Course on Git and GitLab

Yorick Brunet

Nov 15, 2023

CONTENTS:

1 Introduction **1**
1.1 Some Git comics 1
1.2 History 2
1.3 Ressources 2
1.4 Install 2
1.5 Cheat sheets 3

2 Git — Basics **5**
2.1 Preliminary notions 5
2.2 Setup 5
2.3 Init or clone 5
2.4 Working area vs staging area vs repository area 6
2.5 Commit 7
2.6 Log 8
2.7 Diff 10
2.8 Interactive interactions with the working, staging and repository areas 12
2.9 Branch 13
2.10 Merge 16
2.11 Rebase 18
2.12 Stash 22
2.13 Conflict management 24
2.14 Ignore files 28
2.15 Exercises 28

3 Git — GUI **31**
3.1 Linux 31
3.2 MacOS 32
3.3 Windows 32
3.4 All platforms 32

4 Git — Workflow **33**
4.1 Workflow 34
4.2 Useful links to add to project's doc 41

5 GitLab **43**
5.1 Homepage 43
5.2 Project information 44
5.3 Repository 44
5.4 Issues 46
5.5 Merge requests 50

5.6 Conduct a Code Review 52
5.7 Wiki 65
5.8 Settings 65
5.9 Team planning 68

6 Git — Sync **75**
6.1 Remote 75
6.2 A note on history modifications 76
6.3 Fetch 76
6.4 Pull 77
6.5 Push 79
6.6 Exercises 80

7 Git — Advanced **81**
7.1 Reference to current branch and ancestors 81
7.2 Reset 83
7.3 Revert 85
7.4 Worktrees 87
7.5 Clean the repository 89
7.6 Config 89
7.7 LFS 90
7.8 Cherry pick 90
7.9 Squash merge 94
7.10 Rewrite history with interactive rebase 95
7.11 Remove files from history 99

8 Git — Tricks **103**
8.1 Backup and restore a machine's configuration 103
8.2 Useful commands 104

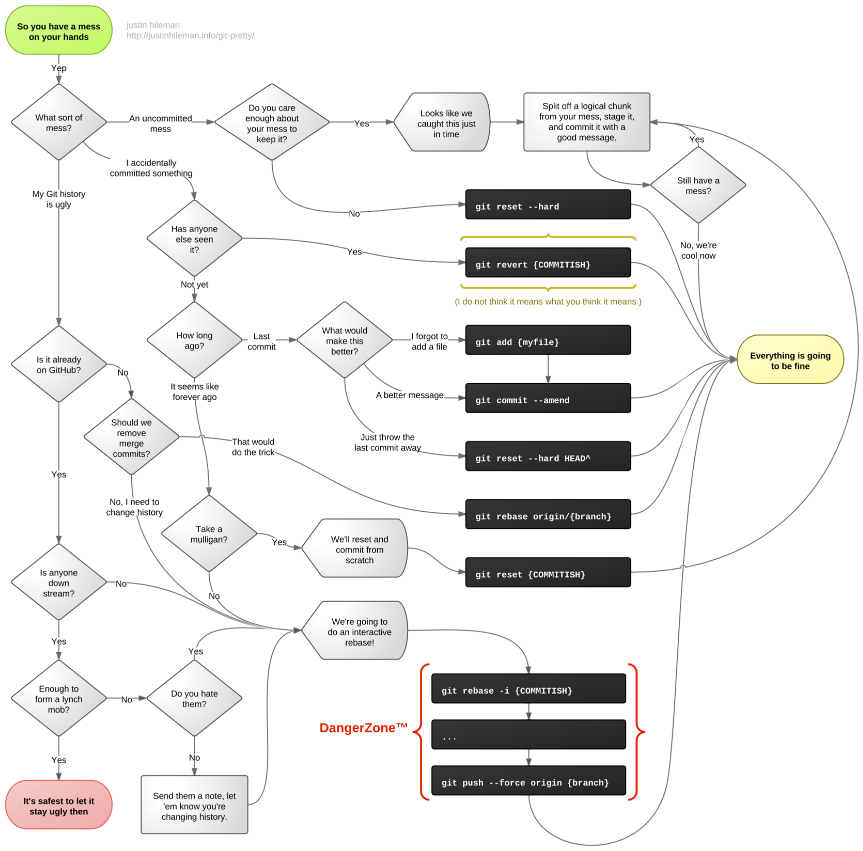
CHAPTER

ONE

INTRODUCTION

This course is on Git and GitLab. The goal is that every participant ends the course with the same knowledge on these two technologies.

1.1 Some Git comics



So you have a mess on your hands
justin hileman
http://justinhileman.info/git-pretty/
Yep
What sort of mess?
An uncommitted mess
Do you care enough about your mess to keep it?
Yes
Looks like we caught this just in time
Split off a logical chunk from your mess, stage it, and commit it with a good message.
Yes
Still have a mess?
No, we're cool now
No
git reset --hard
I accidentally committed something
Has anyone else seen it?
Yes
git revert {COMMITISH}
(I do not think it means what you think it means.)
Not yet
How long ago?
Last commit
What would make this better?
I forgot to add a file
git add {myfile}
A better message
git commit --amend
Just throw the last commit away
git reset --hard HEAD^
Everything is going to be fine
My Git history is ugly
Is it already on GitHub?
No
It seems like forever ago
Should we remove merge commits?
That would do the trick
git rebase origin/{branch}
No, I need to change history
Take a mulligan?
Yes
We'll reset and commit from scratch
git reset {COMMITISH}
Yes
Is anyone down stream?
No
No
We're going to do an interactive rebase!
git rebase -i {COMMITISH}
...
git push --force origin {branch}
DangerZone™
Yes
Enough to form a lynch mob?
No
Do you hate them?
Yes
No
Yes
It's safest to let it stay ugly then
Send them a note, let 'em know you're changing history.

Fig. 1.1: Comics on Git management (but nearly true)

1.2 History

Dixit Git article on Wikipedia:

> Git is free and open source software for distributed version control: tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).

Git was originally written by Linus Torvals, and other kernel developers, in 2005 to be used as version control software for the Linux Kernel. Version 1.0 was released end of 2005.

GitLab is a DevOps platform, founded in 2014, based on Git functionalities but extending them with (not exhaustive):

- web access for Git repositories
- improvement of the interaction on a project thanks to its easy web access
- access rights management
- issue tracking
- merge requests
- CI/CD pipeline
- wiki

1.3 Ressources

- Git home page
- Pro Git book
- GitLab docs
- Tool to learn Git
- GitLab instance of REDS

1.4 Install

1.4.1 Linux

Listing 1.1: Install on Linux

```
$ sudo apt install git gitk git-gui git-lfs
```

See the download page for more details.

1.4.2 MacOS

Listing 1.2: Install on MacOS

```
$ brew install git git-gui git-lfs
```

See the download page for more details.

1.4.3 Windows

Use git for windows to install Git.

For command-line, open **Git Bash**.

See the download page for more details.

1.5 Cheat sheets

- GitLab's cheat sheet → as pdf
- Atlassian's cheat sheet
- GitHub's cheat sheet
- ndpsoftware's cheat sheet
- freecodecamp's cheat sheet

CHAPTER

TWO

GIT — BASICS

2.1 Preliminary notions

A commit is an atomic set of changes that is added to the repository. A commit may have one of multiple parents and one of multiple children, creating a chain of commits. A Git repository is thus a ensemble of commits that are meaningfully linked together.

This document shows a few images representing a chain of commits. The arrow between two commits has the meaning "my parent is" and goes from the child commit to the parent commit.

2.2 Setup

Git needs a initial configuration to correctly display your identity in commits. Use command git-config. If this step is forgotten, Git will just use the information it finds on your machine, which leads to inconcistent identities when commiting across different machines and also to the use of non pretty data.

Listing 2.1: Basic Git configuration

```
$ git config --global user.name "Yorick Brunet"
$ git config --global user.email "yorick.brunet@heig-vd.ch"
$ git config --global pull.rebase true
$ git config --global fetch.prune true
```

The configuration git-config pull.rebase makes Git rebase branches on top of the fetched branch. This explanation will make sense later in the course, once we've seen *Branch* and *Rebase*.

The configuration git-config fetch.prune makes Git remove any remote-tracking references that no longer exist on the remote. This explanation will make sense later in the course, once we've seen *Remote*.

2.3 Init or clone

It exists two ways to set up a repository. The first way is to initialise a repository on his/her machine to add version control to an existing project. Use the command git-init, which creates a `.git/` directory in the project. The repository is now empty but has no remote.

Listing 2.2: Init a new Git repository with the main branch called `main`

```
$ git init --initial-branch=main
```

The second way is to clone an existing repository from a remote repository hosted on a web platform, such as GitLab or GitHub. Use the command git-clone.

Listing 2.3: Clone an existing repository

```
$ git clone https://reds-gitlab.heig-vd.ch/reds-presentations/c-git-and-gitlab.git
```

2.4 Working area vs staging area vs repository area

Git uses three areas for managing files:

- the **working area** contains all untracked and tracked-but-modified files
- the **staging area** contains all newly-added-and-unmodified and tracked-but-modified-and-staged files
  - the staging area is the content of the next commit
  - any changes made to a tracked file but not staged is not going to be committed
- the **repository area** contains the files that Git knows, i.e. the already commited files

The command git-status displays the current status of the untracked, tracked-but-modified and staged files.

Listing 2.4: Example of **`git status`**

```
$ git status
On branch main

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
        new file:   ../.gitignore
        new file:   ../README.md

Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
        modified:   ../README.md

Untracked files:
(use "git add <file>..." to include in what will be committed)
        source/basics.rst
```

Note: Have you seen that a file can be listed in “Changes not staged for commit” (aka. working area) and “Changes to be committed” (aka. staging area) ?

Use the command git-add to:

- track an untracked file
- stage the changes of a tracked file

Use the command git-restore to discard changes of a tracked file from the working area.

Use the command git-rm --cached to unstage changes of a tracked file (from the staging area). The changes are transfered to the working directory. You can also use git reset -- for this action but mind the `--` (more on *Reset* later).

Use the command git-rm (+ `-r` for directory, + `-f` to force) to remove a tracked file from the repository.

2.5 Commit

The command git-commit "saves" the staging area in a block called **commit**. A commit can also be seen as a snapshot of the current development of a batch of files. The commit contains several modifications to the last commit pointed by HEAD, which is a symbolic reference to the branch we are working on. `HEAD` is the parent of the new commit and the new commit is now the child of `HEAD`. Upon commit, the pointer `HEAD` is automatically changed to the new commit.

Commits and their parents (taken from Pro Git book, Fig. 2.1):

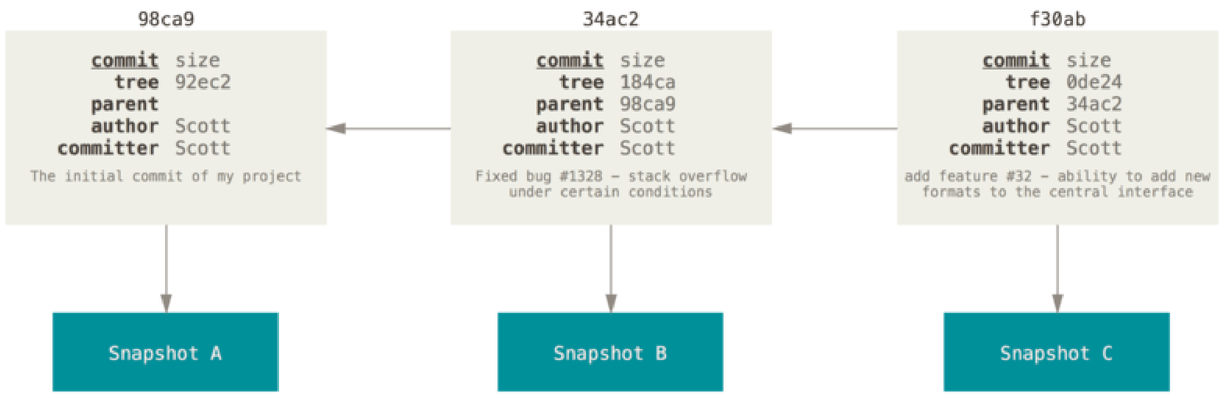


Fig. 2.1: Commit — Commits and their parents

The commit needs a comment to help other (and yourself!) know what you added in this commit. The comment is automatically asked when running **`git commit`**. Alternatively, the comment can be passed directly with **`git commit -m "..."`**.

Listing 2.5: Example of **`git commit`**

```
$ git commit # commit the staging area, the comment is asked by Git
$ git commit -m "..." # commit the staging area, the comment is directly provided by the␣
↪user
```

Some projects may also require to sign off the commit. The command git commit -s must be used for that. This will add a "Signed-off-by" in the commit message:

Listing 2.6: Example of signed-off

```
commit e31ffcff3a71ccb3a80b0755db3cbc741e31dc6f (HEAD -> 1-release-first-version)
Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
Date:   Thu Nov 10 17:26:24 2022 +0100

    Address JMI's comments

    Signed-off-by: Yorick Brunet <yorick.brunet@heig-vd.ch>
```

2.5.1 Rules for good commits

1. Make commit as small as possible; always think about the Single Responsibility Principle
2. A commit can never be too short – a oneliner is fine!
3. There cannot be too many commits
4. Write meaningful commit messages
5. Don't comment unused code, remove it
6. Each commit should compile

2.5.2 Rules for good commit messages

The Git Book provides some commit guidelines. Here's a summary:

1. Capitalize the subject line
2. Use the imperative mood in the subject line
3. Limit the subject line to 50 characters
4. Do not end the subject line with a period
5. Separate subject from body with a blank line
6. Wrap the body at 72 characters
7. Use the body to explain what and why vs. how

2.6 Log

The command git-log shows the history of a branch. By default, `HEAD` is used to determine the current branch. Use options `--graph --all` to show more details. Use options `--oneline` to show only one line per commit.

Listing 2.7: Possible references for **`git log`**

```
$ git log # show history of the current branch (aka HEAD)
$ git log main # show the history of a specific branch
$ git log 1f13d1e # show the history of a specific commit
$ git log v1 # show the history of a specific tag
```

Listing 2.8: Example of **`git log`**

```
$ git log
commit 9f582ed143faf9d3f18bffb0812e42731789cad0 (HEAD -> main)
Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
Date:   Thu Sep 8 14:00:22 2022 +0200

    Add Git GUI

commit 16a2cfb4aa6e6130e6d475cc75e5e3fff361536e
Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
Date:   Thu Sep 8 13:34:39 2022 +0200
```

(continues on next page)

(continued from previous page)

```

    Initial structure + Git Basics
```

Listing 2.9: Example of **git log --graph --all**

```
$ git log --graph --all
* commit 9f582ed143faf9d3f18bffb0812e42731789cad0 (HEAD -> main)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Sep 8 14:00:22 2022 +0200
|
|     Add Git GUI
|
* commit 16a2cfb4aa6e6130e6d475cc75e5e3fff361536e
Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
Date:   Thu Sep 8 13:34:39 2022 +0200

    Initial structure + Git Basics
```

Listing 2.10: Example of **git log --graph --all --oneline**

```
$ git log --graph --all --oneline
* 9f582ed (HEAD -> main) Add Git GUI
* 16a2cfb Initial structure + Git Basics
```

The command git-show shows the content of a commit. The content is the commit hash, author, date, message and diff.

Listing 2.11: Possible references for **git show**

```
$ git show # show the last commit of the current branch (aka HEAD)
$ git show main # show the last commit of a specific branch
$ git show 1f13d1e # show the specific commit
$ git show v1 # show the tagged commit
```

Example:

Listing 2.12: Example of **git show**

```
$ git show
commit 9f582ed143faf9d3f18bffb0812e42731789cad0 (HEAD -> main)
Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
Date:   Thu Sep 8 14:00:22 2022 +0200

    Add Git GUI

diff --git a/docs/source/gui.rst b/docs/source/gui.rst
index b2f762d..28af942 100644
--- a/docs/source/gui.rst
+++ b/docs/source/gui.rst
@@ -2,3 +2,28 @@
Git --- GUI
===========

```

(continues on next page)

(continued from previous page)

```
+Git can be used with command line, but can also be used with a graphical user interface.
+
+Linux
+=====
[...]
```

2.7 Diff

One of the main advantage of a versioning system is to see the difference of our changes over a previous version. **git log** and **git show** allows us to see the different revisions points (commits) in our repository and the changes introduced by each commit.

The command git-diff allows us to see the changes introduced within the working and staging areas, and between commits.

- Use command git-diff to show the changes of the working area
- Use command git-diff --cached to show the changes of the staging area
- Use command git-diff <rev> to show the changes of the working and staging areas relative to *<rev>*
- Use command git-diff <rev1> <rev2> to show the changes between *<rev1>* and *<rev2>* with *<rev1>* being the oldest revision point

Examples:

Listing 2.13: Full example of **git diff** — Current file content

```
Diff
====

One of the main advantage of a versioning system is to see the difference of our changes␣
↪over a previous version.
:program:`git log` and :program:`git show` allows us to see the different revisions␣
↪points (commits) in our repository and
the changes introduced by each commit.
```

Listing 2.14: Full example of **git diff**

```
$ git status
On branch 1-release-first-version
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
      modified:   docs/source/basics.rst

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
      modified:   docs/source/basics.rst

$ git diff --cached
diff --git a/docs/source/basics.rst b/docs/source/basics.rst
index b722e53..3fa5780 100644
```

(continues on next page)

(continued from previous page)

```
--- a/docs/source/basics.rst
+++ b/docs/source/basics.rst
@@ -280,8 +280,9 @@ Example:
     +=====
     [...]

-Differences
-===========
+Diff
+====
+
 Interactive interactions with the working, staging and repository areas
 ======================================================================

(END)

$ git diff
diff --git a/docs/source/basics.rst b/docs/source/basics.rst
index 3fa5780..6758c20 100644
--- a/docs/source/basics.rst
+++ b/docs/source/basics.rst
@@ -283,6 +283,13 @@ Example:
 Diff
 ====

+One of the main advantage of a versioning system is to see the difference of our␣
↪changes over a previous version.
+:program:`git log` and :program:`git show` allows us to see the different revisions␣
↪points (commits) in our repository and
+the changes introduced by each commit.
+
 Interactive interactions with the working, staging and repository areas
 ======================================================================

(END)

$ git diff HEAD

diff --git a/docs/source/basics.rst b/docs/source/basics.rst
index b722e53..6758c20 100644
--- a/docs/source/basics.rst
+++ b/docs/source/basics.rst
@@ -280,8 +280,16 @@ Example:
     +=====
     [...]

-Differences
-===========
+Diff
+====
+
+One of the main advantage of a versioning system is to see the difference of our␣
```

(continues on next page)

(continued from previous page)

```
↪changes over a previous version.
+:program:`git log` and :program:`git show` allows us to see the different revisions␣
↪points (commits) in our repository and
+the changes introduced by each commit.
+
Interactive interactions with the working, staging and repository areas
=======================================================================

(END)
```

2.8 Interactive interactions with the working, staging and repository areas

Most of the commands seen in *Working area vs staging area vs repository area* have an option *-p* which allows an interactive selection of hunks, and thus allows applying the command on the selected hunks (instead of on the whole file). A hunk is a change (composed of one or multiple lines) in a file that can be applied separately from other changes in the same file. A hunk can be split.

Use the command git-add -p to stage changes of a tracked file by interactively choosing which hunks to add.

Listing 2.15: Example of **git add -p**

```
$ git add -p source/basics.rst
diff --git a/docs/source/basics.rst b/docs/source/basics.rst
index bc07bc2..77fc7eb 100644
--- a/docs/source/basics.rst
+++ b/docs/source/basics.rst
@@ -22,9 +19,13 @@ to the use of non pretty data.
    $ git config --global user.name "Brunet Yorick"
    $ git config --global user.email "yorick.brunet@heig-vd.ch"
+   $ git config --global pull.rebase true
+
+`git-config pull.rebase <https://git-scm.com/docs/git-config#Documentation/git-config.
↪txt-pullrebase>`__ makes
+Git rebase branches on top of the fetched branch.

Init or clone
--------------
+=============

It exists two ways to set up a repository.
The first way is to initialise a repository on his/her machine to add version control to␣
↪an
(2/8) Stage this hunk [y,n,q,a,d,K,j,J,g,/,s,e,?]? s
Split into 2 hunks.
@@ -22,5 +19,9 @@

    $ git config --global user.name "Brunet Yorick"
    $ git config --global user.email "yorick.brunet@heig-vd.ch"
+   $ git config --global pull.rebase true
```

(continues on next page)

(continued from previous page)

```
+
+`git-config pull.rebase <https://git-scm.com/docs/git-config#Documentation/git-config.
↪txt-pullrebase>`__ makes
+Git rebase branches on top of the fetched branch.

Init or clone
(2/9) Stage this hunk [y,n,q,a,d,K,j,J,g,/,e,?]? y
@@ -25,6 +26,6 @@

Init or clone
--------------
+=============

It exists two ways to set up a repository.
The first way is to initialise a repository on his/her machine to add version control to␣
↪an
(3/9) Stage this hunk [y,n,q,a,d,K,j,J,g,/,e,?]? n
```

Use the command git-reset -p to unstage changes of a tracked file by interactively choosing which hunks to unstage. This is the opposite of **`git add -p`**.

Use the command git restore -p to discard changes of a tracked file by interactively choosing which hunks to discard.

2.9 Branch

Git provides lightweight branches, which are easy to create, merge and delete. `main/master` is also a branch (it's just the main branch) and is created during the first commit. Ideally, each piece of work should be done in a separate branch.

The command git-branch runs operations on branches, such as create, delete, rename, etc. The command git-checkout allows to switch between branches or "pointers" (tags, commit hash).

Listing 2.16: Possible usages of **`git branch`** and **`git checkout`**

```
$ git branch # list local branches
$ git branch -a # list local and remote branches
$ git branch hello # create branch "hello" from HEAD
$ git checkout hello # switch to branch "hello"
$ git checkout -b hello # create and switch to branch "hello"
```

Example:

1. The repository is initialised (result shown on Fig. 2.2)

2. `$ git branch hello` (result shown on Fig. 2.3)

3. `$ git branch` (result shown on Listing 2.17)

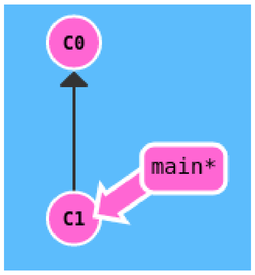


Fig. 2.2: Branch — Repository initialisation

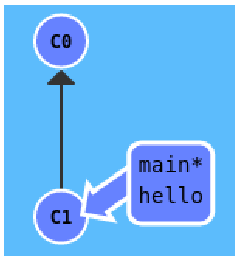


Fig. 2.3: Branch — Create branch *hello*

Listing 2.17: Result of **git branch**

```
* main
hello
```

4. $ `git checkout hello` (result shown on Fig. 2.4)

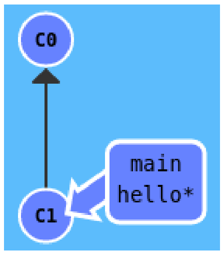


Fig. 2.4: Branch — Switch to branch *hello*

5. Add some changes with **git commit** (result shown on Fig. 2.5)

6. $ `git branch` (result shown on Listing 2.18)

Listing 2.18: Result of **git branch**

```
main
* hello
```

7. $ `git checkout main` (result shown on Fig. 2.6)

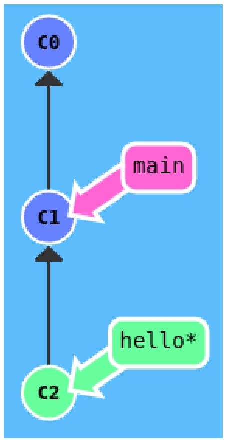


Fig. 2.5: Branch — Create a commit on branch *hello*

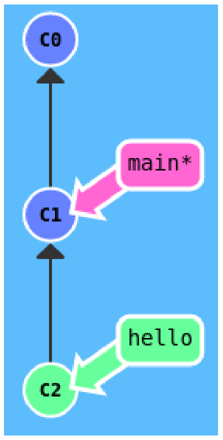


Fig. 2.6: Branch — Switch to branch *main*

8. `$ git checkout -b world` (result shown on Fig. 2.7)

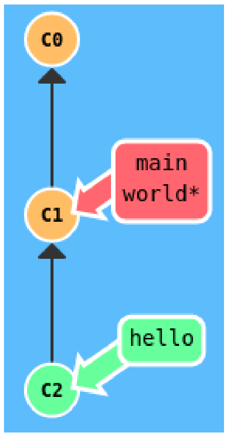


Fig. 2.7: Branch — Create branch *world* and switch to it

9. Add some changes with **`git commit`** (result shown on Fig. 2.8)

2.9.1 Branch naming

A common branch naming helps organise branches when several persons are working together (or even when working alone).

A good pattern is **[kind-of-change]/[what-is-being-changed]**, e.g. feature/adapt-dts-for-new-soc.

We will see this later, but GitLab names branches created from issues with the pattern **[issue-ID]-[issue-title]**, e.g. 1-release-first-version.

2.10 Merge

Branches are useful to work on a task, but the work must then be merged with the main branch. The command git-merge merges branches. The merge can happen either *fast-forward* (default) or not. The *fast-forward* option updates, when possible, the current branch pointer to the pointer of the branch being merged. The non-*fast-forward* option creates an additional merge commit.

You want a clean history? Use the fast-forward option. You want a meaningful history? Do not use the fast-forward option. Your teammates will thank you.

Listing 2.19: Possible usages of **`git merge`**

```
$ git merge my-branch-to-merge # merge my-branch-to-merge into the current branch with␣
↪fast-forward
$ git merge --no-ff my-branch-to-merge # merge my-branch-to-merge into the current␣
↪branch without fast-forward
```

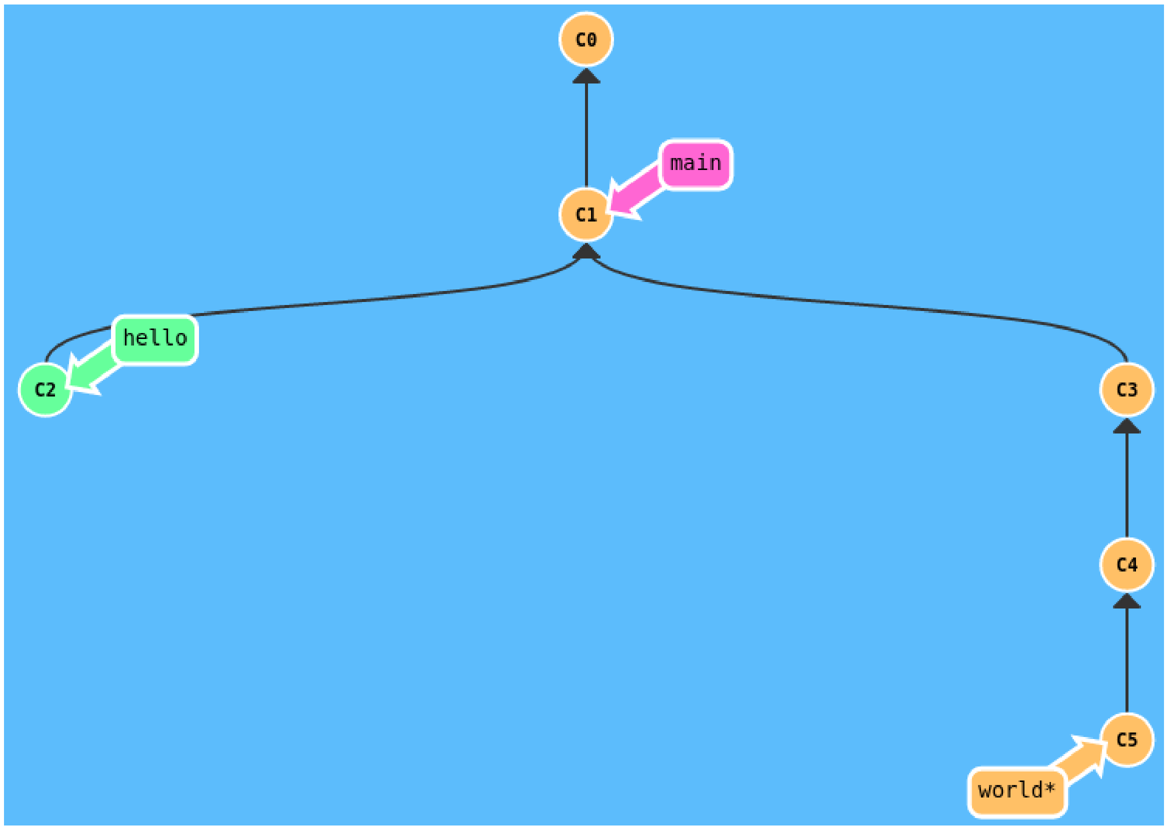


Fig. 2.8: Branch — Add some commits on branch *world*

For local work without issue tracking, choose the *fast-forward* option. It's the one that keeps the cleanest history.

Example:

10. `$ git checkout main` (result shown on Fig. 2.9)

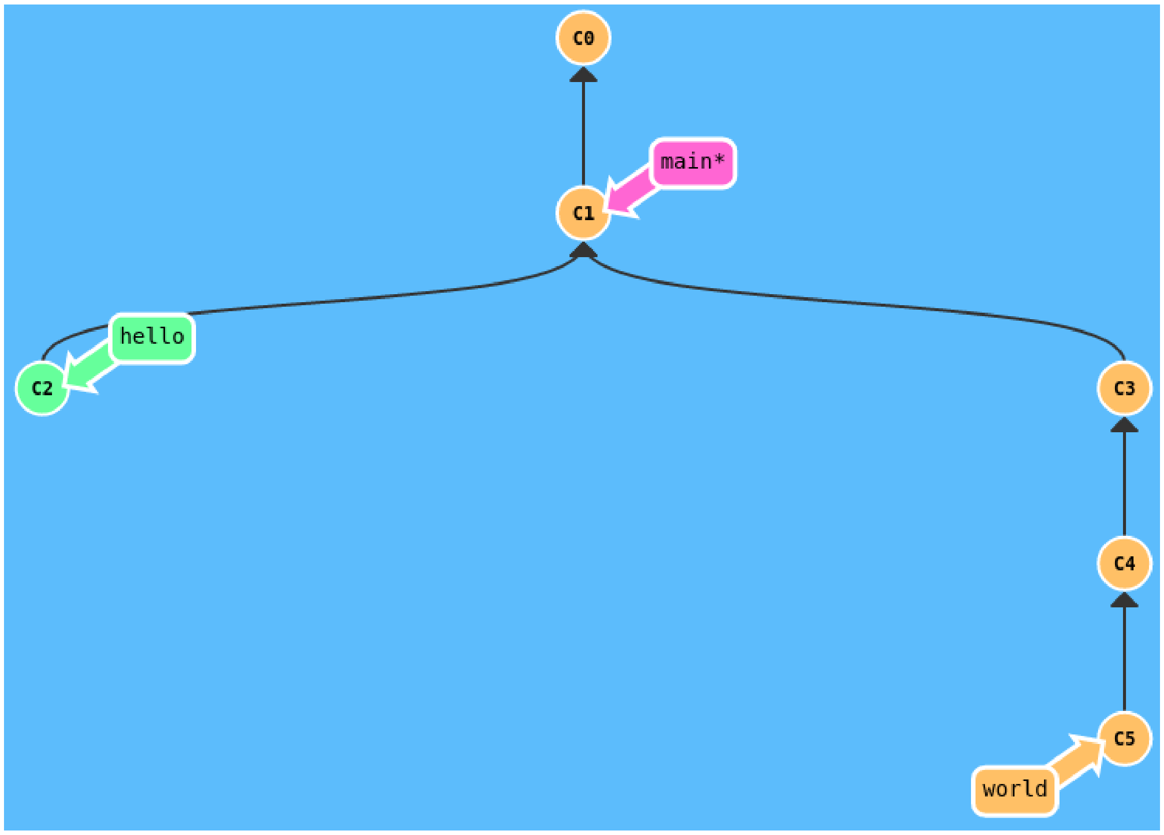


Fig. 2.9: Merge — Switch to branch *main*

11. `$ git merge world` (result shown on Fig. 2.10)
12. `$ git merge --no-ff hello` (result shown on Fig. 2.11)

2.11 Rebase

Rebase is another kind of merge. While the command git-merge applies the commits of another branch to the current branch, the command git-rebase applies commits of the current branch on top of another branch (and updates the current branch's reference).

The advantage of *rebase* over *merge* is that *rebase* only applies the current branch commits on top of another branch, without modifying the other branch. If commits cannot be reapplied on top of the branch, the user who uses *rebase* manages the conflicts. See *Step 8: Align your work on origin/main* for recomendations on the usage of *rebase* and *merge*.

