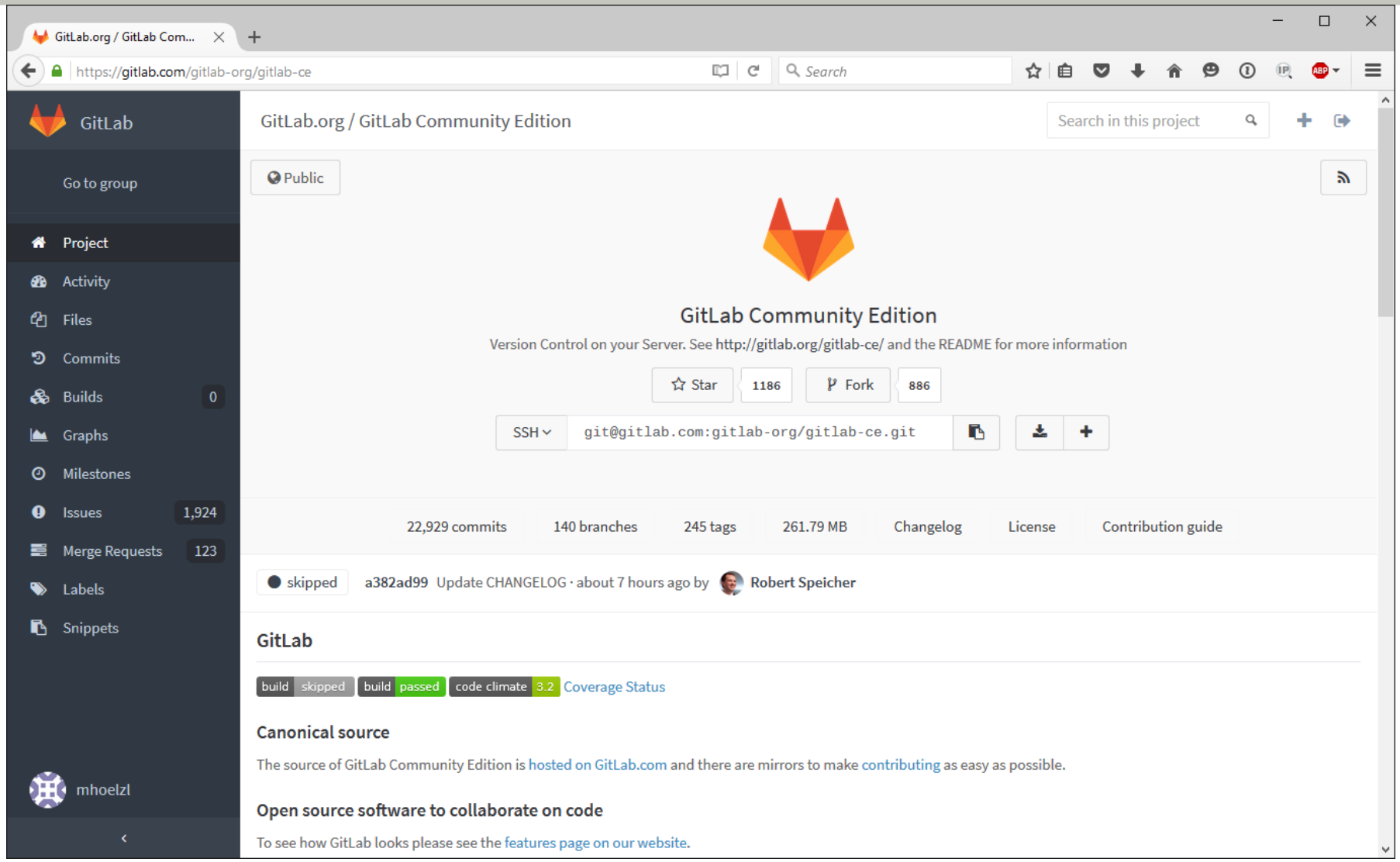
Some words on collaboration…

Merging a user story into master means **integration**

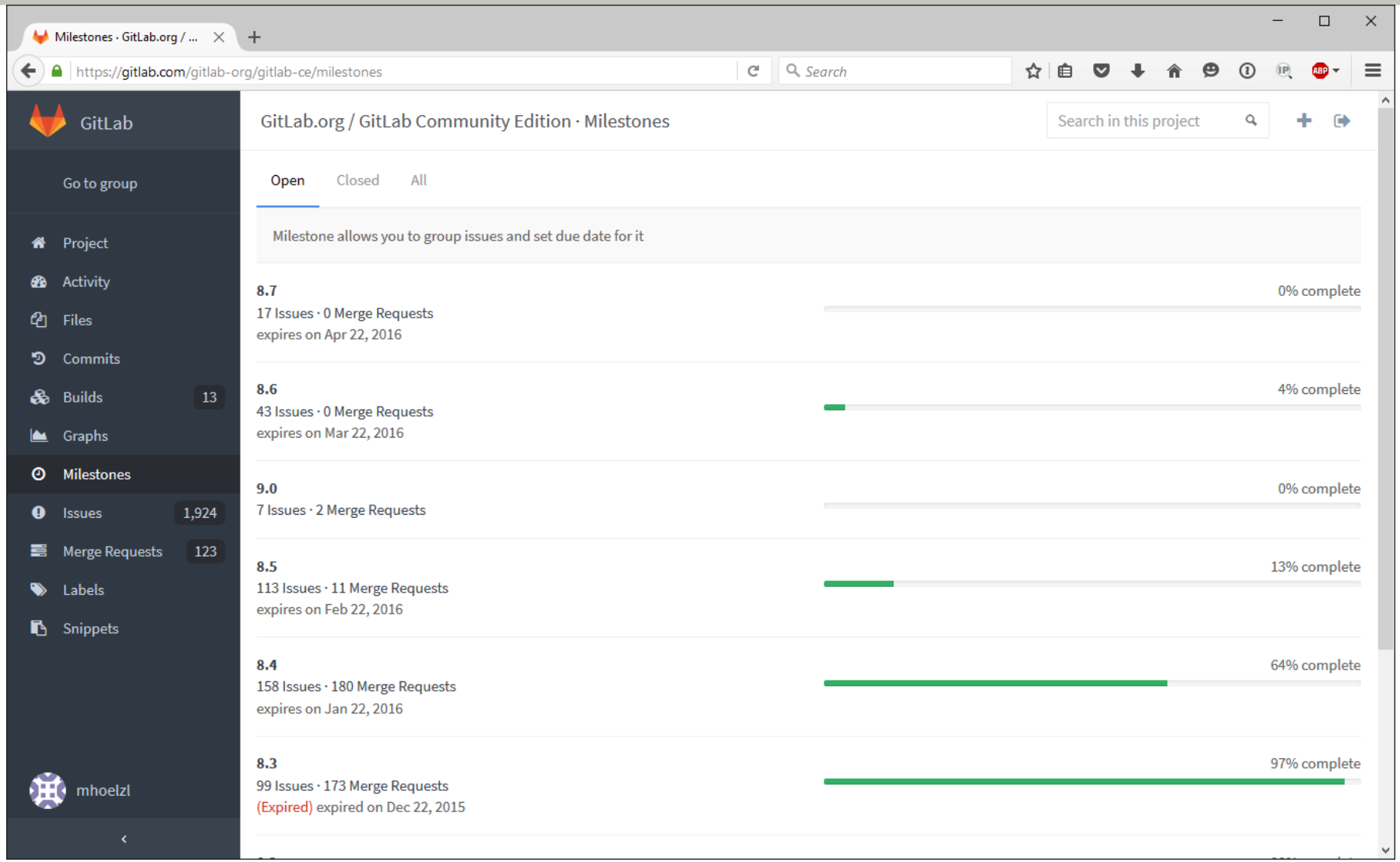
- Conflicts must be carefully **resolved**
- The whole codebase must **compile**
- All tests must **pass**:
  unit tests, integration tests, system tests

Integrate early and opportunistically,
It will not get easier if you wait!

