Software Engineering Processes

Course Code: XB_0089

Fernanda Madeiral & Claudia Raibulet

Lecture: Version control systems & Git & GitHub & Open Source

Bachelor in Computer Science, 2023/2024

Version control systems

Version control systems

Version control systems are software tools that help software teams keep track and manage changes to source code over time.

Version control systems

Change history recording

Independent development

Storage management

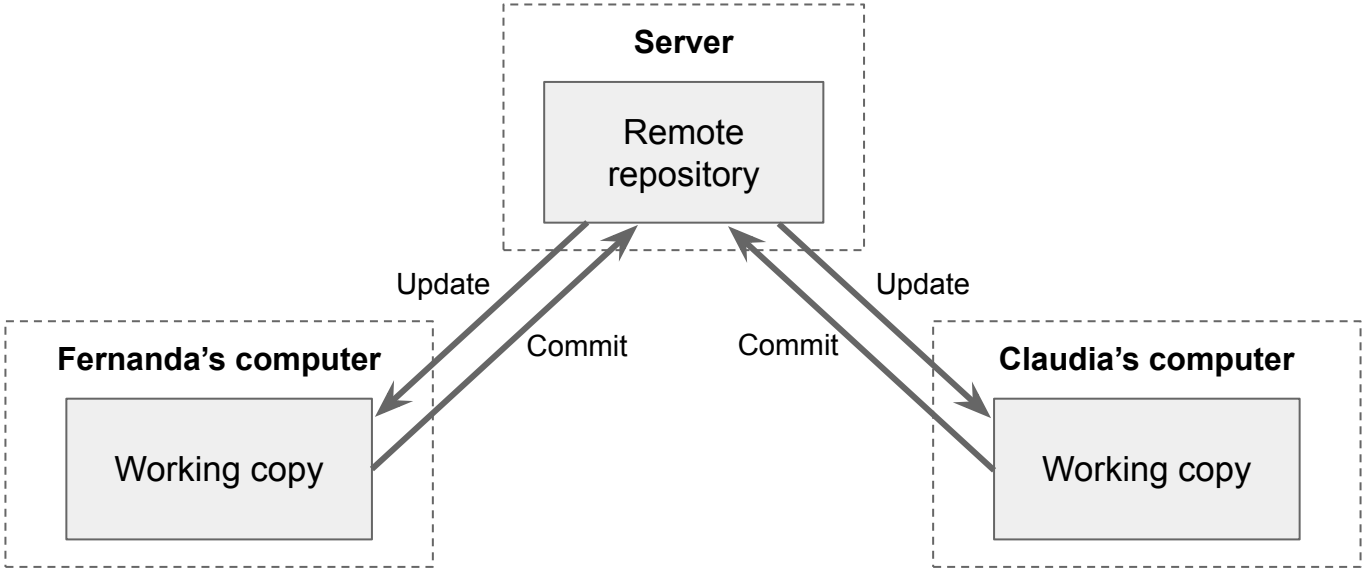
Repositories

Repository: files/folders that contain the project's files and each file's version history.