Example:

1. Branch `foo` is being developed but branch `main` was in the mean time also updated (result shown on Fig. 2.12)

2. Use **`git rebase main`** to rebase `foo` on `main` (result shown on Fig. 2.13)

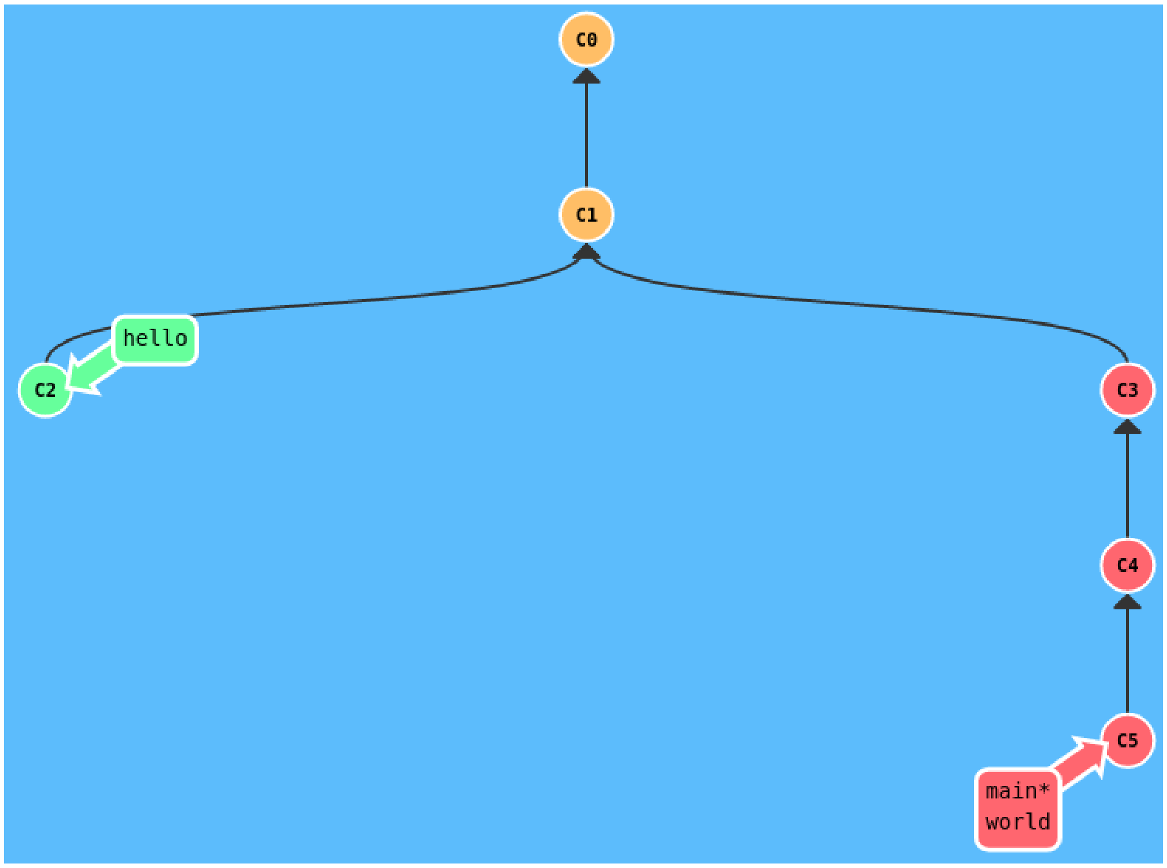


Fig. 2.10: Merge — Merge branch *world* in mode *fast-forward*

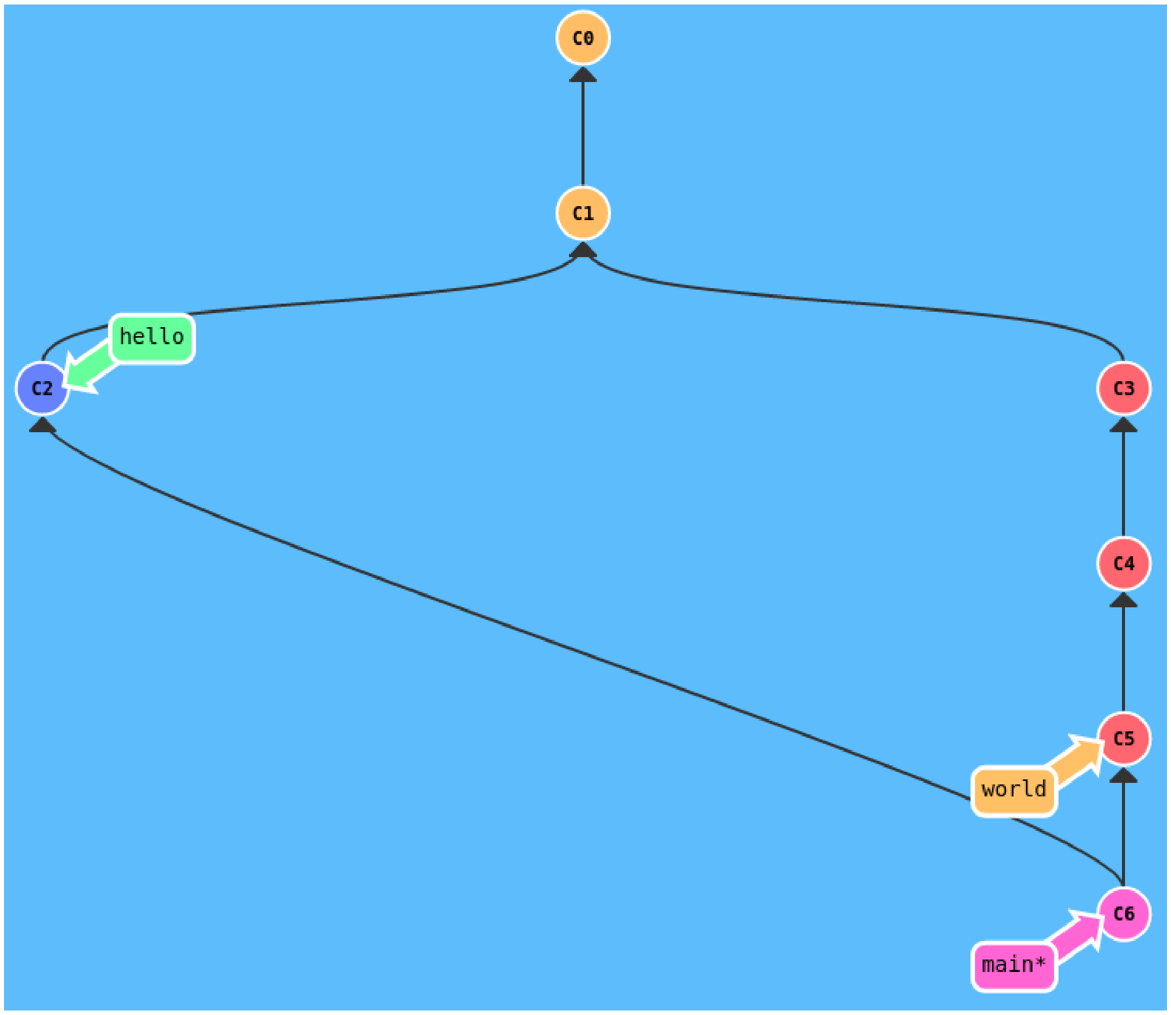


Fig. 2.11: Merge — Merge branch *hello* in mode no *fast-forward*

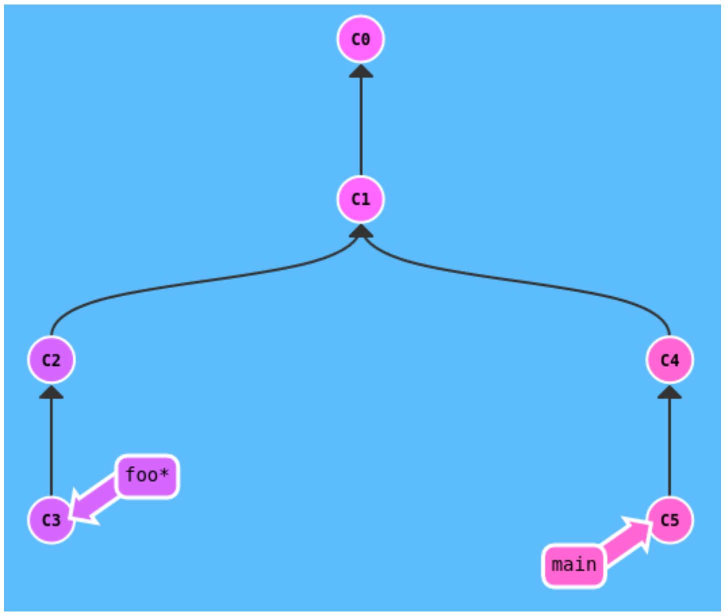


Fig. 2.12: Rebase — Divergence on the repository

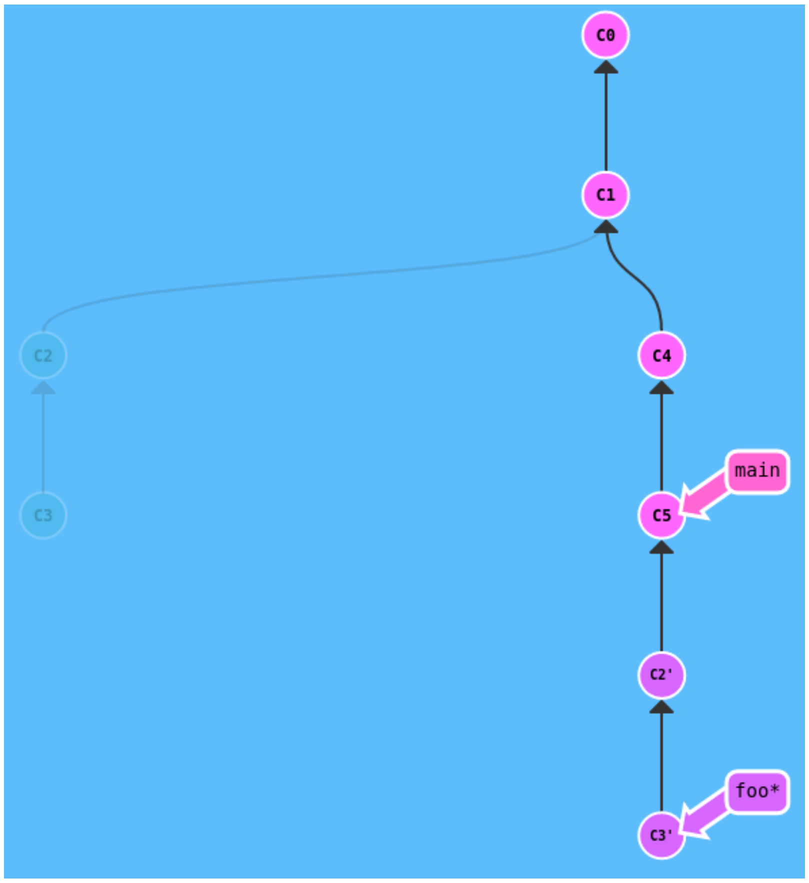


Fig. 2.13: Rebase — Rebase branch *foo* on *main*

2.11.1 What if someone rebased a common branch ?

Rebasing a common branch changes the history of this branch and thus its reference. When someone pulls it, the behaviour depends on the Git configuration. Usually, the remote branch is merged into the local branch, which is not the expected behaviour when rebase is used!

To keep the benefit of the rebase, use instead:

Listing 2.20: Handle a rebased origin branch (1)

```
$ git pull --rebase
```

or

Listing 2.21: Handle a rebased origin branch (2)

```
$ git fetch
$ git reset --hard origin/<branche>
```

2.11.2 What if I have a conflict while rebasing ?

See *Conflict management*.

2.12 Stash

It may happen during the development that one must quickly use another branch or another status of the project. Three options are available to the developer to get rid of his/her current work and switch to another state:

- Reset the working directory with **`git reset --hard HEAD`** which removes all modifications: a bit extrem;
- Create a commit to save the working directory with **`git commit -a -m "WIP"`**: is a solution but allows only one WIP-commit per branch;
- Use the command git-stash which saves the local modifications in a "temporary" commit, allowing to restore a non-dirty working directory.

A stash is associated to a commit, which means that the stash contains the diff with the associated commit, but the stash can be re-applied to any working directory. Conflicts may need to be resolved.

- Use **`git stash list`** to list all stashes of the repository

  Format: `stash@{N}: On BRANCH: MESSAGE`

  - `N`: stash ID
  - `BRANCH`: branch name on which the stash was created
  - `MESSAGE`: stash message given at stash creation

  Example: `stash@{0}: On 1-release-first-version: Working on ...`

- Use **`git stash save`** to stash the working directory; message is optional
- Use **`git stash pop [id]`** to re-apply the selected stash and drop it (unless there is a conflict); if `id` is not mentionned, use the most recent one

- Use **`git stash apply [id]`** to re-apply the selected stash; if `id` is not mentionned, use the most recent one
- Use **`git stash drop [id]`** to drop the selected stash; if `id` is not mentionned, use the most recent one

Example:

Listing 2.22: Full example of **`git stash`**

```
$ git status
On branch 1-release-first-version
Your branch is up to date with 'origin/1-release-first-version'.

Changes to be committed:
(use "git restore --staged <file>..." to unstage)
    modified:   docs/source/basics.rst

Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
    modified:   docs/source/basics.rst

Untracked files:
(use "git add <file>..." to include in what will be committed)
    docs/source/advanced.rst

$ git pull
error: cannot pull with rebase: You have unstaged changes.
error: additionally, your index contains uncommitted changes.
error: please commit or stash them.

$ git stash list

$ git stash save "Working on ..."
Saved working directory and index state On 1-release-first-version: Working on ...

$ git status
On branch 1-release-first-version
Your branch is up to date with 'origin/1-release-first-version'.

Untracked files:
(use "git add <file>..." to include in what will be committed)
    docs/source/advanced.rst

nothing added to commit but untracked files present (use "git add" to track)

$ git stash list
stash@{0}: On 1-release-first-version: Working on ...

$ git stash pop
On branch 1-release-first-version
Your branch is up to date with 'origin/1-release-first-version'.

Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
```

(continues on next page)

(continued from previous page)

```
(use "git restore <file>..." to discard changes in working directory)
    modified:   docs/source/basics.rst

Untracked files:
(use "git add <file>..." to include in what will be committed)
    docs/source/advanced.rst

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (b1bb031a66d6d3b2d5e23411712cb993c8875c45)

$ git stash list
```

2.13 Conflict management

When merging, rebasing, or applying a stash on a code base that differs from the one being applied, conflicts may happen. Git tries to resolve conflicts on its own, but sometimes cannot. In such a case, it will tell you that something requires your attention.

2.13.1 Regular files

The command git-mergetool runs merge conflict resolution tools to resolve merge conflicts for non-binary files.

Example:

On branch *test*, the last commit looks like

Listing 2.23: Diff of branch *test* for example of conflict management

```
diff --git a/docs/source/basics.rst b/docs/source/basics.rst
index bc07bc2..1c64b14 100644
--- a/docs/source/basics.rst
+++ b/docs/source/basics.rst
@@ -6,8 +6,8 @@ Git --- Basics

.. _git-scm-book: https://git-scm.com/book/en/v2/

-Theory
-======
+Theories
+========

Setup
-----
```

On branch *1-release-first-version*, the last commit looks like

Listing 2.24: Diff of branch *1-release-first-version* for example of conflict management

```
diff --git a/docs/source/basics.rst b/docs/source/basics.rst
index bc07bc2..4de96c4 100644
```

(continues on next page)

(continued from previous page)

```
--- a/docs/source/basics.rst
+++ b/docs/source/basics.rst
@@ -6,9 +6,6 @@ Git --- Basics

.. _git-scm-book: https://git-scm.com/book/en/v2/

-Theory
-======
-
Setup
-----
```

The user wants to merge branch *test* into *1-release-first-version* but gets a conflict and must resolve it.

Listing 2.25: Full example of **git mergetool** — start of merge conflict resolution

```
$ git merge test
Auto-merging docs/source/basics.rst
CONFLICT (content): Merge conflict in docs/source/basics.rst
Automatic merge failed; fix conflicts and then commit the result.

$ git status
On branch 1-release-first-version
Your branch is ahead of 'origin/1-release-first-version' by 1 commit.
(use "git push" to publish your local commits)

You have unmerged paths.
(fix conflicts and run "git commit")
(use "git merge --abort" to abort the merge)

Unmerged paths:
(use "git add <file>..." to mark resolution)
    both modified:   basics.rst

$ git mergetool
Merging:
docs/source/basics.rst

Normal merge conflict for 'docs/source/basics.rst':
{local}: modified file
{remote}: modified file
```

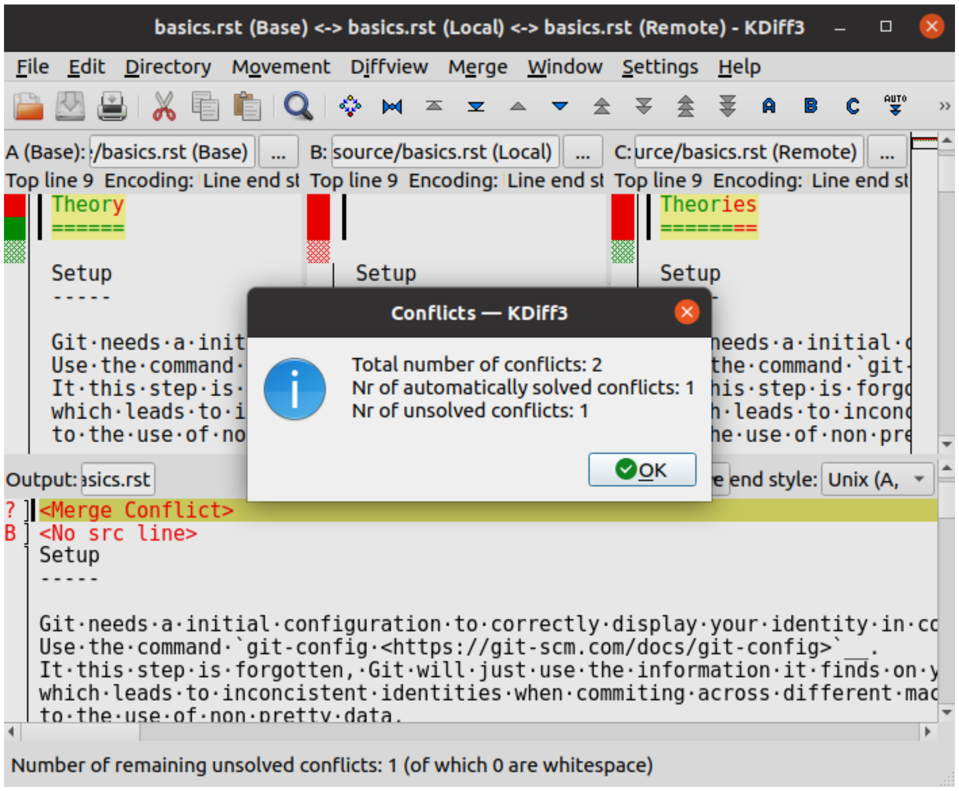
The last command opens the merge tool where you can solve conflicts (result shown on Fig. 2.14 and Fig. 2.15):

Save the file, close the merge tool and finish the conflict resolution process:

Listing 2.26: Full example of **git mergetool** — end of merge conflict resolution

```
$ git status
On branch 1-release-first-version
Your branch is ahead of 'origin/1-release-first-version' by 1 commit.
```

(continues on next page)


basics.rst (Base) <-> basics.rst (Local) <-> basics.rst (Remote) - KDiff3
File Edit Directory Movement Diffview Merge Window Settings Help
A (Base): :/basics.rst (Base)
B: source/basics.rst (Local)
C: urce/basics.rst (Remote)
Top line 9 Encoding: Line end st
Conflicts — KDiff3
OK
Output: ısics.rst
e end style: Unix (A,

```
Theory
======

Setup
-----

Git·needs·a·init
Use·the·command·
It·this·step·is·
which·leads·to·i
to·the·use·of·no
```

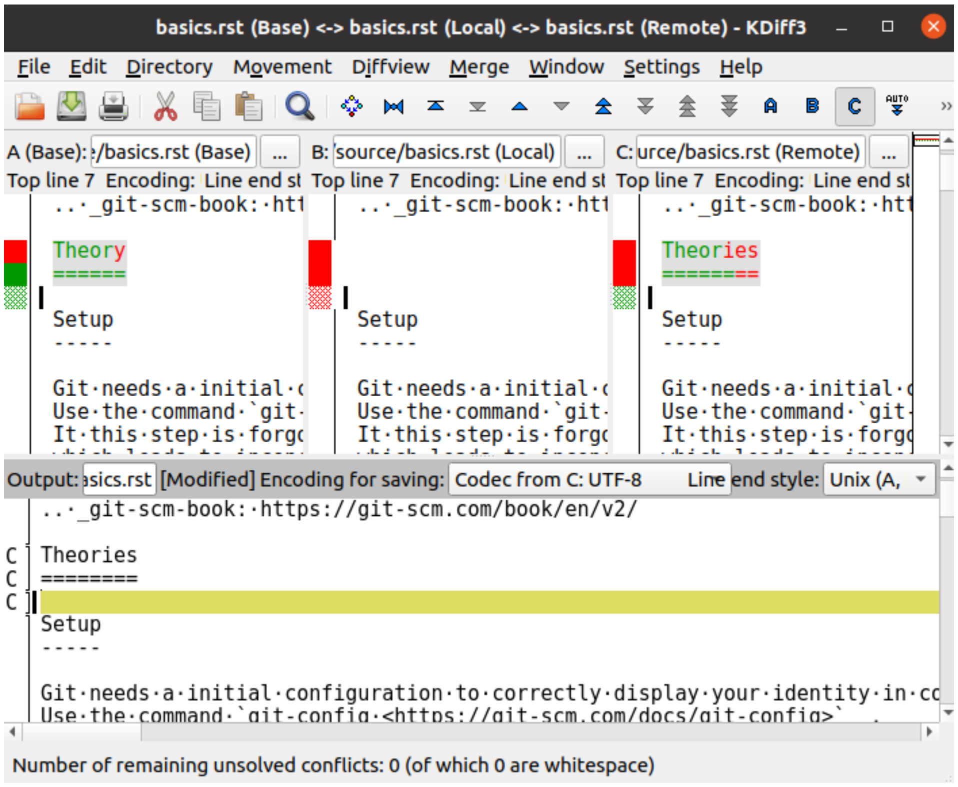
```
Theories
========

Setup
```

Total number of conflicts: 2
Nr of automatically solved conflicts: 1
Nr of unsolved conflicts: 1

```
? <Merge Conflict>
B <No src line>
  Setup
  -----

  Git·needs·a·initial·configuration·to·correctly·display·your·identity·in·co
  Use·the·command·`git-config·<https://git-scm.com/docs/git-config>`__.
  It·this·step·is·forgotten,·Git·will·just·use·the·information·it·finds·on·y
  which·leads·to·inconcistent·identities·when·commiting·across·different·mac
  to·the·use·of·non·pretty·data.
```

Number of remaining unsolved conflicts: 1 (of which 0 are whitespace)

Fig. 2.14: Conflict management — Opening of the merging tool


basics.rst (Base) <-> basics.rst (Local) <-> basics.rst (Remote) - KDiff3
File Edit Directory Movement Diffview Merge Window Settings Help
A (Base): :/basics.rst (Base)
B: source/basics.rst (Local)
C: urce/basics.rst (Remote)
Top line 7 Encoding: Line end st
Output: ısics.rst [Modified] Encoding for saving: Codec from C: UTF-8 Line end style: Unix (A,

```
..·_git-scm-book:·htt

Theory
======

Setup
-----

Git·needs·a·initial·c
Use·the·command·`git-
It·this·step·is·forgc
```

```
..·_git-scm-book:·https://git-scm.com/book/en/v2/

C Theories
C ========
C
  Setup
  -----

  Git·needs·a·initial·configuration·to·correctly·display·your·identity·in·co
  Use·the·command·`git-config·<https://git-scm.com/docs/git-config>`__.
```

Number of remaining unsolved conflicts: 0 (of which 0 are whitespace)

Fig. 2.15: Conflict management — Conflict resolution in the merging tool

(continued from previous page)

```
(use "git push" to publish your local commits)

All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)

Changes to be committed:
    modified:   basics.rst

$ git commit
[1-release-first-version 2410592] Merge branch 'test' into 1-release-first-version
```

2.13.2 Binary files

Unfortunatelly, the command git-mergetool is useless for binary files. Instead use git-checkout [--ours|--theirs] to retrieve the correct version of the file, followed by a git-add.

Note that during a *rebase* operation, `--ours` gives the version from the branch that is being rebased onto, while `--theirs` gives the version from your branch.

Example while rebasing a branch:

Listing 2.27: Example of using **git checkout** to solve conflicts on binary files while rebasing a branch

```
$ git rebase origin/main
warning: Cannot merge binary files: Model/model.ang (HEAD vs. 85fc8d0 (my commit))
Auto-merging Model/model.ang
CONFLICT (content): Merge conflict in Model/model.ang
Auto-merging Simulation/simulation.py
error: could not apply 85fc8d0... my commit
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply 85fc8d0... my commit

$ git status
interactive rebase in progress; onto f0fbaaa
Last command done (1 command done):
   pick 85fc8d0 my commit
No commands remaining.
You are currently rebasing branch 'test' on 'f0fbaaa'.
(fix conflicts and then run "git rebase --continue")
(use "git rebase --skip" to skip this patch)
(use "git rebase --abort" to check out the original branch)

Changes to be committed:
(use "git restore --staged <file>..." to unstage)
   modified:   Simulation/simulation.py

Unmerged paths:
(use "git restore --staged <file>..." to unstage)
```

(continues on next page)

(continued from previous page)

```
(use "git add <file>..." to mark resolution)
   both modified:   Model/model.ang

$ git checkout --theirs -- Model/model.ang # to keep my changes!
$ git add Model/model.ang

$ git status
interactive rebase in progress; onto f0fbaaa
Last command done (1 command done):
   pick 85fc8d0 my commit
No commands remaining.
You are currently rebasing branch 'test' on 'f0fbaaa'.
(all conflicts fixed: run "git rebase --continue")

Changes to be committed:
(use "git restore --staged <file>..." to unstage)
   modified:   Model/model.ang
   modified:   Simulation/simulation.py

$ git rebase --continue
[detached HEAD c030a63] test
 2 files changed, 1 insertion(+)
 rewrite Model/model.ang (95%)
Successfully rebased and updated refs/heads/test.
```

2.14 Ignore files

File `.gitignore` lists files that must be ignored by Git. Git will not show them in the list of untracked files and will not track them. Usually a single `.gitignore` file is present at the root folder of the Git repository but a repository may contain several `.gitignore` in different folders.

Example:

Listing 2.28: Example of content of `.gitignore`

```
docs/build/
__pycache__/
*.o
```

2.15 Exercises

On your machine, configure Git so that it uses your identity.

Create a folder for this exercise and execute the steps below. For each step, write down which command(s) you used.

1. Initialise a Git repository
2. Create a file `hello.txt` and add some content
3. Track `hello.txt` and commit it
4. Display the log

   - on which branch points `HEAD` ?
   - who did the last commit and when ?
   - what was commited ?
5. Modify `hello.txt`
6. Display the modifications you made
7. Stage the current changes to `hello.txt`
8. Modify again `hello.txt`
9. Display the modifications you made
   - have you display all modifications ?
   - which modifications will be part of the next commit ?
   - which modifications will not be part of the next commit ?
10. Commit your changes
11. Are the areas clean ? If not why ?
12. Discard all changes
13. Display the log
    - on which branch points `HEAD` ?
    - who did the last commit and when ?
    - what was commited ?
    - can you understand what was commited ?
14. Create a branch `A`, add some files and make a commit
15. Merge branch `A` into `main`
16. Display the log
    - what happened?
17. Create a branch `C` but do not check it out
18. Create a branch `B`, add some files and make a commit
19. Merge branch `B` into `main` without *fast-forward*
20. Display the log (with graph)
    - what happened?
    - what is different from the previous merge ?
21. Open a file and modify it; finally stash the modifications away
22. Check out branch `C` and rebase it on top of `main`
23. Display the log
    - what happened?
    - what is different from the previous merge ?
24. Pop the stash previously created
    - what happened?

CHAPTER

THREE

GIT — GUI

Git can be used with command line, but can also be used with a graphical user interface.

3.1 Linux

A list of (all) GUI clients on Linux can be found on the Git homepage.

Suggestions for this platform:

- gitg is an external program; it is graphically basic but allows to do most of the work. It is anyway useful to visualize the history of the repository.

  Listing 3.1: Install and use **`gitg`** on Linux

  ```
  $ sudo apt install gitg
  $ gitg
  ```

- qgit is an external program; it is graphically basic but allows to do most of the work. It is anyway useful to visualize the history of the repository.

  Listing 3.2: Install and use **`qgit`** on Linux

  ```
  $ sudo apt install qgit
  $ qgit
  ```

- tig is an ncurses-based external program mainly designed to visualize the history of the repository.

Listing 3.3: Install and use **tig** on Linux

```
$ sudo apt install tig
$ tig
```

3.2 MacOS

A list of (all) GUI clients on MacOS can be found on the Git homepage.

Suggestions for this platform:

- Git Fork: the best client ever!

3.3 Windows

A list of (all) GUI clients on Windows can be found on the Git homepage.

Suggestions for this platform:

- Git Fork: the best client ever!

3.4 All platforms

- gitk and git-gui are built-in GUI tools. **git-gui** is used for committing and **gitk** for browsing.

Listing 3.4: Install and use **gitk** and **git-gui** on Linux

```
$ sudo apt install gitk git-gui
$ gitk
$ git gui
```

- git instaweb browses your working repository in gitweb

Listing 3.5: Use **git instaweb** on Linux

```
$ git instaweb
```

- Extension Git Graph for Visual Studio Code; a nice embedded history graph
- git difftool shows changes using common diff tools

Listing 3.6: Use **git difftool** on Linux: open the repository's root directory in **meld**

```
$ git difftool -t meld -d
```

CHAPTER

FOUR

GIT — WORKFLOW

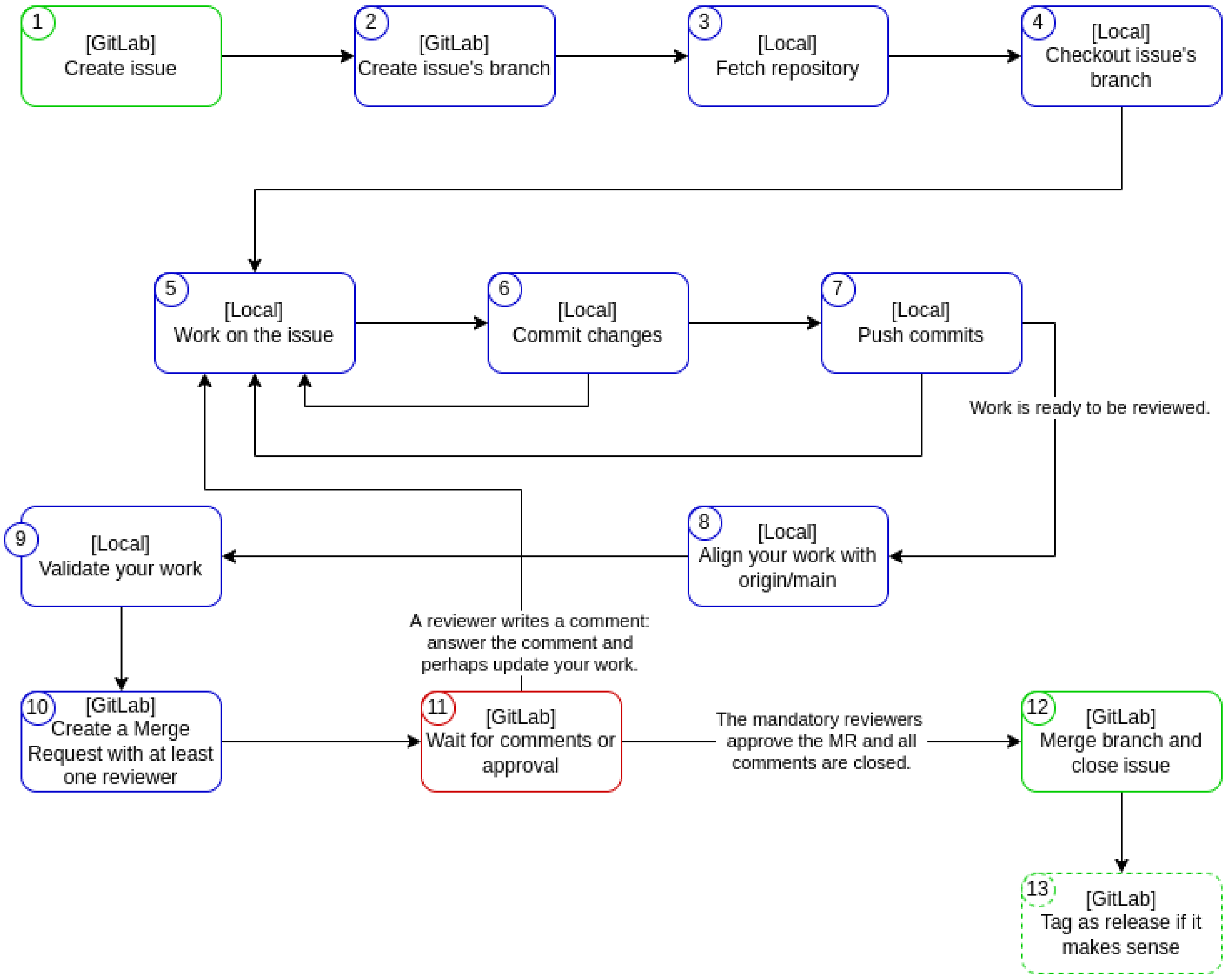
Git has many capabilities to work alone, but Git allows also to work in a team and share resources! Working together requires some workflow to work efficiently though. The workflow presented below is one approach to working together efficiently.

Important: The official reference to the Git Workflow is on Virgil.

- *Workflow*
  - *Step 1: Create an issue in GitLab*
  - *Step 2: Create issue's branch in GitLab*
  - *Steps 3-4: Fetch repository and Checkout issue's branch*
  - *Step 5: Work on the issue*
  - *Step 6: Commit changes*
  - *Step 7: Push commits to the remote*
  - *Step 8: Align your work on* `origin/main`
  - *Step 9: Validate your work*
  - *Step 10: Create a Merge Request in GitLab*
  - *Step 11: Wait for comments or approval*
  - *Step 12: Merge branch and close issue*
  - *Step 13: Tag as release if it makes sense*
- *Useful links to add to project's doc*

4.1 Workflow

To allow working as a team in the GitLab environment, the following git flow shall be used (depicted on Fig. 4.1):



1 [GitLab] Create issue
2 [GitLab] Create issue's branch
3 [Local] Fetch repository
4 [Local] Checkout issue's branch
5 [Local] Work on the issue
6 [Local] Commit changes
7 [Local] Push commits
Work is ready to be reviewed.
8 [Local] Align your work with origin/main
9 [Local] Validate your work
A reviewer writes a comment: answer the comment and perhaps update your work.
10 [GitLab] Create a Merge Request with at least one reviewer
11 [GitLab] Wait for comments or approval
The mandatory reviewers approve the MR and all comments are closed.
12 [GitLab] Merge branch and close issue
13 [GitLab] Tag as release if it makes sense

Fig. 4.1: Workflow

4.1.1 Step 1: Create an issue in GitLab

On the menu panel of the project in GitLab, open the menu *Issues* → *New issue* (see Fig. 4.2).

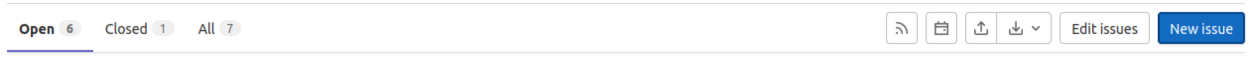


Fig. 4.2: Workflow — Step 1 — Create new issue

Tip: Who executes this step? Usually the project manager.