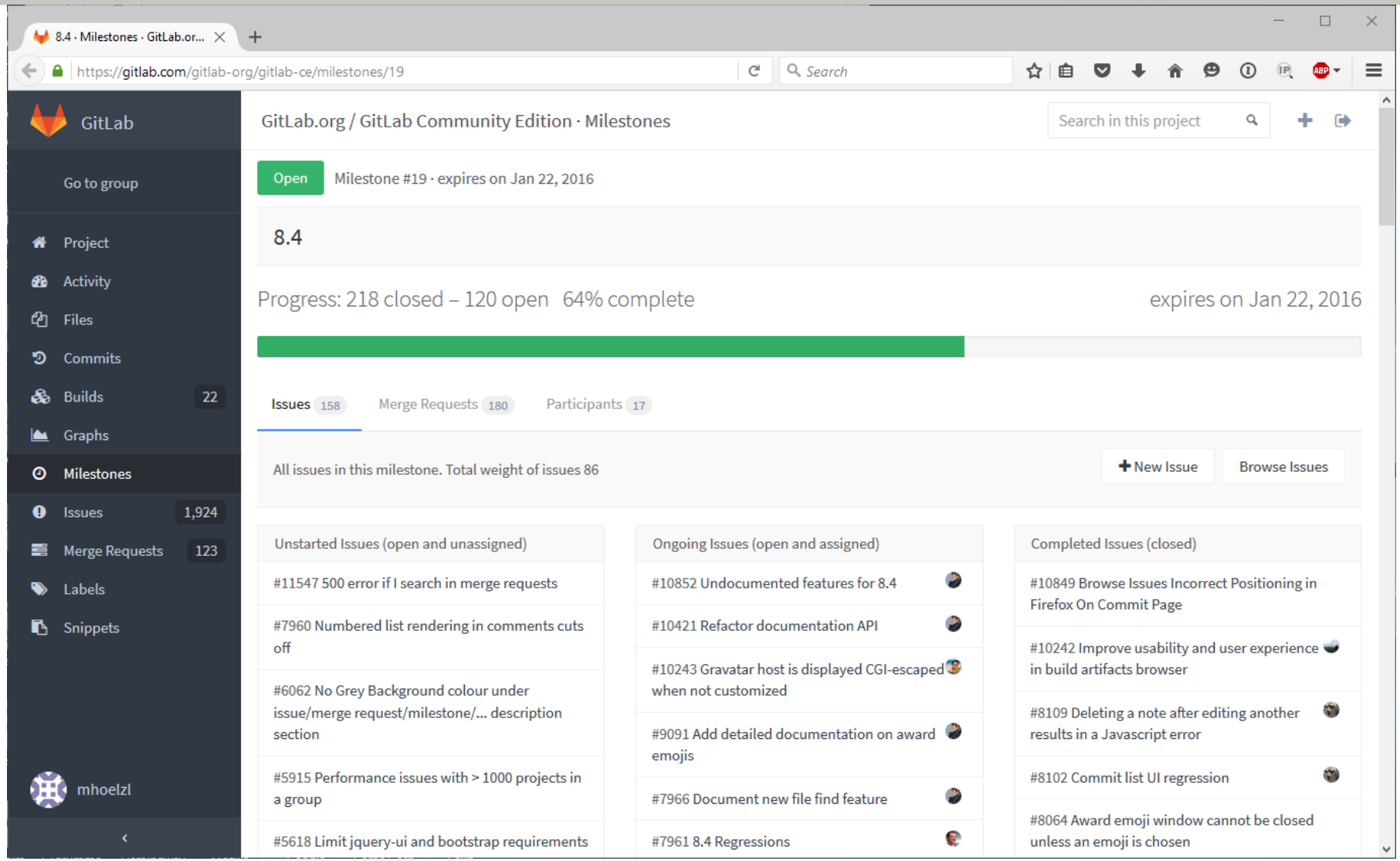
GitLab: Project Overview



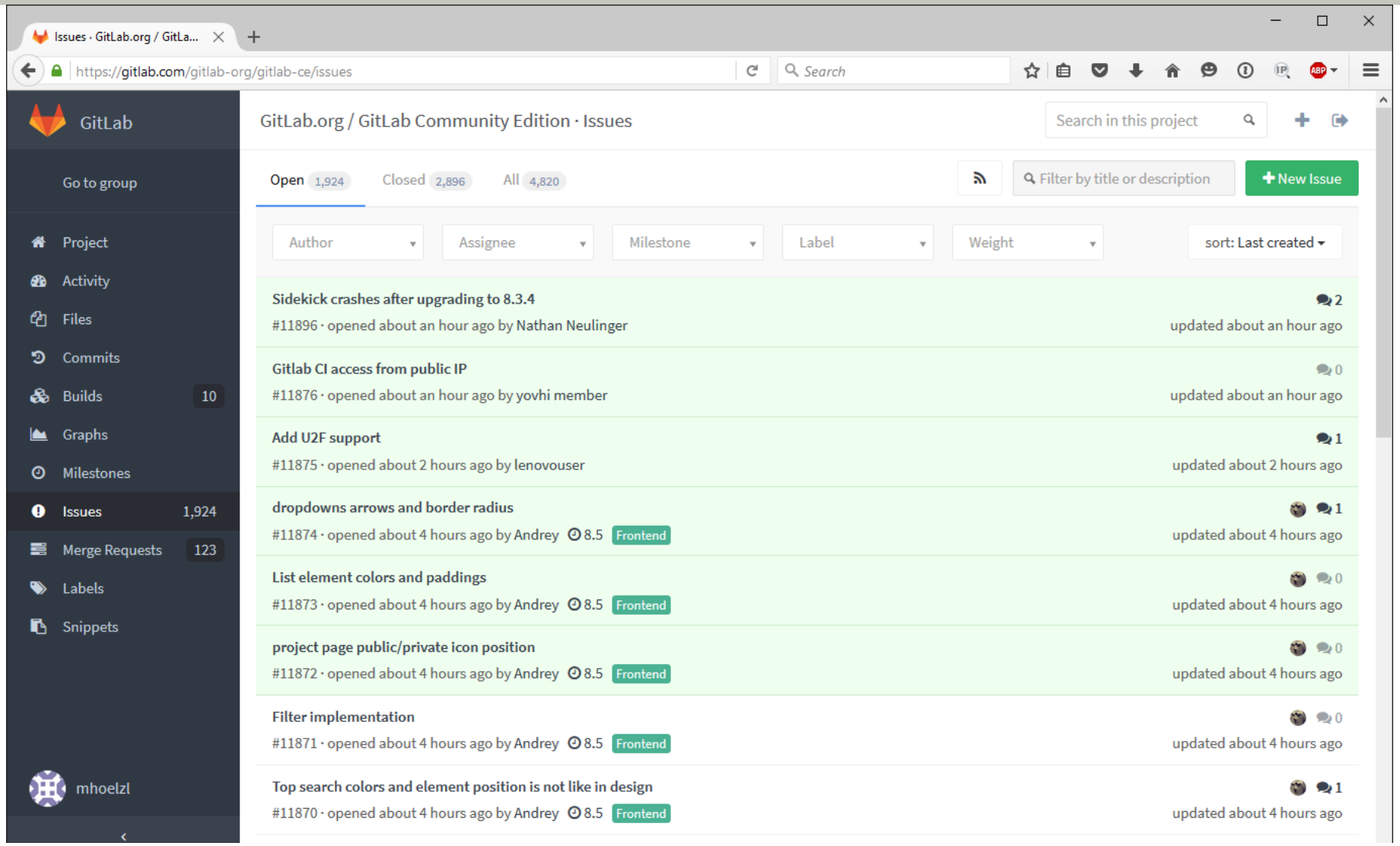
GitLab: List of Milestones



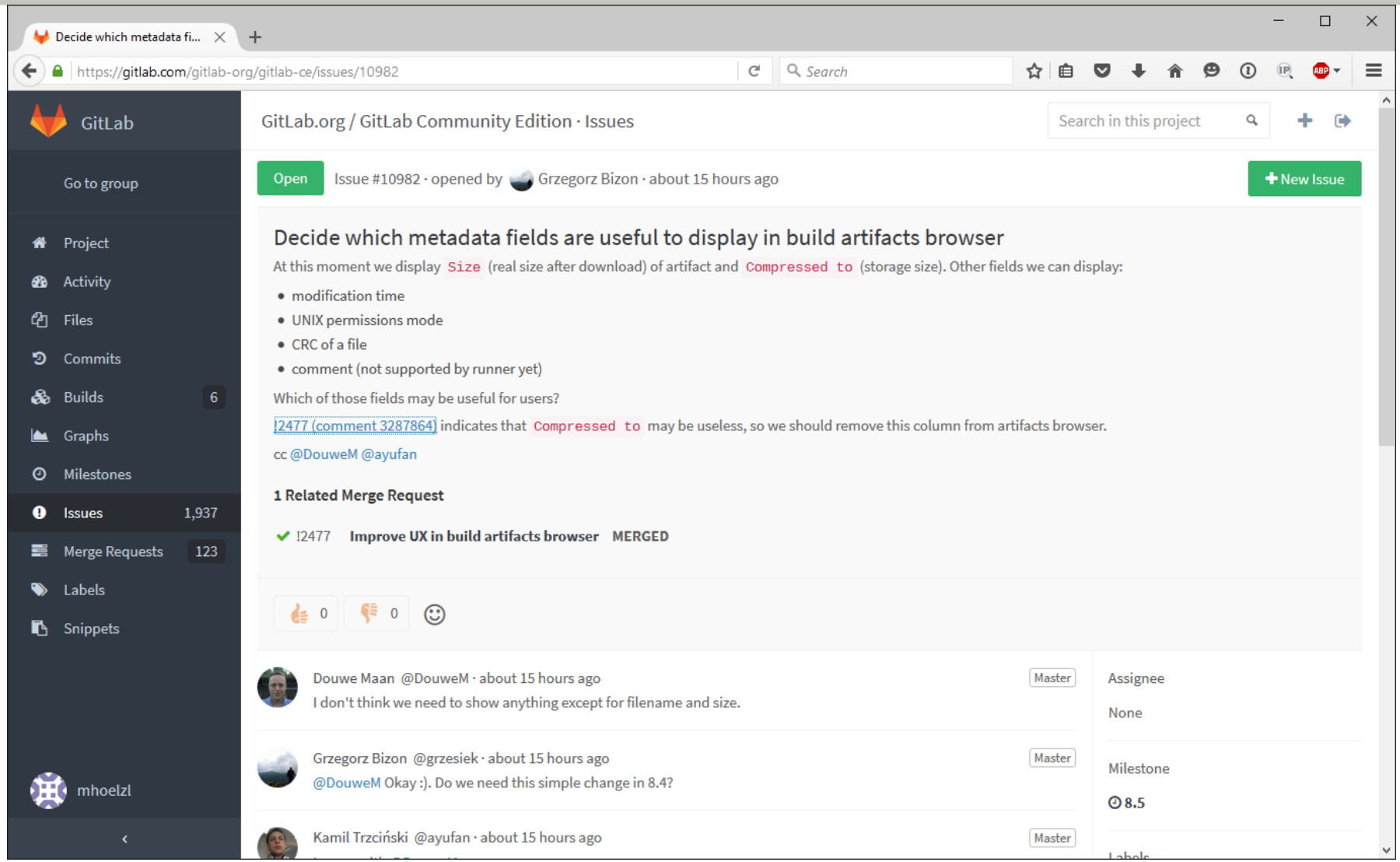
GitLab: Milestone



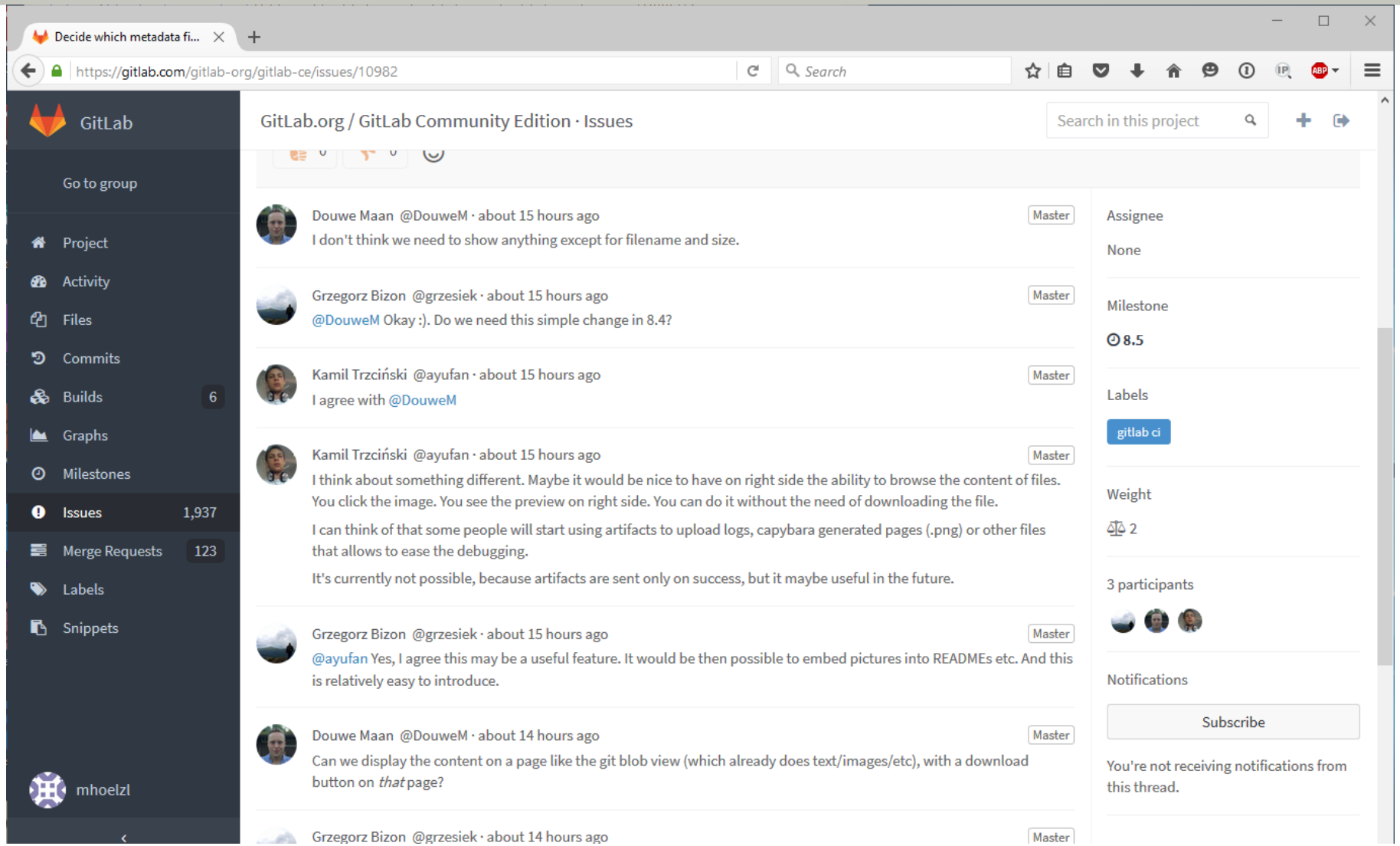
GitLab: Issues



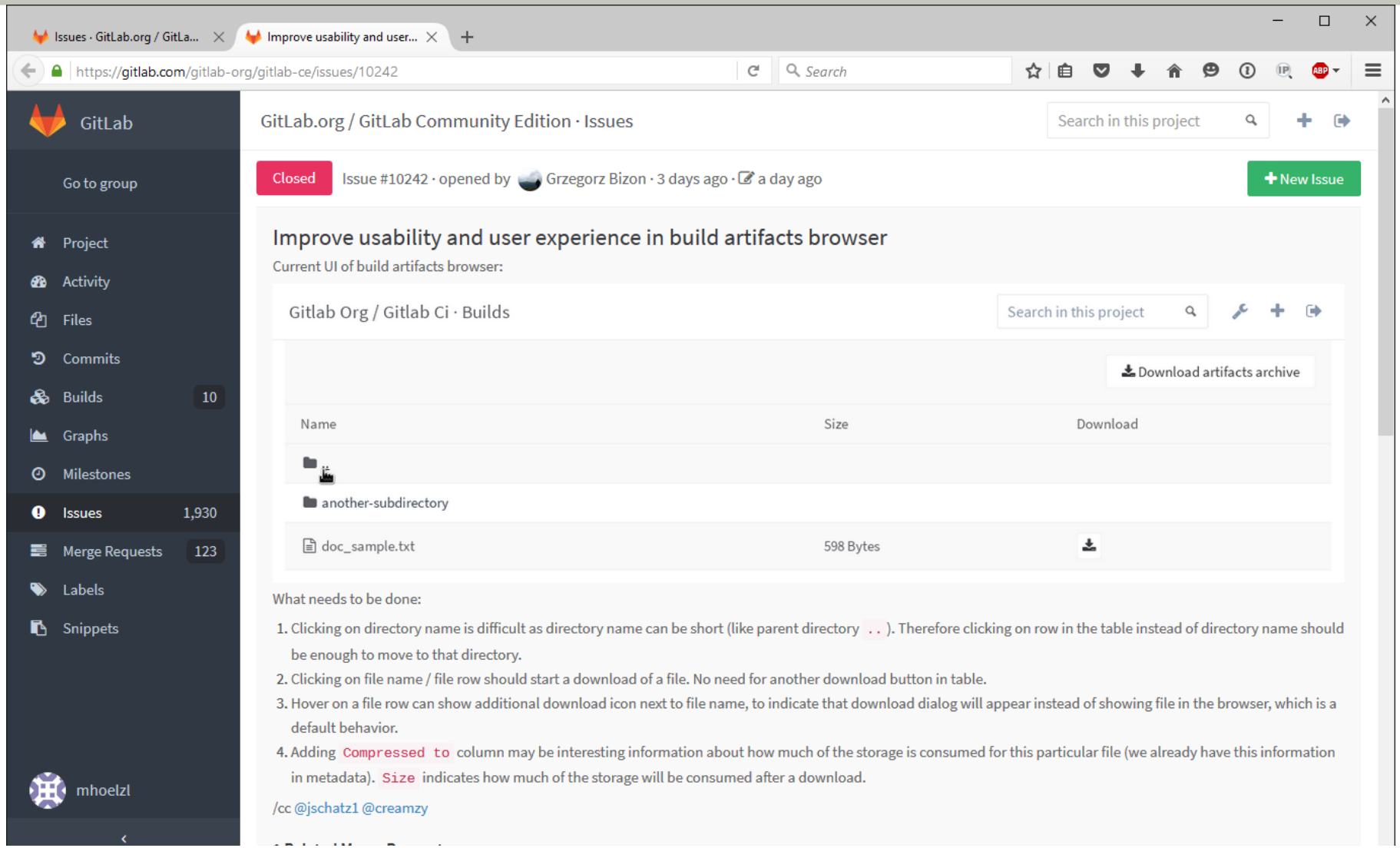
GitLab: Issue



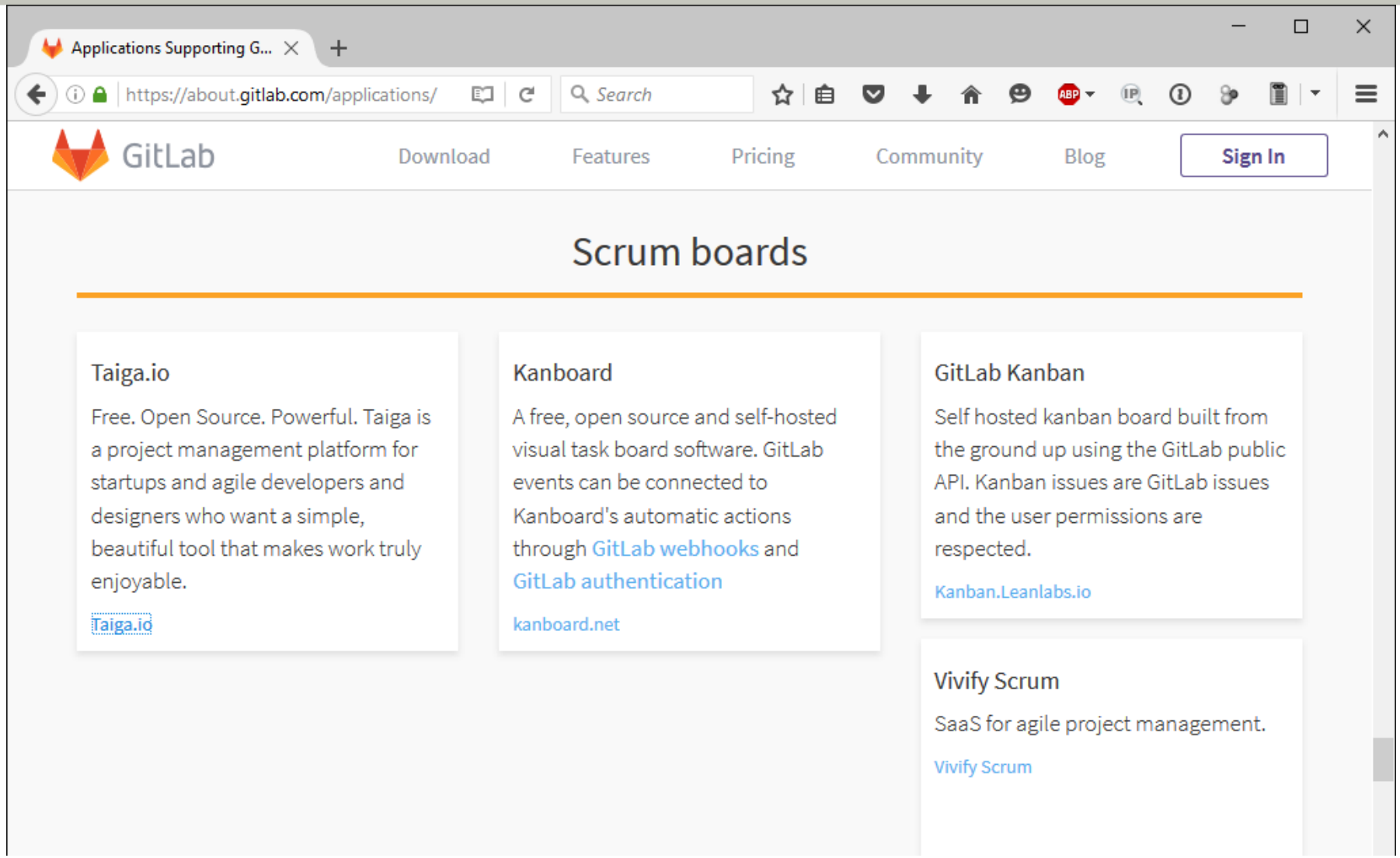
GitLab: Issue



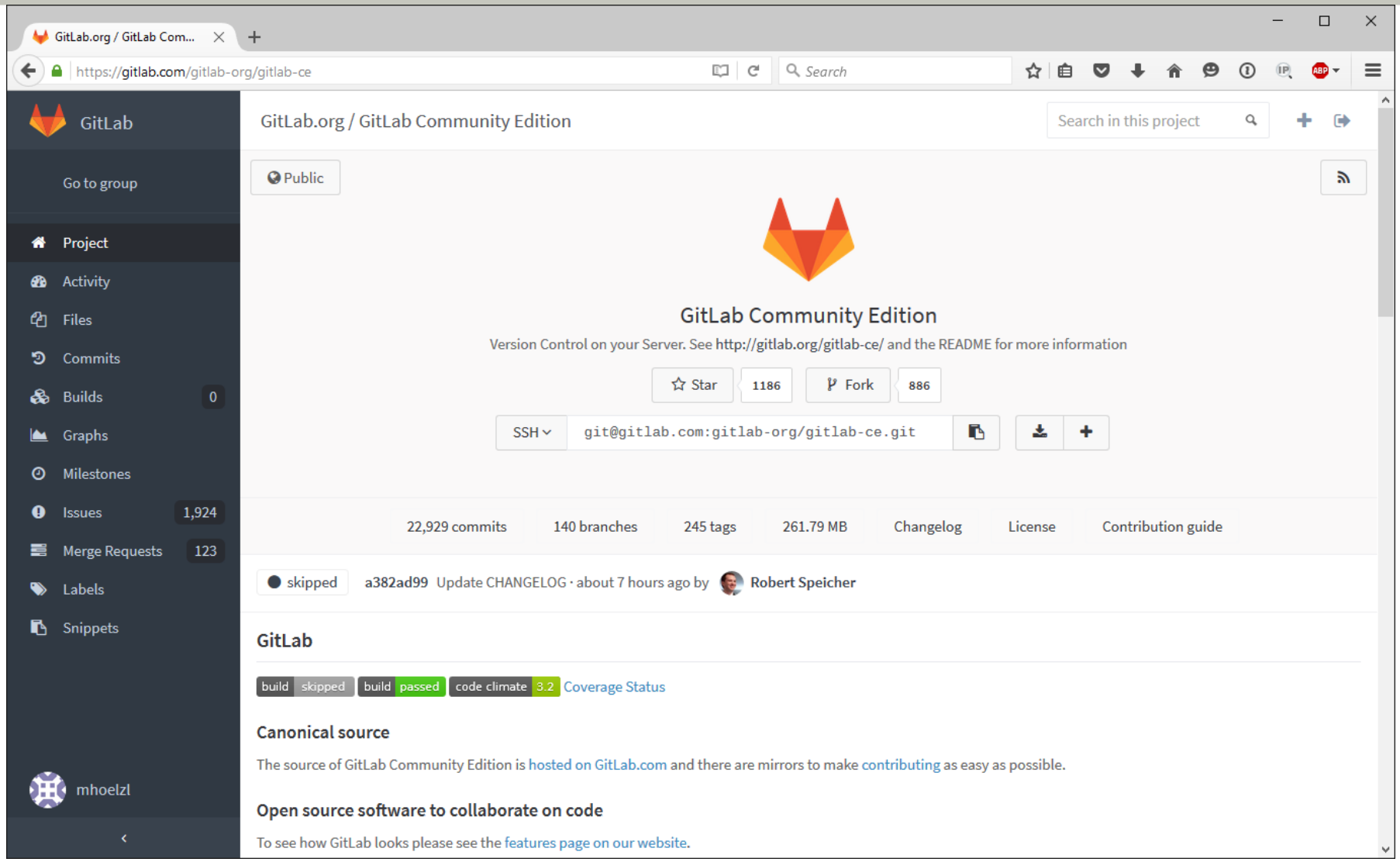
GitLab: Issue



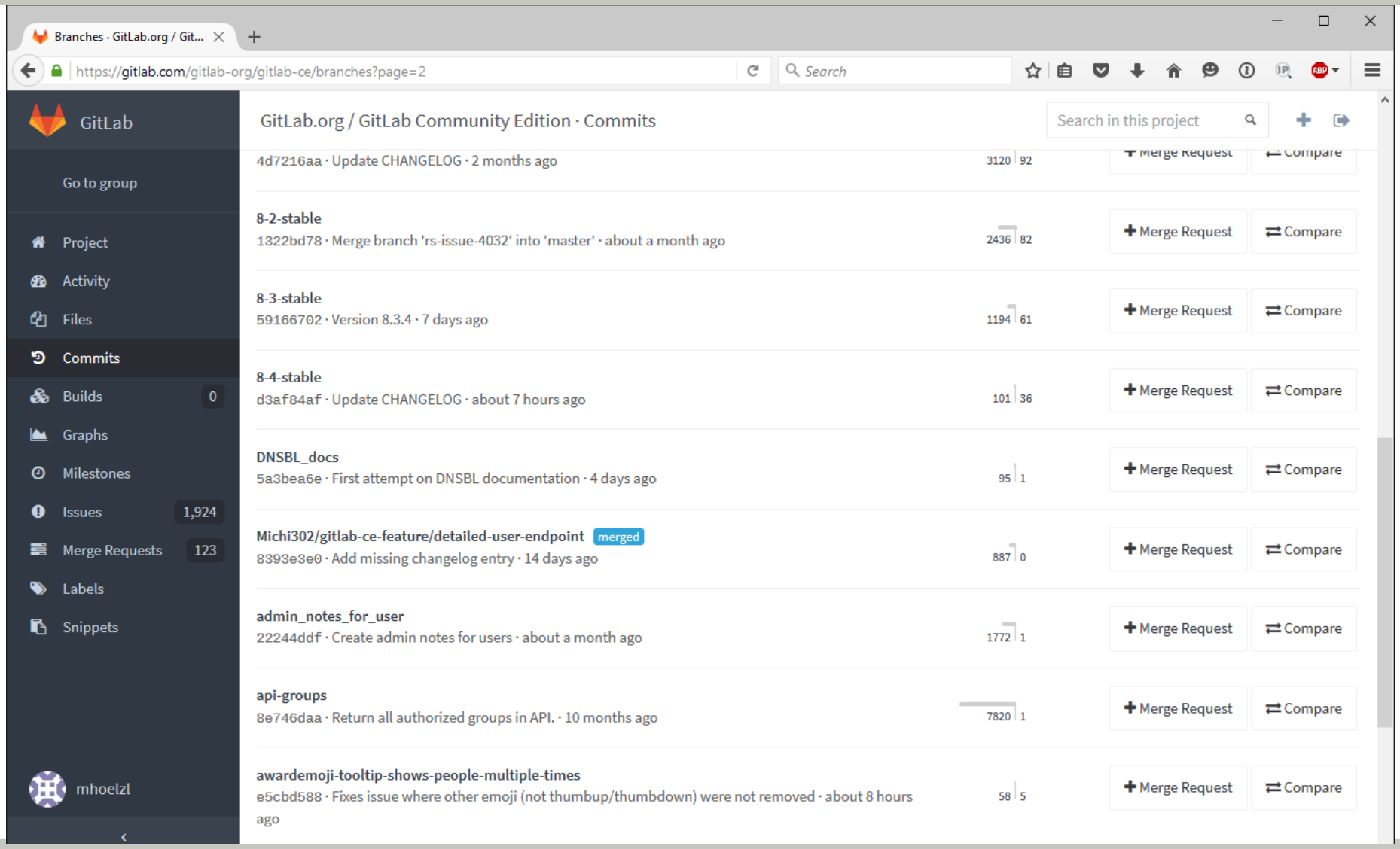
GitLab: Third-Party Scrum Support



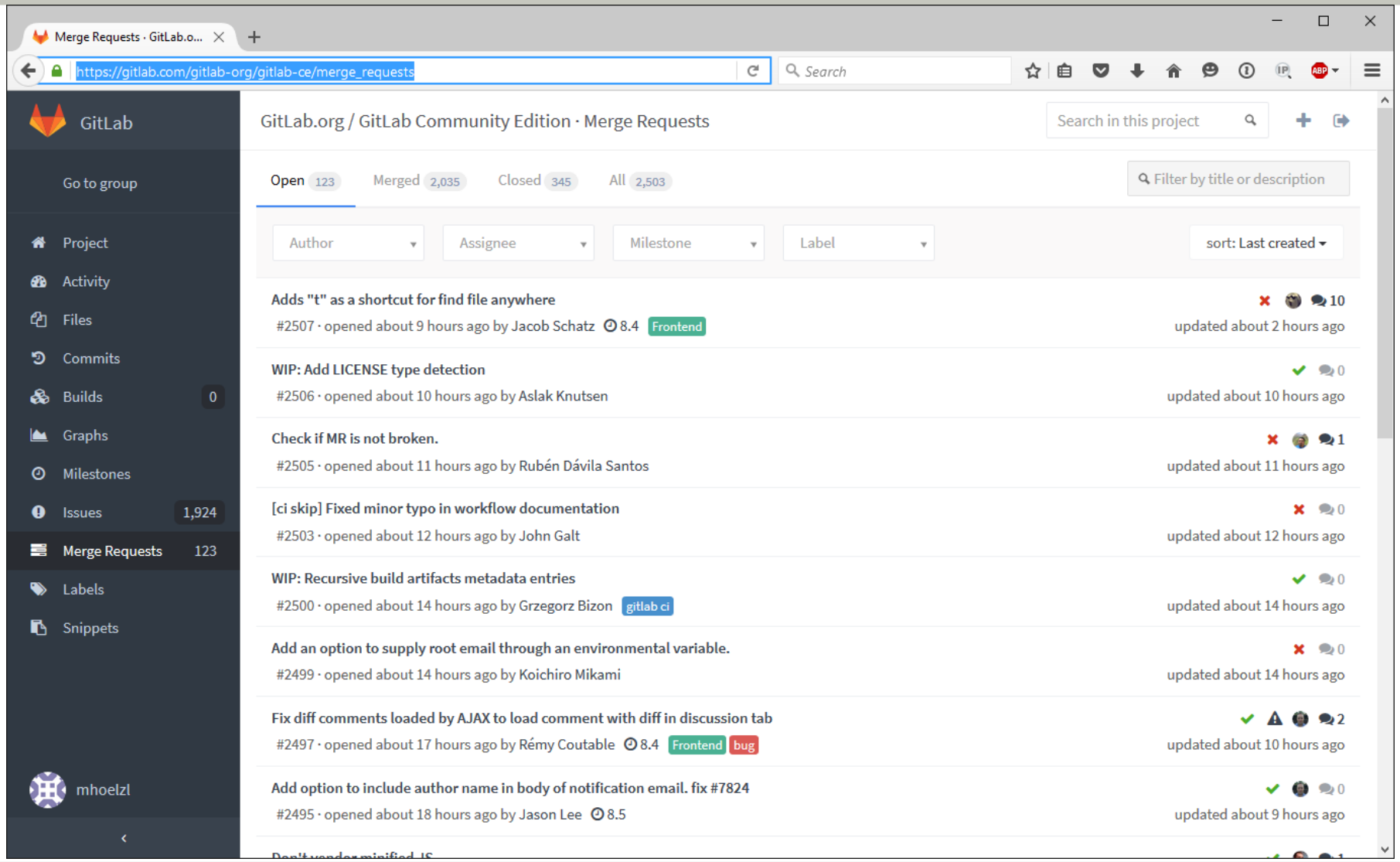
GitLab: Branches



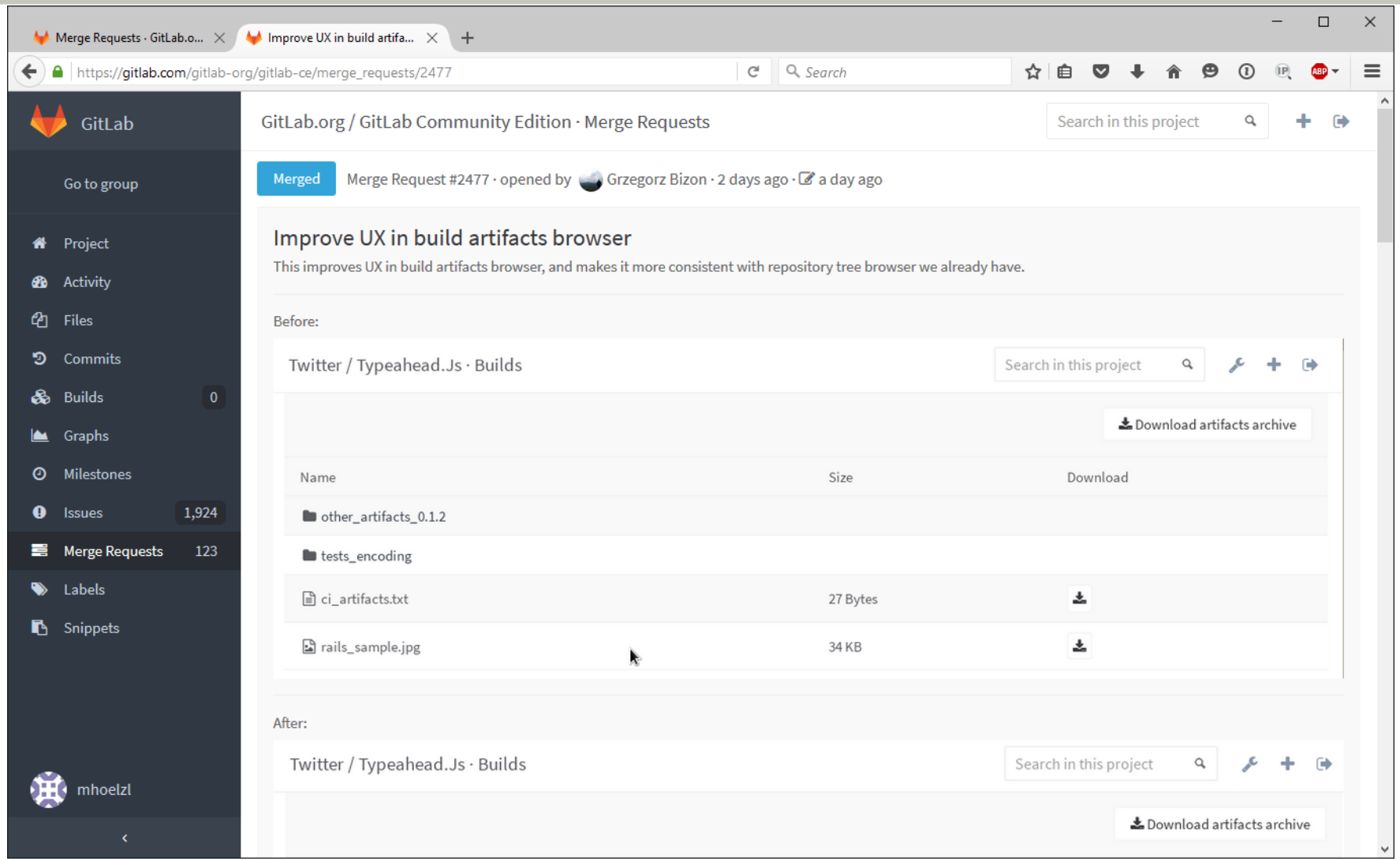
GitLab: Branches



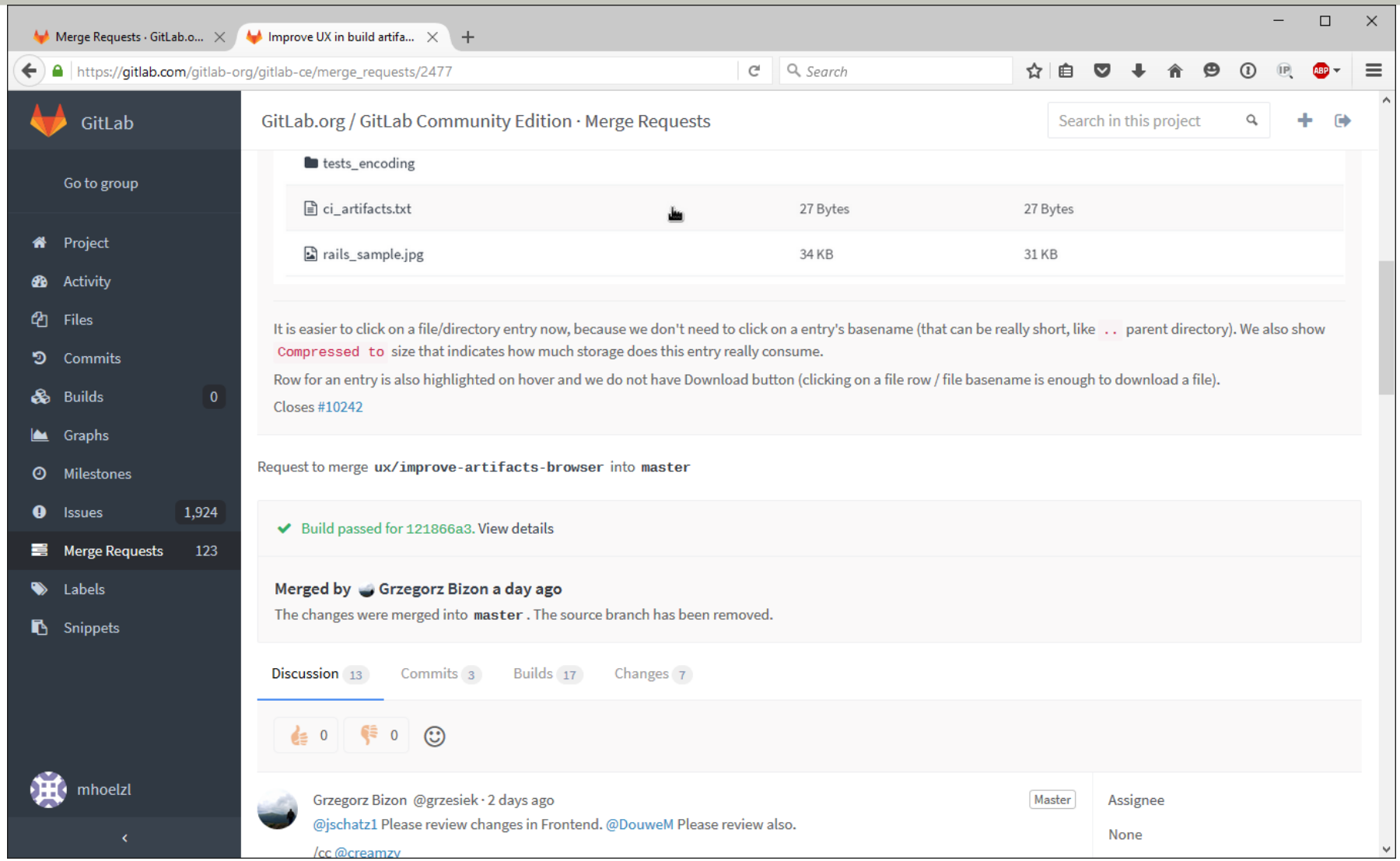
GitLab: Merge Requests



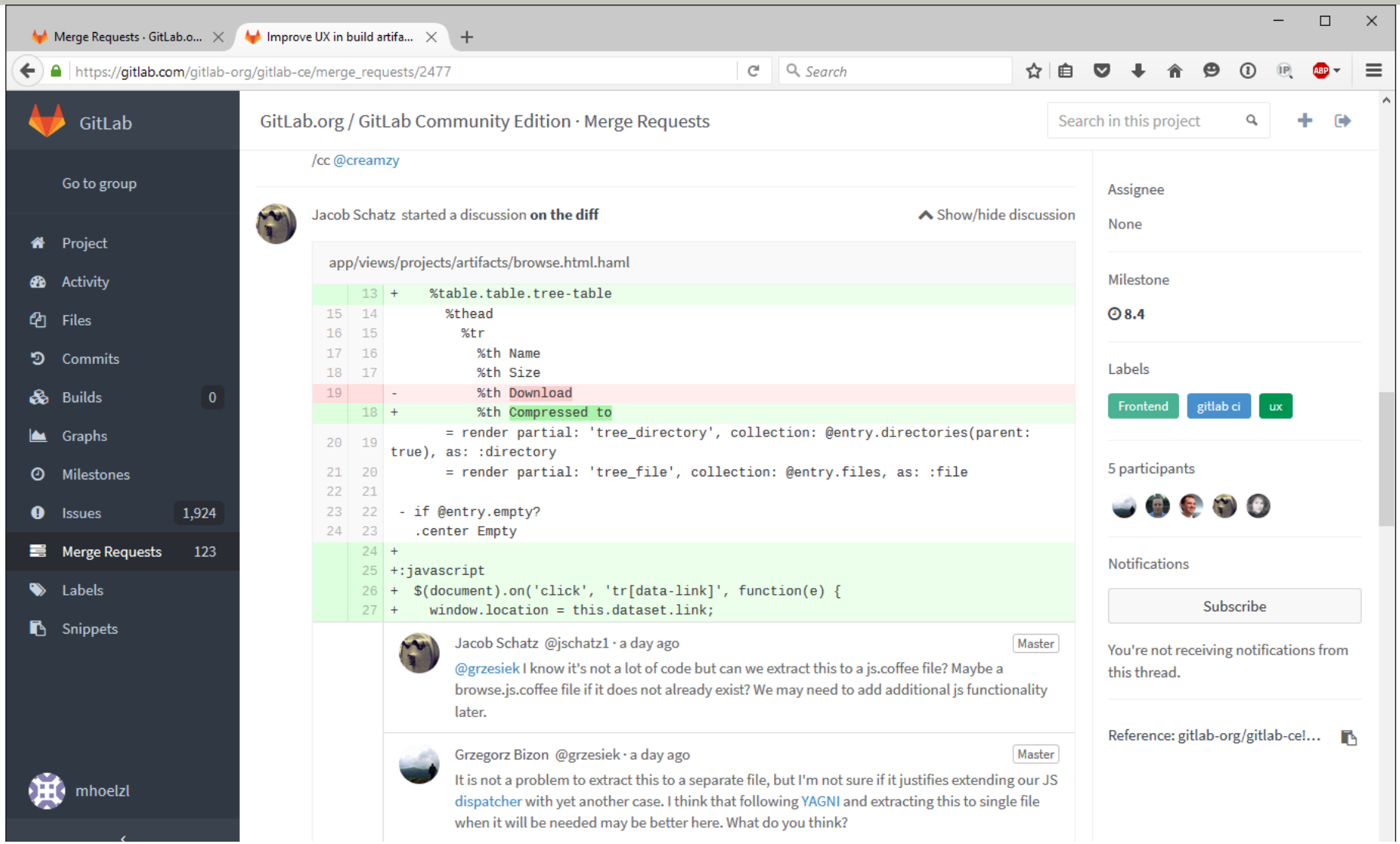
GitLab: Merge Request



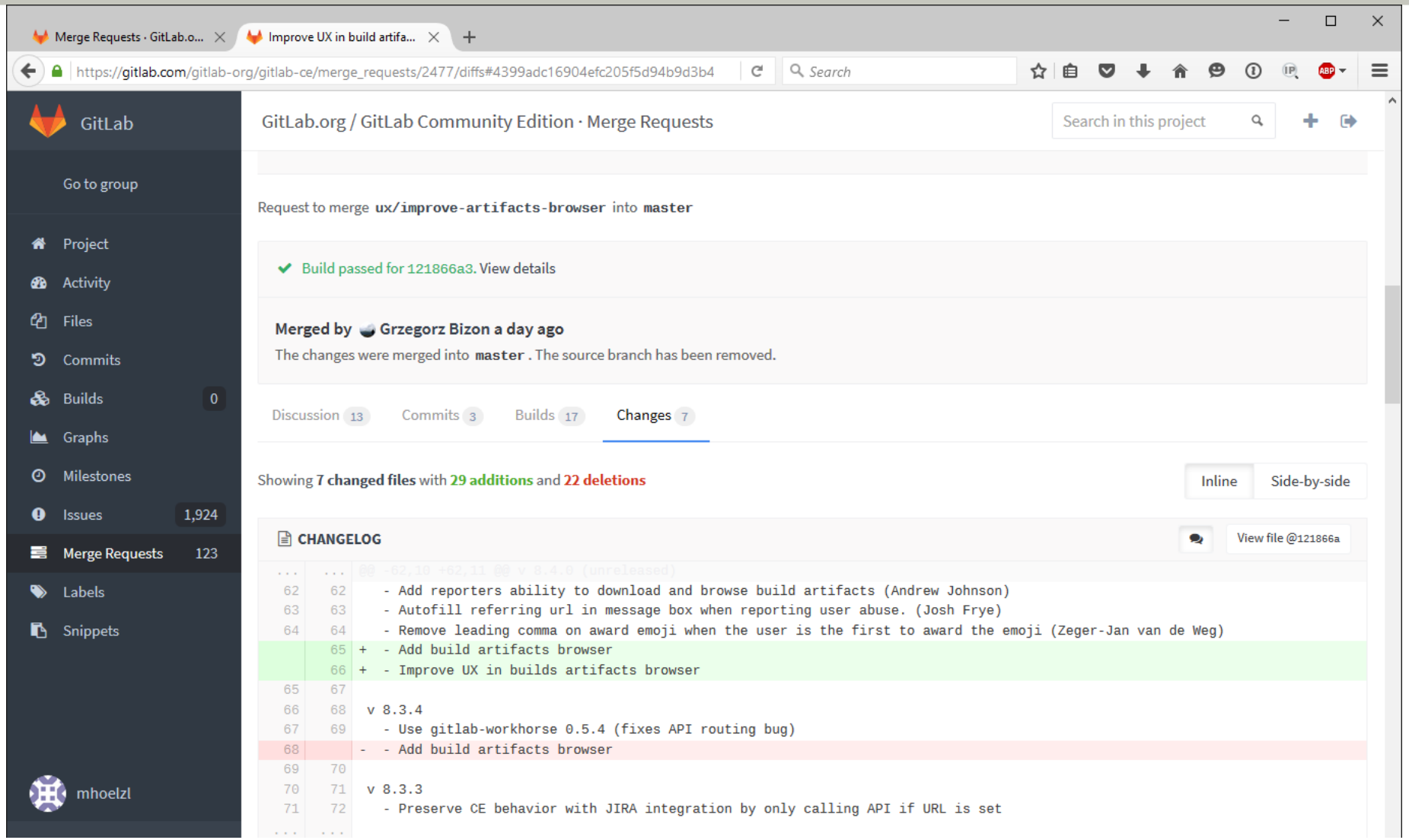
GitLab: Merge Request



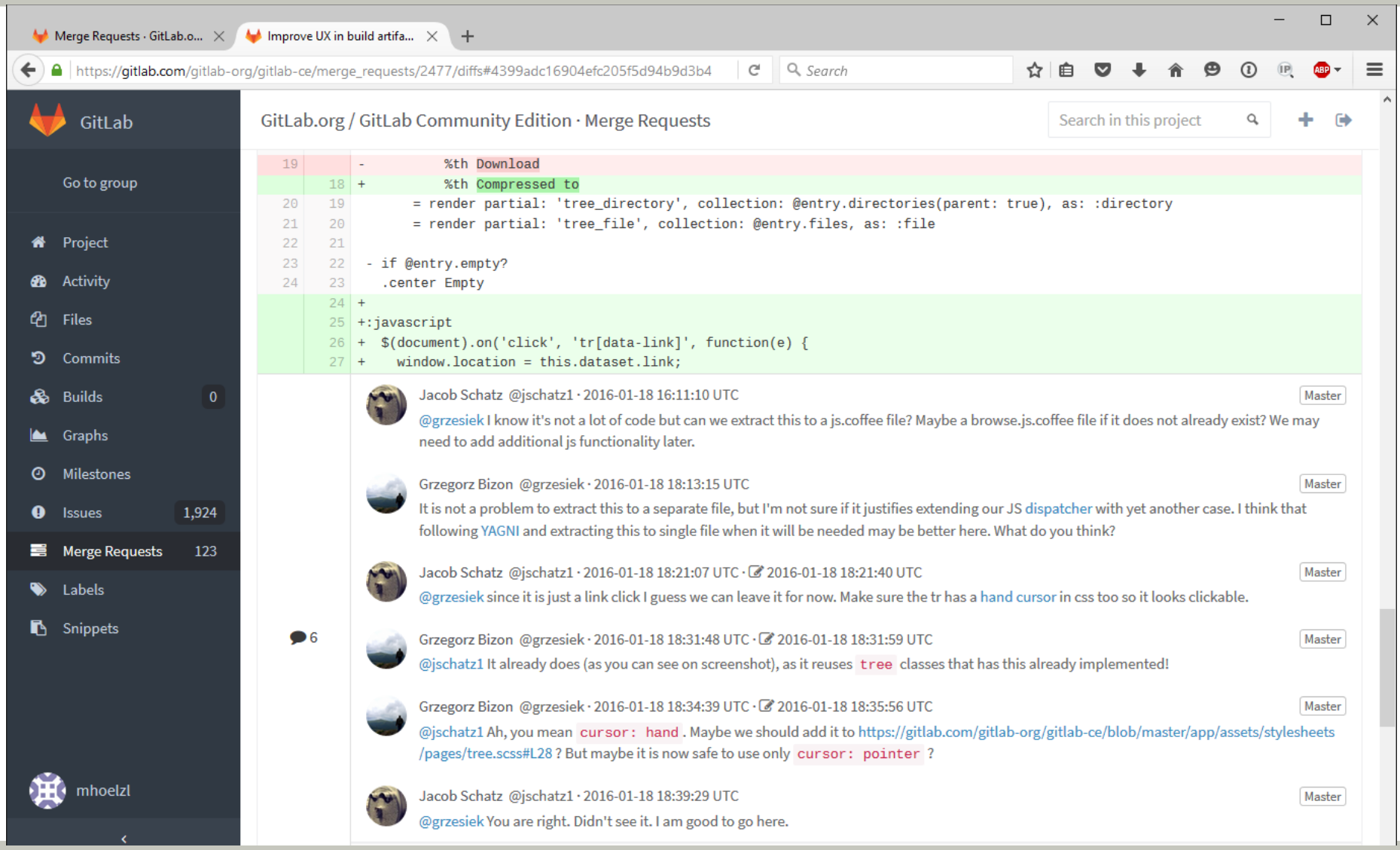
GitLab: Merge Request



GitLab: Merge Request



GitLab: Merge Request



Summary

Summary: Development Environment

- Eclipse