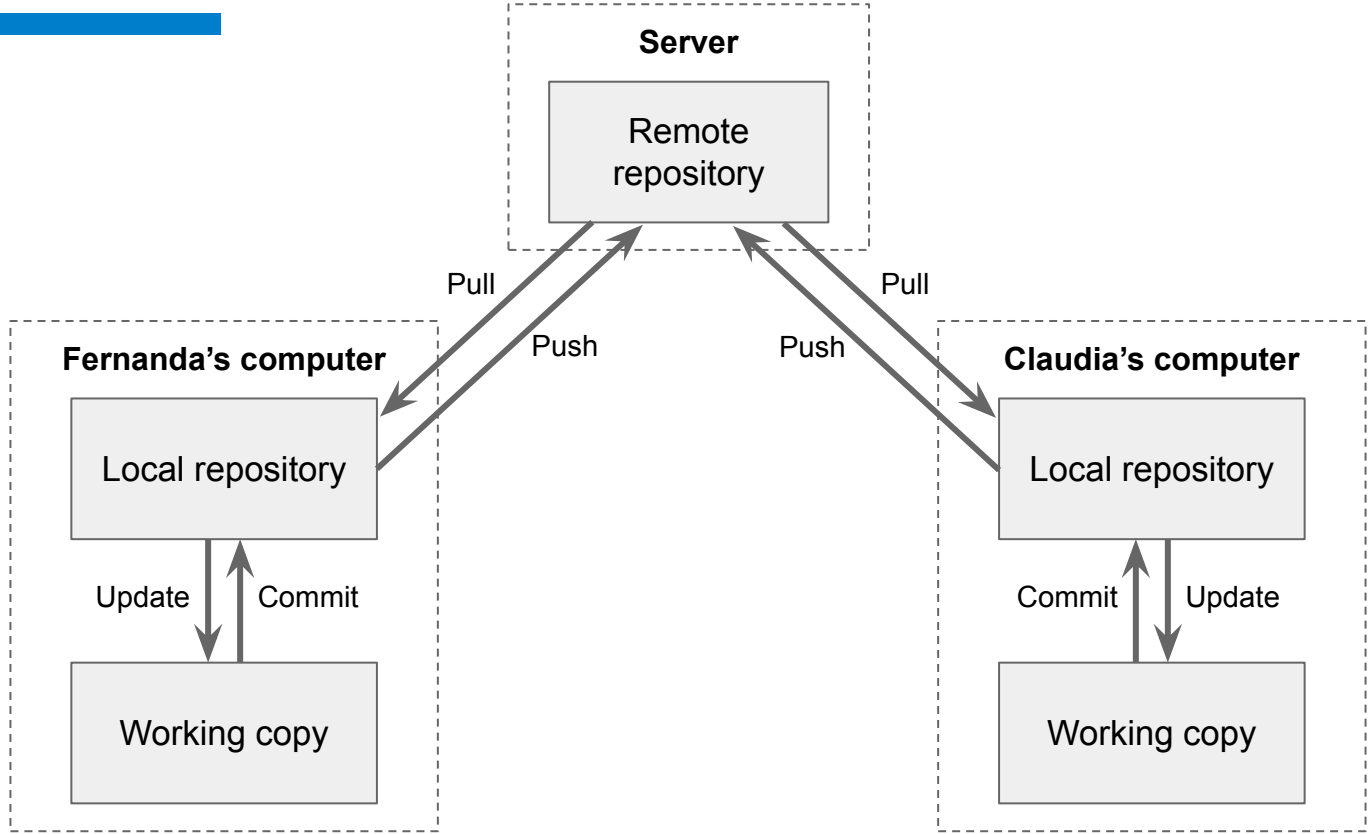
Local repository: a repository stored on a developer computer, only the developer has access to it.

Remote repository: a repository stored on some remote computer, all team members have access to it.

Centralized version control systems



Distributed version control systems



Centralized vs. distributed version control systems

Advantages of distributed version control systems:

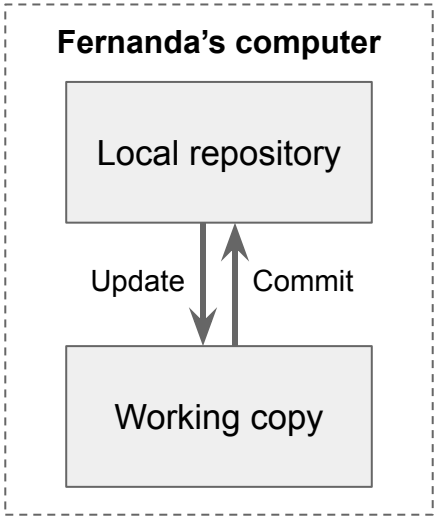
- Flexibility for local work without network connectivity
- Speed to perform commands on the repository
- Resilience to restore a damaged remote repository

Centralized vs. distributed version control systems

Disadvantages of distributed version control systems:

- Storage: if the project has a long history, it might take more disk space in the developers' computers

Local version control



Git

Git is a distributed version control system

Created by Linus Torvalds, the creator of Linux, in 2005

Git

“The best way to learn git is probably to first only do very basic things and not even look at some of the things you can do until you are familiar and confident about the basics.”

– Linus Torvalds

Git

To be studied and practiced:

Slides "SEP2024-Lecture5-git-full.pdf"

Git and GitHub

What is the difference between them?