4.1.2 Step 2: Create issue's branch in GitLab

On the menu panel of the project in GitLab, open the menu *Issues* → *List* if not already open (see Fig. 4.3).

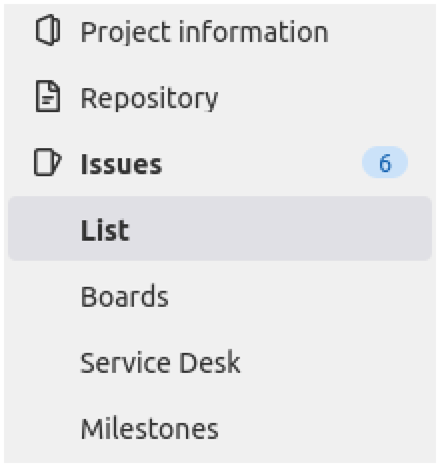
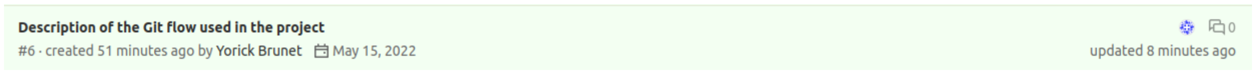


Fig. 4.3: Workflow — Step 2 — Menu to display the list of issues

Select the issue you will work on, for example (see Fig. 4.4).



Description of the Git flow used in the project
#6 · created 51 minutes ago by Yorick Brunet   May 15, 2022
0
updated 8 minutes ago

Fig. 4.4: Workflow — Step 2 — Display of an issue in the list

Then click on **the little arrow 'v' of the button** *Create merge request* (see Fig. 4.5).

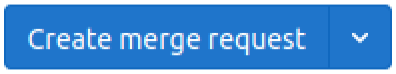


Fig. 4.5: Workflow — Step 2 — Use the little arrow on the button *Create merge request* to create a branch

Select **Create branch** instead of *Create merge request and branch* and click on the button *Create branch* (see Fig. 4.6).

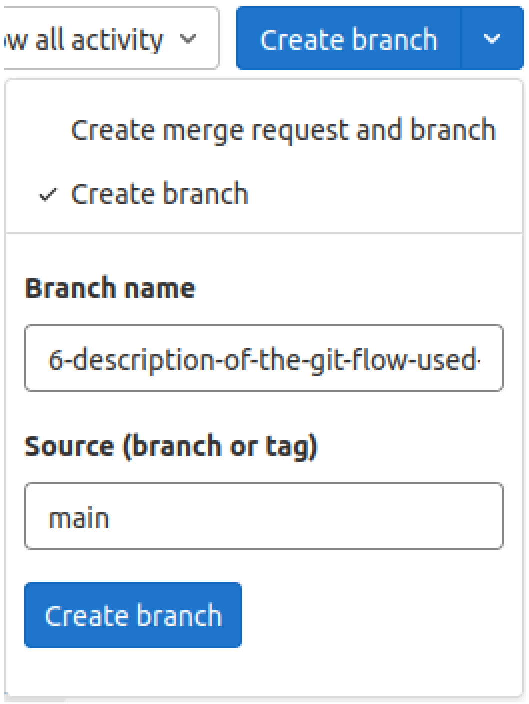
Tip: Who executes this step? The developer.

4.1.3 Steps 3-4: Fetch repository and Checkout issue's branch

Fetch the repository to see the new branch on the remote repository and check it out.

```
$ git fetch
$ git checkout <branch> # e.g. 6-description-of-the-git-flow-used-in-the-project
```

Tip: Who executes this step? The developer.



Branch name

6-description-of-the-git-flow-used

Source (branch or tag)

main

Fig. 4.6: Workflow — Step 2 — Create branch

4.1.4 Step 5: Work on the issue

Work on the issue, develop your code and update the documentation with your favorite IDE. For each step of the issue, execute *Step 6: Commit changes* to make small atomic changes.

Tip: Who executes this step? The developer.

4.1.5 Step 6: Commit changes

Commit regularly when a step is done. Then continue *Step 5: Work on the issue* (push later) or *Step 7: Push commits to the remote* now.

Tip: Who executes this step? The developer.

4.1.6 Step 7: Push commits to the remote

Push the commits regularly to have a “backup”. Then continue *Step 5: Work on the issue* or *Step 8: Align your work on origin/main* if the work is ready to be reviewed.

Tip: Who executes this step? The developer.

4.1.7 Step 8: Align your work on `origin/main`

Before creating the Merge Request (and after having worked on the comments of the reviewers if this action took too much time), re-align your work on `origin/main` to avoid merge conflicts.

There are three possibilities:

1. there were no merge since you created your branch or merges do not affect the same piece of code: **nothing to do**;
2. there were merges which affect the same piece of code and only you contributed to your branch: **rebase** your branch on `origin/main`;

   ```
   # Make a commit, the workspace must not be dirty
   $ git fetch
   $ git rebase origin/main
   ```

3. there were merges which affect the same piece of code and other persons also contributed to your branch: **merge** `origin/main` into your branch.

   ```
   # Make a commit, the workspace must not be dirty
   $ git fetch
   $ git merge origin/main
   ```

In case of conflict, GitLab will anyway forbid the merge (not the Merge Request) and you will have to go through possibilities 2 or 3 to solve these conflicts.

Continue with *Step 9: Validate your work*.

Tip: Who executes this step? The developer.

4.1.8 Step 9: Validate your work

Validate your work by running the different tests and linters provided by the project, even though a CI exists and runs the tests anyway.

Continue with *Step 10: Create a Merge Request in GitLab*, or *Step 11: Wait for comments or approval* if the Merge Request exists already.

Tip: Who executes this step? The developer.

4.1.9 Step 10: Create a Merge Request in GitLab

When the work is ready to be reviewed, create the Merge Request.

On the menu panel of the project in GitLab, open the menu *Merge Request* (see Fig. 4.7).

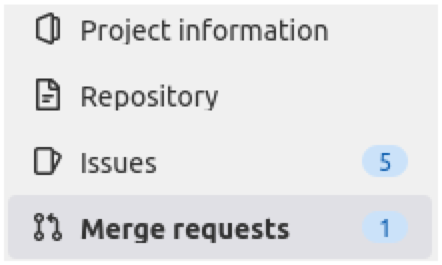


Fig. 4.7: Workflow — Step 10 — Menu to display the list of merge requests

Click on *Create merge request* in the box that appeared at the top of the page (see Fig. 4.8).

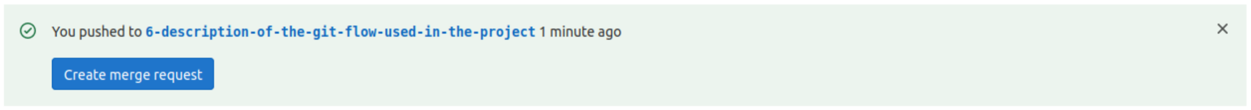
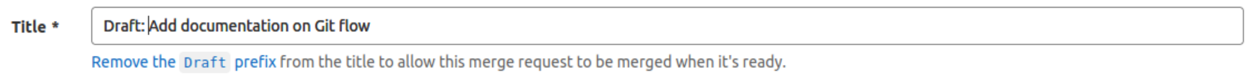

You pushed to 6-description-of-the-git-flow-used-in-the-project 1 minute ago

Fig. 4.8: Workflow — Step 10 — Initiate the creation of the merge request

Fill in the *title field* with a short self-explaining text (see Fig. 4.9).


Title * Draft: Add documentation on Git flow

Remove the Draft prefix from the title to allow this merge request to be merged when it's ready.

Fig. 4.9: Workflow — Step 10 — Fill in merge request title

Note: Adding 'Draft: ' to the title is a good behaviour as it forbids an unfortunate merge while the request is being reviewed.

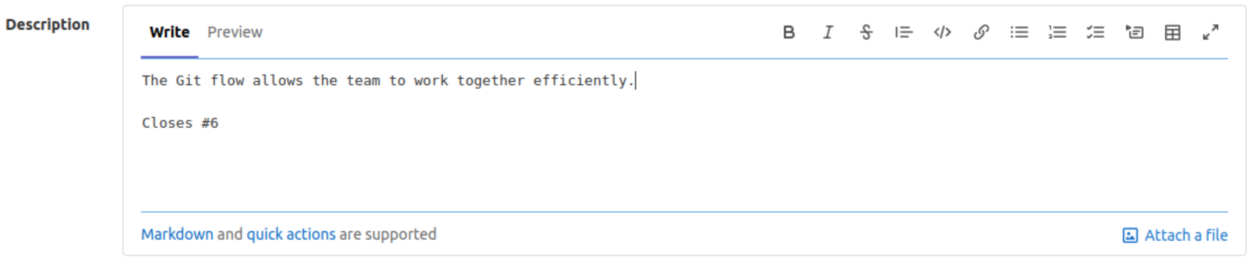
And complete the *description field* if necessary (see Fig. 4.10).

Finally, assign a least one reviewer (see Fig. 4.11).

And click on the button *Create merge request* (see Fig. 4.12).

Warning: The current license of GitLab allows to assign only one reviewer.

Tip: Who executes this step? The developer.



```
The Git flow allows the team to work together efficiently.

Closes #6
```

Fig. 4.10: Workflow — Step 10 — Fill in merge request description

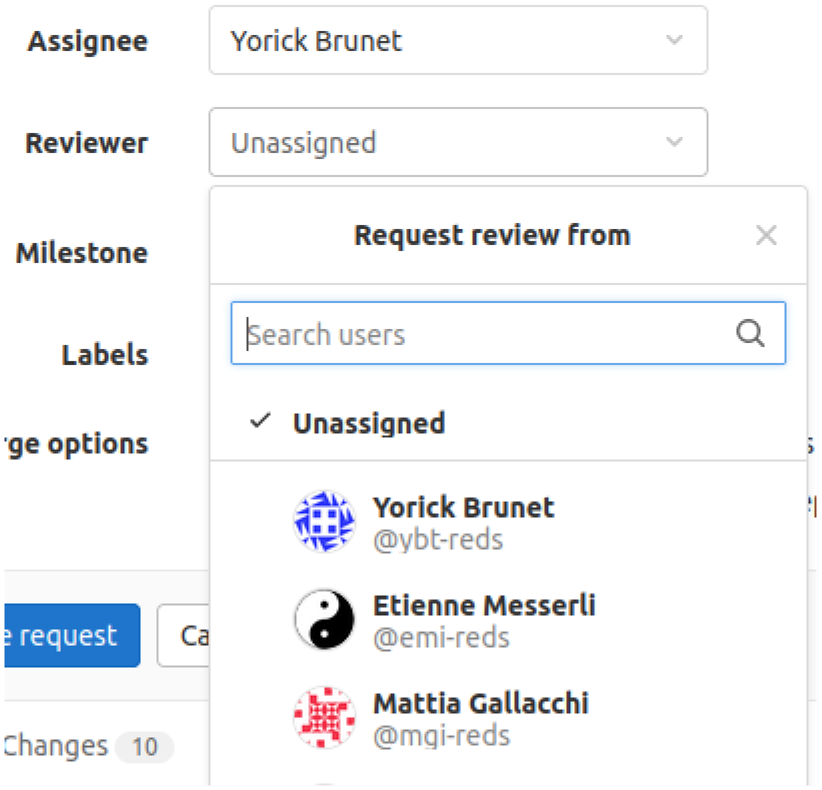


Fig. 4.11: Workflow — Step 10 — Assign reviewer

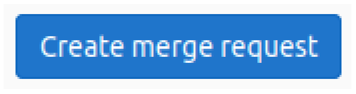


Fig. 4.12: Workflow — Step 10 — Create the merge request

4.1.10 Step 11: Wait for comments or approval

Wait for comments or approval of the reviewers.

In case of comments, go back to *Step 5: Work on the issue* to solve those comments:

- the developer answers the comment in GitLab
- the reviewer closes the comment in GitLab if the answer is accepted

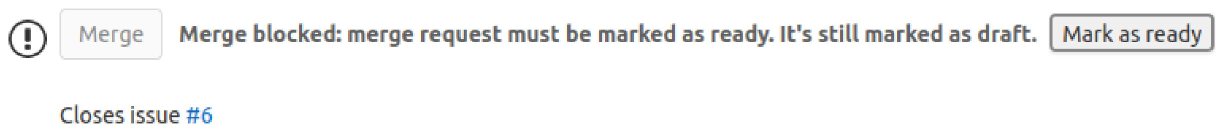
When everything is solved, proceed with *Step 12: Merge branch and close issue*.

Tip: Who execute this step? The reviewers.

4.1.11 Step 12: Merge branch and close issue

When the mandatory reviewers have approved the Merge Request and all comments are closed, the merge request can be merged.

If the merge request was created as draft, first mark it as ready (see Fig. 4.13).



Merge blocked: merge request must be marked as ready. It's still marked as draft.

Closes issue #6

Fig. 4.13: Workflow — Step 12 — Mark merge request as ready if it's still in draft mode

And then validate the merge (see Fig. 4.14).

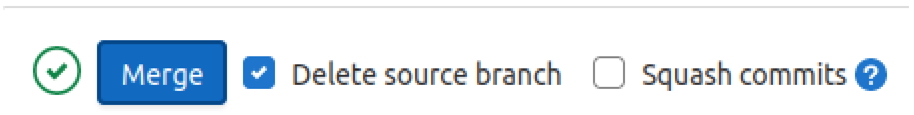


Fig. 4.14: Workflow — Step 12 — Execute merge

The **source branch should be deleted** (keep box checked) so that obsolete branches are not kept on the repository. The commits may be squashed depending on the work that has been done.

Depending on the state of the project, continue with *Step 13: Tag as release if it makes sense*.

Tip: Who executes this step? Usually the project manager (who has a maintainer role).

4.1.12 Step 13: Tag as release if it makes sense

Sometimes in the life of a project, it makes sense to tag a known and working state of the project as *release*.

On the menu panel of the project in GitLab, open the menu *Repository* → *Tags*. A list of all tags is shown.

To add a tag, click on the button *New tag* on the top right corner of the page (see Fig. 4.15):

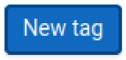


Fig. 4.15: Workflow — Step 13 — Button to create new tag

and provide:

- *Tag name* → suggested format is *v<MAJOR>.<MINOR>.<BUGFIX>*, e.g. *v1.2.3*
- *Release note* → this creates an official release and not only a tag (e.g. REDS Tools releases)
- *Message*, optionaly

Create the tag from the `main` branch.

Validate by clicking on *Create tag*.

Tip: Who executes this step? Usually the project manager (who has a maintainer role).

4.2 Useful links to add to project's doc

Listing 4.1: Useful links to target actions on GitLab

```
open `a new issue <link_newissue_>`_
open `issues <link_issues_>`_
open `the list of merge requests <link_mergerequest_>`_
open `the pipelines list <link_pipelines_>`_
open `the release list <link_releases_>`_

.. _link_newissue:       https://reds-gitlab.heig-vd.ch/reds-heigvd/<project name>/-/
↪issues/new
.. _link_issues:         https://reds-gitlab.heig-vd.ch/reds-heigvd/<project name>/-/
↪issues
.. _link_mergerequest:   https://reds-gitlab.heig-vd.ch/reds-heigvd/<project name>/-/
↪merge_requests
.. _link_pipelines:      https://reds-gitlab.heig-vd.ch/reds-heigvd/<project name>/-/
↪pipelines
.. _link_releases:       https://reds-gitlab.heig-vd.ch/reds-heigvd/<project name>/-/
↪releases
```

CHAPTER

FIVE

GITLAB

This chapter presents the interaction with the DevOps platform GitLab, because the REDS uses a GitLab instance. However, the fundamental notions presented here are applicable for other DevOps platforms, such as GitHub, Bitbucket, Azure DevOps, etc.

The reference page on Virgil about our GitLab instance is here.

A DevOps platform agregates tools around Git repositories and provides a web interface. The platform uses access rights for groups and users. The platform hosts many Git repositories, named *projects*, and manages their access rights. Each user can have several roles depending on the project: owner, maintainer, developer, reporter or guest. In addition, the platform usually provides the following tools:

- issue tracking
- merge requests
- CI/CD pipeline
- wiki

Issue tracking and merge requests are typically used for the *Git — Workflow*.

The CI/CD pipeline allows to run actions on Git events such as push and merge.

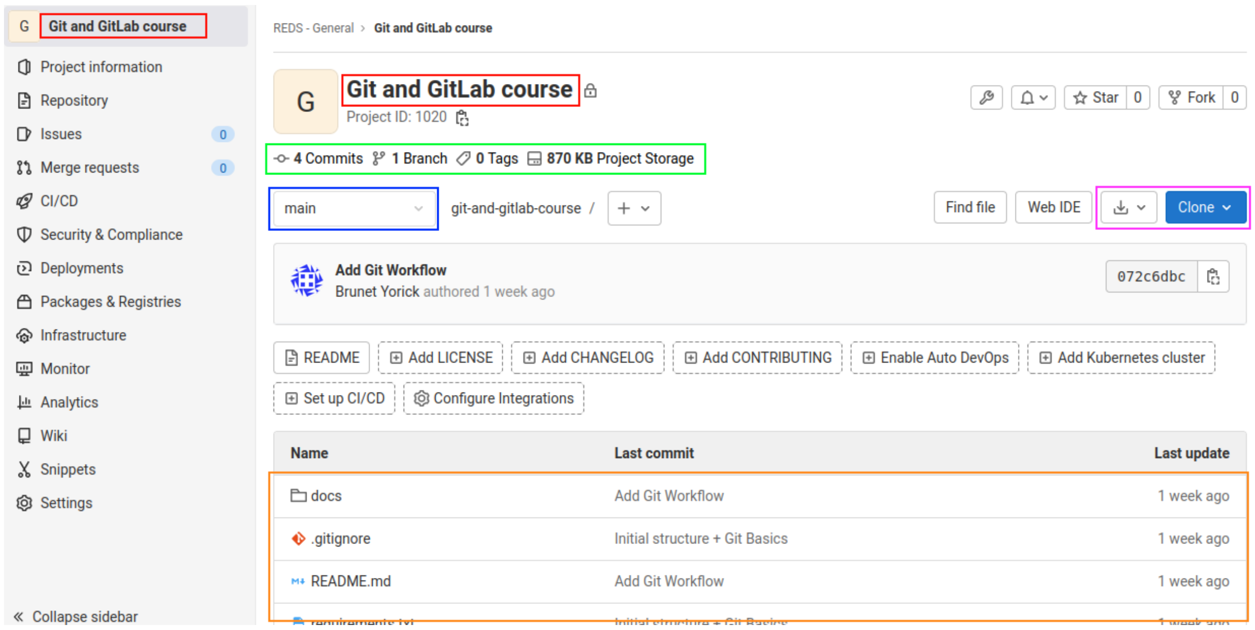
The wiki allows easy access for all members of the projects to instant documentation.

The synchronisation between GitLab and the local Git repository will be seen in chapter *Git — Sync*.

5.1 Homepage

When opening a project, GitLab displays the project's homepage which contains many details (see Fig. 5.1); among others:

- project name
- commits, branches, tags, releases and size
- branches available
- list of files of the selected branch
- download and clone options



Name	Last commit	Last update
docs	Add Git Workflow	1 week ago
.gitignore	Initial structure + Git Basics	1 week ago
README.md	Add Git Workflow	1 week ago

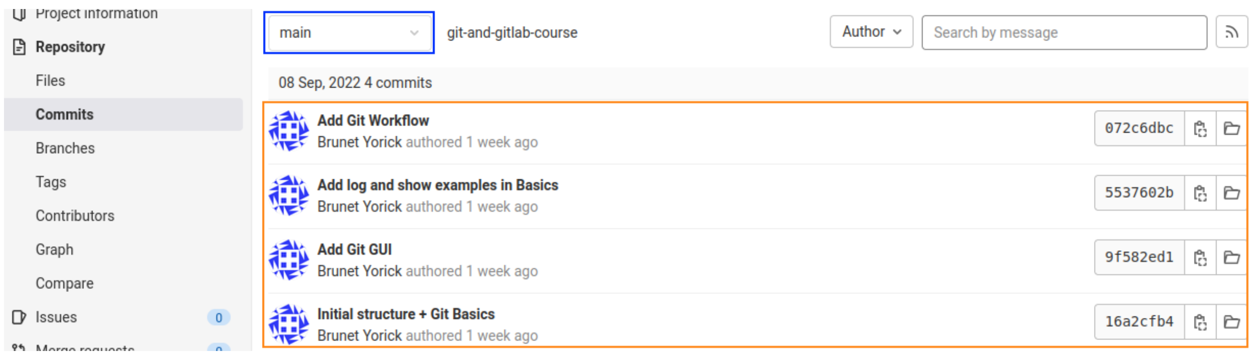
Fig. 5.1: GitLab — Homepage — Overview

5.2 Project information

- Use menu *Project information* → *Members* to manage project's members

5.3 Repository

- Use menu *Repository* → *Files* to see the same files overview as on the homepage
- Use menu *Repository* → *Commits* to show the commits of the selected branch (see Fig. 5.2)
  - branches available
  - list of commits of the selected branch



08 Sep, 2022 4 commits

Add Git Workflow
Brunet Yorick authored 1 week ago
072c6dbc

Add log and show examples in Basics
Brunet Yorick authored 1 week ago
5537602b

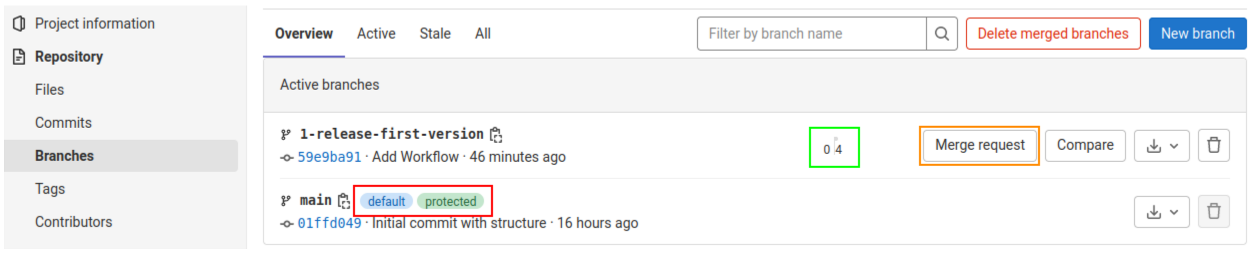
Add Git GUI
Brunet Yorick authored 1 week ago
9f582ed1

Initial structure + Git Basics
Brunet Yorick authored 1 week ago
16a2cfb4

Fig. 5.2: GitLab — Repository — List of commits

- Use menu *Repository* → *Branches* to show the repository branches (see Fig. 5.3)
  - labels of branch

- behind/ahead commits based on the default branch, usually `main` or `master`
- create a merge request

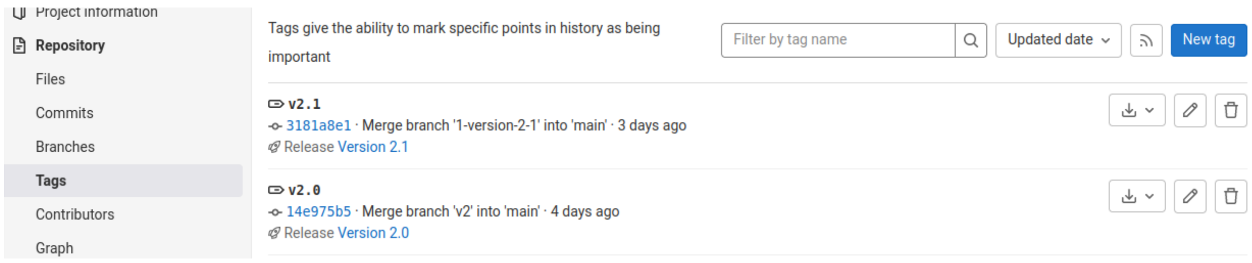


1-release-first-version
59e9ba91 · Add Workflow · 46 minutes ago

main
01ffd049 · Initial commit with structure · 16 hours ago

Fig. 5.3: GitLab — Repository — List of branches

- Use menu *Repository* → *Tags* to show the repository tags (see Fig. 5.4)



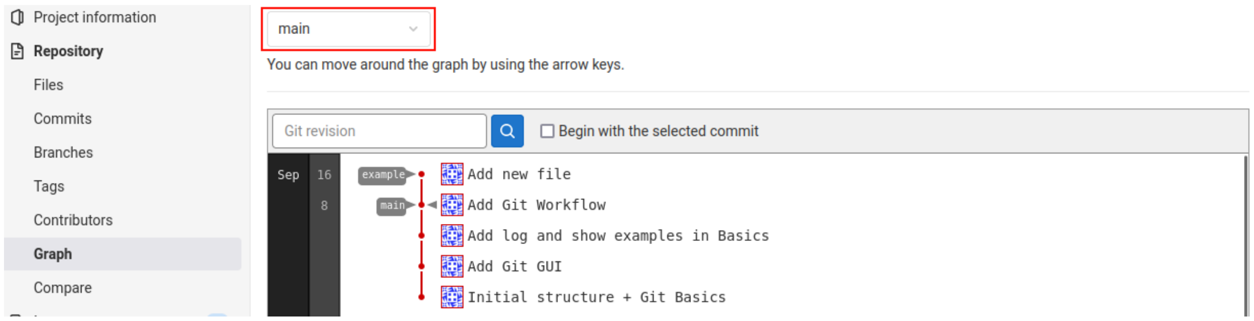
Tags give the ability to mark specific points in history as being important

v2.1
3181a8e1 · Merge branch '1-version-2-1' into 'main' · 3 days ago
Release Version 2.1

v2.0
14e975b5 · Merge branch 'v2' into 'main' · 4 days ago
Release Version 2.0

Fig. 5.4: GitLab — Repository — List of tags

- Use menu *Repository* → *Graph* to show the repository graph (see Fig. 5.5)
  - branches available

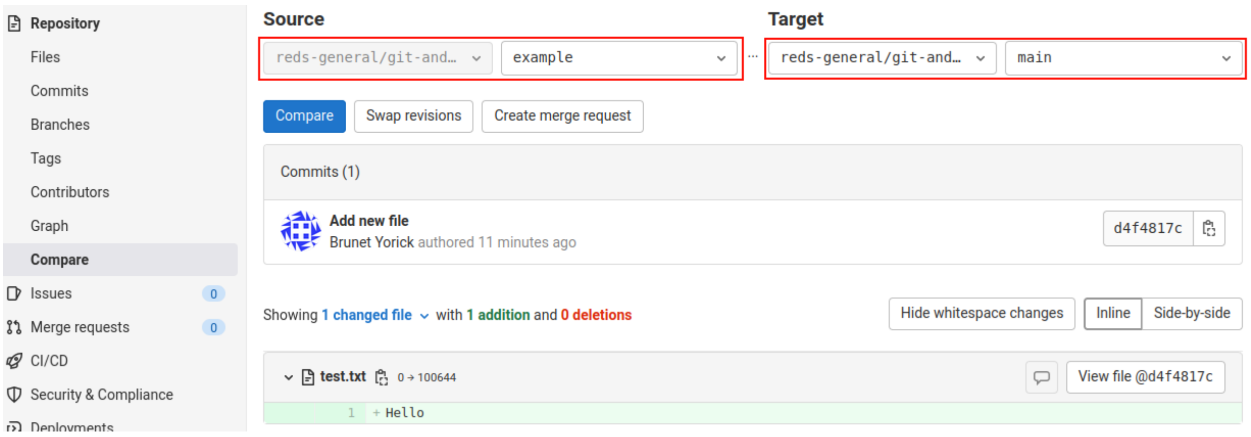


You can move around the graph by using the arrow keys.

```
Add new file
Add Git Workflow
Add log and show examples in Basics
Add Git GUI
Initial structure + Git Basics
```

Fig. 5.5: GitLab — Repository — Branch commits graph

- Use menu *Repository* → *Compare* to compare two branches (see Fig. 5.6)
  - branches to compare, with the dev branch as source and main branch as target



```
+ Hello
```

Fig. 5.6: GitLab — Repository — Branch compare

5.4 Issues

Issues allow to track work to do on the project. They contain a title, an optional content which describes the expected result of the issue, and may contain subtasks. Labels can also be used on issues to categorise them. Ideally an issue should be assigned to one of the project members.

- Use menu *Issues* or menu *Issues* → *List* to display the list of issues (see Fig. 5.7)
  - for each issue are displayed the title, its ID, its author and its labels
  - a new issue can be created with the top right button, see *Step 1: Create an issue in GitLab* from the *Git — Workflow*

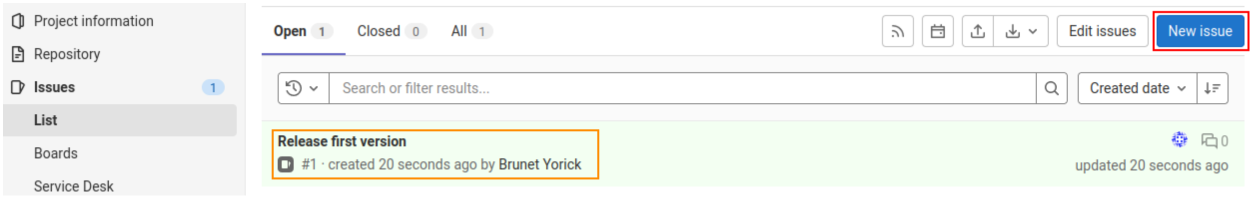
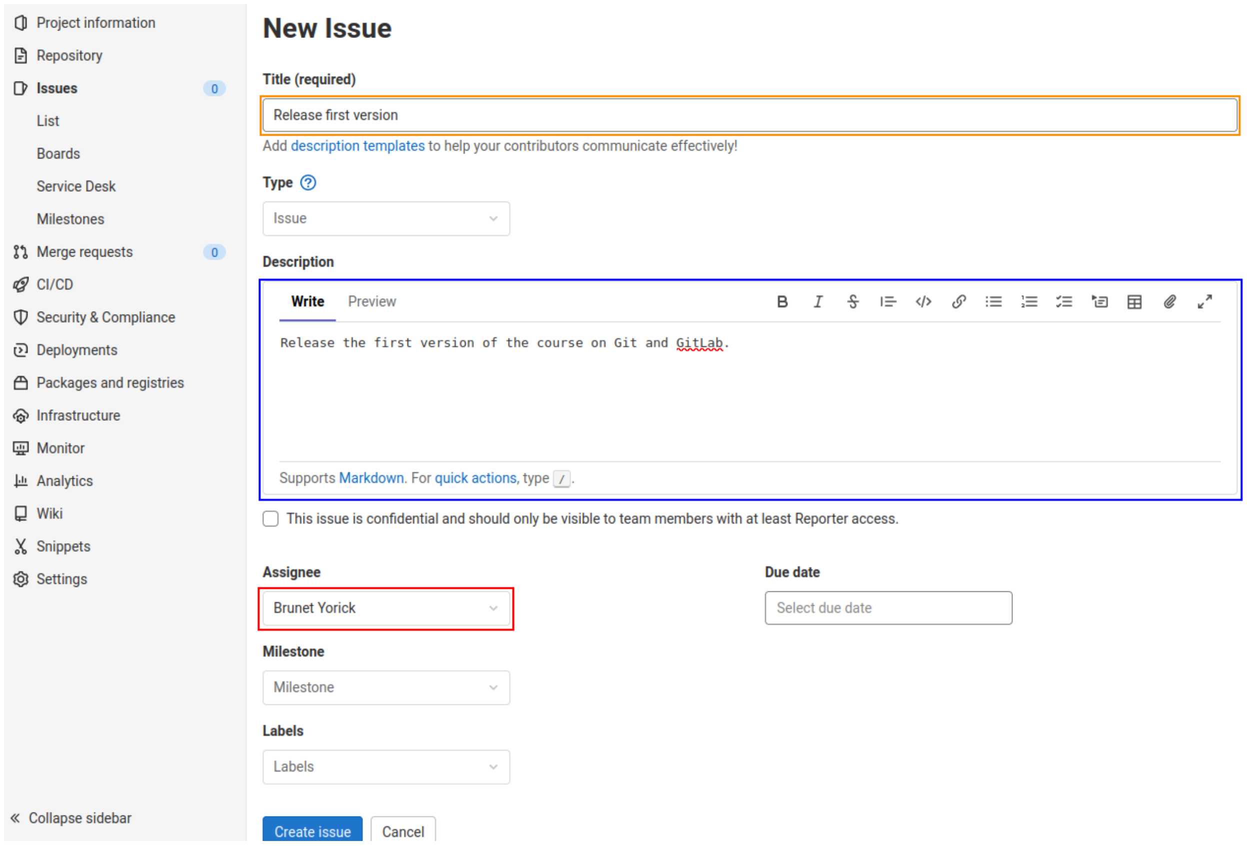


Fig. 5.7: GitLab — Issues — List of issues

  - when creating the new issue, provide (see Fig. 5.8):
    - ∗ its title,
    - ∗ its description, and
    - ∗ its assignation (if it is already known who is going to work on it)
- When opening an issue, all details are displayed (see Fig. 5.9)
  - the issue title,
  - the issue description,
  - the issue comments,
  - the issue assignee,
  - a merge request or a branch should be created from the issue, see also *Step 2: Create issue's branch in GitLab* from the *Git — Workflow*



New Issue

Title (required)

Release first version

Add description templates to help your contributors communicate effectively!

Type: Issue

Description

Release the first version of the course on Git and GitLab.

Supports Markdown. For quick actions, type /.

This issue is confidential and should only be visible to team members with at least Reporter access.