  - The IDE for our project
- Git
  - Distributed version control system
  - Built-in branching facilities

Technology Background
Development environment, Skeleton and Libraries

Slides by Prof. Dr. Matthias Hölzl
(based on material from Dr. Andreas Schröder)

Outline

Lecture 1

I. Eclipse
II. Git

Lecture 2

IV. Skeleton Overview
V. Libraries Overview
VI. Game Rules and Coding Task

Part I. Eclipse

Part I: Goals

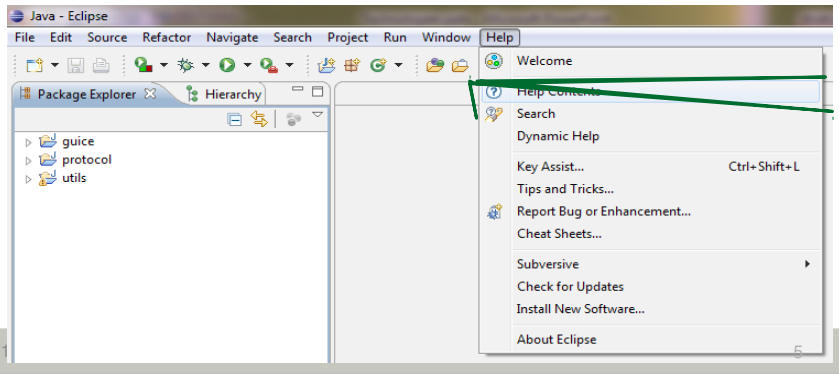
Learning Target

- Recognize the power of Eclipse
- Identify what you did not know yet
- Know where to find tutorials and help
- Being able to set up the Eclipse IDE for the lab

Eclipse

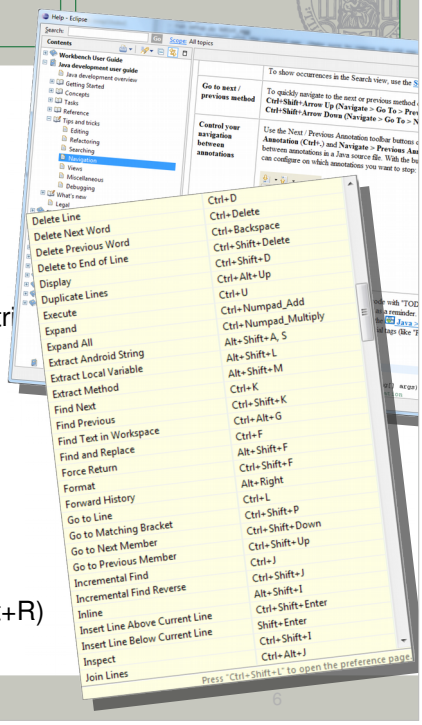
Eclipse is far more than a Java editor

- Code navigation and exploration
- Refactoring
- Background compilation
- Customizable build system
- Extensibile: Git, JUnit, Code Coverage, Web development tools, …



Eclipse

- Recommended **reads**
  - Workbench user guide > Tips and tricks
  - Java development user guide > Tips and tri