Git and GitHub

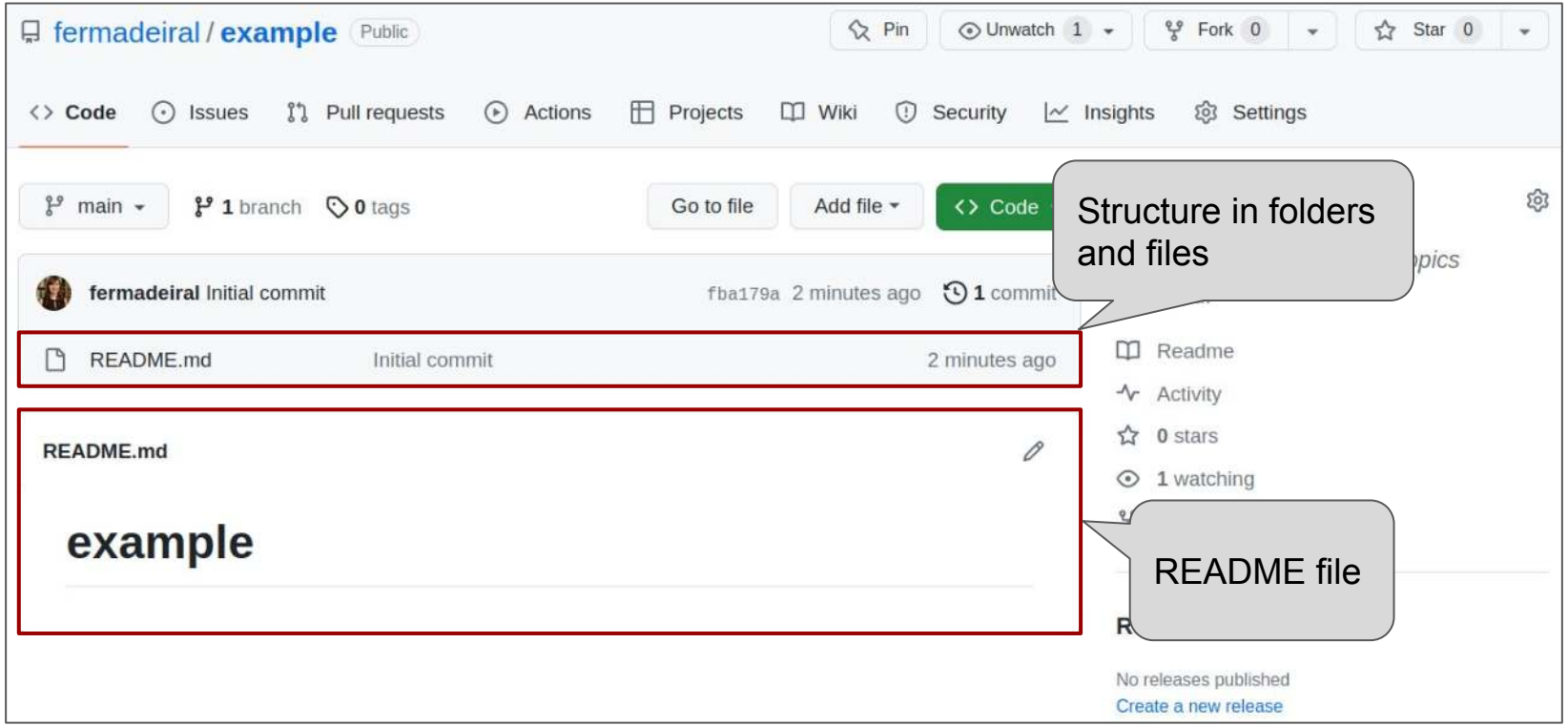
Git	**GitHub**
It’s a system	It’s a service
It’s locally installed in the computer	It’s on the web
It’s a command line tool	It has a graphical interface
It’s a version control system	It’s a code hosting platform for git repositories
It has features to control the version of files	It has features to control the version of files plus other features such as issue tracking, code review, and continuous integration

Types of repositories

Public: anyone on the internet can see it.