Assignee: Brunet Yorick

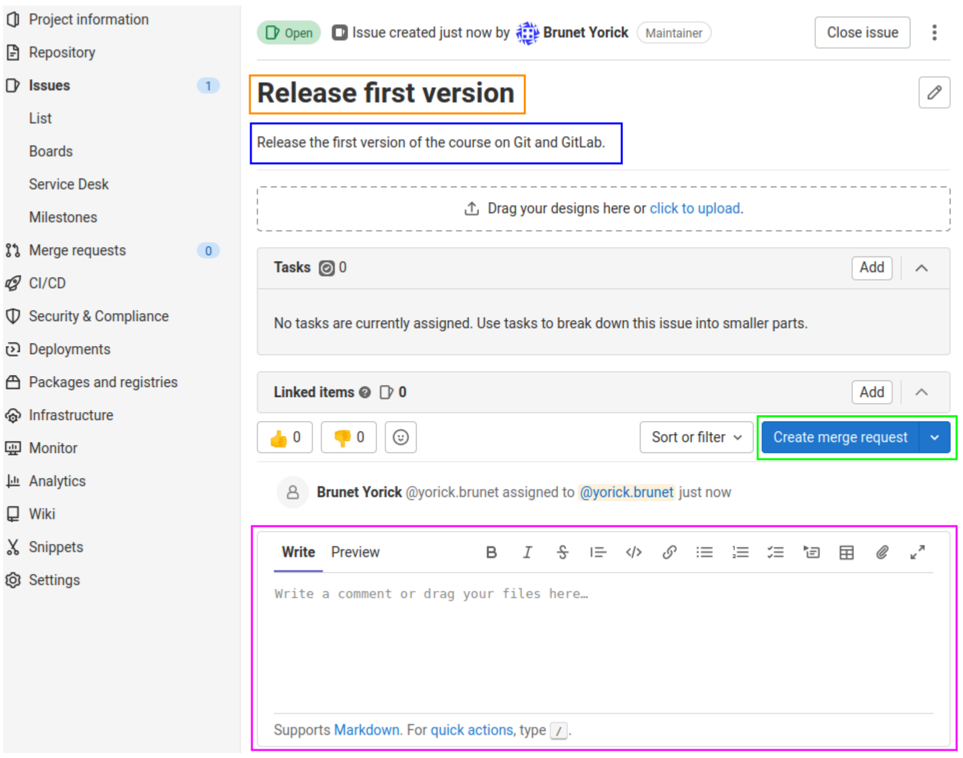
Due date: Select due date

Milestone: Milestone

Labels: Labels

Fig. 5.8: GitLab — Issues — New Issue

Note: It is recommended to create a development branch from an issue, and later a merge request from a development branch when the work is ready to be reviewed.



Release first version

Release the first version of the course on Git and GitLab.

No tasks are currently assigned. Use tasks to break down this issue into smaller parts.

Brunet Yorick @yorick.brunet assigned to @yorick.brunet just now

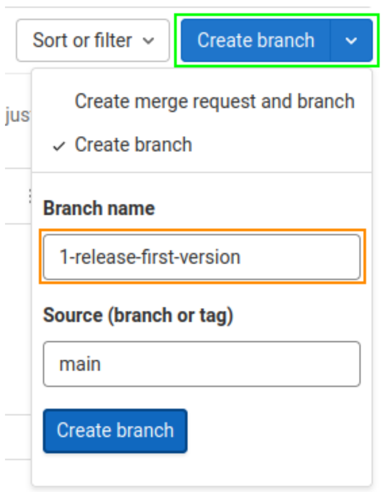
Supports Markdown. For quick actions, type /.

Fig. 5.9: GitLab — Issues — Issue overview

- To create the development branch, and based on the previous figure, click on the arrow on the right of button *Create merge request* and select *Create branch* (see Fig. 5.10):
  - the button name changes to Create branch
  - the branch name can be chosen, but leave it by default which uses the pattern "issue number"-"issue title"

  The new development branch is now linked to the branch (see Fig. 5.11).

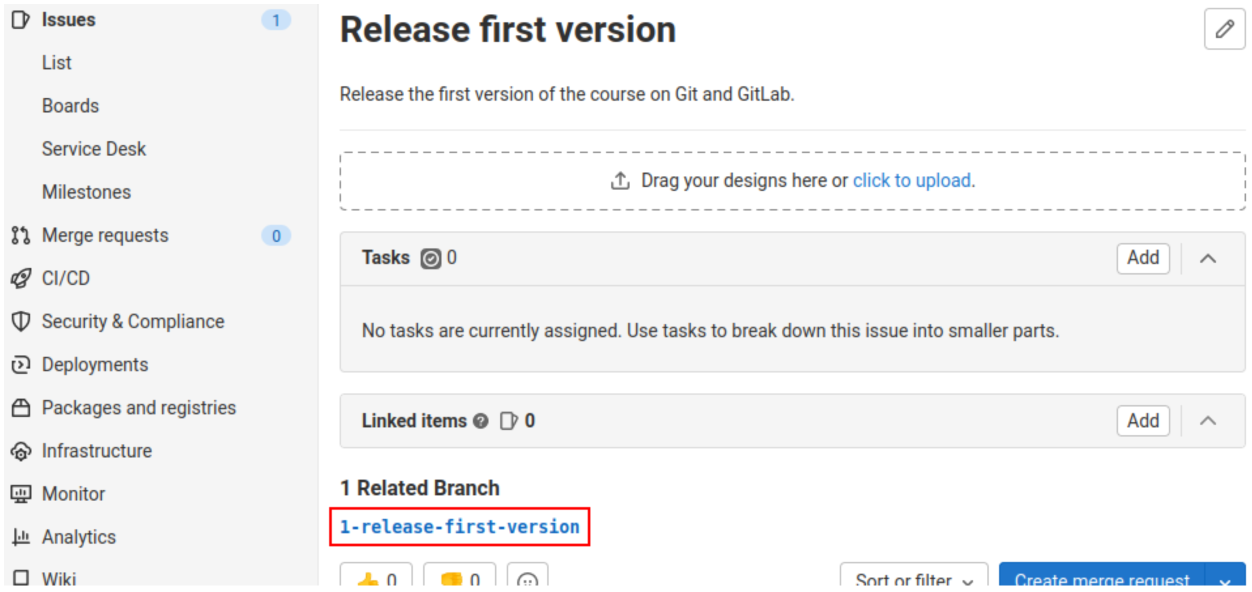
  The development branch can be fetched from the remote repository (more details later in *Fetch*):



Branch name: 1-release-first-version

Source (branch or tag): main

Fig. 5.10: GitLab — Issues — Create issue's branch



Release first version

Release the first version of the course on Git and GitLab.

Tasks 0

No tasks are currently assigned. Use tasks to break down this issue into smaller parts.

Linked items 0

1 Related Branch

1-release-first-version

Fig. 5.11: GitLab — Issues — Issue with linked branch

Listing 5.1: Example of `git fetch`

```
$ git fetch
From https://reds-gitlab.heig-vd.ch/reds-presentations/c-git-and-gitlab
* [new branch]      1-release-first-version -> origin/1-release-first-
↪version
$ git checkout 1-release-first-version
Branch '1-release-first-version' set up to track remote branch '1-release-
↪first-version' from 'origin'.
Switched to a new branch '1-release-first-version'
```

5.5 Merge requests

Merge requests are used to merge a branch into another branch, the latter being in general the main branch or an important development branch. This allows two complementary actions:

- have reviewers give their feedback on the work done, which usually improves its quality
- have only some persons on the project who can agree that the work is worth being merged, usually the maintainers

To create a merge request, use an existing branch:

- either from the list of repository branches (see Fig. 5.12)
  - create a merge request

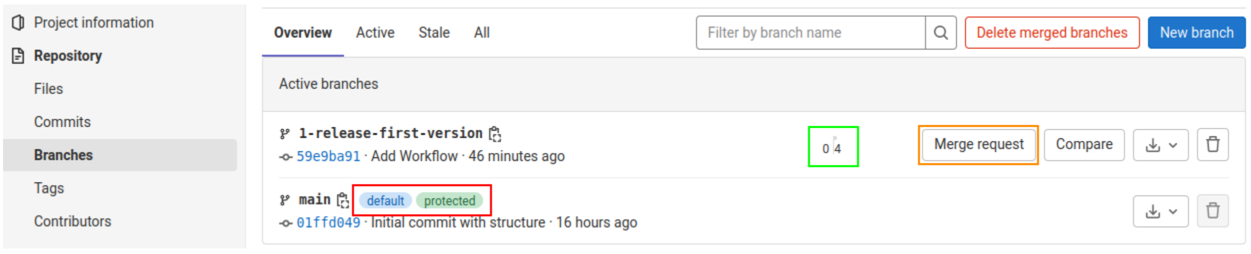


Fig. 5.12: GitLab — Merge requests — List of branches

- or from a "popup" that GitLab displays when you push a branch (see Fig. 5.13)

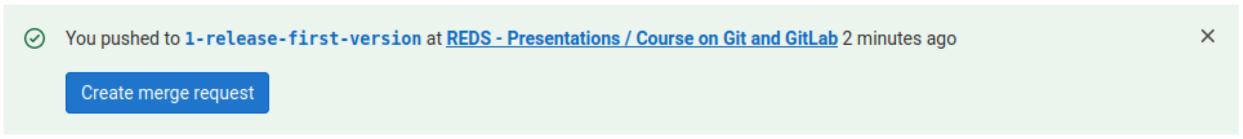

You pushed to 1-release-first-version at REDS - Presentations / Course on Git and GitLab 2 minutes ago
Create merge request

Fig. 5.13: GitLab — Merge requests — Popup to create a new merge request

When creating the merge request, a new page is displayed where details on the merge request must be provided (see Fig. 5.14):

- its title,
- its description,
- its assignation (usually the person who worked on the development branch),

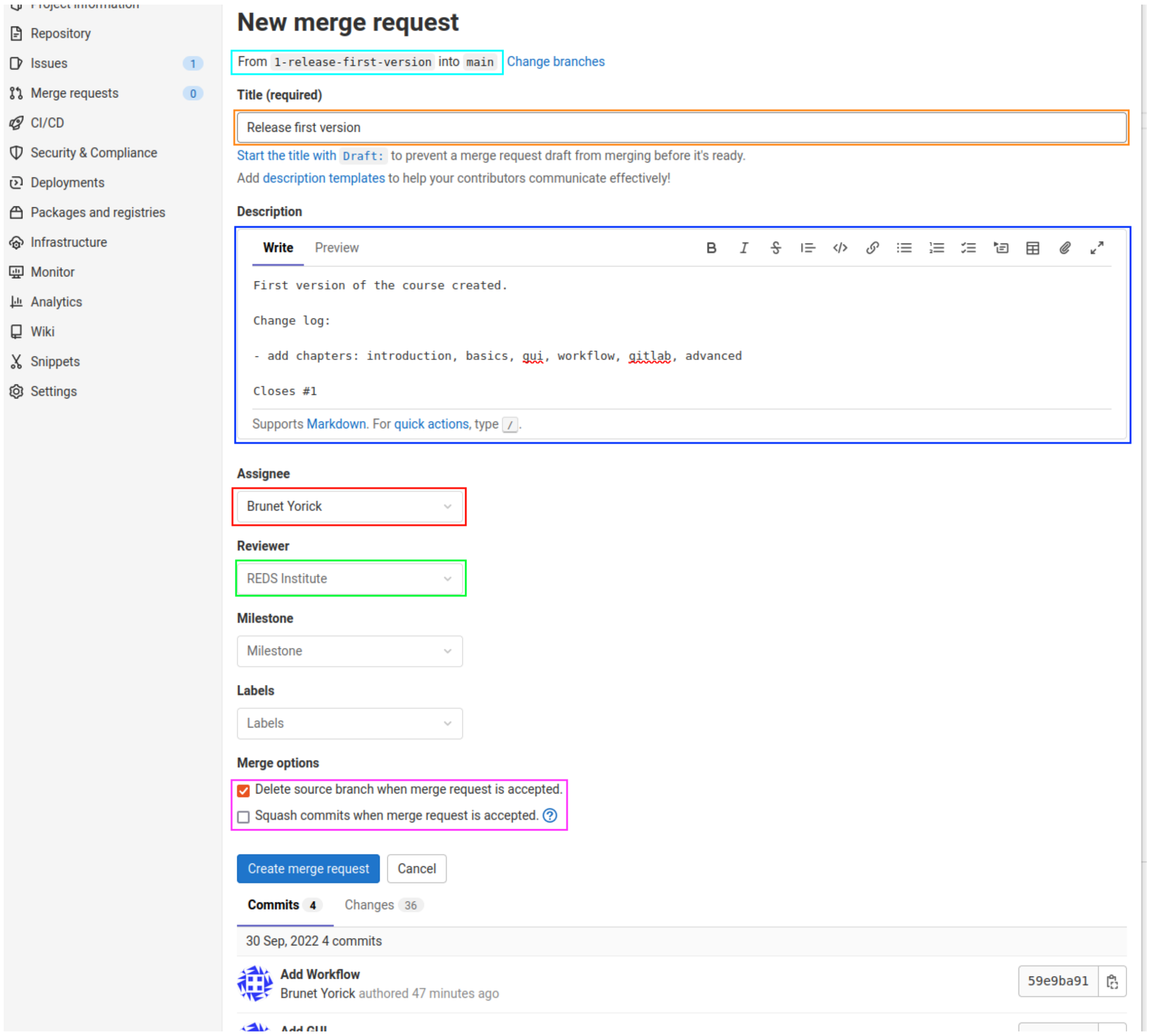
- its reviewer(s).

Note: https://reds-gitlab.heig-vd.ch uses the Free Tier, which allows only one assigned reviewer. It is, however, not forbidded that more persons review the merge request. Open source projects use the Premium Tier, which allows many reviewers.

In addition are also available:

- the merge options but, in general, keep the default settings (delete source branch, no squash commit),
- the source and destination branches.

By convention, create a draft merge request if the feature is not ready to be reviewed and merged. The draft status blocks any merge of the merge request until it has been marked as ready.



New merge request

From `1-release-first-version` into `main` Change branches

Title (required)

Release first version

Start the title with `Draft:` to prevent a merge request draft from merging before it's ready.
Add description templates to help your contributors communicate effectively!

Description

```
First version of the course created.

Change log:

- add chapters: introduction, basics, gui, workflow, gitlab, advanced

Closes #1
```

Supports Markdown. For quick actions, type `/`.

Assignee: Brunet Yorick

Reviewer: REDS Institute

Milestone: Milestone

Labels: Labels

Merge options

[✓] Delete source branch when merge request is accepted.
[ ] Squash commits when merge request is accepted.

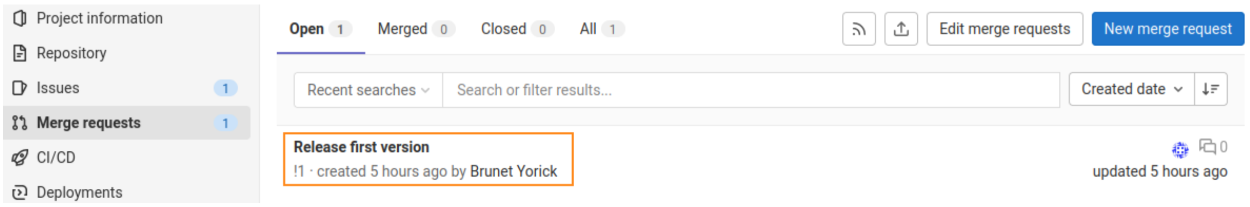
30 Sep, 2022 4 commits

Add Workflow
Brunet Yorick authored 47 minutes ago 59e9ba91

Fig. 5.14: GitLab — Merge requests — New merge request

- Use menu *Merge requests* to display the list of merge requests (see Fig. 5.15)

- for each issue are displayed the title, which issue it resolves and its author



Release first version
!1 · created 5 hours ago by Brunet Yorick
0
updated 5 hours ago

Fig. 5.15: GitLab — Merge requests — New merge request

- When opening a merge request, all details are displayed (see Fig. 5.16)
  - the merge request title,
  - the merge request description,
  - the merge information, merge options and merge button,
  - the merge request assignee,
  - the merge request reviewer(s).

  The merge request uses several tabs to display:
  - the overview (shown above, see Fig. 5.16)
  - the commits (see Fig. 5.17)
  - the pipelines
  - the changes (see Fig. 5.18)

5.6 Conduct a Code Review

A code review is conducted on tab *Changes* of the *Merge requests* page.

Be aware that the author, the reviewer and the maintainer (who merges the MR) have responsibilities. Refer to your project's guideline for these responsibilities. Some examples are:

- The responsibility of the merge request author
- The responsibility of the reviewer
- The responsibility of the maintainer

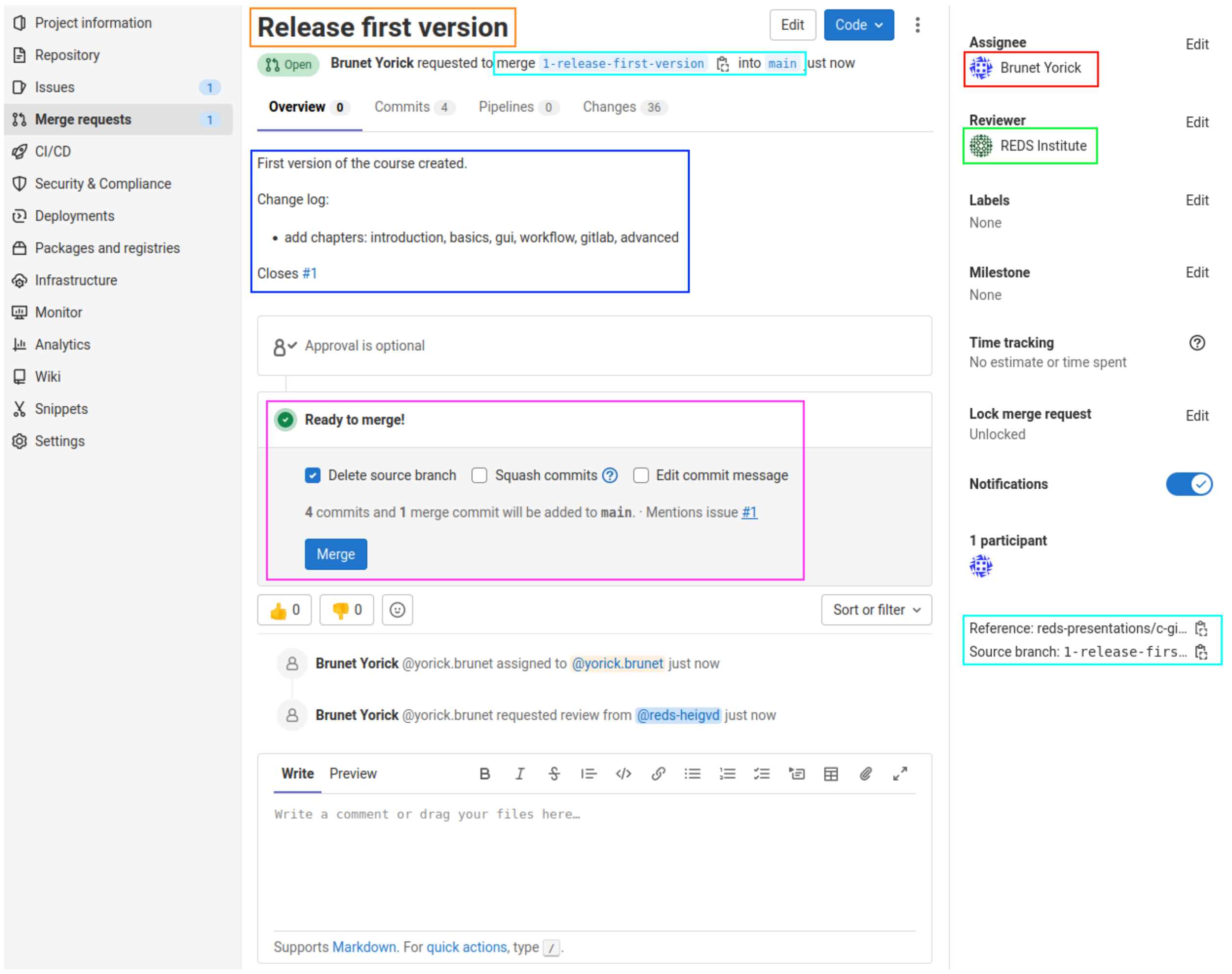
Different form of comments are available to help the reviewer communicate his/her thoughts:

- A regular comment
- A Change Suggestion

In general, a comment is always added to a thread, which is marked as unresolved after creation, and which can be resolved once the comment has been taken care of.

Furthermore, these comments can be added individually or grouped in a review.

To start a Code Review, open the tab *Changes* of the Merge Request.



Release first version

Open Brunet Yorick requested to merge `1-release-first-version` into `main` just now

First version of the course created.

Change log:

- add chapters: introduction, basics, gui, workflow, gitlab, advanced

Closes #1

Approval is optional

Ready to merge!

[✓] Delete source branch [ ] Squash commits [ ] Edit commit message

4 commits and 1 merge commit will be added to **main**. · Mentions issue #1

👍 0 👎 0

Brunet Yorick @yorick.brunet assigned to @yorick.brunet just now

Brunet Yorick @yorick.brunet requested review from @reds-heigvd just now

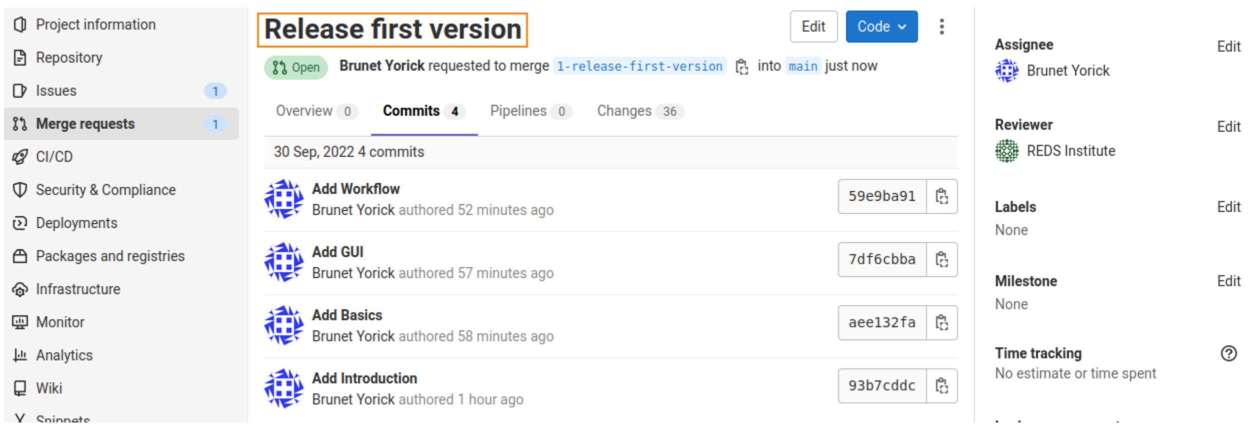
Write a comment or drag your files here…

Supports Markdown. For quick actions, type `/`.

Assignee	Brunet Yorick
Reviewer	REDS Institute
Labels	None
Milestone	None
Time tracking	No estimate or time spent
Lock merge request	Unlocked
Notifications	✓
1 participant	

Reference: reds-presentations/c-gi…
Source branch: `1-release-firs…`

Fig. 5.16: GitLab — Merge requests — List of merge requests



Release first version

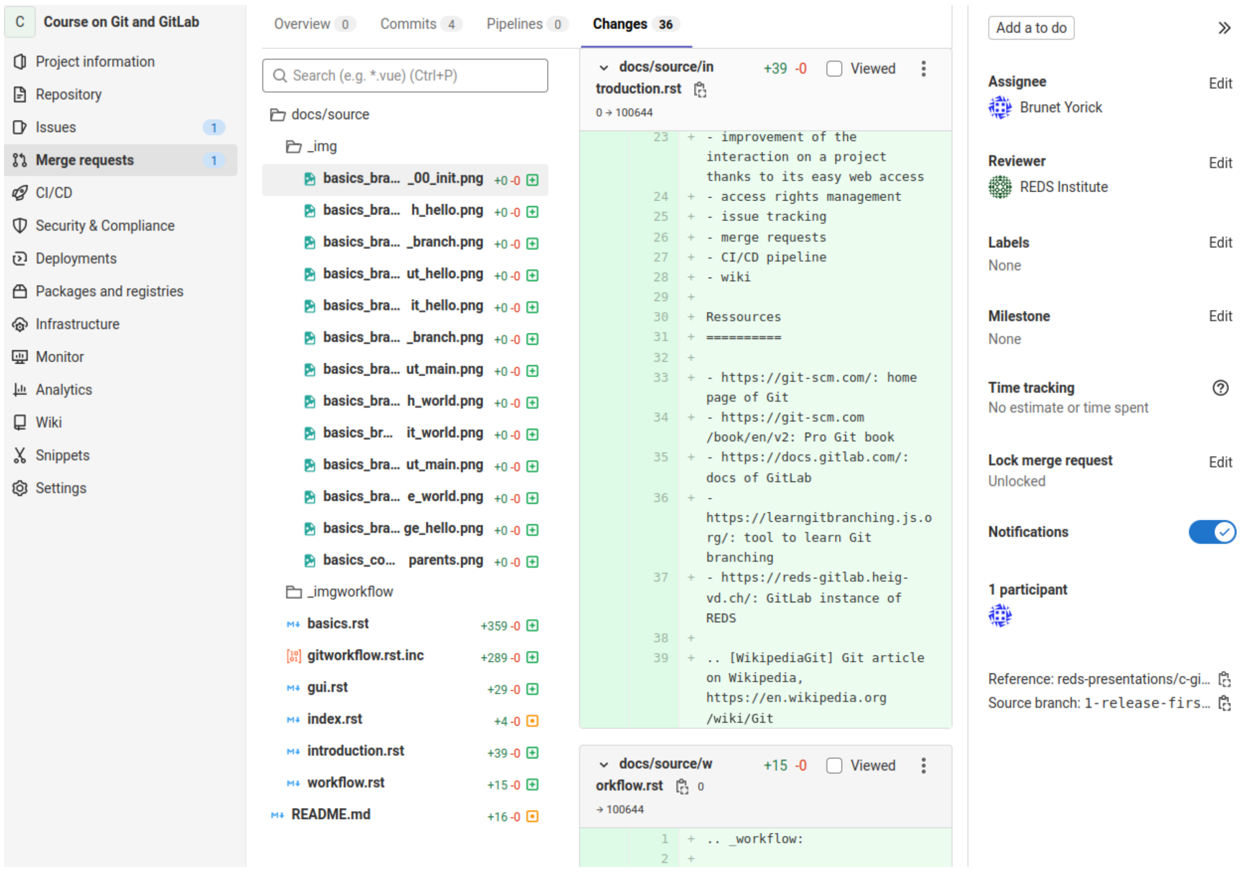
Open Brunet Yorick requested to merge `1-release-first-version` into `main` just now

30 Sep, 2022 4 commits

Add Workflow Brunet Yorick authored 52 minutes ago	59e9ba91
Add GUI Brunet Yorick authored 57 minutes ago	7df6cbba
Add Basics Brunet Yorick authored 58 minutes ago	aee132fa
Add Introduction Brunet Yorick authored 1 hour ago	93b7cddc

Assignee	Brunet Yorick
Reviewer	REDS Institute
Labels	None
Milestone	None
Time tracking	No estimate or time spent

Fig. 5.17: GitLab — Merge requests — Merge request view (general overview)



C Course on Git and GitLab

- Project information
- Repository
- Issues 1
- Merge requests 1
- CI/CD
- Security & Compliance
- Deployments
- Packages and registries
- Infrastructure
- Monitor
- Analytics
- Wiki
- Snippets
- Settings

Overview 0 Commits 4 Pipelines 0 **Changes 36**

Search (e.g. *.vue) (Ctrl+P)

- docs/source
  - _img
    - basics_bra… _00_init.png +0 -0
    - basics_bra… h_hello.png +0 -0
    - basics_bra… _branch.png +0 -0
    - basics_bra… ut_hello.png +0 -0
    - basics_bra… it_hello.png +0 -0
    - basics_bra… _branch.png +0 -0
    - basics_bra… ut_main.png +0 -0
    - basics_bra… h_world.png +0 -0
    - basics_br… it_world.png +0 -0
    - basics_bra… ut_main.png +0 -0
    - basics_bra… e_world.png +0 -0
    - basics_bra… ge_hello.png +0 -0
    - basics_co… parents.png +0 -0
  - _imgworkflow
  - basics.rst +359 -0
  - gitworkflow.rst.inc +289 -0
  - gui.rst +29 -0
  - index.rst +4 -0
  - introduction.rst +39 -0
  - workflow.rst +15 -0
- README.md +16 -0

docs/source/introduction.rst +39 -0 Viewed
0 → 100644

```
+ - improvement of the
  interaction on a project
  thanks to its easy web access
+ - access rights management
+ - issue tracking
+ - merge requests
+ - CI/CD pipeline
+ - wiki
+
+ Ressources
+ ==========
+
+ - https://git-scm.com/: home
  page of Git
+ - https://git-scm.com
  /book/en/v2: Pro Git book
+ - https://docs.gitlab.com/:
  docs of GitLab
+ -
  https://learngitbranching.js.o
  rg/: tool to learn Git
  branching
+ - https://reds-gitlab.heig-
  vd.ch/: GitLab instance of
  REDS
+
+ .. [WikipediaGit] Git article
  on Wikipedia,
  https://en.wikipedia.org
  /wiki/Git
```

docs/source/workflow.rst 0 +15 -0 Viewed
→ 100644

```
+ .. _workflow:
+
```

Add a to do

Assignee Edit
Brunet Yorick

Reviewer Edit
REDS Institute

Labels Edit
None

Milestone Edit
None

Time tracking
No estimate or time spent

Lock merge request Edit
Unlocked

Notifications

1 participant

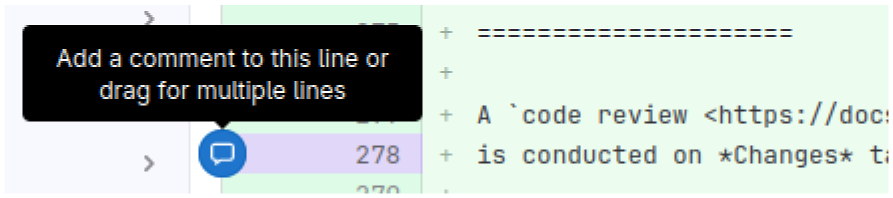
Reference: reds-presentations/c-gi…
Source branch: 1-release-firs…

Fig. 5.18: GitLab — Merge requests — Merge request view (changes)

5.6.1 Add an individual regular comment

The first form of comment is a regular comment. The reviewer can write a free text to communicate his/her thought. The text can be formatted in GitLab flavored Markdown.

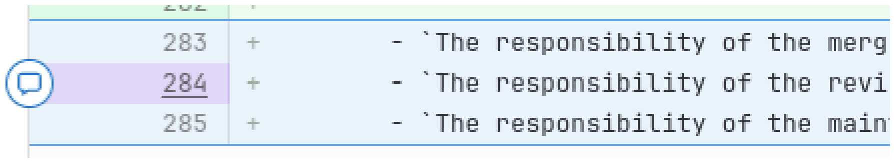
In the diff, select a line and click on the icon "comment" (see Fig. 5.19).



```
+ =====================
+
+ A `code review <https://doc
278 + is conducted on *Changes* t
```

Fig. 5.19: GitLab — Code review — Add a single or multi-line comment

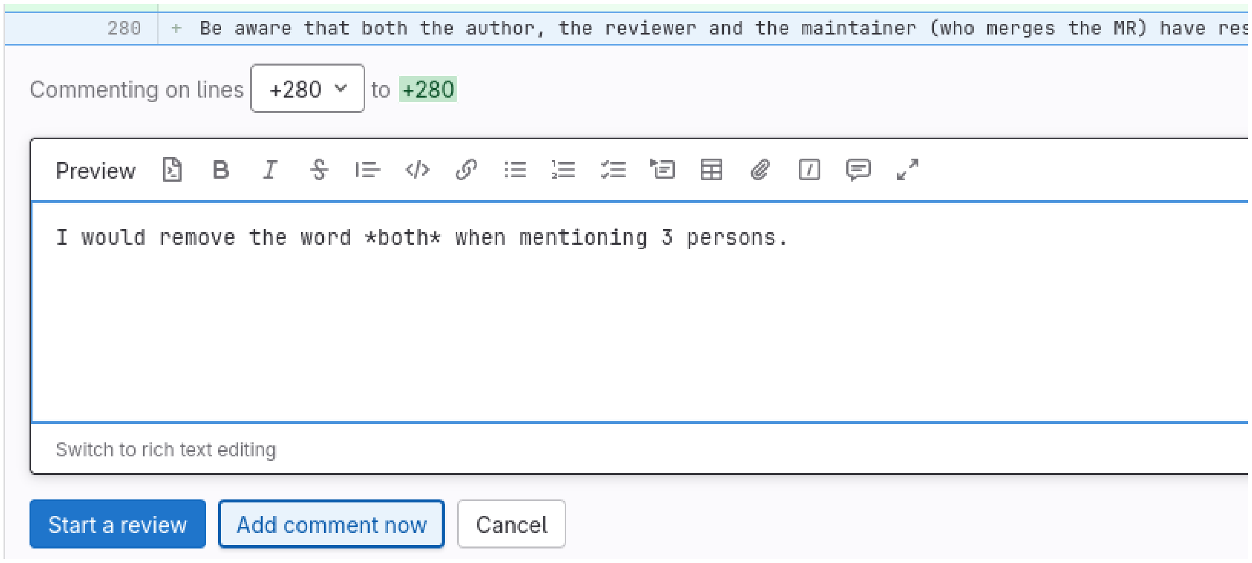
It is possible to comment multiple lines at once by keeping the icon clicked and dragging it on the corresponding lines (see Fig. 5.20).



```
283 +         - `The responsibility of the merg
284 +         - `The responsibility of the revi
285 +         - `The responsibility of the main
```

Fig. 5.20: GitLab — Code review — Comment multiple lines

A single-line comment looks like Fig. 5.21 while a multi-lines comment looks like Fig. 5.22.