- Recommended **shortcuts**
  - Quick Fix (Ctrl+1), Quick Access (Ctrl+3)
  - Open Type / resource (Ctrl+Shift+T / Ctrl+Shift+R)
  - Open declaration / Javadoc (F3 / F2)
  - Quick type hierarchy (Ctrl+T)
  - Quick outline (Ctrl+O)
  - Refactor / Rename (Alt+Shift+T / Alt+Shift+R)
  - … Key bindings overview (Ctrl+Shift+L) J



Setting up Eclipse

To setup your Eclipse, you need to:

1. Download and install eclipse
2. Setup basics: code styles, save actions, file encoding, …
3. Setup Tomcat
4. Setup Git
5. Setup DB
6. Setup Launch configurations
7. Install other plugins/extensions as needed (e.g., EclEmma)

Part II. Git

- Git is a modern **distributed** version control system (VCS)
- Initial release **2005** by Linus Torwalds
- Widely adopted in open source communities: Linux Kernel, Ruby on Rails, Android, Debian, …

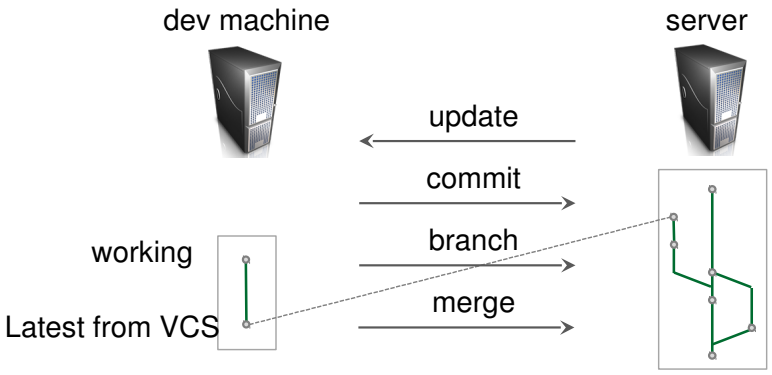
- Can best be learned if you **forget everything you know** about how (centralized) version control works!

Fundamentals of version control

- Repository – a database containing files under version control and the history of these files.
- Working Copy – a local copy of files from the repository. May be modified, and may not represent the most recent repository revision.
- Revision – the state of a file (CVS), of a branch (Git), or of the whole repository (SVN) as committed to the version control system.
- Change Set – a set of modifications to files under version control.
- Commit – the act of writing a change set from the working copy to the repository.
- Update – the act of fetching changes that have been performed on the repository since the last update and applying them to the working copy.

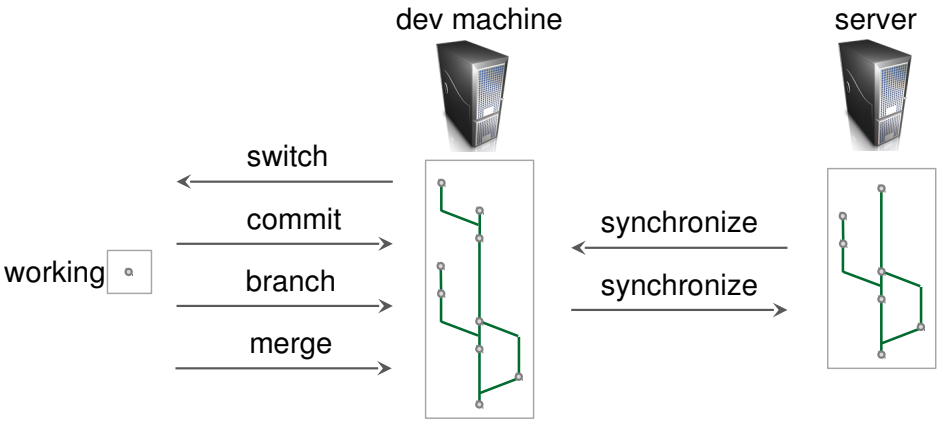
- Branch – a set of files under version control that evolve independently of other file sets. Often defines an own line of development of a product.
- Tag – a human-readable link to a specific revision. Is often used to mark the source code of released versions (e.g. tag v_2_0_3).
- Trunk/Master – the branch denoting the main line of development of a product.
- Merge – the act of reconciling change sets from parallel branches.
- Switch – the act of changing the working copy from a branch to another.
- Conflict – occurs when a file was changed concurrently, and the VCS cannot reconcile the changes automatically. Conflicts must be **resolved** manually.

Centralized VCS



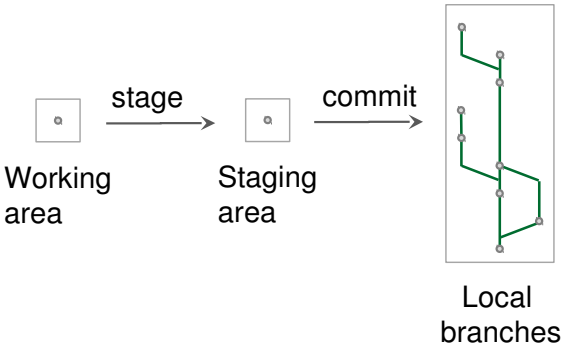
- Cannot work **without connectivity**
- Needs **server** to branch and merge
- Cannot save experimental features **locally**

Decentralized VCS

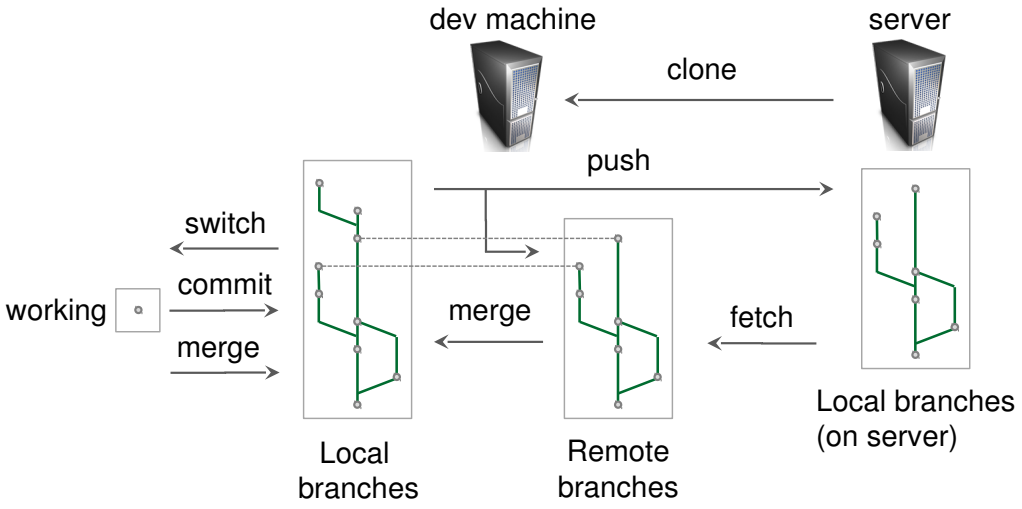


- Works **without connectivity**
- Can branch and merge against **local VCS**
- Needs **synchronization** among multiple VCS

Git Staging

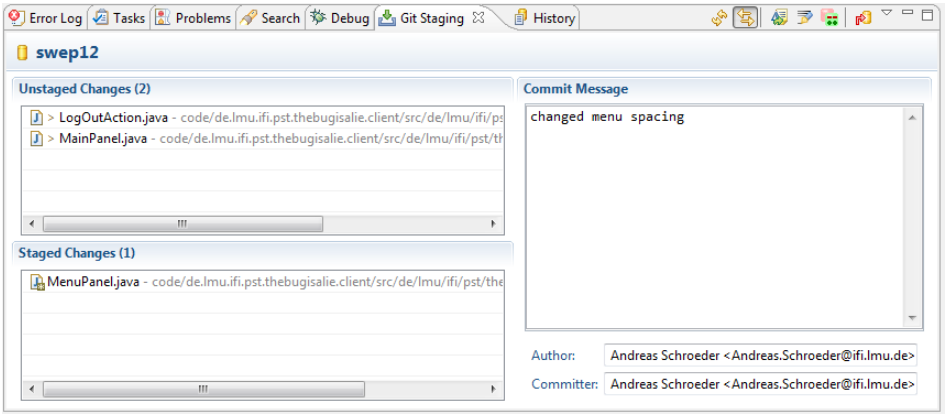


- Git allows to select changes for commit
- “Staging area” lies between working area and local branches



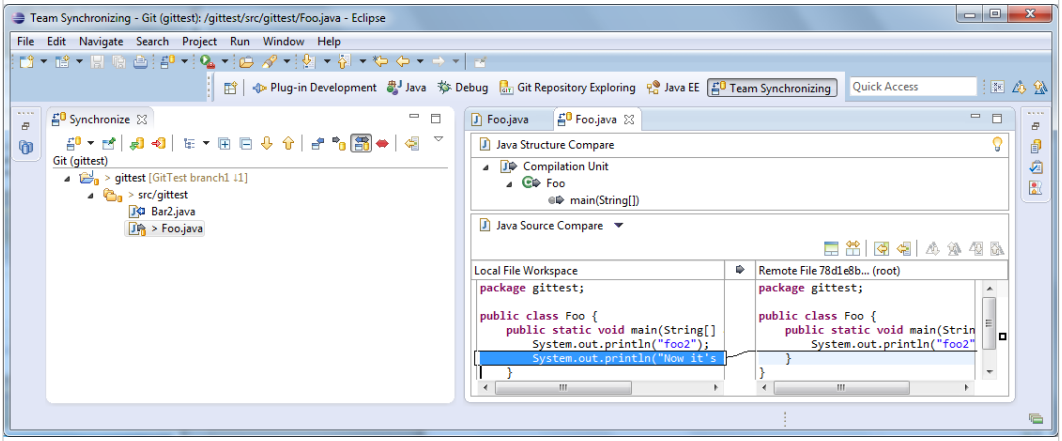
pull = fetch & merge

eGit Staging View



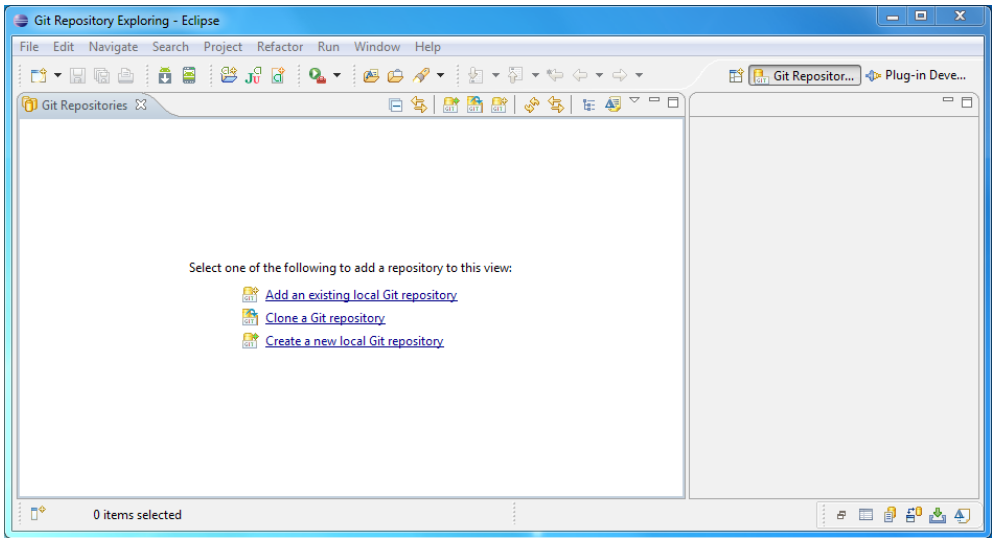
- Helpful tool for creating commits
- Faster than Menu > Team > Add

Synchronize Workspace View

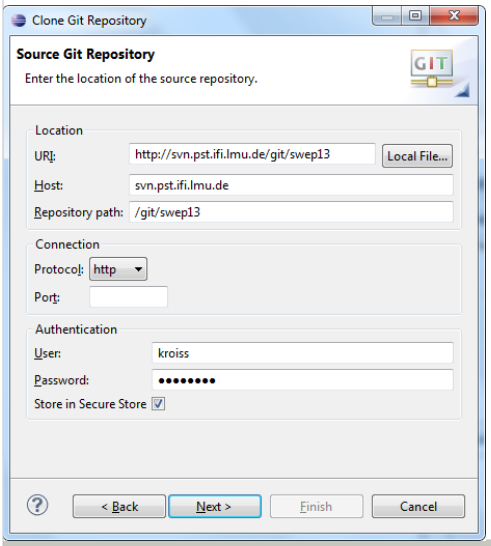


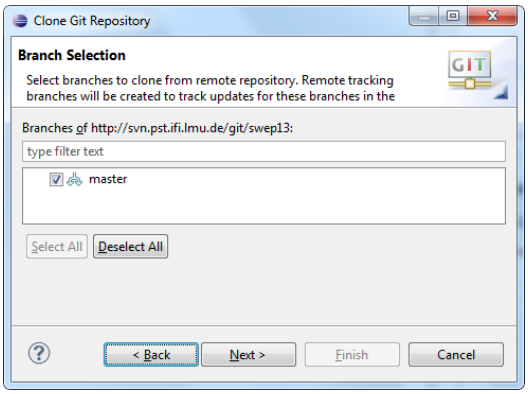
- Menu Team Synchronize Workspace (or change to Team Synchronizing perspective)