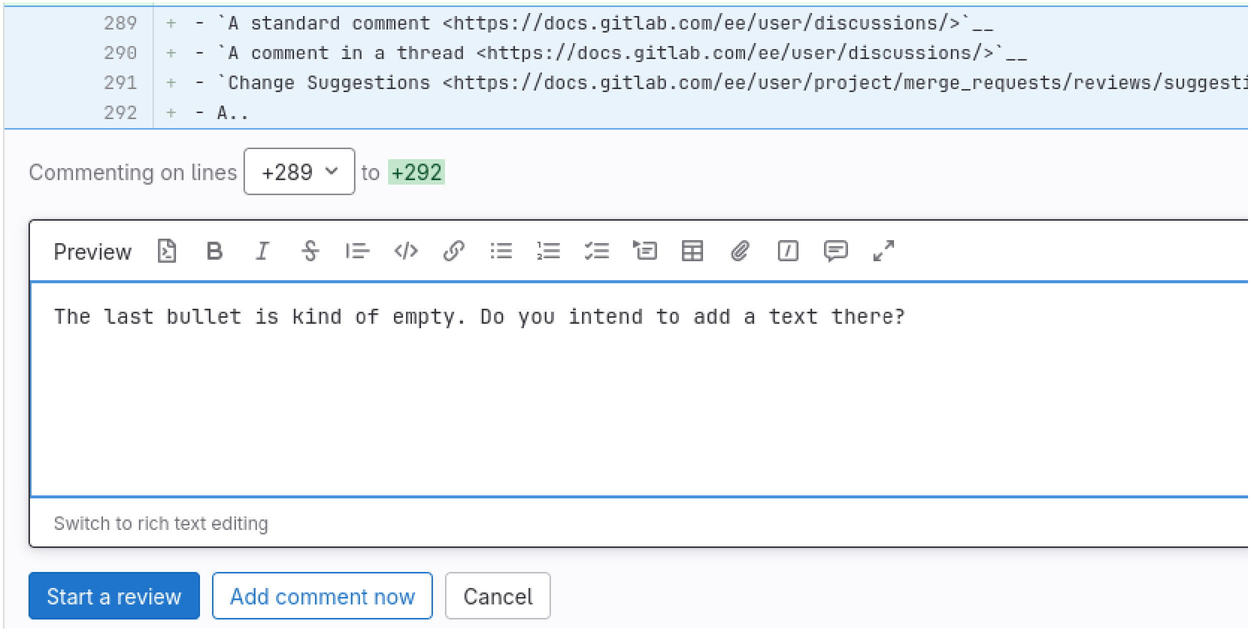
```
280 + Be aware that both the author, the reviewer and the maintainer (who merges the MR) have res
```

Commenting on lines +280 to +280

```
I would remove the word *both* when mentioning 3 persons.
```

Fig. 5.21: GitLab — Code review — Single line comment

Write the comment's text, review it by clicking on **Preview**, and finally click on **Add comment now** to create the comment.



```
+ - `A standard comment <https://docs.gitlab.com/ee/user/discussions/>`__
+ - `A comment in a thread <https://docs.gitlab.com/ee/user/discussions/>`__
+ - `Change Suggestions <https://docs.gitlab.com/ee/user/project/merge_requests/reviews/suggesti
+ - A..
```

```
The last bullet is kind of empty. Do you intend to add a text there?
```

Fig. 5.22: GitLab — Code review — Multi lines comment

5.6.2 Add an individual change suggestion

The second form of comment is a change suggestion. The comment may contain a text, but the key part of a change suggestion is the… suggested change.

First, start by adding a regular comment as shown in the previous section, and insert a code change suggestion (see Fig. 5.23).

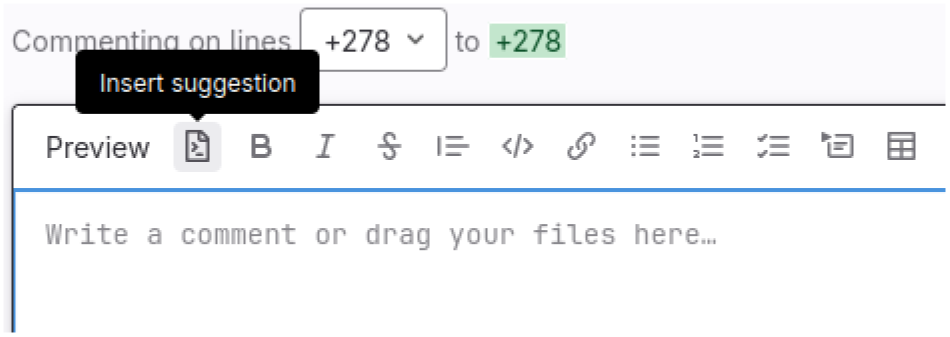
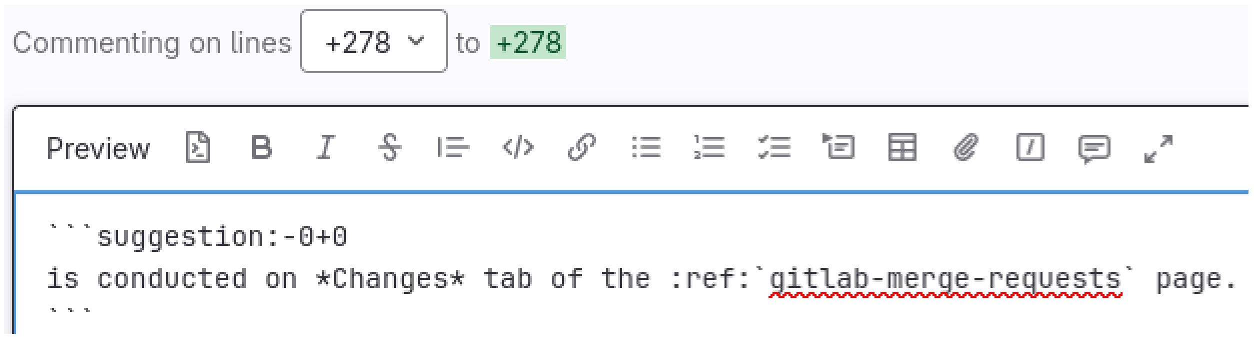
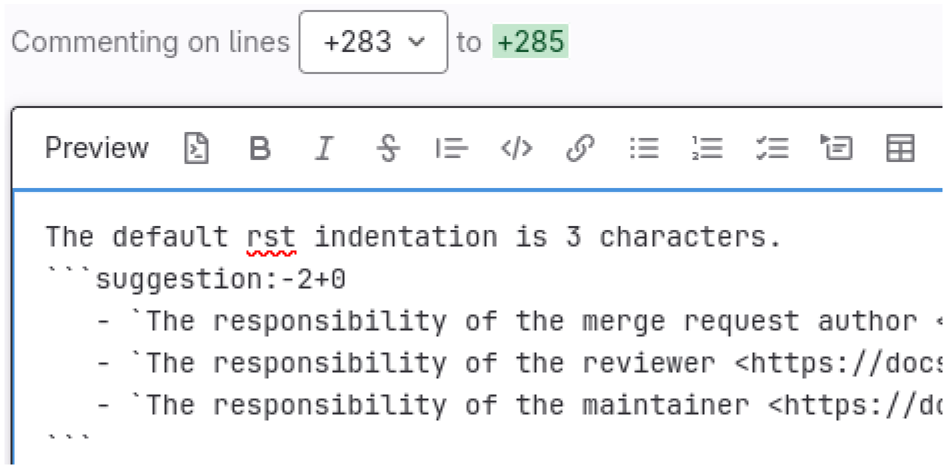
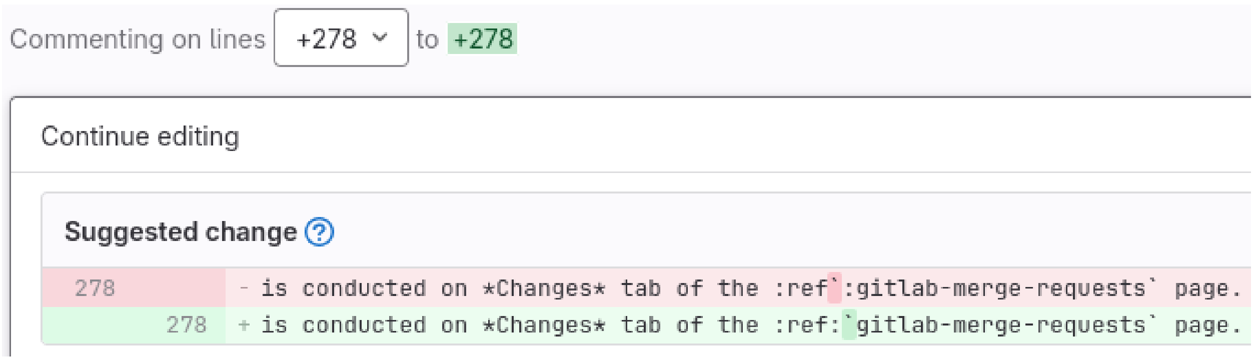


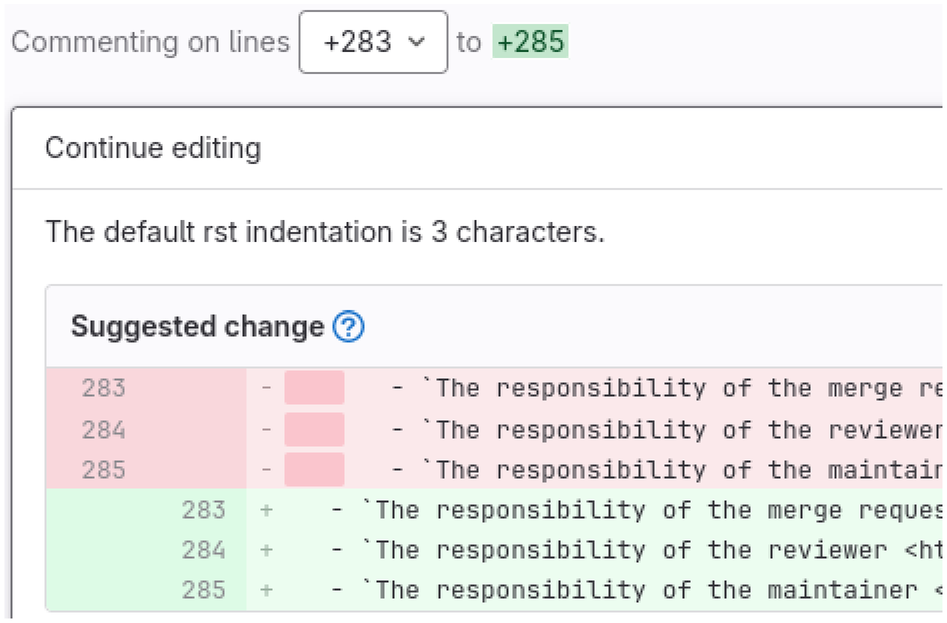
Fig. 5.23: GitLab — Code review — Insert a suggestion code block

Second, modify the code so that it reflects your suggestion (see Fig. 5.24 for single-line suggestion and Fig. 5.25 for multi-lines suggestion).

Write an optional comment's text, review the change suggestion by clicking on **Preview** (see Fig. 5.26 for single-line suggestion and Fig. 5.27 for multi-lines suggestion), and finally click on **Add comment now** to create the comment.



~~~
```suggestion:-0+0
is conducted on *Changes* tab of the :ref:`gitlab-merge-requests` page.
~~~
```

Fig. 5.24: GitLab — Code review — Add single-line suggestion


Commenting on lines
+283
to
+285
Preview
B
I

```
The default rst indentation is 3 characters.
```suggestion:-2+0
   - `The responsibility of the merge request author <
   - `The responsibility of the reviewer <https://docs
   - `The responsibility of the maintainer <https://dc
```
```

Fig. 5.25: GitLab — Code review — Add multi-lines suggestion


Commenting on lines
+278
to
+278
Continue editing
Suggested change

```
278      - is conducted on *Changes* tab of the :ref`:gitlab-merge-requests` page.
    278  + is conducted on *Changes* tab of the :ref:`gitlab-merge-requests` page.
```

Fig. 5.26: GitLab — Code review — Preview single-line suggestion
```


Commenting on lines +283 to +285

Continue editing

The default rst indentation is 3 characters.

Suggested change

```
283      -      - `The responsibility of the merge re
284      -      - `The responsibility of the reviewer
285      -      - `The responsibility of the maintair
     283 +    - `The responsibility of the merge reques
     284 +    - `The responsibility of the reviewer <ht
     285 +    - `The responsibility of the maintainer <
```

Fig. 5.27: GitLab — Code review — Preview multi-lines suggestion

5.6.3 Add comments in review mode

It is possible to add comments or change suggestions in “review mode”. Instead of creating comments one by one, this mode allows to add several comments in bulk mode. It means that, when you enter the review mode (upon creating the first comment), you can add comments which are effectively created only when the review is submitted. When you submit your review, GitLab:

- publishes the comments in your review,
- sends a single email to every notifiable user of the merge request, with your review comments attached,
- performs any quick actions you added to your review comments.

To enter in mode review, create a first comment (see Fig. 5.19), and click on **Start a review** (see Fig. 5.21) instead of add comment now.

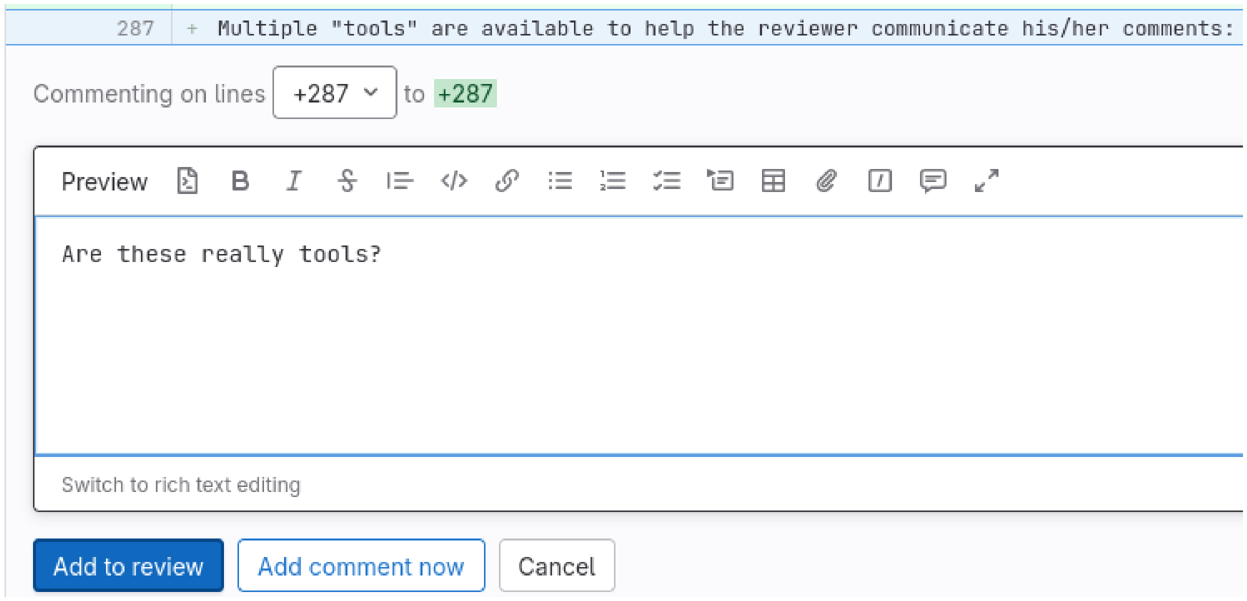
Continue adding comments, each time adding the comment to the review with button **Add to review** (see Fig. 5.28).

At any time, you can see all your pending comments (see Fig. 5.29):

- pending comments are labelled with the *pending* flag,
- at the bottom of the *Changes* tab, a button **Pending comments** provides a summary and a quick link.

When the review is done, publish your review by clicking on the button **Finish review** (see Fig. 5.30). You have the possibility to add an optional summary comment before submitting the review by clicking on button **Submit review**.

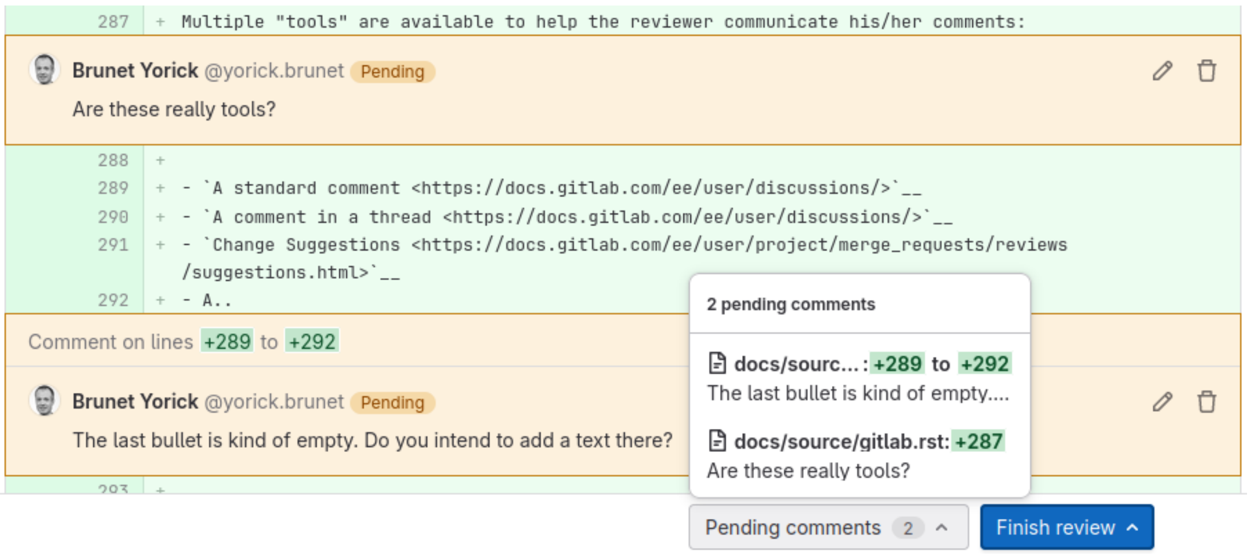
In the tab *Overview*, all comments will be resolvable as individual comments but the last optional summary comment (see Fig. 5.31).



```
287  + Multiple "tools" are available to help the reviewer communicate his/her comments:
```

Are these really tools?

Fig. 5.28: GitLab — Code review — Add a new comment in review mode



```
287  + Multiple "tools" are available to help the reviewer communicate his/her comments:
```

Brunet Yorick @yorick.brunet Pending

Are these really tools?

```
288  +
289  + - `A standard comment <https://docs.gitlab.com/ee/user/discussions/>`__
290  + - `A comment in a thread <https://docs.gitlab.com/ee/user/discussions/>`__
291  + - `Change Suggestions <https://docs.gitlab.com/ee/user/project/merge_requests/reviews
       /suggestions.html>`__
292  + - A..
```

Comment on lines +289 to +292

Brunet Yorick @yorick.brunet Pending

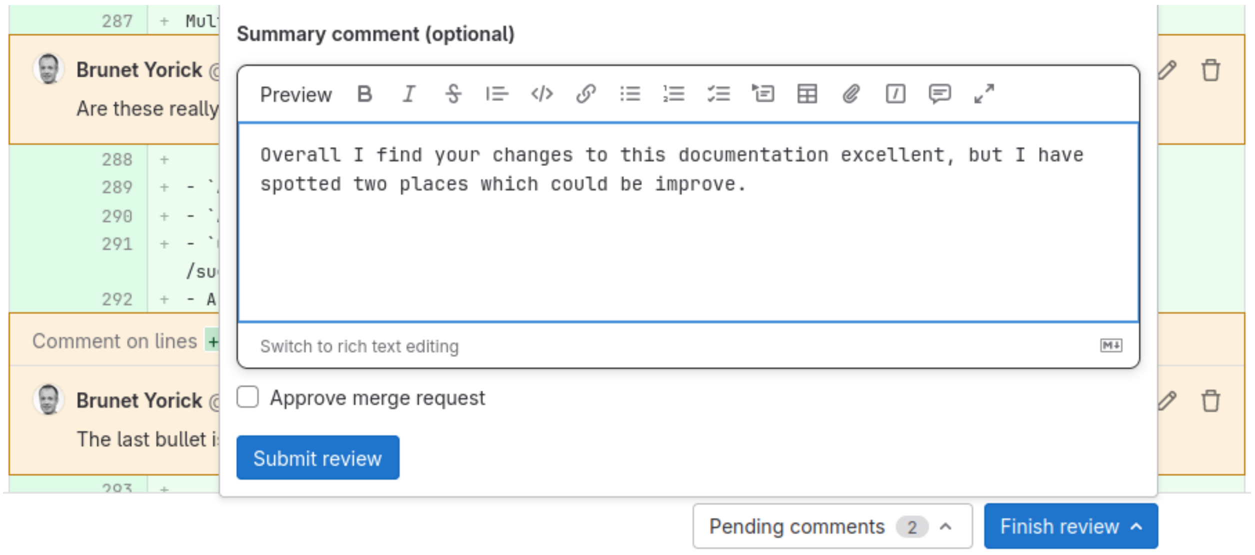
The last bullet is kind of empty. Do you intend to add a text there?

2 pending comments

docs/sourc...: +289 to +292
The last bullet is kind of empty....

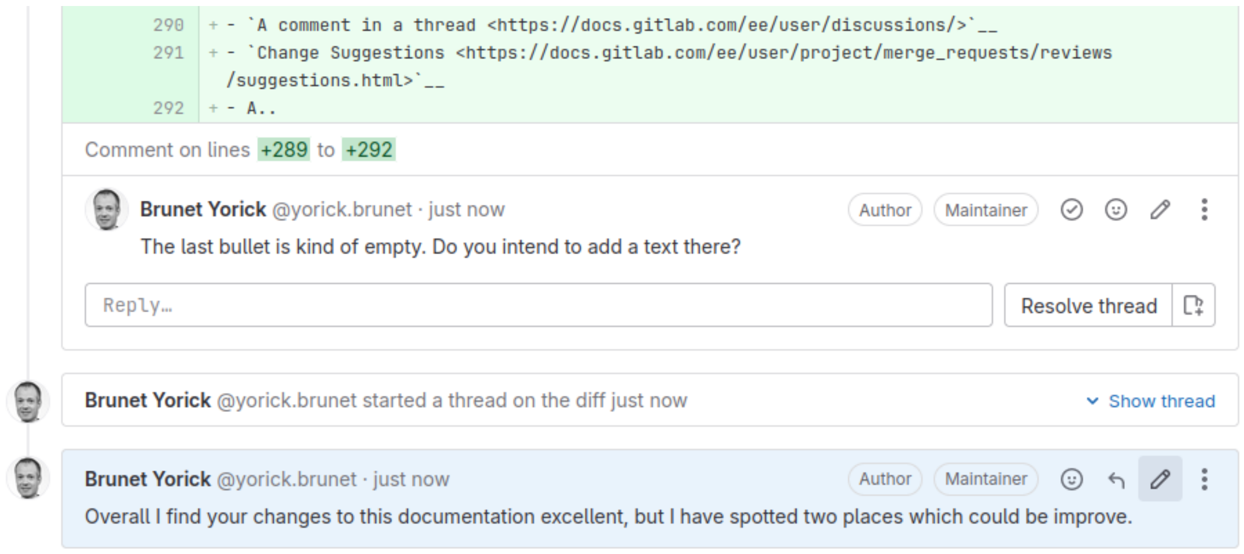
docs/source/gitlab.rst: +287
Are these really tools?

Fig. 5.29: GitLab — Code review — Overview of pending comments in review mode



```
Overall I find your changes to this documentation excellent, but I have
spotted two places which could be improve.
```

Fig. 5.30: GitLab — Code review — Publish pending comments in review mode (1)



```
+ - `A comment in a thread <https://docs.gitlab.com/ee/user/discussions/>`__
+ - `Change Suggestions <https://docs.gitlab.com/ee/user/project/merge_requests/reviews
/suggestions.html>`__
+ - A..
```

Comment on lines +289 to +292

Brunet Yorick @yorick.brunet · just now

The last bullet is kind of empty. Do you intend to add a text there?

Brunet Yorick @yorick.brunet started a thread on the diff just now

Brunet Yorick @yorick.brunet · just now

Overall I find your changes to this documentation excellent, but I have spotted two places which could be improve.

Fig. 5.31: GitLab — Code review — Publish pending comments in review mode (2)

5.6.4 Count of unresolved comments

In the top right corner of the *Overview* tab, a count of the unresolved comments is provided (see Fig. 5.32). The up and down arrows allow the user to navigate among unresolved comments.

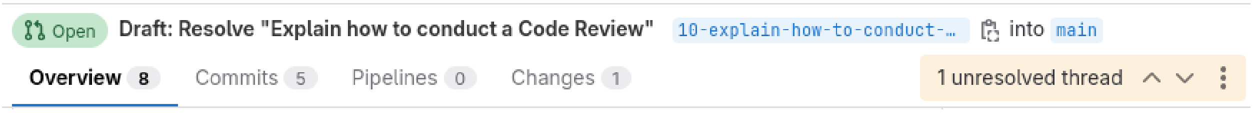

Open
Draft: Resolve "Explain how to conduct a Code Review"
10-explain-how-to-conduct-…
into main
Overview 8
Commits 5
Pipelines 0
Changes 1
1 unresolved thread

Fig. 5.32: GitLab — Code review — Comments count

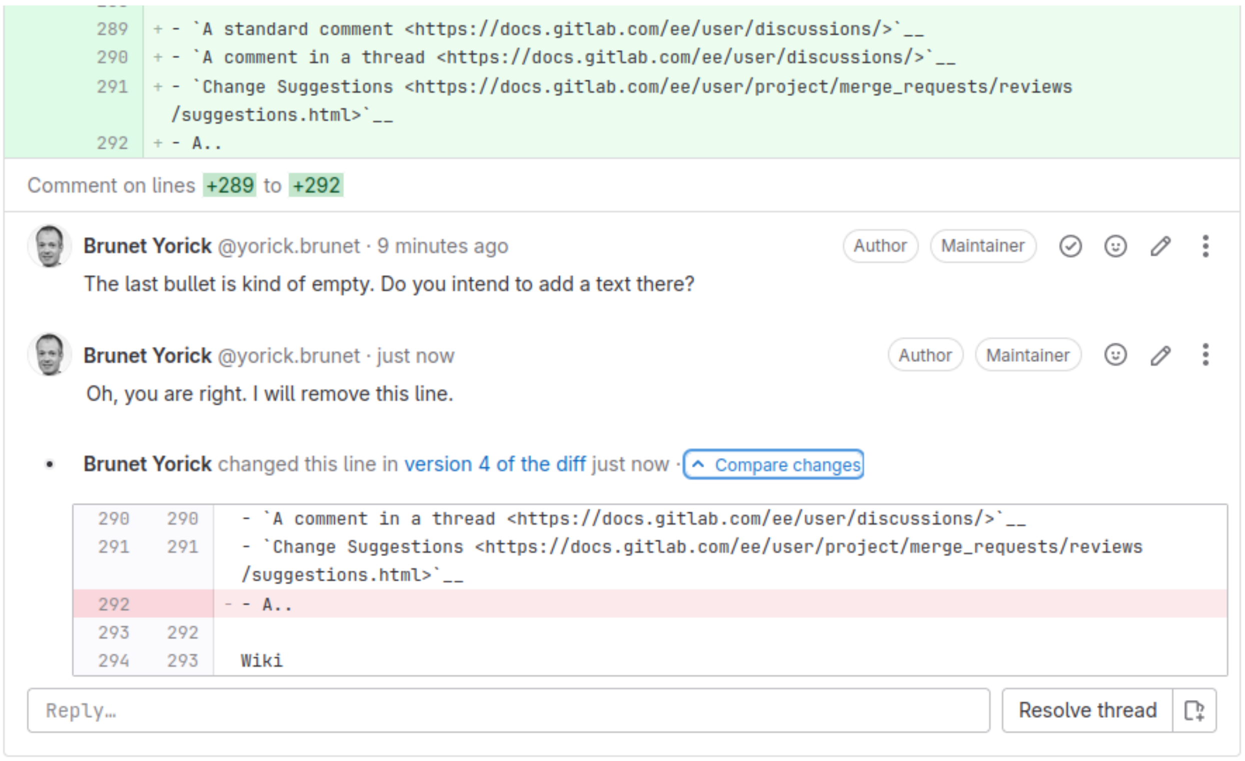
5.6.5 Resolve a regular comment

First of all, a comment of a reviewer communicates his/her thoughts on a piece of code. The reviewer must be clear about his/her request:

- some comments require a change,
- some comments only suggest a change,
- some comments don't ask for any change but require some clarifications from the code author.

The code author may disagree with the comment, but must say it, and agree with the reviewer on how to resolve the comment.

If the comment requires/suggests a change and the author agrees with the change, the resolution of the comment is done via the push of a new commit. The reviewer is then able to see the change by opening the **compare changes** folded block (see Fig. 5.33).



```
+ - `A standard comment <https://docs.gitlab.com/ee/user/discussions/>`__
+ - `A comment in a thread <https://docs.gitlab.com/ee/user/discussions/>`__
+ - `Change Suggestions <https://docs.gitlab.com/ee/user/project/merge_requests/reviews
  /suggestions.html>`__
+ - A..
```

Comment on lines +289 to +292

Brunet Yorick @yorick.brunet · 9 minutes ago
The last bullet is kind of empty. Do you intend to add a text there?

Brunet Yorick @yorick.brunet · just now
Oh, you are right. I will remove this line.

Brunet Yorick changed this line in version 4 of the diff just now · Compare changes

```
290 290    - `A comment in a thread <https://docs.gitlab.com/ee/user/discussions/>`__
291 291    - `Change Suggestions <https://docs.gitlab.com/ee/user/project/merge_requests/reviews
           /suggestions.html>`__
292      - - A..
293 292
294 293    Wiki
```

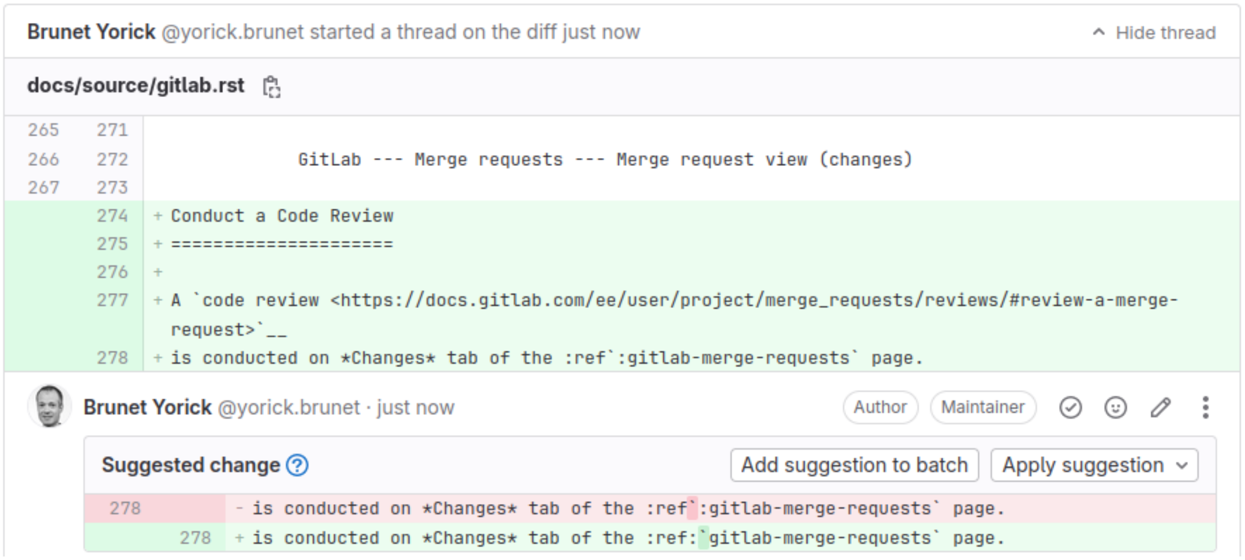
Fig. 5.33: GitLab — Code review — Solve comment

Note: Who clicks on **Resolve thread** to definitively close the comment must be decided within the project. Is it the developer after pushing the change? Is it the reviewer after verifying whether he agrees with the pushed change?

5.6.6 Resolve a change suggestion

The intention of the reviewer is clear with a change suggestion. It is clearly defined how he expects the change to be done.

Again, the code author may disagree with the change, which would lead to more discussions. But if he agrees with the change, he can simply click on the button **Apply suggestion** (see Fig. 5.34).


Brunet Yorick @yorick.brunet started a thread on the diff just now
Hide thread
docs/source/gitlab.rst
Brunet Yorick @yorick.brunet · just now
Author
Maintainer
Suggested change
Add suggestion to batch
Apply suggestion

```
                GitLab --- Merge requests --- Merge request view (changes)

+ Conduct a Code Review
+ ====================
+
+ A `code review <https://docs.gitlab.com/ee/user/project/merge_requests/reviews/#review-a-merge-
  request>`__
+ is conducted on *Changes* tab of the :ref`:gitlab-merge-requests` page.
```

```
- is conducted on *Changes* tab of the :ref`:gitlab-merge-requests` page.
+ is conducted on *Changes* tab of the :ref:`gitlab-merge-requests` page.
```

Fig. 5.34: GitLab — Code review — Solve suggestion – Accept change

Applying the suggestion leads to the creation of a new commit, to which a commit message must be given (see Fig. 5.35 and Fig. 5.36).

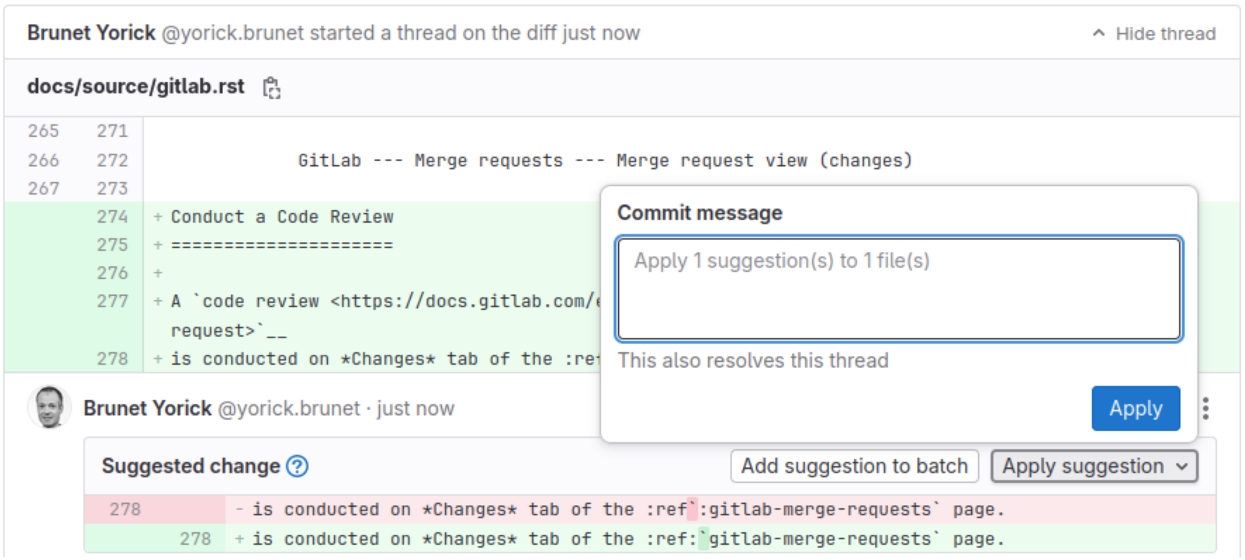
The thread is automatically resolved (see Fig. 5.37).

5.6.7 Resolve change suggestions in batch

It may happen that several change suggestions are part of the same resolution of an issue. In which case, it makes sense to group them in the same resolution commit. To do so, when processing the first change suggestion, click on button **Add suggestion to batch** instead of apply suggestion (see Fig. 5.34).

Add the next change suggestions to the batch (see Fig. 5.38).

And finally click on **Apply … suggestions** when processing the last change suggestion of the batch. Again, provide a commit message and click on **Apply** (see Fig. 5.39). This action creates a single commit with all batched change suggestions.


Brunet Yorick @yorick.brunet started a thread on the diff just now

docs/source/gitlab.rst