eGit from the start

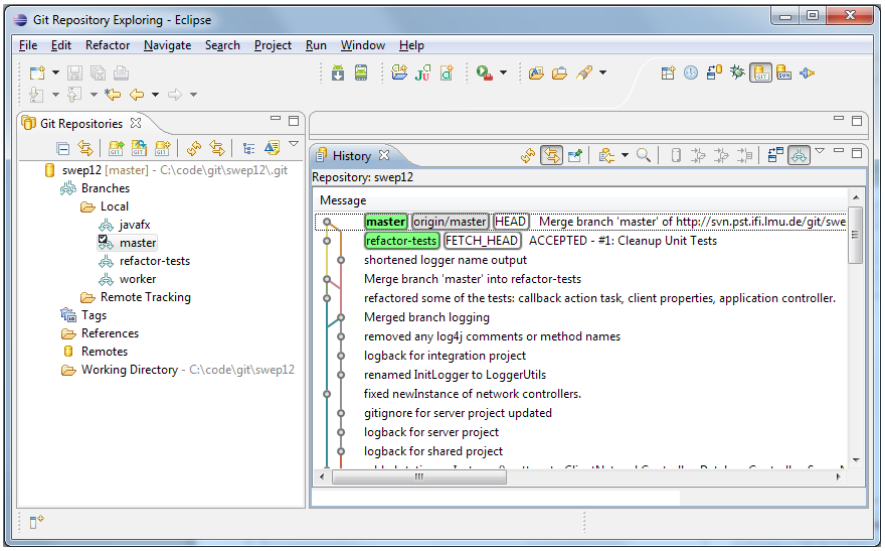


eGit: cloning

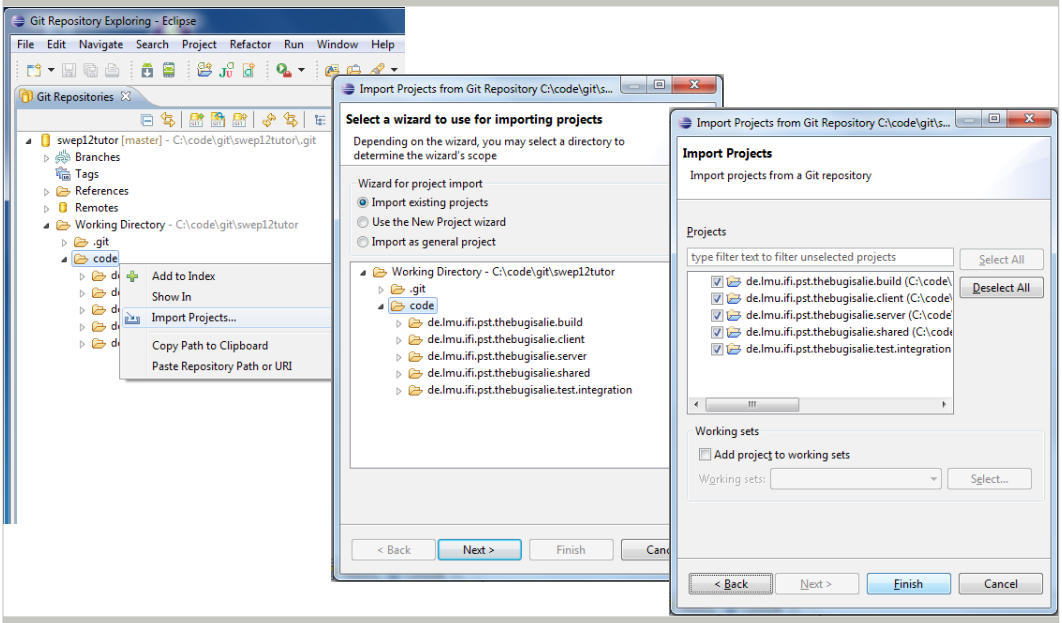




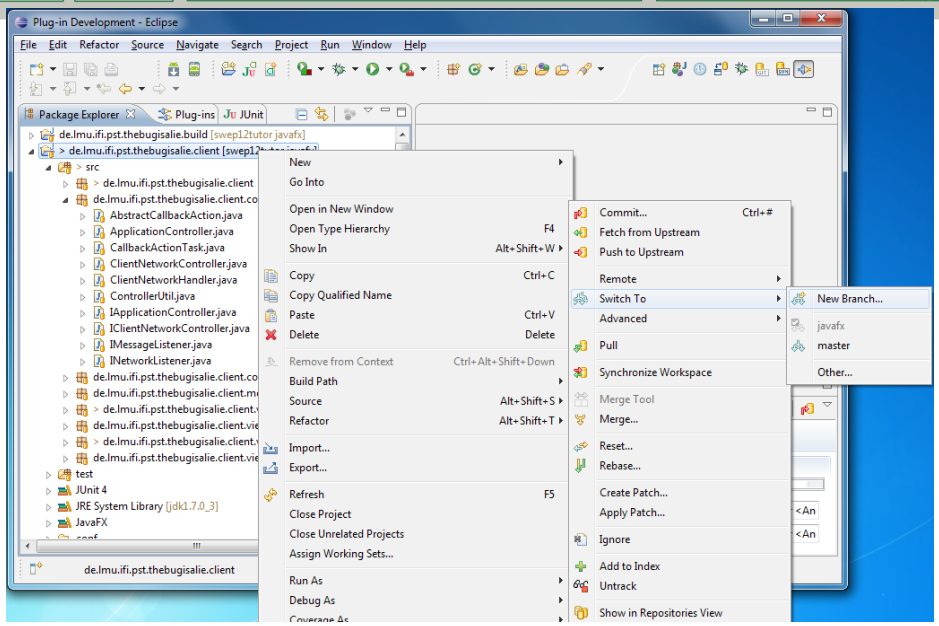
eGit: clone results



eGit: Import Projects



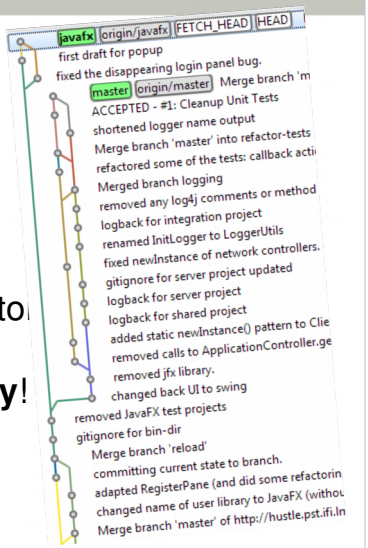
Creating a new branch



LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

The art of branching

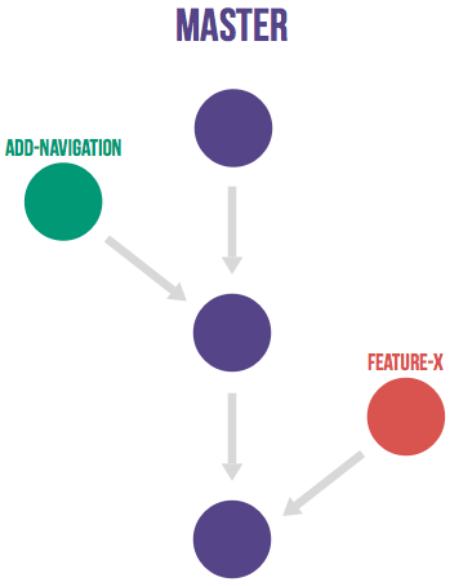
- When working on a user story
  - Create a **branch** for your story (= feature branch)
  - Work on the branch
  - **Merge** the branch into master
- Don't disconnect from the reposito (= team): fetch & merge master changes to your branch **regularly**!
- Read the Git **tutorials**



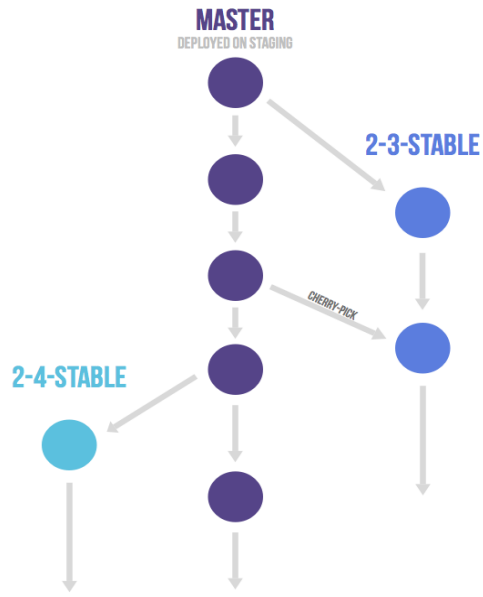
Viel Erfolg mit Git J

Und jetzt zum nächsten Thema, MyLyn

GitLab Flow: Feature Branches



GitLab Flow: Release Branches



Commit messages

- What does the following history tell you?

```
ea42b79 There were some bugs in the code!!!! Didn't know
2db0f12 fixed two build-breaking issues: the ant task co
147709f Tweaks to some files
22b25e0
7f96f57 polishing
59a2ed6 update
```

- And the following?

```
5ba3db6 Fix failing CompositePropertySourceTests
84564a0 Rework @PropertySource early parsing logic
e142fd1 Add tests for ImportSelector meta-data
887815f Update docbook dependency and generate epub
ac8326d Polish mockito usage
```


Viel Erfolg mit Git J

Und jetzt zum nächsten Thema, MyLyn

The art of writing commit messages

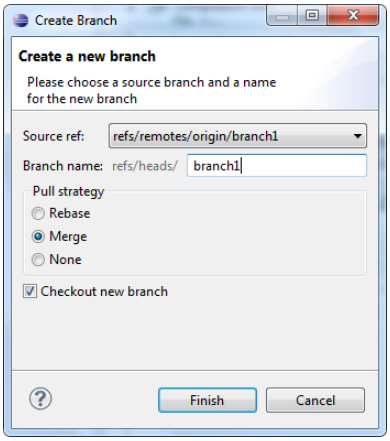
- Good commit messages are important
  - Separate subject from body with a blank line
  - Limit the subject line to 50 characters
  - Capitalize the subject line
  - Do not end the subject line with a period
  - Use the imperative mood in the subject line
  - Wrap the body at 72 characters
  - Use the body to explain *what* and *why* vs. *How*
- See http://chris.beams.io/posts/git-commit/ for more

Viel Erfolg mit Git J

Und jetzt zum nächsten Thema, MyLyn

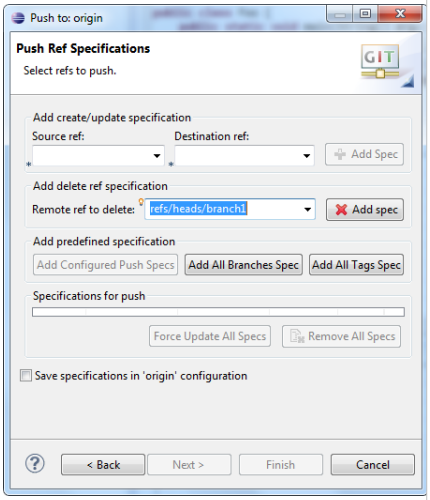
Tracking remote branches

- Instead of checking out a remote branch directly, it's better to create a local branch that **tracks** the remote branch.
- After that, you can pull and push directly to/from the original remote branch.
- If you push a branch that you created locally, the local branch starts tracking the remote copy.



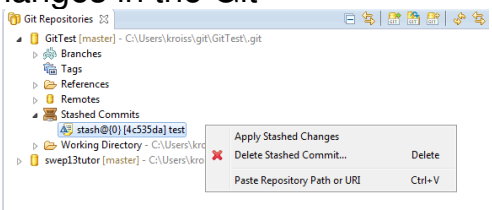
Deleting remote branches

- Deleting the remote reference doesn't do anything on the server.
- To remove a remote branch you have to push "nothing" to the branch:
  `git push origin --delete branch`
  or with the push dialog in eGit (see picture).
- **Deleting remote branches is most fun when others are tracking it J**



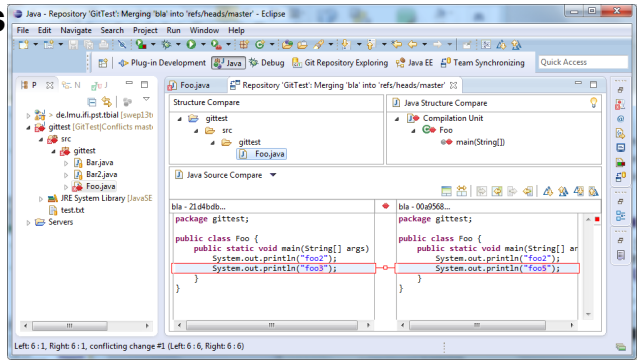
Switching between branches

- Git allows to switch between branches at any time
  - eGit: Menu Team Switch To
- If you have uncommitted changes, you have to either commit them first or **stash your changes**
  - eGit: Menu in Git Repository View Stash Changes
  - Those changes get “stored" and you can switch branches
  - You can restore stashed changes in the Git Repository View



Resolving conflicts

- Merging branches may lead to conflicts. When this happens you end up in a "merging-state" where you have to resolve the conflicts.
- **Resolve conflicts**
  - Menu Team Merge Tool
  - Or/and edit manually
- **Then:**
  - Add to index
  - Commit



Other (e)Git hints

- Undoing changes:
  - Use Menu Replace (HEAD|Commit|…) to replace files/folders with previous versions from the repository
- If you're used to SVN: **don't forget to push**
  - Commit only writes to your local repository. Use "commit and push" in Commit dialog or push explicitly.
- Resolving conflicts without editing (e.g. for binary files)
  - `git checkout --ours <path>`
  - `git checkout --theirs <path>`

Other (e)Git hints

- Use own repository (e.g. GitHub/GitLab) to experiment!
- Install full Git distribution (www.git-scm.com) and (optionally) GUI like TortoiseGit, etc.
- Have a look at the git command line

Some words on collaboration…

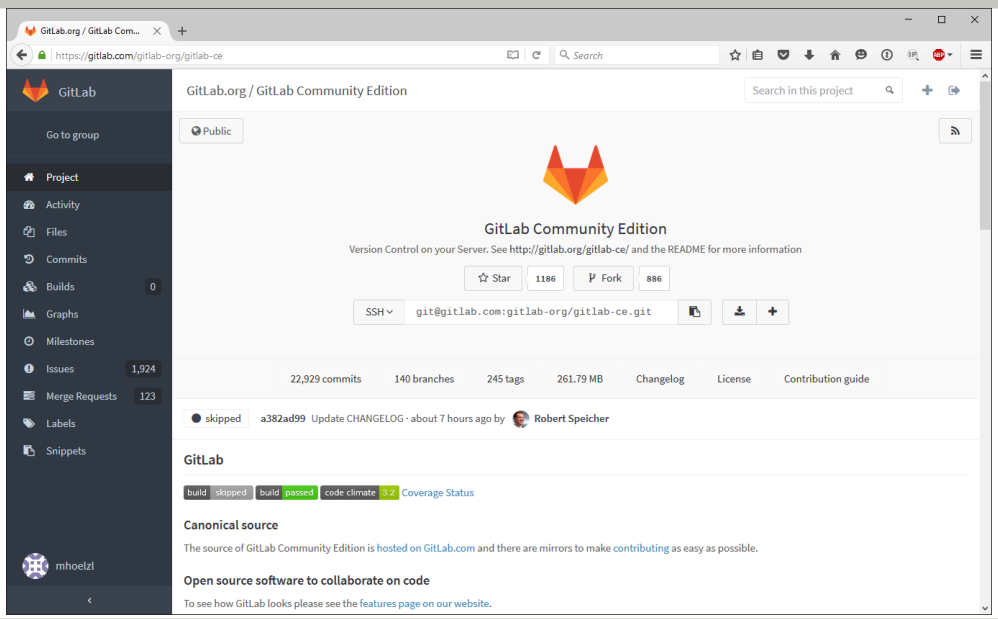
Merging a user story into master means **integration**

- Conflicts must be carefully **resolved**
- The whole codebase must **compile**
- All tests must **pass**:
  unit tests, integration tests, system tests

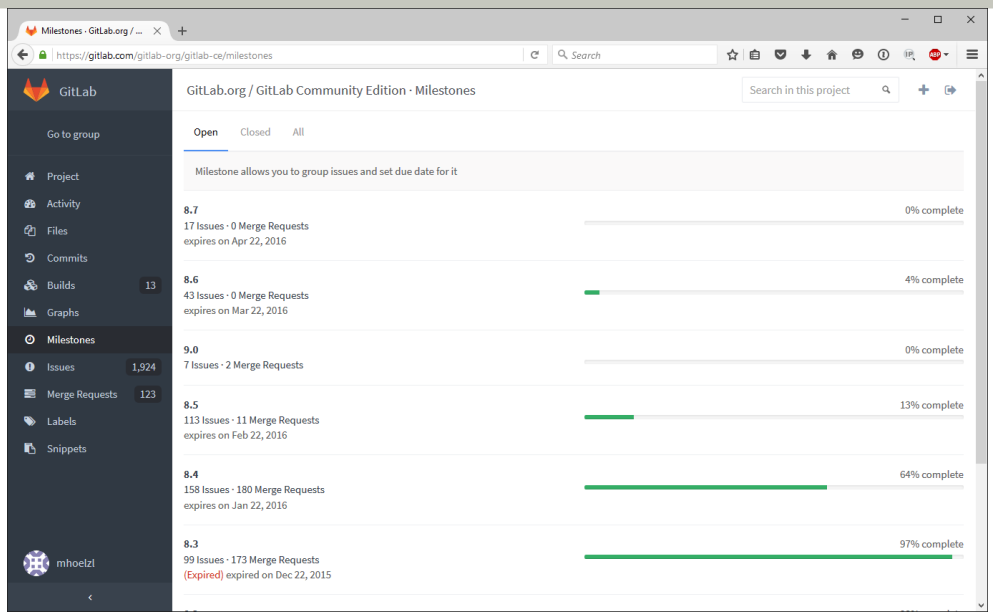
Integrate early and opportunistically,
It will not get easier if you wait!

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

GitLab: Project Overview

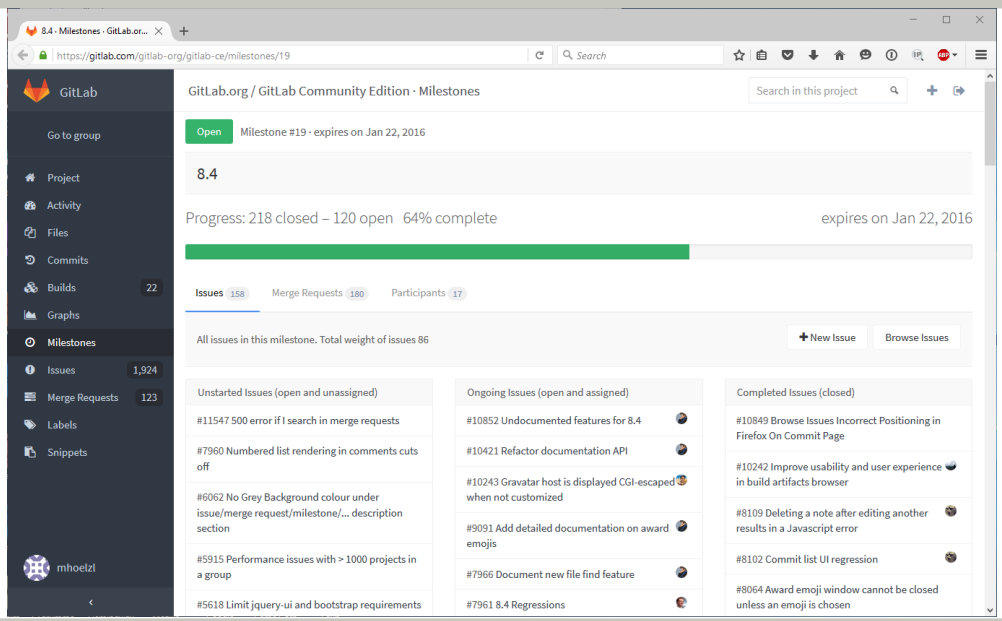


GitLab: List of Milestones

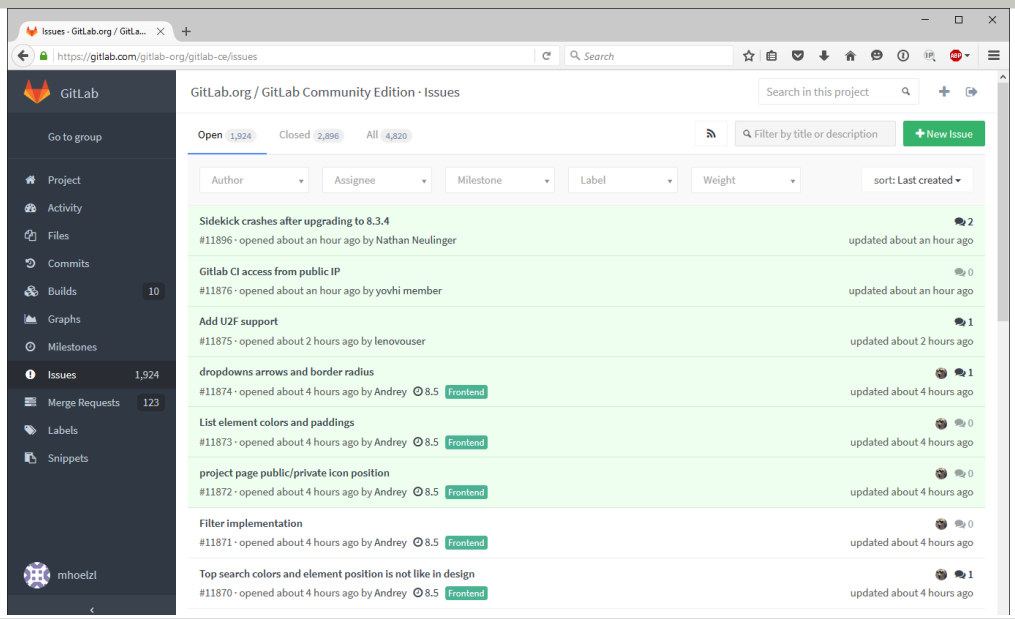


LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

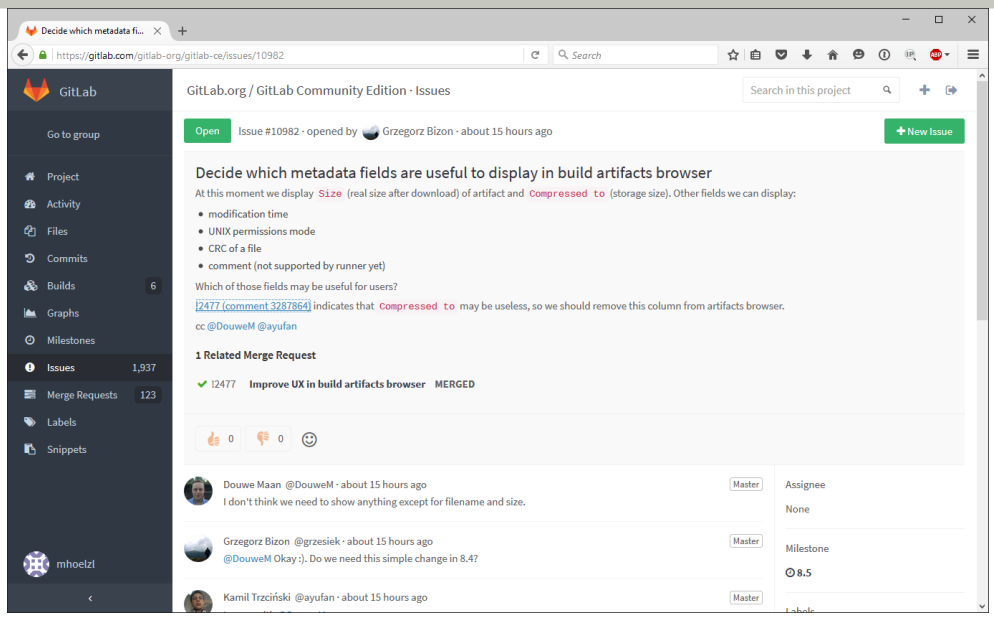
GitLab: Milestone



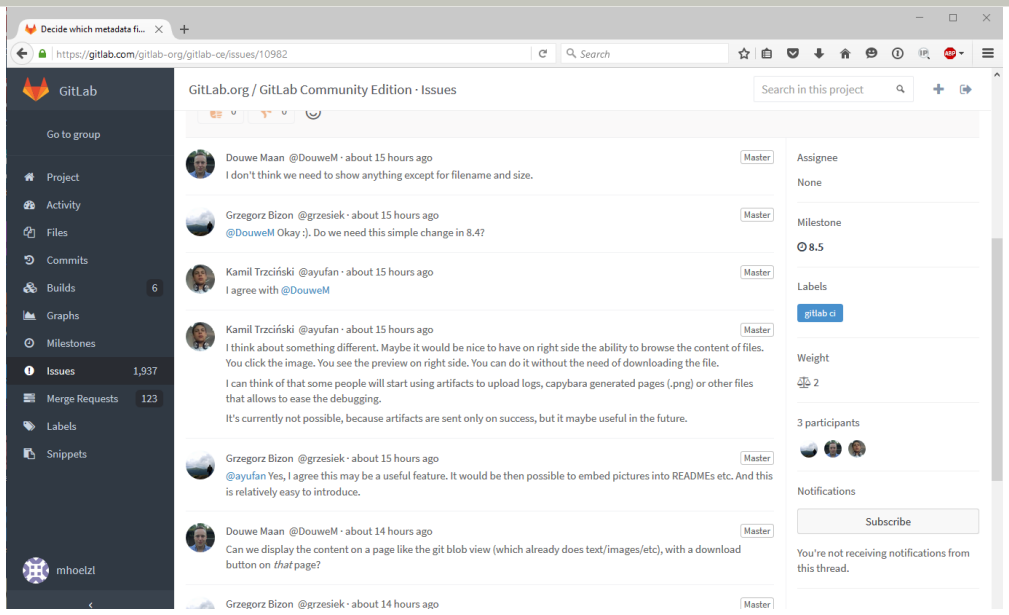
GitLab: Issues



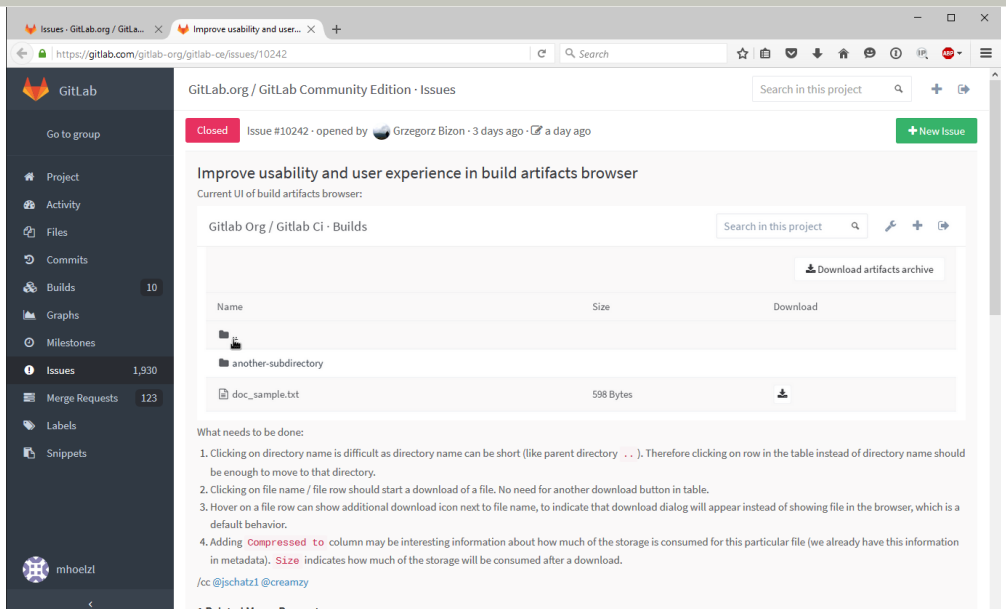
GitLab: Issue



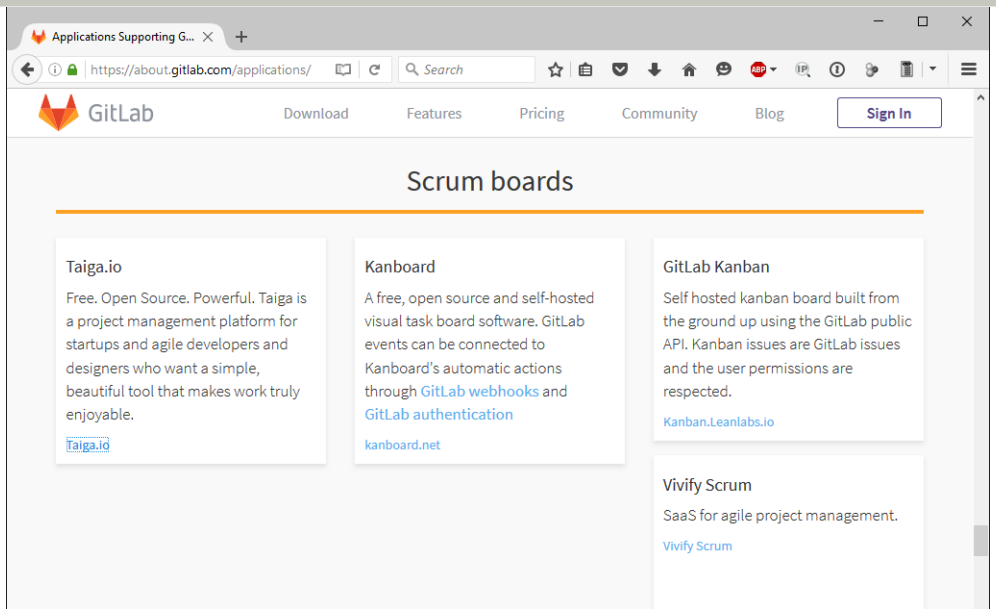
GitLab: Issue



GitLab: Issue

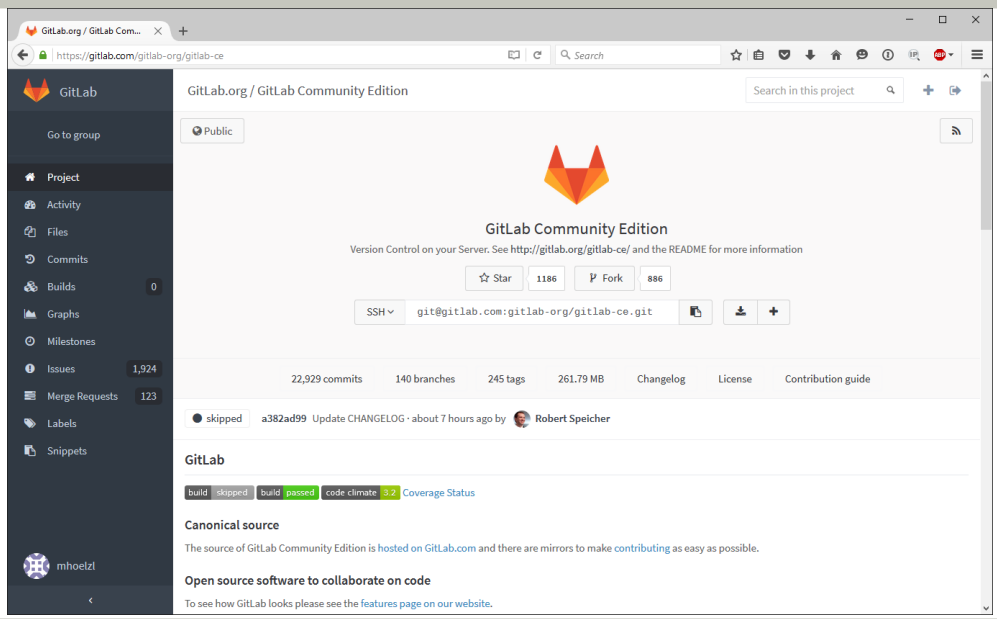


GitLab: Third-Party Scrum Support



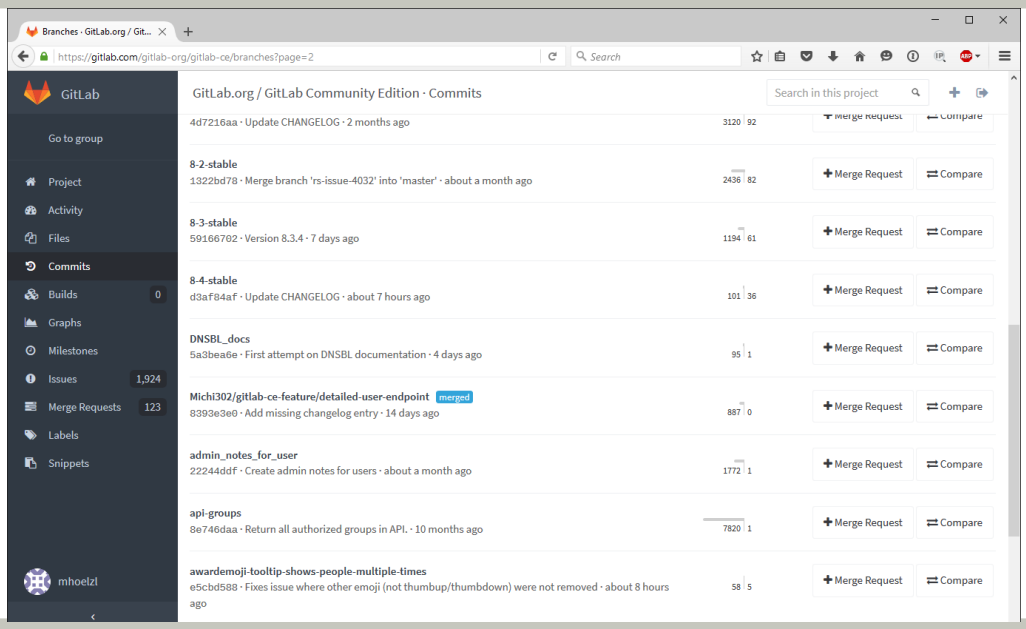
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