```
GitLab --- Merge requests --- Merge request view (changes)

+ Conduct a Code Review
+ ====================
+
+ A `code review <https://docs.gitlab.com/e
request>`__
+ is conducted on *Changes* tab of the :ref
```

Commit message

Apply 1 suggestion(s) to 1 file(s)

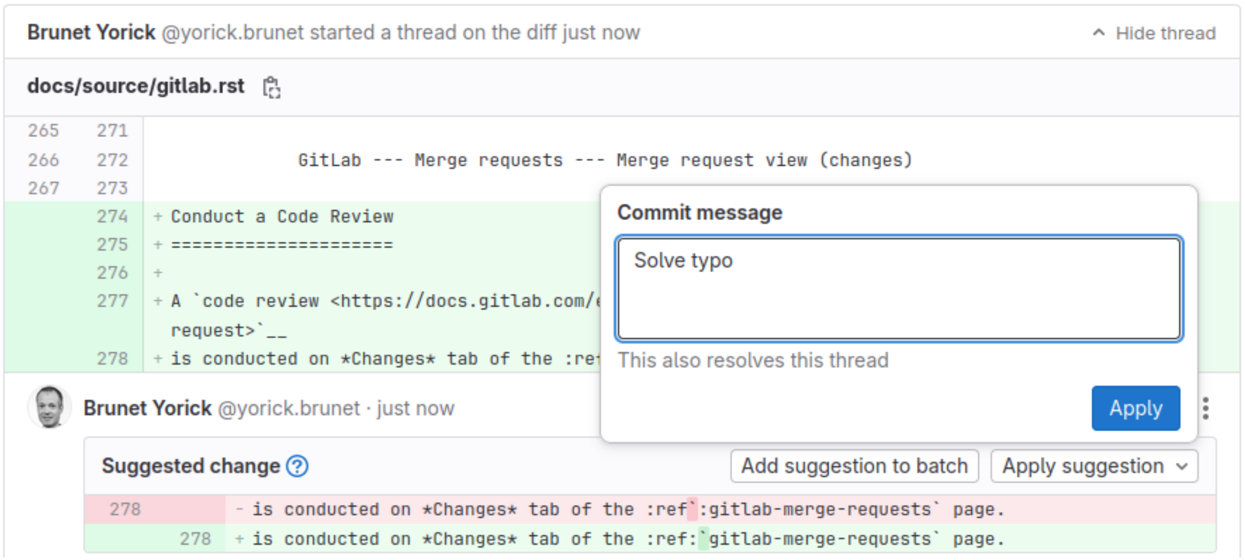
This also resolves this thread

Brunet Yorick @yorick.brunet · just now

Suggested change

```
278    - is conducted on *Changes* tab of the :ref`:gitlab-merge-requests` page.
278    + is conducted on *Changes* tab of the :ref:`gitlab-merge-requests` page.
```

Fig. 5.35: GitLab — Code review — Solve suggestion – Add commit message (1)


Brunet Yorick @yorick.brunet started a thread on the diff just now

docs/source/gitlab.rst

```
GitLab --- Merge requests --- Merge request view (changes)

+ Conduct a Code Review
+ ====================
+
+ A `code review <https://docs.gitlab.com/e
request>`__
+ is conducted on *Changes* tab of the :ref
```

Commit message

Solve typo

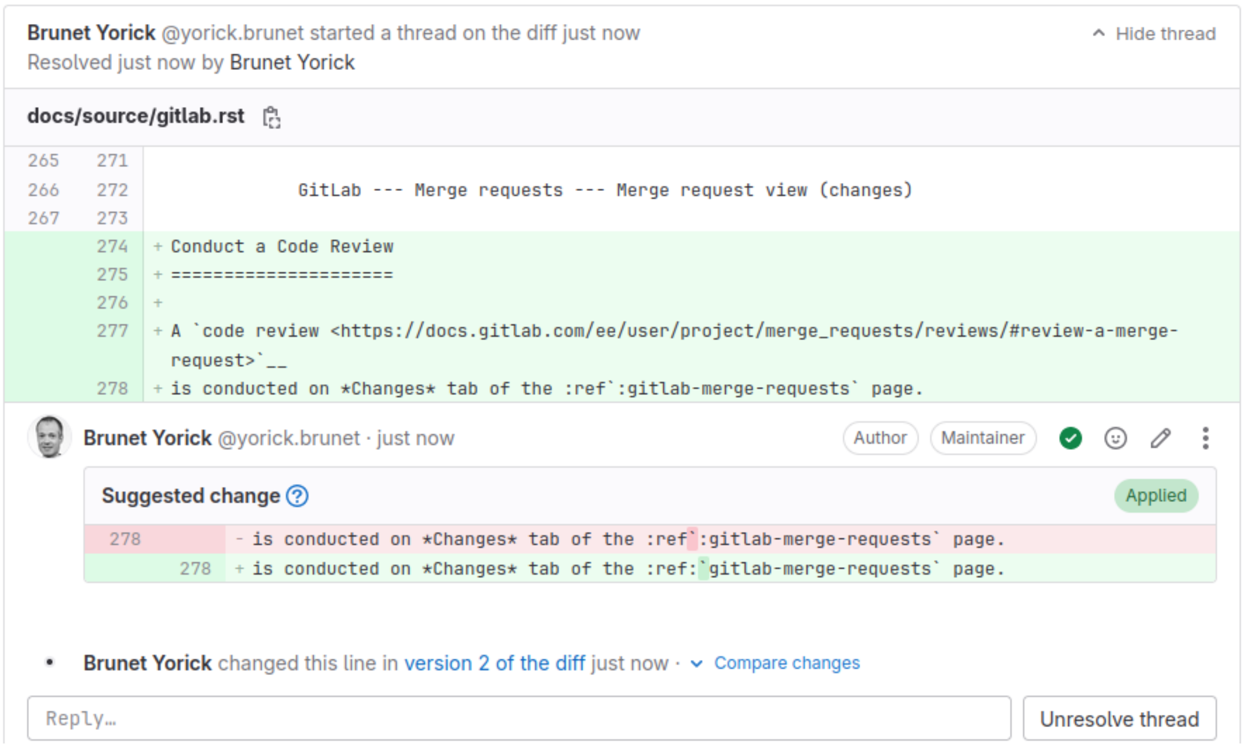
This also resolves this thread

Brunet Yorick @yorick.brunet · just now

Suggested change

```
278    - is conducted on *Changes* tab of the :ref`:gitlab-merge-requests` page.
278    + is conducted on *Changes* tab of the :ref:`gitlab-merge-requests` page.
```

Fig. 5.36: GitLab — Code review — Solve suggestion – Add commit message (2)



Brunet Yorick @yorick.brunet started a thread on the diff just now
Resolved just now by Brunet Yorick

docs/source/gitlab.rst

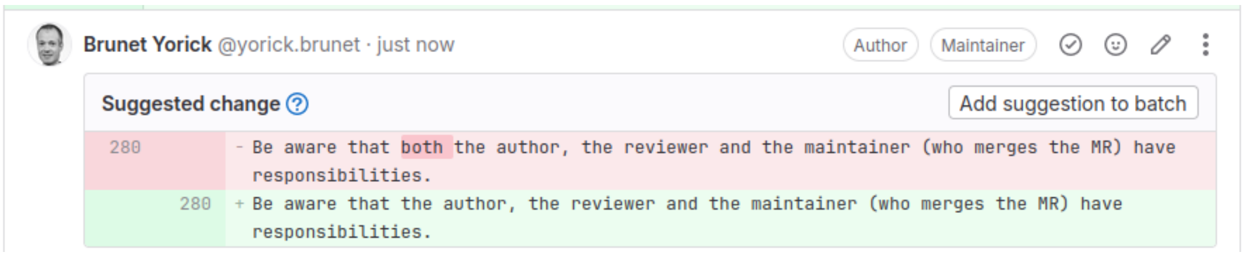
```
                GitLab --- Merge requests --- Merge request view (changes)

+ Conduct a Code Review
+ ====================
+
+ A `code review <https://docs.gitlab.com/ee/user/project/merge_requests/reviews/#review-a-merge-
  request>`__
+ is conducted on *Changes* tab of the :ref`:gitlab-merge-requests` page.
```

Brunet Yorick @yorick.brunet · just now

```
278      - is conducted on *Changes* tab of the :ref`:gitlab-merge-requests` page.
    278  + is conducted on *Changes* tab of the :ref:`gitlab-merge-requests` page.
```

Brunet Yorick changed this line in version 2 of the diff just now · Compare changes

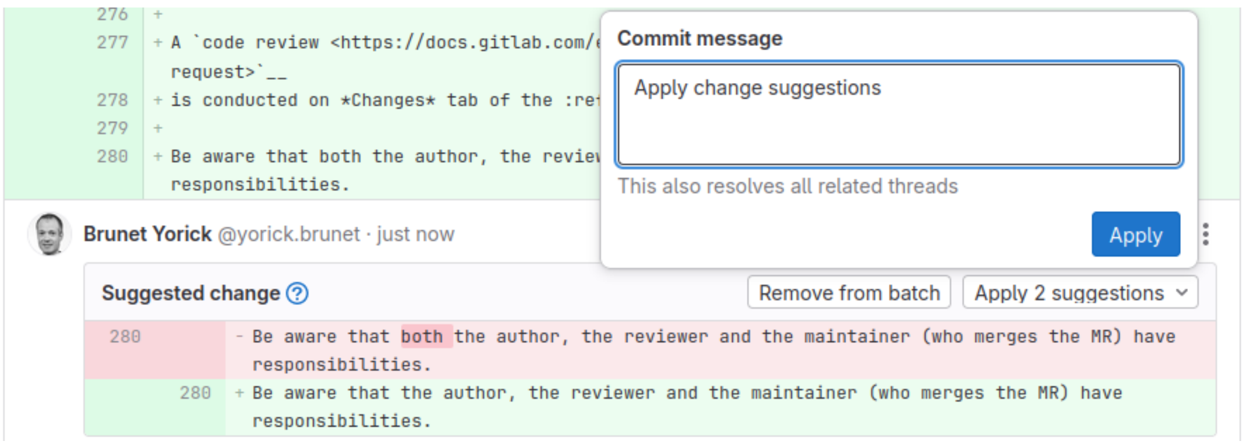
Fig. 5.37: GitLab — Code review — Solve suggestion – Thread resolved



Brunet Yorick @yorick.brunet · just now

```
280      - Be aware that both the author, the reviewer and the maintainer (who merges the MR) have
           responsibilities.
    280  + Be aware that the author, the reviewer and the maintainer (who merges the MR) have
           responsibilities.
```

Fig. 5.38: GitLab — Code review — Solve batch – Add next changes to batch



```
+
+ A `code review <https://docs.gitlab.com/
  request>`__
+ is conducted on *Changes* tab of the :re
+
+ Be aware that both the author, the reviev
  responsibilities.
```

Commit message

Apply change suggestions

This also resolves all related threads

Brunet Yorick @yorick.brunet · just now

```
280      - Be aware that both the author, the reviewer and the maintainer (who merges the MR) have
           responsibilities.
    280  + Be aware that the author, the reviewer and the maintainer (who merges the MR) have
           responsibilities.
```

Fig. 5.39: GitLab — Code review — Solve batch – Apply batch

5.7 Wiki

A wiki is available for every project, is enabled by default and uses a Git repository to version itself.

The wiki should not be the main documentation of the project; a Sphinx documentation should be used for that. However, the wiki can contain data and information that are not directly related to the project, such as meeting minutes, machine configurations, etc. As the wiki is not versioned in the main Git repository, its update does not need to implement a *Git — Workflow*.

- Use menu *Wiki* to display the wiki (see Fig. 5.40):
  - the home page must be created with button *Create your first page*,
  - next pages can be created with button *New page* from any page,
  - page title,
  - page content,
  - edit page,
  - page history.

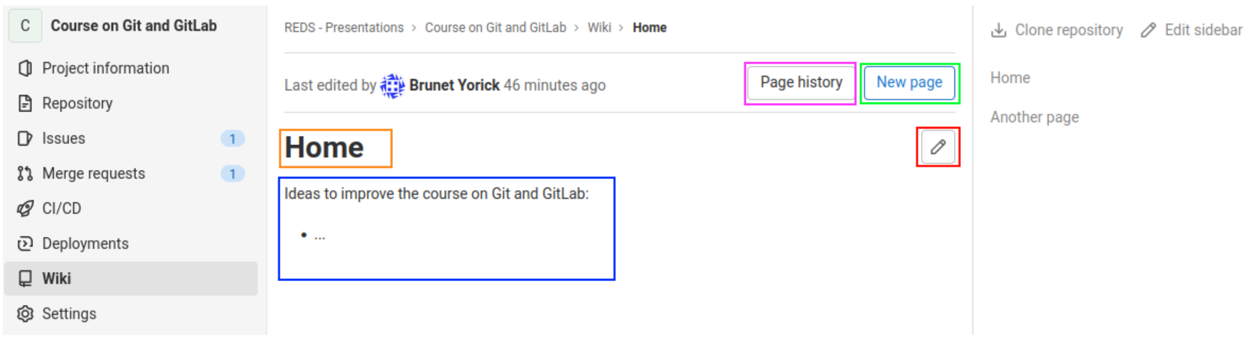

C Course on Git and GitLab
Project information
Repository
Issues 1
Merge requests 1
CI/CD
Deployments
Wiki
Settings
REDS - Presentations > Course on Git and GitLab > Wiki > Home
Last edited by Brunet Yorick 46 minutes ago
Page history
New page
Home
Ideas to improve the course on Git and GitLab:
• ...
Clone repository
Edit sidebar
Home
Another page

Fig. 5.40: GitLab — Wiki — Page overview

- When creating a page, some details must be provided (see Fig. 5.41):
  - page title,
  - page content.

  The text can be toggled between edition and preview.
- Use button *Edit sidebar* to edit the sidebar (see Fig. 5.42). This allows reordering files.

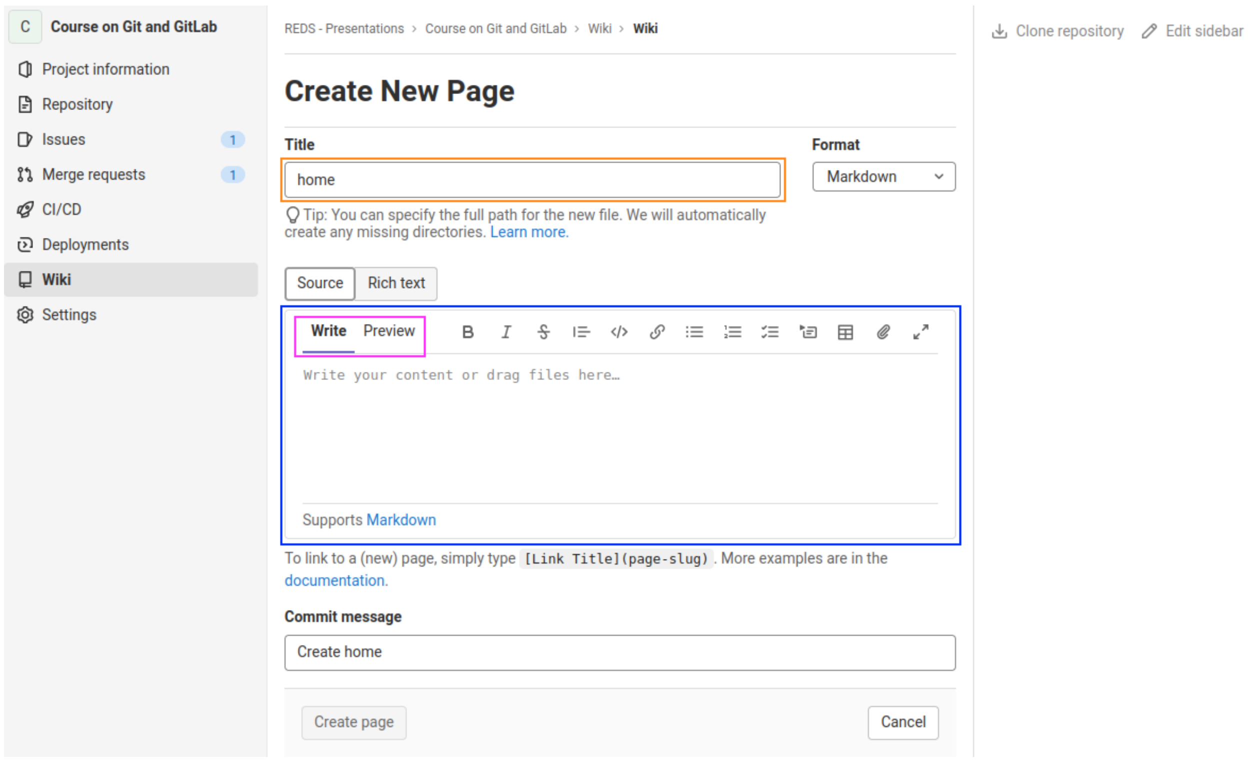
  Keep the sidebar title as it is, but modify the sidebar content. The text can be toggled between edition and preview.

5.8 Settings

The settings are usually reserved for maintainers and owners of a project. The settings do not generally need to be modified, but two pages are still important to know.

- Use *Settings* → *Merge requests* to modify settings associated to merge requests (see Fig. 5.43).

  Keep the merge commit and the capacity to delete source branch. But if you set up a CI pipeline, do modify the merge checks options.



Create New Page

Title: home

Format: Markdown

Tip: You can specify the full path for the new file. We will automatically create any missing directories. Learn more.

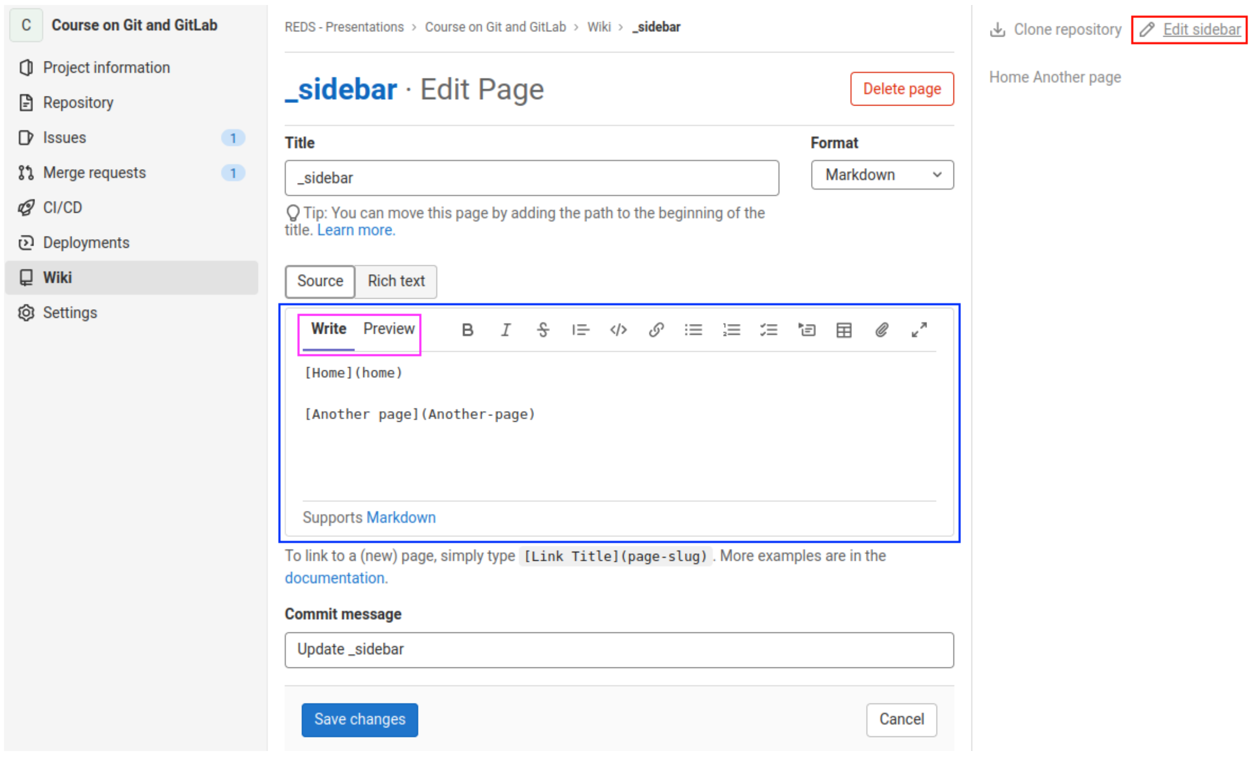
Write your content or drag files here…

Supports Markdown

To link to a (new) page, simply type `[Link Title](page-slug)`. More examples are in the documentation.

Commit message: Create home

Fig. 5.41: GitLab — Wiki — Create new page



_sidebar · Edit Page

Title: _sidebar

Format: Markdown

Tip: You can move this page by adding the path to the beginning of the title. Learn more.

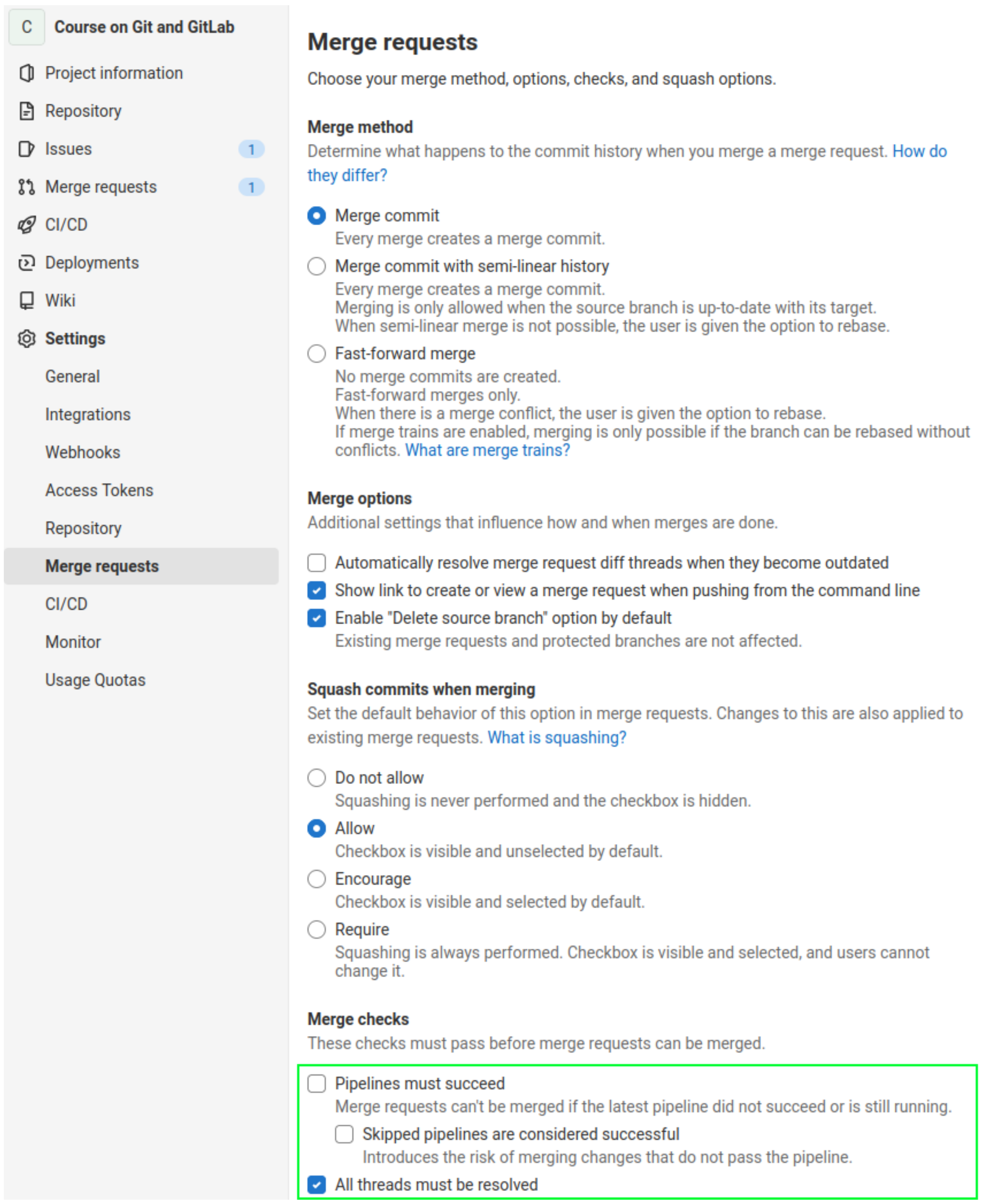
```
[Home](home)

[Another page](Another-page)
```

Supports Markdown

To link to a (new) page, simply type `[Link Title](page-slug)`. More examples are in the documentation.

Commit message: Update _sidebar

Fig. 5.42: GitLab — Wiki — Edit sidebar



Merge requests

Choose your merge method, options, checks, and squash options.

Merge method

Determine what happens to the commit history when you merge a merge request. How do they differ?

[✓ Merge commit]
Every merge creates a merge commit.

Merge commit with semi-linear history
Every merge creates a merge commit.
Merging is only allowed when the source branch is up-to-date with its target.
When semi-linear merge is not possible, the user is given the option to rebase.

Fast-forward merge
No merge commits are created.
Fast-forward merges only.
When there is a merge conflict, the user is given the option to rebase.
If merge trains are enabled, merging is only possible if the branch can be rebased without conflicts. What are merge trains?

Merge options

Additional settings that influence how and when merges are done.

- [ ] Automatically resolve merge request diff threads when they become outdated
- [✓] Show link to create or view a merge request when pushing from the command line
- [✓] Enable "Delete source branch" option by default
  Existing merge requests and protected branches are not affected.

Squash commits when merging

Set the default behavior of this option in merge requests. Changes to this are also applied to existing merge requests. What is squashing?

Do not allow
Squashing is never performed and the checkbox is hidden.

[✓ Allow]
Checkbox is visible and unselected by default.

Encourage
Checkbox is visible and selected by default.

Require
Squashing is always performed. Checkbox is visible and selected, and users cannot change it.

Merge checks

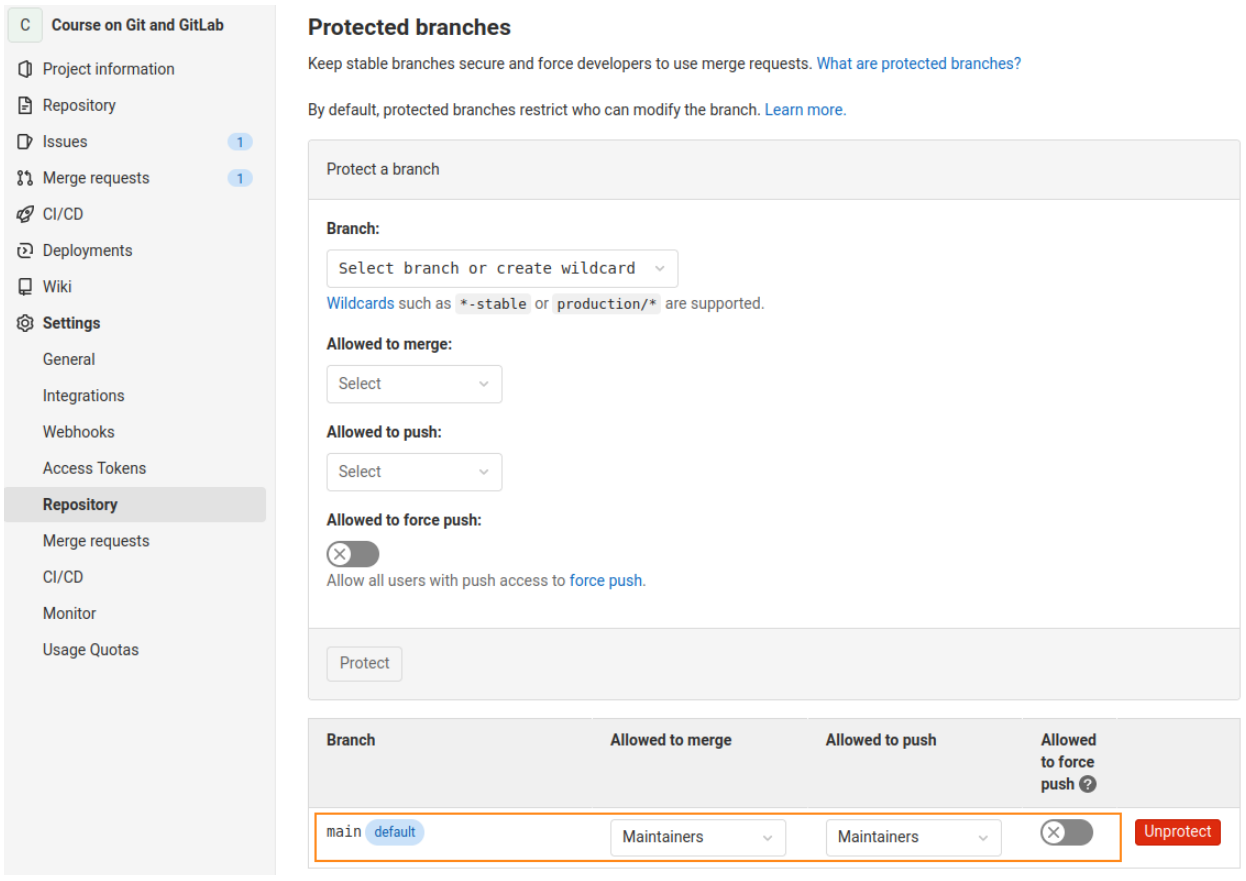
These checks must pass before merge requests can be merged.

- [ ] Pipelines must succeed
  Merge requests can't be merged if the latest pipeline did not succeed or is still running.
  - [ ] Skipped pipelines are considered successful
    Introduces the risk of merging changes that do not pass the pipeline.
- [✓] All threads must be resolved

Fig. 5.43: GitLab — Settings — Settings for merge requests

- Use *Settings* → *Repository* to modify settings associated to branches (see Fig. 5.43).

  One important setting is the protected branches. The list of protected branches is displayed, which allows to update the default settings.



Protected branches

Keep stable branches secure and force developers to use merge requests. What are protected branches?

By default, protected branches restrict who can modify the branch. Learn more.

Protect a branch

Branch:

Select branch or create wildcard

Wildcards such as `*-stable` or `production/*` are supported.

Allowed to merge:

Select

Allowed to push:

Select

Allowed to force push:

Allow all users with push access to force push.

Protect

Branch	Allowed to merge	Allowed to push	Allowed to force push	
main default	Maintainers	Maintainers		Unprotect

Fig. 5.44: GitLab — Settings — Settings for branches

5.9 Team planning

GitLab offers a way to plan and track work of a team. The solution combines labels, *Issues* and issue boards.

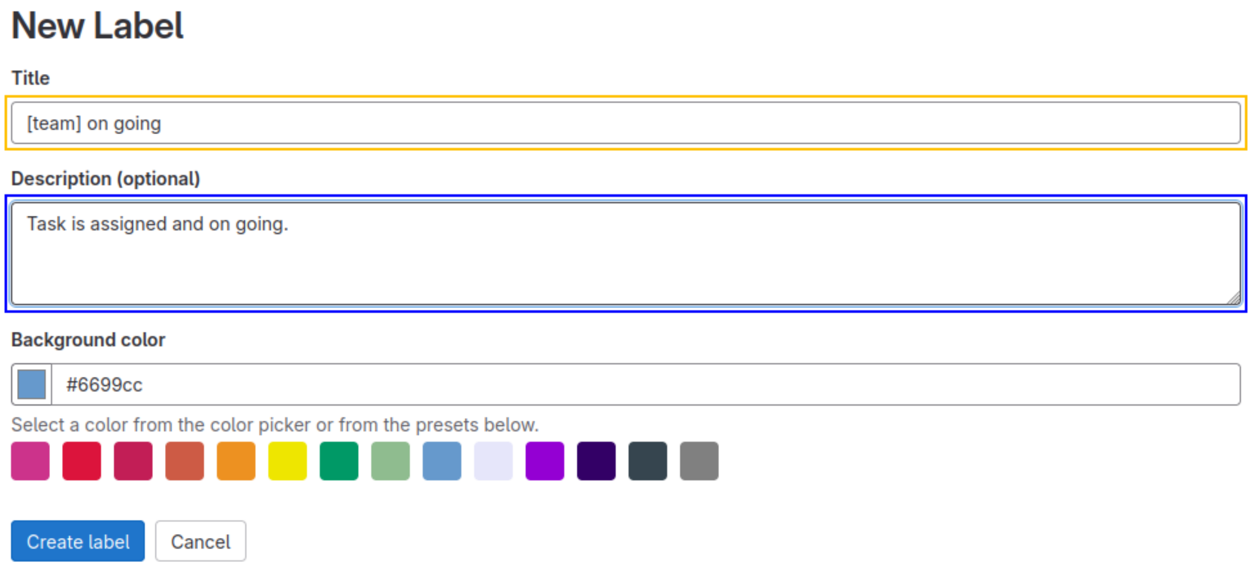
The first step is to create labels in *Manage* → *Labels*. The issue boards contain already lists for *Open* and *Closed* issues. Other planning statuses are managed with labels, e.g.:

- [team] ready
- [team] on going
- [team] on review

Click on button *New label* to create a new label. Specify the title and optionally a description (see Fig. 5.45). Finally click on *Create label*.

Labels can also describe *activity* statuses, e.g. bug, discussion, documentation, enhancement. Thus the complexity is to combine both kind of statuses.

Some labels can be prioritised to give them more weights and priority over others, this is especially useful when a project uses many labels. Fig. 5.46 shows the list of labels.



New Label

Title

[team] on going

Description (optional)

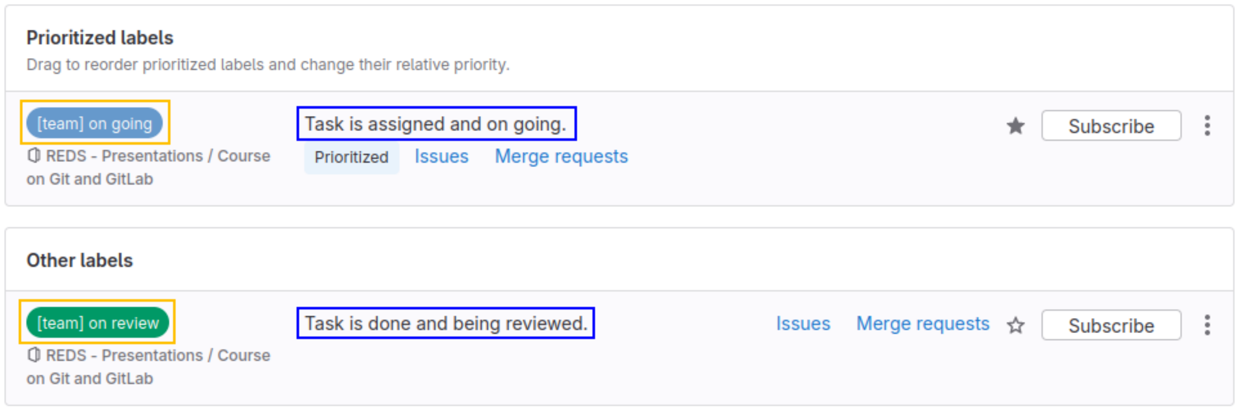
Task is assigned and on going.

Background color

#6699cc

Select a color from the color picker or from the presets below.

Fig. 5.45: GitLab — Team Planning — Create a label



Prioritized labels

Drag to reorder prioritized labels and change their relative priority.

[team] on going

REDS - Presentations / Course on Git and GitLab

Task is assigned and on going.

Other labels

[team] on review

REDS - Presentations / Course on Git and GitLab

Task is done and being reviewed.

Fig. 5.46: GitLab — Team Planning — List of labels

The second step is to open new lists in the issue boards. The issue boards are available in *Plan* → *Issue boards* or from the pinned shortcuts.

Click on button *New list* to create a new list, select the associated label and finally click on button *Add to board* (Fig. 5.47 and Fig. 5.48).

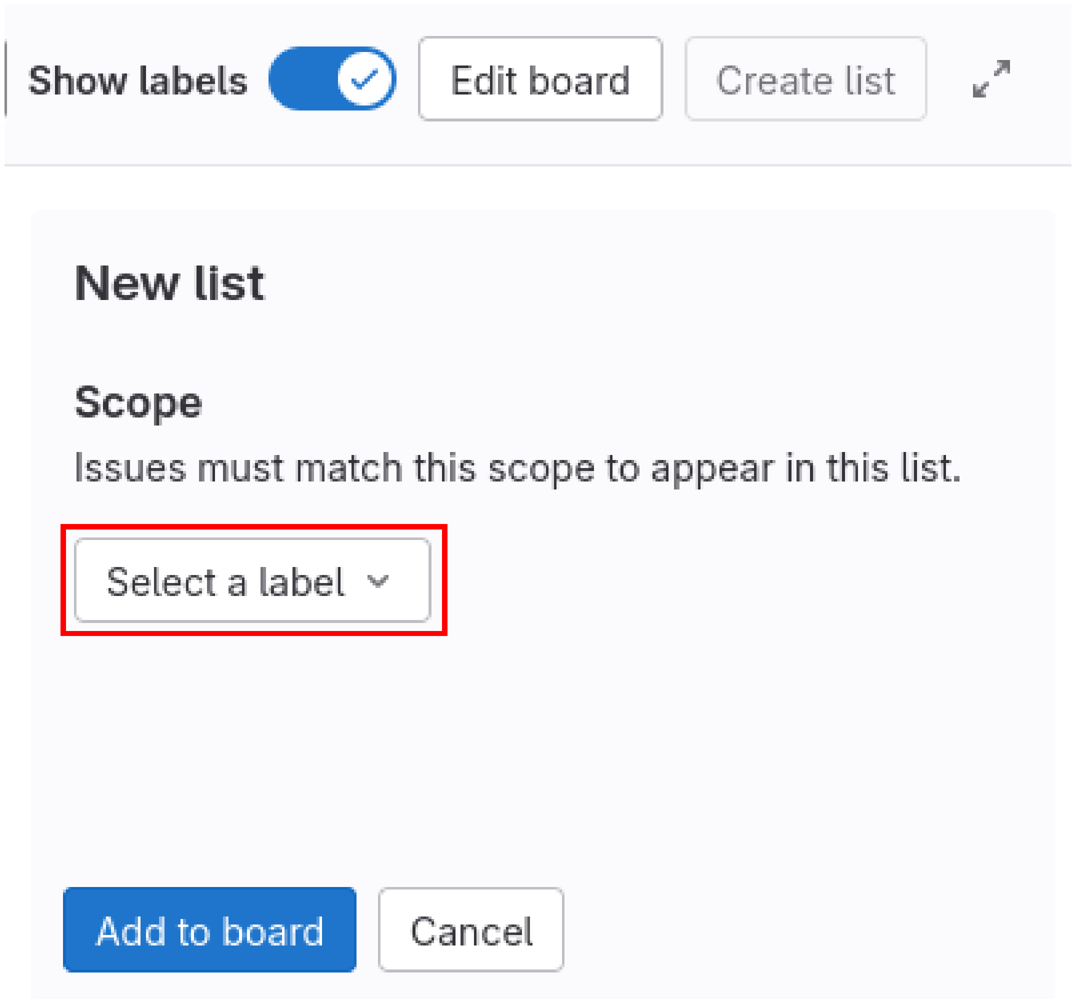


Fig. 5.47: GitLab — Team Planning — Create a new board list (1/2)

Fig. 5.49 shows the board after the creation of two lists.

By default, only the board *Development* exists. But other boards can be created, e.g. management or personal board. Click on the current board's name and then on *Create new board* (see Fig. 5.50). In addition to the title, display options are given, such as which default lists are shown and other specific scopes: only 1 defined label, a milestone, an assignee, etc., or a combination of all these scopes (see Fig. 5.51).

The third step is to create an issue (see *Issues*) and to assign the issue to a board. The assignation is done either via drag-n-drop on the board (see Fig. 5.52) or directly by assigning a label to an issue. It is also possible to order issues within a list, to set some priority for example (see Fig. 5.53).

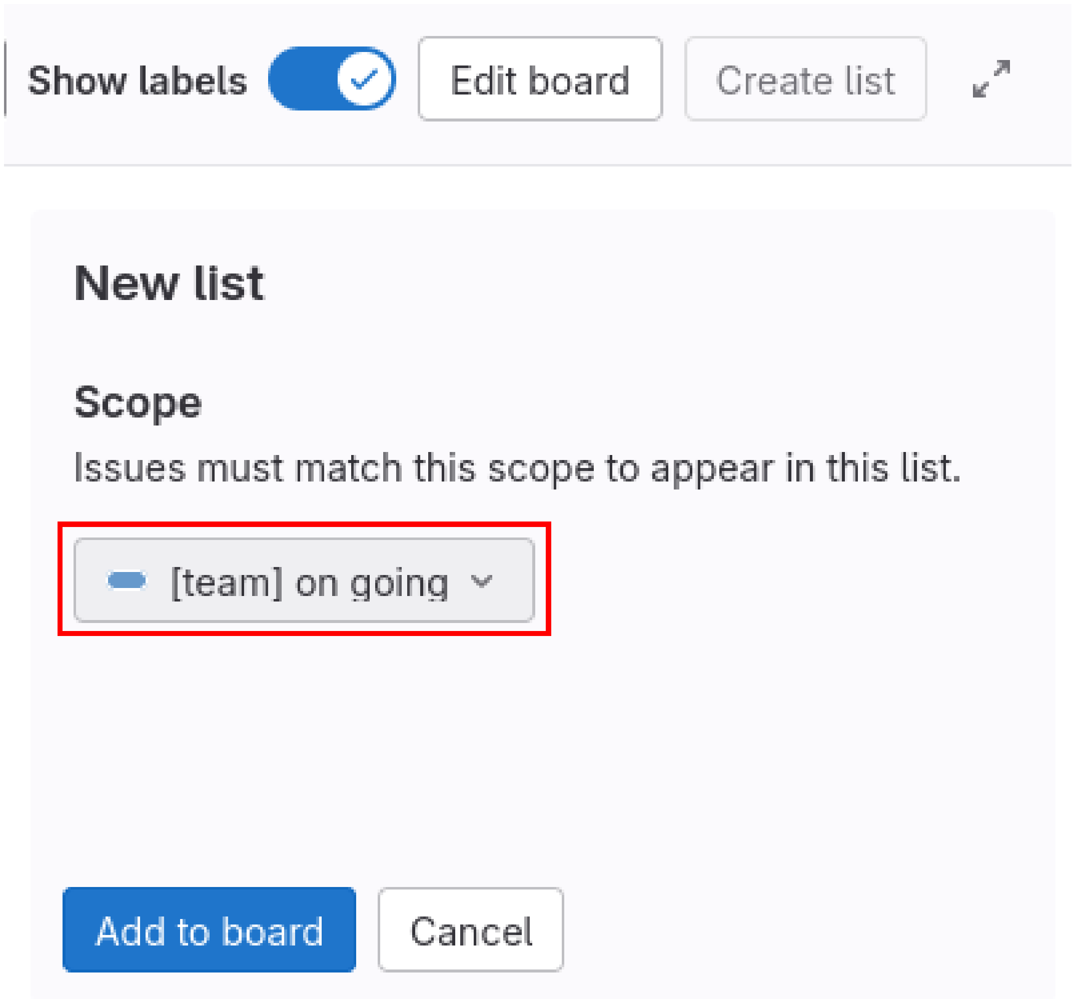


Fig. 5.48: GitLab — Team Planning — Create a new board list (2/2)

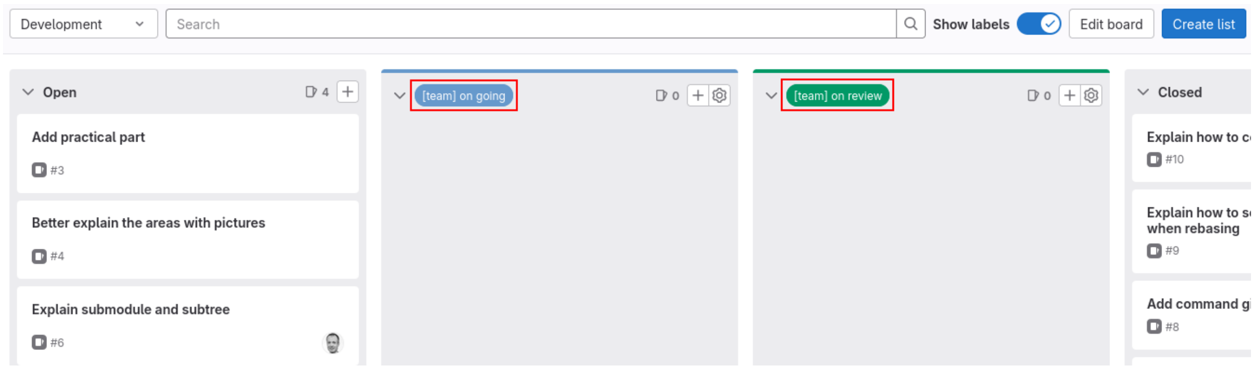


Fig. 5.49: GitLab — Team Planning — Board lists

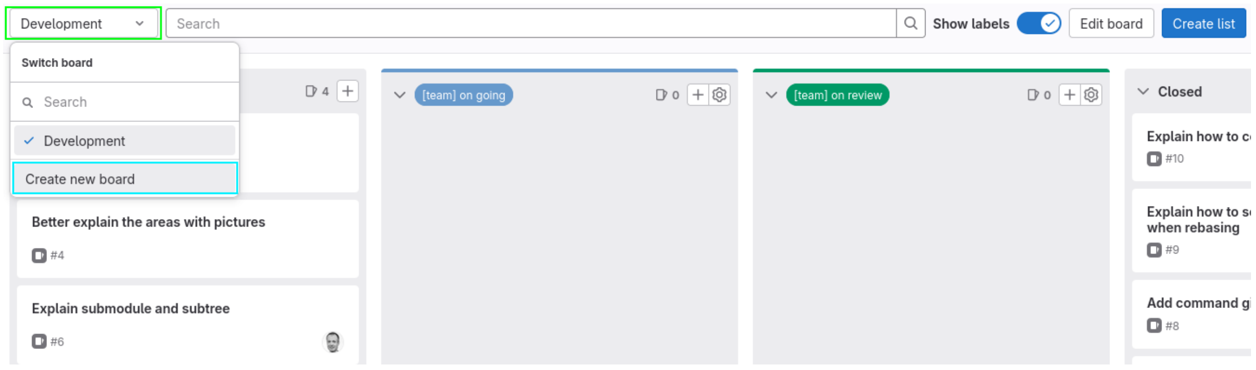
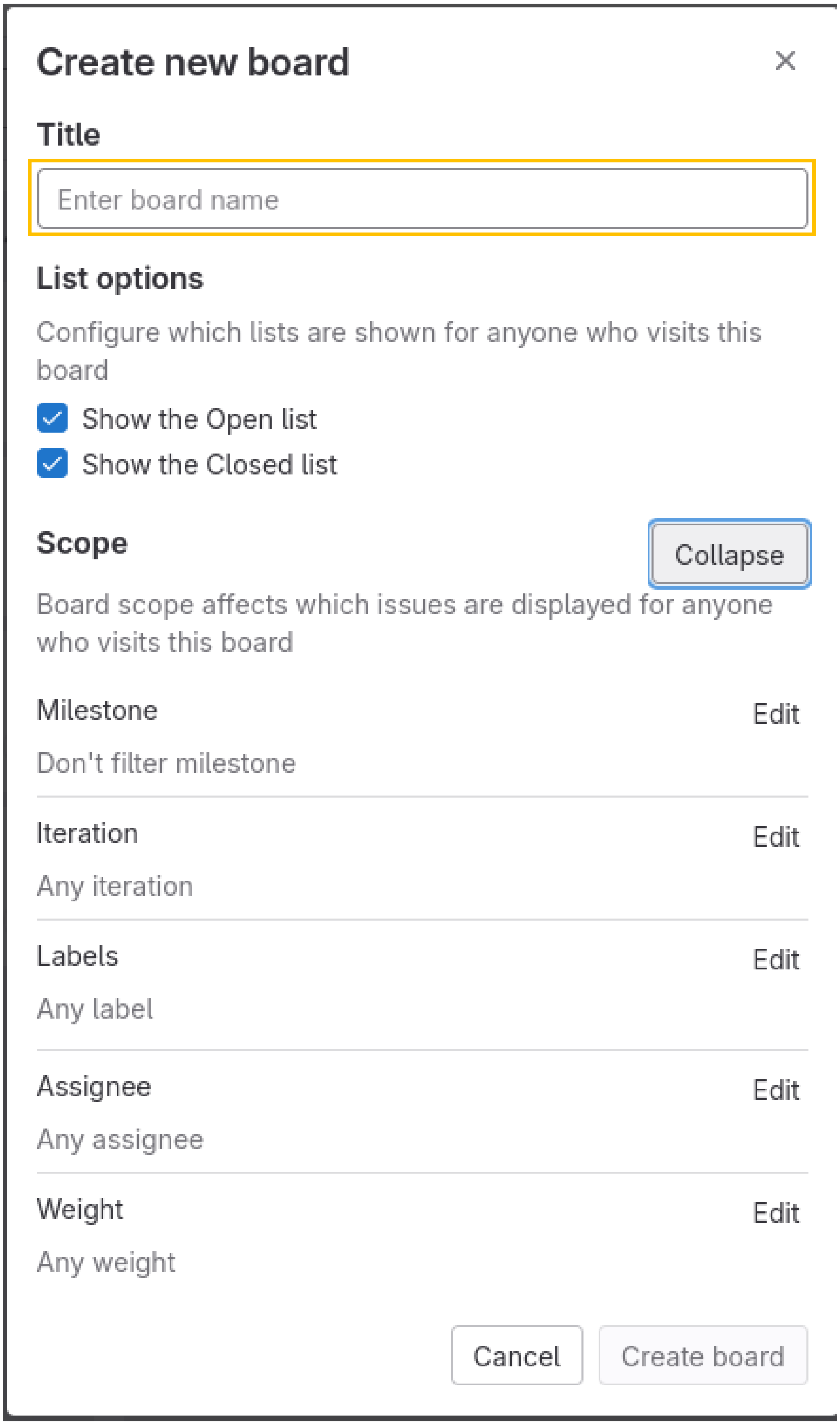


Fig. 5.50: GitLab — Team Planning — New board (1/2)



Create new board

Title

Enter board name

List options

Configure which lists are shown for anyone who visits this board

✓ Show the Open list

✓ Show the Closed list

Scope

Board scope affects which issues are displayed for anyone who visits this board

Milestone: Don't filter milestone

Iteration: Any iteration

Labels: Any label

Assignee: Any assignee

Weight: Any weight

Fig. 5.51: GitLab — Team Planning — New board (2/2)

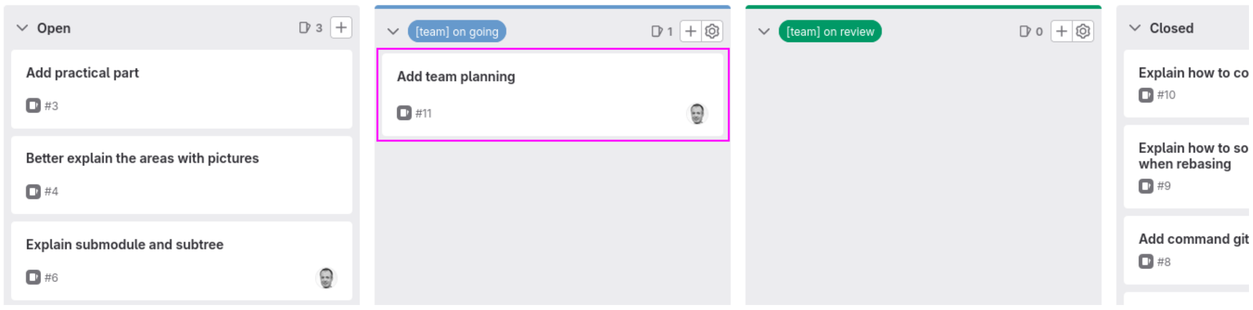


Fig. 5.52: GitLab — Team Planning — Move an issue to list "on going"

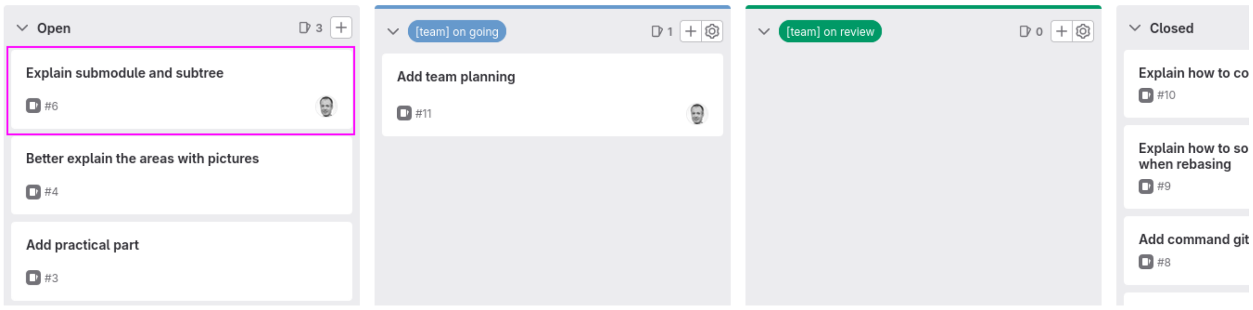


Fig. 5.53: GitLab — Team Planning — Move an issue at the top of the list "Open"

CHAPTER

SIX

GIT — SYNC

6.1 Remote

Git is a decentralized versionning software. It means that, contrary to SVN for example, a repository can have several entities, each existing at a different location with its own life-cycle.

The entities may be sometimes synchronised, which is actually recommended to keep these entities part of the same repository.

The synchronisation can happen via different methods, but in general all including a "main" repository remotely available with which the synchronisation occurs.

The "main" repository is called *remote*. A repository can have several remotes, the default being usually named `origin`. The use of multiple remotes may happen when a development process does not use shared branches as we are used to use with GitLab: people are not working on shared branches in a single remote repository but each person has its own remote where it pushes its "private" branch. Or another example could be when forking a project and you still want to pull commits from the forked project while updating yours. That being said avoid using multiple remotes just for fun!

We saw in *Git — Basics* the two ways to initialise a local repository, either using git-init or git-clone.

`git clone` actually clones an existing "main" repository and thus uses it as origin.

`git init` only initialises a local repository, the remote configuration being missing. To configure the remote repository of a local repository, use the following commands:

Listing 6.1: Example of **`git remote add`** for repository `origin`

```
$ git remote add origin https://reds-gitlab.heig-vd.ch/reds-general/git-and-gitlab-
↪course.git
$ git push -u origin --all
$ git push -u origin --tags
```

A remote's details can be shown with:

Listing 6.2: Example of **`git remote show`** for repository `origin`

```
$ git remote show origin
* remote origin
Fetch URL: https://reds-gitlab.heig-vd.ch/reds-presentations/c-git-and-gitlab.git
Push  URL: https://reds-gitlab.heig-vd.ch/reds-presentations/c-git-and-gitlab.git
HEAD branch: main
Remote branches:
    1-release-first-version tracked
    main                    tracked
```

(continues on next page)

(continued from previous page)

```
Local branches configured for 'git pull':
    1-release-first-version merges with remote 1-release-first-version
    main                    merges with remote main
Local refs configured for 'git push':
    1-release-first-version pushes to 1-release-first-version (up to date)
    main                    pushes to main                    (up to date)
```

6.2 A note on history modifications

As long as a commit is only on the local repository (non-published), the user can freely update the history of his/her local repository. But as soon as the commit is pushed (published) to the remote repository, playing with the history becomes hazardous and is discouraged.

We have seen *Rebase* and will see in the next chapter *Rewrite history with interactive rebase*.

These commands do modify the history!

It is therefore important to know when to use them:

- on non-published commits: totally fine, use it!
- on published commits: hazardous situation foreseen, be sure that all concerned persons are aware of your intentions!

Git — Workflow – *Step 8: Align your work on origin/main* gives some recomendations on the usage of *Rebase* and *Merge*.

6.3 Fetch

The command git-fetch downloads the changes of references from the remote. The content of the local repository is not modified, but the new references are available for use. The new and deleted branches or tags are displayed as well as which branches have updates on the remote.

State of the local (left) and remote (right) repositories before *fetch* (see Fig. 6.1):

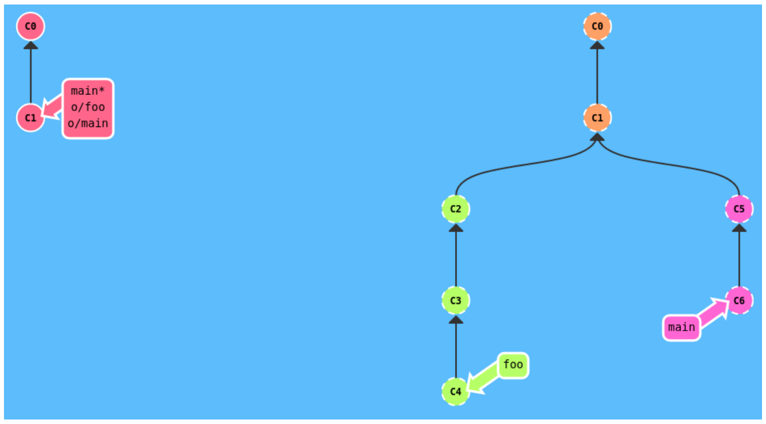


Fig. 6.1: Git — Fetch, pull and push — State before *fetch*

Fetch remote repository:

Listing 6.3: Example of **git fetch**

```
$ git fetch
From https://reds-gitlab.heig-vd.ch/reds-general/python-course
- [deleted]         (none)     -> origin/v2
remote: Enumerating objects: 46, done.
remote: Counting objects: 100% (43/43), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 30 (delta 20), reused 23 (delta 13), pack-reused 0
Unpacking objects: 100% (30/30), 4.00 KiB | 256.00 KiB/s, done.
e14f87c..76e03c2  main       -> origin/main
* [new branch]      enable_ci  -> origin/enable_ci
* [new tag]         v2.0       -> v2.0
* [new tag]         v2.1       -> v2.1
```

State of the local (left) and remote (right) repositories after *fetch* (see Fig. 6.2):

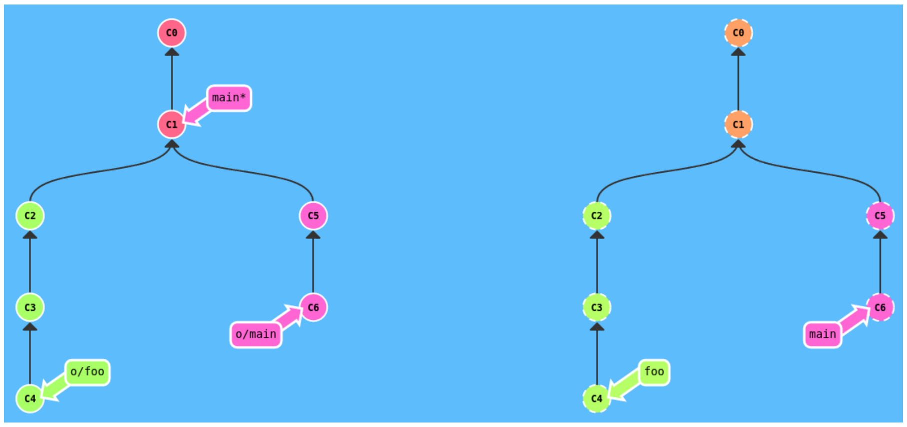


Fig. 6.2: Git — Fetch, pull and push — State after *fetch*

6.4 Pull

The command git-pull executes the actions of **git fetch** and, in addition, applies the changes received to the current branch either with a merge or a rebase, depending on the `pull.rebase` config (see *Setup*). The command displays to which commit the branch is being updated as well as which files are being updated.

Pull remote repository:

Listing 6.4: Example of **git pull** — pull with preceeding fetch

```
$ git pull
Updating e14f87c..76e03c2
Fast-forward
.vscode/settings.json |   2 +
README.md             |   2 +-
code/decorator.py     | 131 ++++++++++++++++++++++++++++++++++
course_python.tex     |  12 ++++
img/autoapi1.png      | Bin 0 -> 24407 bytes
```

(continues on next page)

(continued from previous page)

```
tex/environment.tex    |  74 ++++++++++---------
23 files changed, 1524 insertions(+), 87 deletions(-)
create mode 100644 code/decorator.py
create mode 100644 img/autoapi1.png
Successfully rebased and updated refs/heads/main.
```

State of the local (left) and remote (right) repositories after *pull main* (see Fig. 6.3):

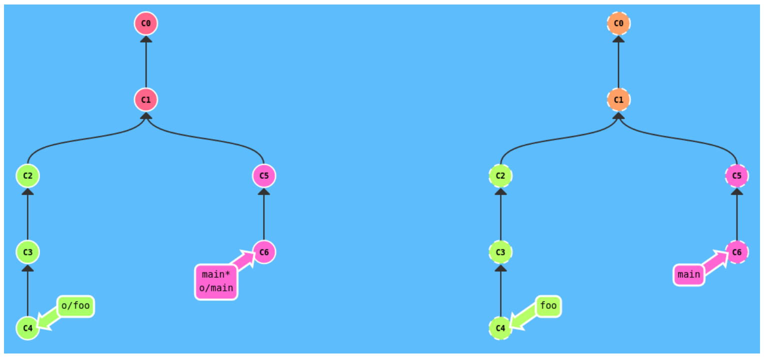


Fig. 6.3: Git — Fetch, pull and push — State after *pull main*

State of the local (left) and remote (right) repositories after *pull foo* (see Fig. 6.4):

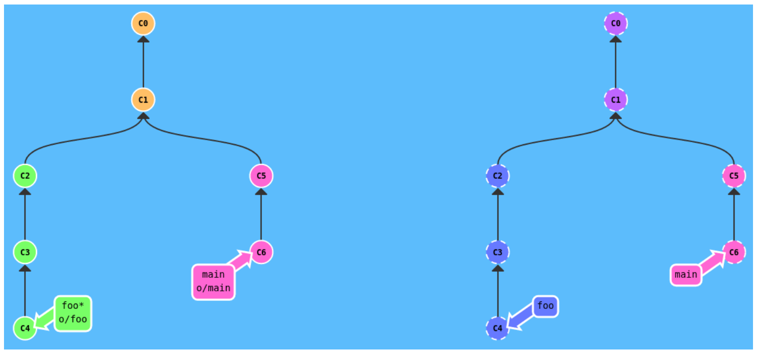


Fig. 6.4: Git — Fetch, pull and push — State after *pull foo*

Listing 6.5: Example of **`git pull`** — pull without preceeding fetch

```
$ git pull
From https://reds-gitlab.heig-vd.ch/reds-general/python-course
- [deleted]         (none)     -> origin/v2
remote: Enumerating objects: 46, done.
remote: Counting objects: 100% (43/43), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 30 (delta 20), reused 23 (delta 13), pack-reused 0
Unpacking objects: 100% (30/30), 4.00 KiB | 292.00 KiB/s, done.
e14f87c..76e03c2  main       -> origin/main
* [new branch]      enable_ci  -> origin/enable_ci
* [new tag]         v2.0       -> v2.0
* [new tag]         v2.1       -> v2.1
```

(continues on next page)

(continued from previous page)

```
Updating e14f87c..76e03c2
Fast-forward
README.md             |   2 +-
code/decorator.py     | 131 ++++++++++++++++++++++++++++++++++
course_python.tex     |  12 ++++
img/autoapi1.png      | Bin 0 -> 24407 bytes
tex/environment.tex   |  74 ++++++++++----------
23 files changed, 1524 insertions(+), 87 deletions(-)
create mode 100644 code/decorator.py
create mode 100644 img/autoapi1.png
Successfully rebased and updated refs/heads/main.
```

Listing 6.6: Example of **`git pull`** — pull without changes to apply

```
$ git pull
Already up to date.
Successfully rebased and updated refs/heads/main.
```

6.5 Push

The command git-push sends the current changes of the branch or a tag to the remote repository.

Listing 6.7: Full example of **`git push`**

```
$ git log --graph --all
* commit 534e252bba2534170f849ddda754d61a449fd480 (HEAD -> 1-release-first-version)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Wed Nov 16 07:13:55 2022 +0100
|
|     Fix repr of --
|
* commit 0a84647c72a34737c08447d098d1fef297c60d67 (origin/1-release-first-version)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Wed Nov 9 10:04:54 2022 +0100
|
|     Add captions to code-block

$ git push
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 519 bytes | 519.00 KiB/s, done.
Total 6 (delta 5), reused 0 (delta 0)
To https://reds-gitlab.heig-vd.ch/reds-presentations/c-git-and-gitlab.git
   0a84647..534e252  1-release-first-version -> 1-release-first-version

$ git log --graph --all
* commit 534e252bba2534170f849ddda754d61a449fd480 (HEAD -> 1-release-first-version,␣
↪origin/1-release-first-version)
```

(continues on next page)

(continued from previous page)

```
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Wed Nov 16 07:13:55 2022 +0100
|
|     Fix repr of --
```

It may happen that new branches, not created from GitLab, are created locally and pushed to the remote repository. In this case, the *upstream* branch (the one on the remote repository) must be created with option --set-upstream <remote> <branch>.

Listing 6.8: Full example of **git push --set-upstream**

```
$ git checkout -b test
Switched to a new branch 'test'

$ git push
fatal: The current branch test has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin test

$ git push --set-upstream origin test
Total 0 (delta 0), reused 0 (delta 0)
To https://reds-gitlab.heig-vd.ch/reds-presentations/c-git-and-gitlab.git
 * [new branch]      test -> test
Branch 'test' set up to track remote branch 'test' from 'origin'.
```

6.6 Exercises

Pair with your neighbour(s) for these exercises. For each step, write down which command(s) you used.

1. Go to the sandbox and create a project for your group; add the all members of the group to the project
2. First round: each team member works on his/her own branch
3. Following the *Git — Workflow*, create one issue per team member and assign the team member to the issue
4. Following the *Git — Workflow*, the developer works on his/her branch and pushes some times his/her work
5. Following the *Git — Workflow*, the developer opens a merge request and assigns another team member as reviewer
6. Following the *Git — Workflow*, the reviewe reviews the merge request and add at least one comment per merge request
7. Following the *Git — Workflow*, the developer answers the comment
8. Following the *Git — Workflow*, the reviewer validates the merge request and merges it
9. Second round: all team members work on the same branch
10. Following the *Git — Workflow*, create one issue and assign all team members to the issue
11. Try to create conflicts while pushing/pulling/merging/rebasing on the same branch
    - Resolve the conflicts by following *Conflict management*

CHAPTER

SEVEN

GIT — ADVANCED

7.1 Reference to current branch and ancestors

`HEAD` is the generic reference to the current checked out branch/commit/tag.

Listing 7.1: Example of `HEAD`

```
$ git checkout 9253217
Note: switching to '9253217'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

git switch -c <new-branch-name>

Or undo this operation with:

git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 9253217 Add material to Basics

$ git log
commit 925321748e8373cd1b51af2a5ee2e1445d1707a0 (HEAD)
Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
Date:   Thu Oct 13 16:07:08 2022 +0200

    Add material to Basics

$ git switch -
Previous HEAD position was 9253217 Add material to Basics
Switched to branch '1-release-first-version'

$ git log
commit 3b7422eae46e3abbc69261ee8ee7fd9924b7ca2e (HEAD -> 1-release-first-version, origin/
↪1-release-first-version)
```

(continues on next page)

(continued from previous page)

```
Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
Date:   Thu Oct 13 16:10:33 2022 +0200

    Include some files
```

A suffix ^ to a revision object means the first parent of that object. A suffix ^<n> means the <n>th parent of that object. ^ is equivalent to ^1.

A suffix ~ to a revision object means the first parent of that object. A suffix ~<n> means the <n>th generation of first parent ancestor. ~3 is equivalent to ^^^ which is equivalent to ^1^1^1.

The example below should help make more sense about ^ and ~:

Listing 7.2: Example of suffixes for ^ and ~ (src doc rev-parse)

```
G   H   I   J
 \ /     \ /
  D   E   F
   \  |  / \
    \ | /   |
     \|/    |
      B     C
       \   /
        \ /
         A

A =      = A^0
B = A^   = A^1     = A~1
C =      = A^2
D = A^^  = A^1^1   = A~2    = A~^
E = B^2  = A^^2             = A~^2
F = B^3  = A^^3             = A~^3
G = A^^^ = A^1^1^1 = A~3
H = D^2  = B^^2    = A^^^2  = A~2^2
I = F^   = B^3^    = A^^3^  = A~^3~
J = F^2  = B^3^2   = A^^3^2
```

The command git-revparse provides some information on the repository, such as the commit hash.

Listing 7.3: Full example of **git rev-parse**

```
$ git log --oneline
dc2790d (HEAD -> 1-release-first-version) Add material to GUI
9253217 Add material to Basics
72ef095 Add GitLab
59e9ba9 Add Workflow
7df6cbb Add GUI
aee132f Add Basics
93b7cdd Add Introduction
01ffd04 (origin/main, main) Initial commit with structure

$ git rev-parse HEAD
dc2790d00e6d96db25b71c2f35410f90818ea048
$ git rev-parse HEAD~1
```

(continues on next page)

(continued from previous page)

```
925321748e8373cd1b51af2a5ee2e1445d1707a0
$ git rev-parse --short HEAD~2
72ef095
```

Further information is available on specifying revisions and ranges for Git.

7.2 Reset

The developper may want to reset its repository to some previous states, either to cancel some commits or to reorganise them. The command git-reset resets the current HEAD to the specified state.

Three main possibilities:

- `--soft`: does not touch the index file or the working tree at all but resets the head to <commit>,
- `--mixed` (default): resets the index but not the working tree, changed files are preserved but not marked for commit,
- `--hard`: resets the index and working tree and any changes to tracked files in the working tree since <commit> are discarded.

Example:

Listing 7.4: Example of **`git reset --soft`**

```
$ git log -n 2
commit 3b7422eae46e3abbc69261ee8ee7fd9924b7ca2e (HEAD -> 1-release-first-version, origin/
↪1-release-first-version)
Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
Date:   Thu Oct 13 16:10:33 2022 +0200

    Include some files

commit dc2790d00e6d96db25b71c2f35410f90818ea048
Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
Date:   Thu Oct 13 16:10:11 2022 +0200

    Add material to GUI

$ git status
On branch 1-release-first-version
Your branch is up to date with 'origin/1-release-first-version'.