GitLab: Branches

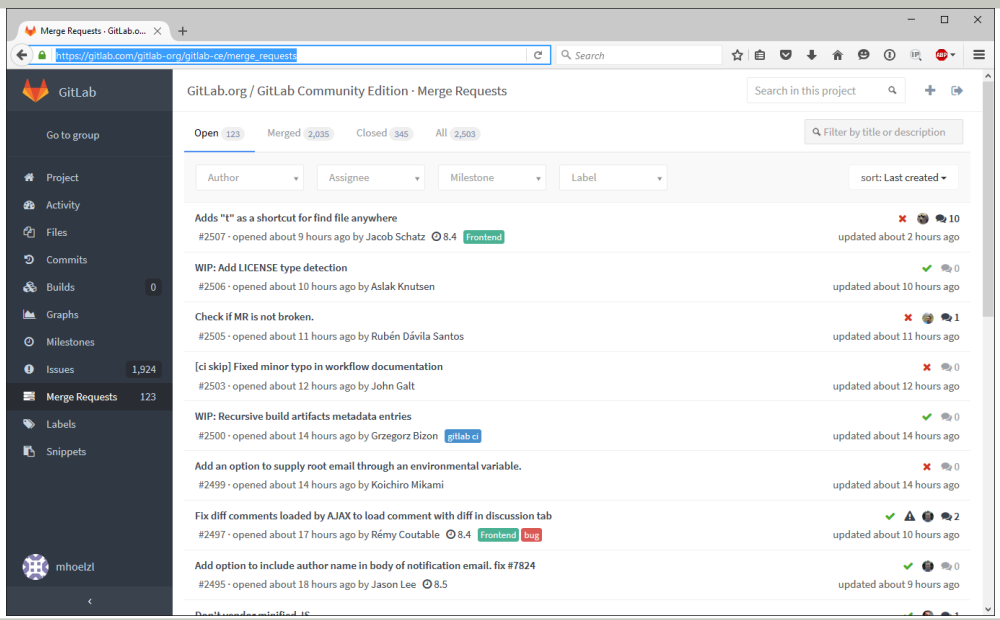


LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

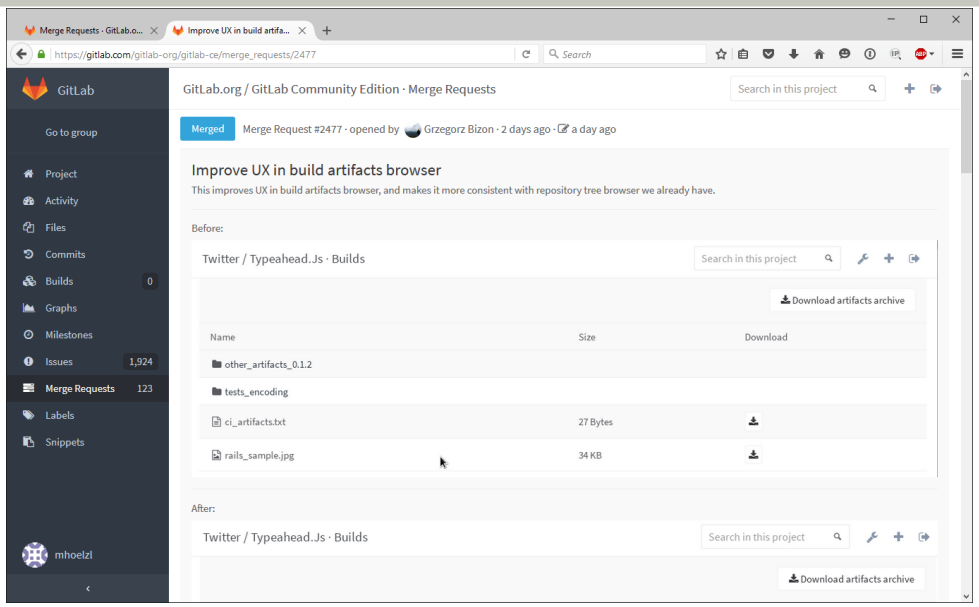
GitLab: Branches



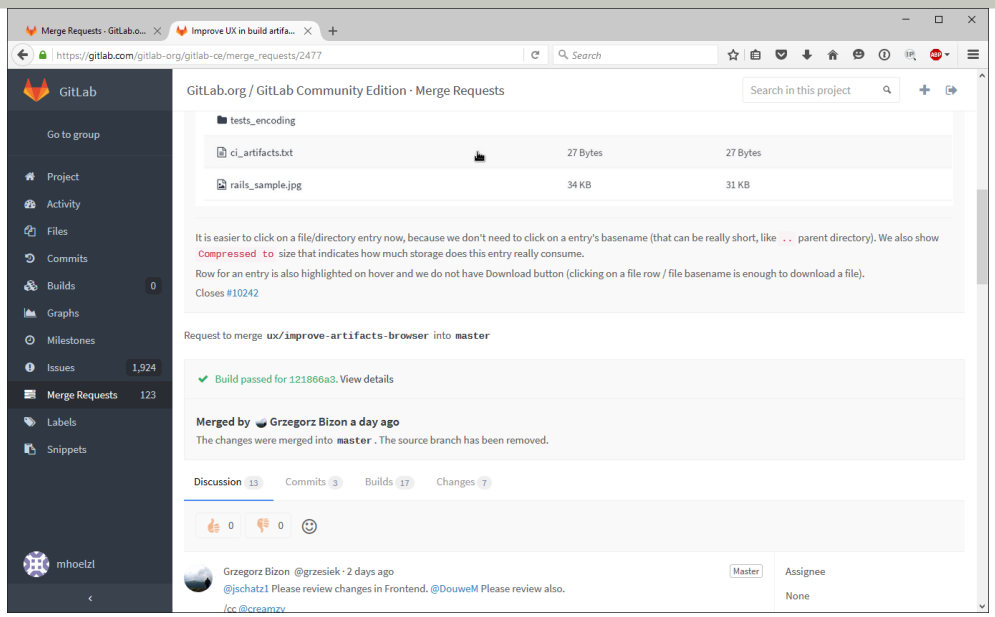
GitLab: Merge Requests



GitLab: Merge Request

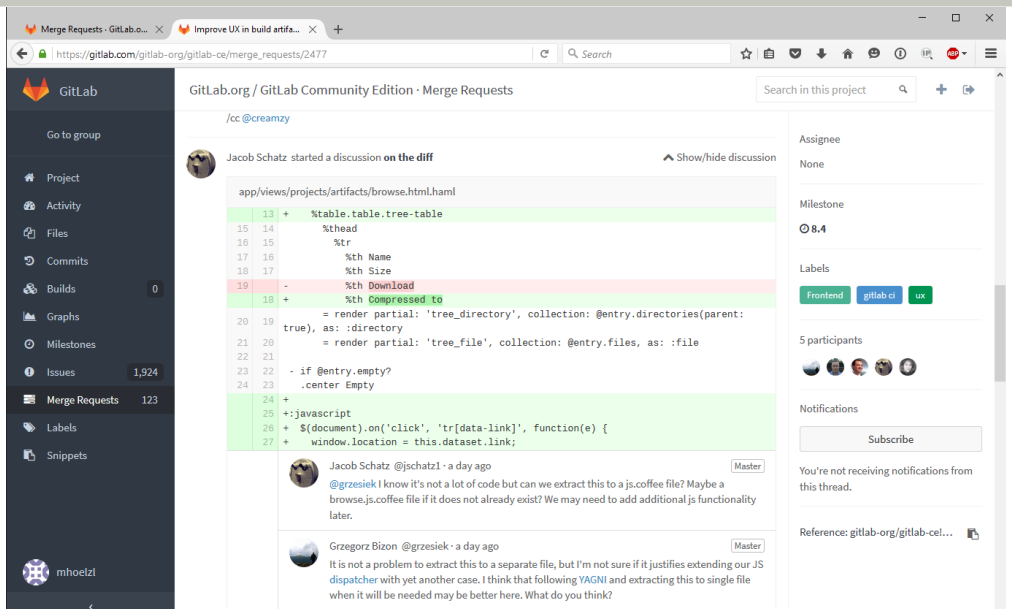


GitLab: Merge Request

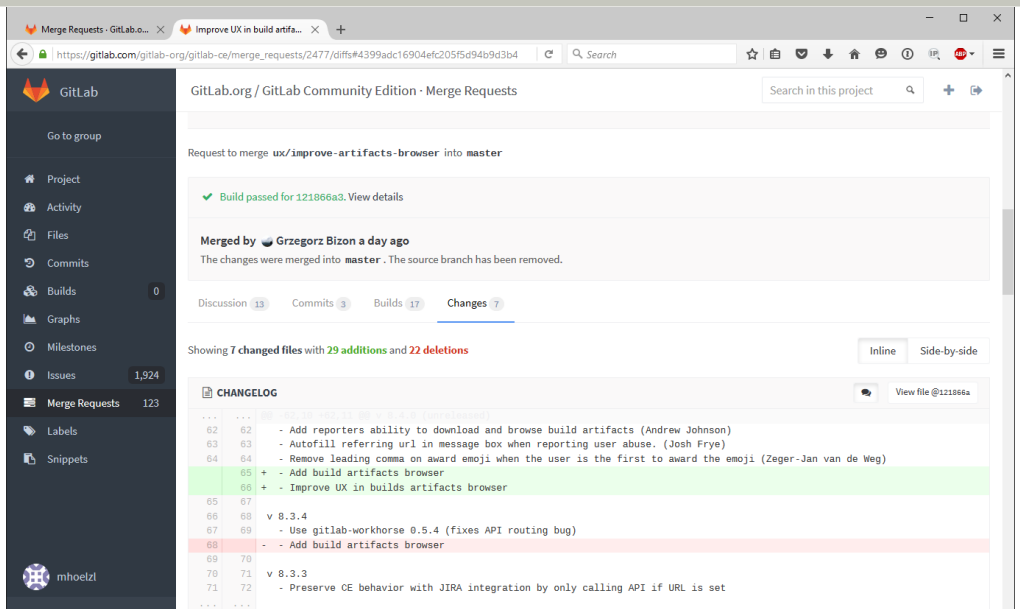


LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

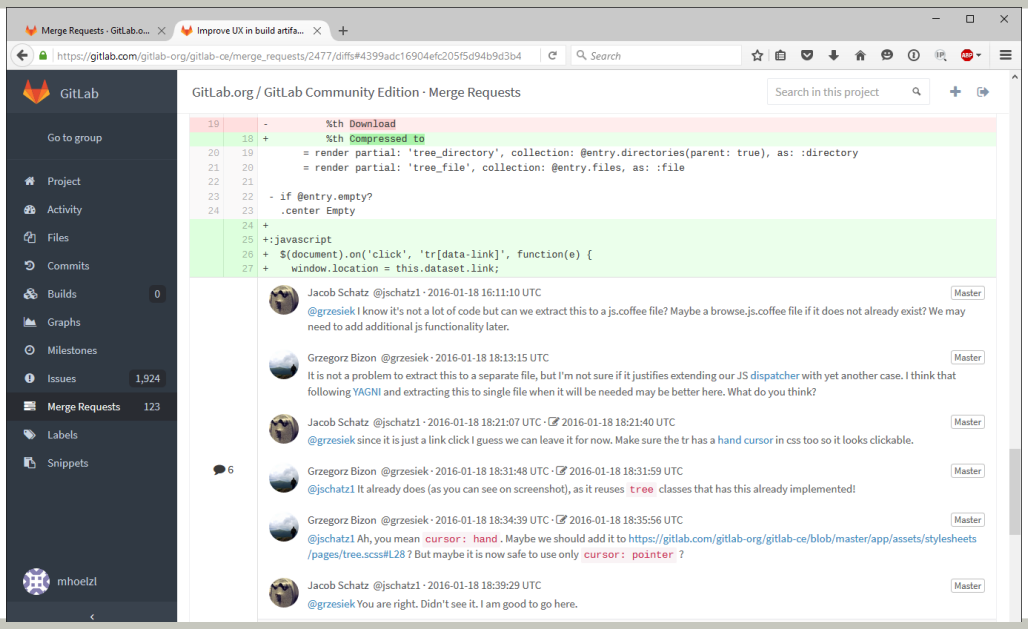
GitLab: Merge Request



GitLab: Merge Request

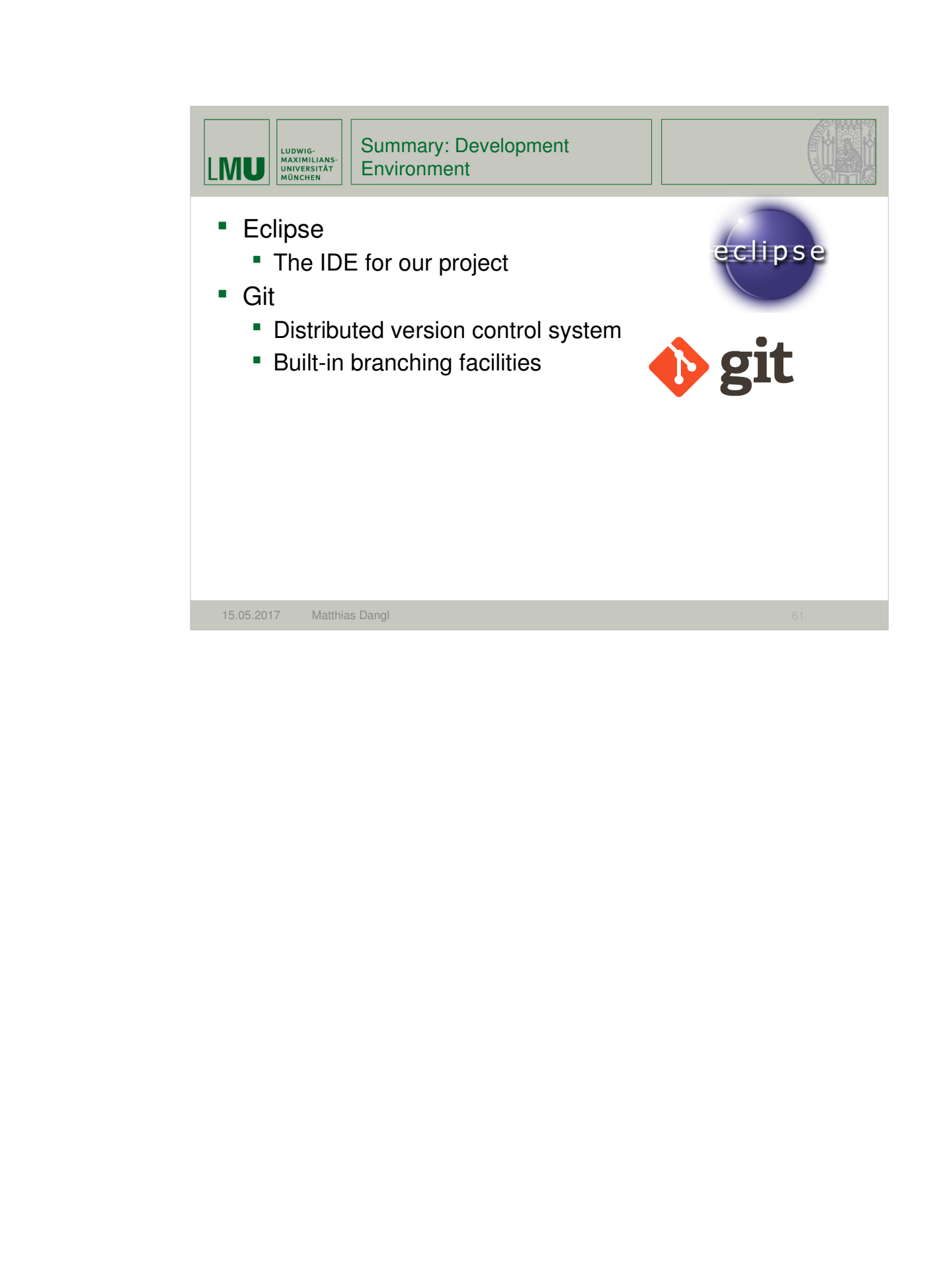


GitLab: Merge Request



Summary

Summary: Development Environment



- Eclipse
  - The IDE for our project
- Git
  - Distributed version control system
  - Built-in branching facilities