Changes to be committed:
(use "git restore --staged <file>..." to unstage)
    new file:   docs/source/advanced.rst
    modified:   docs/source/index.rst

Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
    modified:   docs/source/advanced.rst
    modified:   docs/source/introduction.rst
```

(continues on next page)

(continued from previous page)

```
$ git reset --soft HEAD~1

$ git log -n 1
commit dc2790d00e6d96db25b71c2f35410f90818ea048 (HEAD -> 1-release-first-version)
Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
Date:   Thu Oct 13 16:10:11 2022 +0200

    Add material to GUI

$ git status
On branch 1-release-first-version
Your branch is behind 'origin/1-release-first-version' by 1 commit, and can be fast-
↪forwarded.
(use "git pull" to update your local branch)

Changes to be committed:
(use "git restore --staged <file>..." to unstage)
    new file:   docs/source/advanced.rst
    modified:   docs/source/conf.py
    modified:   docs/source/index.rst

Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
    modified:   docs/source/advanced.rst
    modified:   docs/source/introduction.rst
```

Listing 7.5: Example of **`git reset --mixed`**

```
$ git status
On branch 1-release-first-version
Your branch is up to date with 'origin/1-release-first-version'.

Changes to be committed:
(use "git restore --staged <file>..." to unstage)
    new file:   docs/source/advanced.rst
    modified:   docs/source/index.rst

Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
    modified:   docs/source/advanced.rst
    modified:   docs/source/introduction.rst

$ git reset --mixed HEAD~1
Unstaged changes after reset:
M    docs/source/conf.py
M    docs/source/index.rst
M    docs/source/introduction.rst

$ git status
```

(continues on next page)

(continued from previous page)

```
On branch 1-release-first-version
Your branch is behind 'origin/1-release-first-version' by 1 commit, and can be fast-
↪forwarded.
(use "git pull" to update your local branch)

Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
    modified:   docs/source/conf.py
    modified:   docs/source/index.rst
    modified:   docs/source/introduction.rst
```

Listing 7.6: Example of **`git reset --hard`**

```
$ git status
On branch 1-release-first-version
Your branch is up to date with 'origin/1-release-first-version'.

Changes to be committed:
(use "git restore --staged <file>..." to unstage)
    new file:   docs/source/advanced.rst
    modified:   docs/source/index.rst

Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
    modified:   docs/source/advanced.rst
    modified:   docs/source/introduction.rst

$ git reset --hard HEAD~1
HEAD is now at dc2790d Add material to GUI

$ git status
On branch 1-release-first-version
Your branch is behind 'origin/1-release-first-version' by 1 commit, and can be fast-
↪forwarded.
(use "git pull" to update your local branch)
```

7.3 Revert

The command git-revert reverts some existing commits by inverting the changes. This is safe to use on published commits.

Example:

Listing 7.7: Full example of **`git revert`**

```
$ git log --oneline
3b7422e (HEAD -> 1-release-first-version, origin/1-release-first-version) Include some␣
↪files
dc2790d Add material to GUI
```

(continues on next page)

(continued from previous page)

```
9253217 Add material to Basics
72ef095 Add GitLab
59e9ba9 Add Workflow
7df6cbb Add GUI
aee132f Add Basics
93b7cdd Add Introduction
01ffd04 (origin/main, main) Initial commit with structure

$ git show dc2790d
commit dc2790d00e6d96db25b71c2f35410f90818ea048
Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
Date:   Thu Oct 13 16:10:11 2022 +0200

    Add material to GUI

diff --git a/docs/source/gui.rst b/docs/source/gui.rst
index 28af942..b73c6aa 100644
--- a/docs/source/gui.rst
+++ b/docs/source/gui.rst
@@ -7,23 +7,54 @@ Git can be used with command line, but can also be used with a␣
↪graphical user in
[...]
 A list of (all) GUI clients on Linux can be found on the `Git homepage <https://git-scm.
↪com/download/gui/mac>`__.

+Suggestions for this platform:
+
+- `Git Fork <https://git-fork.com/>`__: the best client ever!
+
[...]

$ git revert dc2790d
[1-release-first-version 41a60d0] Revert "Add material to GUI"
1 file changed, 29 insertions(+), 60 deletions(-)
rewrite docs/source/gui.rst (67%)

$ git log --oneline
41a60d0 (HEAD -> 1-release-first-version) Revert "Add material to GUI"
3b7422e (origin/1-release-first-version) Include some files
dc2790d Add material to GUI
9253217 Add material to Basics
72ef095 Add GitLab
59e9ba9 Add Workflow
7df6cbb Add GUI
aee132f Add Basics
93b7cdd Add Introduction
01ffd04 (origin/main, main) Initial commit with structure

$ git show 41a60d0
commit 41a60d0885cec0543d46f44d2b965c1e4087bc4d (HEAD -> 1-release-first-version)
Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
Date:   Thu Oct 20 16:43:06 2022 +0200
```

(continues on next page)

(continued from previous page)

```
    Revert "Add material to GUI"

    This reverts commit dc2790d00e6d96db25b71c2f35410f90818ea048.

diff --git a/docs/source/gui.rst b/docs/source/gui.rst
index b73c6aa..28af942 100644
--- a/docs/source/gui.rst
+++ b/docs/source/gui.rst
@@ -7,54 +7,23 @@ Git can be used with command line, but can also be used with a␣
↪graphical user in
[...]
 A list of (all) GUI clients on Linux can be found on the `Git homepage <https://git-scm.
↪com/download/gui/mac>`__.

-Suggestions for this platform:
-
-- `Git Fork <https://git-fork.com/>`__: the best client ever!
-
[...]
```

7.4 Worktrees

The command git-worktree manages multiple working trees, allowing you to check out multiple branches at a time.

The worktree containing the main branch is called *main worktree*. It is available in the directory where the repository was cloned. The worktree containing another branch is called *linked worktree*. Its path depends on the user's choice when using the command **`git worktree add`**, but the path **should** be outside of the main worktree.

Worktrees make the use of *Stash* less useful. It is no longer necessary to stash some ungoing work to switch on another branch.

Listing 7.8: Full example of **`git worktree`** (part 1)

```
$ git worktree list
/path/to/course_gitlab  a66bf8c [main]

$ git fetch
From https://reds-gitlab.heig-vd.ch/reds-presentations/c-git-and-gitlab
 * [new branch]      7-add-command-worktree -> origin/7-add-command-worktree

$ git worktree add ../course_gitlabwt7 7-add-command-worktree
Preparing worktree (new branch '7-add-command-worktree')
Branch '7-add-command-worktree' set up to track remote branch '7-add-command-worktree'␣
↪from 'origin'.
HEAD is now at a66bf8c Merge branch '5-explain-how-to-manage-a-rebased-branch-for-other-
↪devs' into 'main'

$ git worktree list
/path/to/course_gitlab     a66bf8c [main]
/path/to/course_gitlabwt7  a66bf8c [7-add-command-worktree]
```

(continues on next page)

(continued from previous page)

```
$ git branch -v
+ 7-add-command-worktree a66bf8c Merge branch '5-explain-how-to-manage-a-...' into 'main'
* main                   a66bf8c Merge branch '5-explain-how-to-manage-a-...' into 'main'

$ cd ../course_gitlabwt7
$ git status
On branch 7-add-command-worktree
```

Work on your branch. If needed, you can go back to the main branch. Both worktrees are kept independent.

Listing 7.9: Full example of **`git worktree`** (part 2)

```
$ git status
On branch 7-add-command-worktree
Your branch is up to date with 'origin/7-add-command-worktree'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
      modified:   docs/source/advanced.rst
      modified:   docs/source/basics.rst

$ cd ../course_gitlab
$ git status
On branch main
Your branch is up to date with 'origin/main'.
```

When the work on the branch is done, the linked worktree can be removed. If you modified the branch, make sure to push the changes. In the example below, the branch as been modified and pushed.

Listing 7.10: Full example of **`git worktree`** (part 3)

```
$ git status
On branch main

$ git worktree remove ../course_gitlabwt7
$ git worktree list
/path/to/course_gitlab     a66bf8c [main]

$ git branch -v
+ 7-add-command-worktree d355ff5 Add section worktree
* main                   a66bf8c Merge branch '5-explain-how-to-manage-a-...' into 'main'
```

7.5 Clean the repository

The command git-clean removes untracked files from the repository.

Listing 7.11: Possible usages for **git clean**

```
$ git clean -fx # removes all untracked files
$ git clean -fX # removes files ignored by Git
```

The command git-count-objects counts unpacked number of objects and their disk consumption. The command git-repack packs unpacked objects in a repository. The command git-gc compresses file revisions, removes unreachable objects, packs references, prunes reflog, etc.

Listing 7.12: Example of **git count-objects** and **git gc**

```
$ git count-objects
253 objects, 33980 kilobytes
$ git repack
Enumerating objects: 227, done.
Counting objects: 100% (227/227), done.
Delta compression using up to 8 threads
Compressing objects: 100% (199/199), done.
Writing objects: 100% (227/227), done.
Total 227 (delta 69), reused 0 (delta 0)

$ git count-objects
253 objects, 33980 kilobytes

$ git gc
Enumerating objects: 227, done.
Counting objects: 100% (227/227), done.
Delta compression using up to 8 threads
Compressing objects: 100% (130/130), done.
Writing objects: 100% (227/227), done.
Total 227 (delta 69), reused 227 (delta 69)

$ git count-objects
9 objects, 40 kilobytes
```

7.6 Config

The command git-config allows to configure and personalize Git.

The file `.gitconfig` contains the global Git configuration of the machine. The file `.git/config` contains the local Git configuration of the repository.

A useful config:

Listing 7.13: Example of **git config** with suggested aliases

```
$ git config --global alias.st "status"
$ git config --global alias.di "diff"
```

(continues on next page)

(continued from previous page)

```
$ git config --global alias.dc "diff --cached"
$ git config --global alias.co "count-objects"
$ git config --global alias.lg "log --graph --all"
$ git config --global alias.h "rev-parse HEAD"
$ git config --global alias.hs "rev-parse --short HEAD"
$ git config --global alias.wt "worktree"
$ git config --global color.ui true # Colorize the command line results
$ git config --global fetch.prune true # Prune local branches that were removed on remote
```

Add yours!

7.7 LFS

Git Large File Storage (LFS) is an extension for versioning large files.

Git LFS replaces large files with text pointers inside Git, while storing the file contents on a remote server like GitLab. Large files are still versioned, but cloning a repository is much faster because the files are not saved directly within Git.

Listing 7.14: Init of Git LFS

```
$ git lfs install
Updated git hooks.
Git LFS initialized.
$ git lfs track "*.xcf"
Tracking "*.xcf"
$ git add .gitattributes
```

A new file `.gitattributes` is created in the root folder of the repository. It contains the files tracked by LFS and must be versioned, e.g.:

Listing 7.15: Example of LFS tracking line in `.gitattributes` for XCF files

```
*.xcf filter=lfs diff=lfs merge=lfs -text
```

7.8 Cherry pick

Git allows the creation of many branches and an easy merge from a one branch into another. But what if one wants to apply only part of a branch?

The command git-cherry-pick is there for this alternative. It applies the changes introduced by some existing commits by copying them onto another branch.

Example:

Listing 7.16: Full example of **`git cherry-pick`** — without conflict

```
$ git log --graph --all
* commit 7cea952501597938bef62a4414230b55781c48c2 (dev)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 20 17:32:48 2022 +0200
```

(continues on next page)

(continued from previous page)

```
|
|     b
|
* commit 47fa50706becebde4a79c3c6e34042daf86db053
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 20 17:32:33 2022 +0200
|
|     a
|
* commit 3b7422eae46e3abbc69261ee8ee7fd9924b7ca2e (HEAD -> 1-release-first-version,␣
↪origin/1-release-first-version)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 13 16:10:33 2022 +0200
|
|     Include some files

$ git cherry-pick dev~1
[1-release-first-version 248b25f] a
 Date: Thu Oct 20 17:32:33 2022 +0200
 1 file changed, 1 insertion(+)

$ git log --graph --all
* commit 248b25f4624f8e36bd57020fc313437e2e091ccb (HEAD -> 1-release-first-version)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 20 17:32:33 2022 +0200
|
|     a
|
| * commit 7cea952501597938bef62a4414230b55781c48c2 (dev)
| | Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| | Date:   Thu Oct 20 17:32:48 2022 +0200
| |
| |     b
| |
| * commit 47fa50706becebde4a79c3c6e34042daf86db053
|/  Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
|   Date:   Thu Oct 20 17:32:33 2022 +0200
|
|       a
|
* commit 3b7422eae46e3abbc69261ee8ee7fd9924b7ca2e (origin/1-release-first-version)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 13 16:10:33 2022 +0200
|
|     Include some files
```

Listing 7.17: Full example of **`git cherry-pick`** — with conflict

```
$ git log --graph --all
* commit 7cea952501597938bef62a4414230b55781c48c2 (dev)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 20 17:32:48 2022 +0200
```

(continues on next page)

(continued from previous page)

```
|
|     b
|
* commit 47fa50706becebde4a79c3c6e34042daf86db053
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 20 17:32:33 2022 +0200
|
|     a
|
* commit 3b7422eae46e3abbc69261ee8ee7fd9924b7ca2e (HEAD -> 1-release-first-version,␣
↪origin/1-release-first-version)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 13 16:10:33 2022 +0200
|
|     Include some files

$ git cherry-pick dev
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
error: could not apply 7cea952... b
hint: after resolving the conflicts, mark the corrected paths
hint: with 'git add <paths>' or 'git rm <paths>'
hint: and commit the result with 'git commit'

$ git status
On branch 1-release-first-version
Your branch is up to date with 'origin/1-release-first-version'.

You are currently cherry-picking commit 7cea952.
    (fix conflicts and run "git cherry-pick --continue")
    (use "git cherry-pick --skip" to skip this patch)
    (use "git cherry-pick --abort" to cancel the cherry-pick operation)

Unmerged paths:
    (use "git add <file>..." to mark resolution)
    both modified:   README.md

$ git mergetool
Merging:
README.md

Normal merge conflict for 'README.md':
{local}: modified file
{remote}: modified file

$ git status
On branch 1-release-first-version
Your branch is up to date with 'origin/1-release-first-version'.

You are currently cherry-picking commit 7cea952.
    (all conflicts fixed: run "git cherry-pick --continue")
    (use "git cherry-pick --skip" to skip this patch)
```

(continues on next page)

(continued from previous page)

```
    (use "git cherry-pick --abort" to cancel the cherry-pick operation)

Changes to be committed:
    modified:   README.md

$ git cherry-pick --continue
[1-release-first-version bdb3f6e] b
Date: Thu Oct 20 17:32:48 2022 +0200
1 file changed, 1 insertion(+)

$ git log --graph --all
* commit bdb3f6e3df88f8d630e8fcbc728f5024c0afa61b (HEAD -> 1-release-first-version)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 20 17:32:48 2022 +0200
|
|     b
|
| * commit 7cea952501597938bef62a4414230b55781c48c2 (dev)
| | Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| | Date:   Thu Oct 20 17:32:48 2022 +0200
| |
| |     b
| |
| * commit 47fa50706becebde4a79c3c6e34042daf86db053
|/  Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
|   Date:   Thu Oct 20 17:32:33 2022 +0200
|
|       a
|
* commit 3b7422eae46e3abbc69261ee8ee7fd9924b7ca2e (origin/1-release-first-version)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 13 16:10:33 2022 +0200
|
|     Include some files

$ git show
commit bdb3f6e3df88f8d630e8fcbc728f5024c0afa61b (HEAD -> 1-release-first-version)
Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
Date:   Thu Oct 20 17:32:48 2022 +0200

    b

diff --git a/README.md b/README.md
index 461e207..2802b55 100644
--- a/README.md
+++ b/README.md
@@ -60,3 +60,4 @@ cp ${VIRGIL_ROOT}source/manuals/development/gitworkflow.rst $COPIED_GWF
 tail -n +7 $COPIED_GWF | sponge $COPIED_GWF
 cp -r ${VIRGIL_ROOT}source/manuals/development/_imgworkflow/ $COPY_DIR
 ```
+bbbbbbbbbbbbbbbbb
(END)
```
```

7.9 Squash merge

Sometimes it is not worth merging/pushing all commits because they are only backup work increments and only the end result must be kept.

The command git-merge --squash can be used to squash the commits of the merged branch. This command, however, doesn't make a commit. A real commit must be made after the merge.

Example:

Listing 7.18: Full example of **git merge --squash**

```
$ git log --graph --all
* commit 7cea952501597938bef62a4414230b55781c48c2 (dev)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 20 17:32:48 2022 +0200
|
|     b
|
* commit 47fa50706becebde4a79c3c6e34042daf86db053
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 20 17:32:33 2022 +0200
|
|     a
|
* commit 3b7422eae46e3abbc69261ee8ee7fd9924b7ca2e (HEAD -> 1-release-first-version,␣
↪origin/1-release-first-version)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 13 16:10:33 2022 +0200
|
|     Include some files
|

$ git merge --squash dev
Updating 3b7422e..7cea952
Fast-forward
Squash commit -- not updating HEAD
README.md | 2 ++
1 file changed, 2 insertions(+)

$ git status
On branch 1-release-first-version
Your branch is up to date with 'origin/1-release-first-version'.

Changes to be committed:
(use "git restore --staged <file>..." to unstage)
    modified:   README.md

$ git diff --cached
diff --git a/README.md b/README.md
index 461e207..86e90cc 100644
--- a/README.md
+++ b/README.md
@@ -60,3 +60,5 @@
```

(continues on next page)

(continued from previous page)

```
[...]
+aaaaaaaaaaaaaaaa
+bbbbbbbbbbbbbbbb
(END)

$ git commit -m "..."
[1-release-first-version 0919702] ...
1 file changed, 2 insertions(+)

$ git log --graph -all
* commit 0919702139215c2439913bd7158092ab60d1ecc8 (HEAD -> 1-release-first-version)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 20 17:37:22 2022 +0200
|
|     ...
|
| * commit 7cea952501597938bef62a4414230b55781c48c2 (dev)
| | Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| | Date:   Thu Oct 20 17:32:48 2022 +0200
| |
| |     b
| |
| * commit 47fa50706becebde4a79c3c6e34042daf86db053
|/  Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
|   Date:   Thu Oct 20 17:32:33 2022 +0200
|
|       a
|
* commit 3b7422eae46e3abbc69261ee8ee7fd9924b7ca2e (origin/1-release-first-version)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 13 16:10:33 2022 +0200
|
|     Include some files
|
```

7.10 Rewrite history with interactive rebase

The squash merge allows to squash a branch while merging. But it may sometimes be useful to rewrite the history of your own branch.

The command git-rebase --interactive initiates the interactive rebase of the branch. A revision object must be provided that is used as oldest revision point for the rebase, i.e.:

- a commit hash: `47fa50706becebde4a79c3c6e34042daf86db053`
- a tag: `v0.9.3`
- a branch: `main`
- the commit from where the current branch was forked: `git merge-base --fork-point <BASE BRANCH>`

Listing 7.19: Full example of **git rebase --interactive** — start of rebase operation

```
$ git log --graph --all
* commit 7cea952501597938bef62a4414230b55781c48c2 (HEAD -> dev)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 20 17:32:48 2022 +0200
|
|     b
|
* commit 47fa50706becebde4a79c3c6e34042daf86db053
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 20 17:32:33 2022 +0200
|
|     a
|
* commit 3b7422eae46e3abbc69261ee8ee7fd9924b7ca2e (origin/1-release-first-version, 1-
↪release-first-version)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 13 16:10:33 2022 +0200
|
|     Include some files

$ git rebase -i `git merge-base --fork-point 1-release-first-version`
```

Listing 7.20: Full example of **git rebase --interactive** — initial proposition of operations on commits

```
pick 47fa507 a
pick 7cea952 b

# Rebase 3b7422e..7cea952 onto 3b7422e (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
```

(continues on next page)

(continued from previous page)

```
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

Listing 7.21: Full example of **git rebase --interactive** — user-selected operations on commits: reword first commit, squash second commit

```
r 47fa507 a
s 7cea952 b

# Rebase 3b7422e..7cea952 onto 3b7422e (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

Listing 7.22: Full example of **git rebase --interactive** — reword first commit (state shown is after rewording)

```
a+b

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Thu Oct 20 17:32:33 2022 +0200
#
# interactive rebase in progress; onto 3b7422e
# Last command done (1 command done):
#    r 47fa507 a
# Next command to do (1 remaining command):
```

(continues on next page)

(continued from previous page)

```
#    s 7cea952 b
# You are currently editing a commit while rebasing branch 'dev' on '3b7422e    '.
#
# Changes to be committed:
#   modified:   README.md
```

Listing 7.23: Full example of **git rebase --interactive** — final commit message for first+second commit as second commit is squashed, the second commit original message is actually ignored (could have used fixup instead)

```
# This is a combination of 2 commits.
# This is the 1st commit message:

a+b

# This is the commit message #2:

#b

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Thu Oct 20 17:32:33 2022 +0200
#
# interactive rebase in progress; onto 3b7422e
# Last commands done (2 commands done):
#    r 47fa507 a
#    s 7cea952 b
# No commands remaining.
# You are currently rebasing branch 'dev' on '3b7422e'.
#
# Changes to be committed:
#   modified:   README.md
```

Listing 7.24: Full example of **git rebase --interactive** — end of rebase operation

```
$ git rebase -i `git merge-base --fork-point 1-release-first-version`
[detached HEAD db9f85b] a+b
 Date: Thu Oct 20 17:32:33 2022 +0200
 1 file changed, 1 insertion(+)
[detached HEAD 1064639] a+b
 Date: Thu Oct 20 17:32:33 2022 +0200
 1 file changed, 2 insertions(+)
Successfully rebased and updated refs/heads/dev.

$ git log --graph --all
* commit 106463921ebc7e724b36eff3e8e6fd8820586f06 (HEAD -> dev)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 20 17:32:33 2022 +0200
|
```

(continues on next page)

(continued from previous page)

```
|     a+b
|
* commit 3b7422eae46e3abbc69261ee8ee7fd9924b7ca2e (origin/1-release-first-version, 1-
↪release-first-version)
| Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
| Date:   Thu Oct 13 16:10:33 2022 +0200
|
|     Include some files

$ git show
commit 106463921ebc7e724b36eff3e8e6fd8820586f06 (HEAD -> dev)
Author: Yorick Brunet <yorick.brunet@heig-vd.ch>
Date:   Thu Oct 20 17:32:33 2022 +0200

    a+b

diff --git a/README.md b/README.md
index 461e207..86e90cc 100644
--- a/README.md
+++ b/README.md
@@ -60,3 +60,5 @@ cp ${VIRGIL_ROOT}source/manuals/development/gitworkflow.rst $COPIED_GWF
 tail -n +7 $COPIED_GWF | sponge $COPIED_GWF
 cp -r ${VIRGIL_ROOT}source/manuals/development/_imgworkflow/ $COPY_DIR
```
+aaaaaaaaaaaaaaaaa
+bbbbbbbbbbbbbbbbb
(END)
```

7.11 Remove files from history

Using git-rm removes files from the current working area and, when committed, from the repository area. But not from history.

git-filter-repo allows to fully remove a file from history by rewriting it! This is especially useful when a file-that-should-not-be-committed has been committed. See doc for more usage.

Note: **`git filter-repo`** works only on a freshly cloned repository.

Listing 7.25: Split of a repository with **`git filter-repo`**

```
$ git clone https://reds-gitlab.heig-vd.ch/vna4ct/vna4ct_full.git vna4ct
$ cp -r vna4ct vna4ct_pcb
$ cd vna4ct_pcb
$ redstools dir-tree --root-dir . --max-depth 3  --do-not-show .git --do-not-enter␣
↪vivado_test_git publi doc admin
./
|── admin/
|── dev/
|   |── hard/
```

(continues on next page)
```

(continued from previous page)

```
|   |   |── pcb/
|   |   |── pcb_stackup/
|   |   └── vna_bf/
|   |── hard_fpga/
|   |   └── ...
|   |── matlab/
|   |   └── ...
|   └── soft/
|       └── ...
|── doc/
|── publi/
|── tiersdoc/
|   |── 74lvc1g04/
|   ...
|   └── vc709/
|       └── vc709_DDR3_MIG_generation/
└── vivado_test_git/

$ git filter-repo --path dev/hard --path tiersdoc --path-rename dev/hard/:dev/
Parsed 432 commits
New history written in 2.74 seconds; now repacking/cleaning...
Repacking your repo and cleaning out old unneeded objects
HEAD is now at bc545a9 Merge branch 'main' of https://reds-gitlab.heig-vd.ch/reds-heigvd/
↪vna4ct
Enumerating objects: 2767, done.
Counting objects: 100% (2767/2767), done.
Delta compression using up to 16 threads
Compressing objects: 100% (789/789), done.
Writing objects: 100% (2767/2767), done.
Total 2767 (delta 1894), reused 2754 (delta 1891), pack-reused 0
Completely finished after 4.55 seconds.

$ redstools dir-tree --root-dir . --max-depth 2  --do-not-show .git --do-not-enter␣
↪vivado_test_git publi doc admin
./
|── dev/
|   |── pcb/
|   |── pcb_stackup/
|   └── vna_bf/
└── tiersdoc/
   |── 74lvc1g04/
   ...
   └── vc709/
      └── vc709_DDR3>_MIG_generation

$ git remote add origin https://reds-gitlab.heig-vd.ch/vna4ct/vna4ct-pcb.git
$ git push -u origin main
Enumerating objects: 2767, done.
Counting objects: 100% (2767/2767), done.
Delta compression using up to 16 threads
Compressing objects: 100% (786/786), done.
Writing objects: 100% (2767/2767), 334.17 MiB | 1.93 MiB/s, done.
```

(continues on next page)

(continued from previous page)

```
Total 2767 (delta 1894), reused 2767 (delta 1894), pack-reused 0
remote: Resolving deltas: 100% (1894/1894), done.
To https://reds-gitlab.heig-vd.ch/vna4ct/vna4ct-pcb.git
* [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.

$ cd ../vna4ct
$ git filter-repo --invert-paths --path dev/hard --path tiersdoc
Parsed 432 commits
ew history written in 0.20 seconds; now repacking/cleaning...
Repacking your repo and cleaning out old unneeded objects
HEAD is now at f4601e8 Merge branch 'lfr_dev' into 'main'
Enumerating objects: 2890, done.
Counting objects: 100% (2890/2890), done.
Delta compression using up to 16 threads
Compressing objects: 100% (1593/1593), done.
Writing objects: 100% (2890/2890), done.
Total 2890 (delta 1215), reused 2874 (delta 1210), pack-reused 0
Completely finished after 1.15 seconds.

$ redstools dir-tree --root-dir . --max-depth 2  --do-not-show .git --do-not-enter␣
↪vivado_test_git publi doc admin
./
|── admin/
|── dev/
|   |── hard_fpga/
|   |── matlab/
|   └── soft/
|── doc/
|── publi/
└── vivado_test_git/

$ git remote add origin https://reds-gitlab.heig-vd.ch/vna4ct/vna4ct.git
$ git push -u origin main
Enumerating objects: 2764, done.
Counting objects: 100% (2764/2764), done.
Delta compression using up to 16 threads
Compressing objects: 100% (1526/1526), done.
Writing objects: 100% (2764/2764), 127.60 MiB | 1.93 MiB/s, done.
Total 2764 (delta 1153), reused 2759 (delta 1151), pack-reused 0
remote: Resolving deltas: 100% (1153/1153), done.
To https://reds-gitlab.heig-vd.ch/vna4ct/vna4ct.git
* [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

CHAPTER

EIGHT

GIT — TRICKS

8.1 Backup and restore a machine’s configuration

The configuration of a machine is stored in a Git repository. The example given here apply to the home of a regular user.

To keep the backup in a remote repository, create a project in your own namespace.

8.1.1 Backup config

Set the URL of the remote repository on the first line of Listing 8.1.

Listing 8.1: Backup configuration — Setup repository

```
$ CONFIG_URL="..."
$ CONFIG_DIR="/home/${USER}/configs/"
$ SHELL_RC=".`basename $SHELL`rc"
$ mkdir -p $CONFIG_DIR
$ cd $CONFIG_DIR
$ git init --initial-branch=main
$ git config core.worktree "/home/${USER}/"
$ git config status.showUntrackedFiles no
$ git config status.relativePaths false
$ git remote add origin $CONFIG_URL
$ echo "alias gitc='GIT_DIR=${CONFIG_DIR}.git git'\n" >> ~/$SHELL_RC
```

Add configuration files to the repository and save the current state, either using the command of line 1 or line 2 of Listing 8.2, depending what your current working directory is.

Listing 8.2: Backup configuration — Add configuration files and commit

```
$ git add ../<file> # from CONFIG_DIR
$ gitc add <file> # from ~
$ git commit -m "Initial commit"
$ git push -u origin main
```

8.1.2 Restore config

Set the URL of the remote repository on the first line of Listing 8.3.

Listing 8.3: Restore configuration

```
$ CONFIG_URL="..."
$ CONFIG_DIR="/home/${USER}/configs/"
$ git clone --no-checkout $CONFIG_URL `basename $CONFIG_DIR`
$ cd $CONFIG_DIR
$ git config core.worktree "/home/${USER}/"
$ git config status.showUntrackedFiles no
$ git config status.relativePaths false
$ git reset --hard origin/main
```

8.2 Useful commands

8.2.1 Change the author of the last commit

Listing 8.4: Change the author of the last commit

```
$ git commit --amend --author="... ... <...@heig-vd.ch>" --no-edit
```

8.2.2 Change the author of several commits

This example makes an interactive rebase (see also *Rewrite history with interactive rebase*) to change the author of all commits of the branch.

The lines of Listing 8.5 do:

- Line 1 and line 2 are (in most cases) equivalent
- Line 1 sets the starting revision from the fork with branch “main” while line 2 sets the starting revision from the fork with the parent branch
- After executing line 1 **or** line 2, set the commits to be modified with the keyword `edit` and keep the commits with the right author as `pick`
- Execute line 3 for all commits
- Line 4 shows whether the interactive rebase is still on-going

Listing 8.5: Change the author of several commits within the same branch

```
$ git rebase -i `git merge-base --octopus main`
$ git rebase -i `git show-branch --merge-base`
$ git commit --amend --author="... ... <...@heig-vd.ch>" --no-edit && git rebase --
↪continue
$ git status
```

8.2.3 Remove multiple files after deleting them from the disk

Listing 8.6: Remove multiple files after deletion from disk

```
$ git ls-files --deleted -z | xargs -0 git rm
```

8.2.4 Restore a file from a previous commit

Use either line 1 or line 2 of Listing 8.7 to restore a file on the working area.

Listing 8.7: Restore a file on the working area

```
$ git checkout <revision point> -- /path/to/file
$ git restore -s <revision point> --worktree -- /path/to/file
```

Listing 8.8: Restore a file on the staging area

```
$ git restore -s <revision point> --staged -- /path/to/file
```