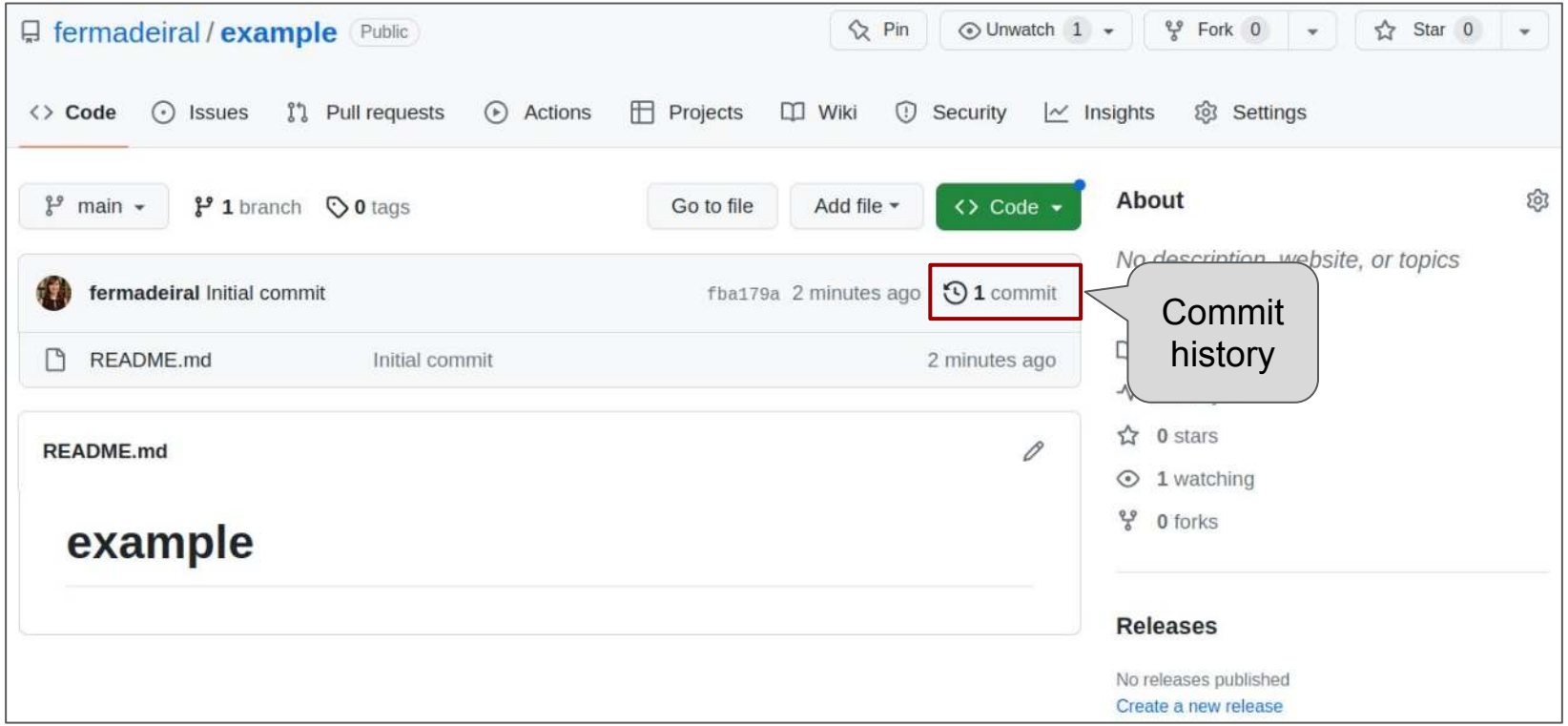
Private: you choose who can see and commit to this repository.

GitHub repository “body”



Repository URL: https://github.com/fermadeiral/example

GitHub repository commit history



Repository URL: https://github.com/fermadeiral/example

GitHub – how to perform changes

fermadeiral / example Public

Pin | Unwatch 1 | Fork 0 | Star 0

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

main | 1 branch | 0 tags | Go to file | Add file | Code

fermadeiral Initial commit fba179a 2 minutes ago 1 commit

README.md Initial commit 2 minutes ago

README.md

example

About

No description, website, or topics provided.

Readme

Activity

0 stars

1 watching

0 forks

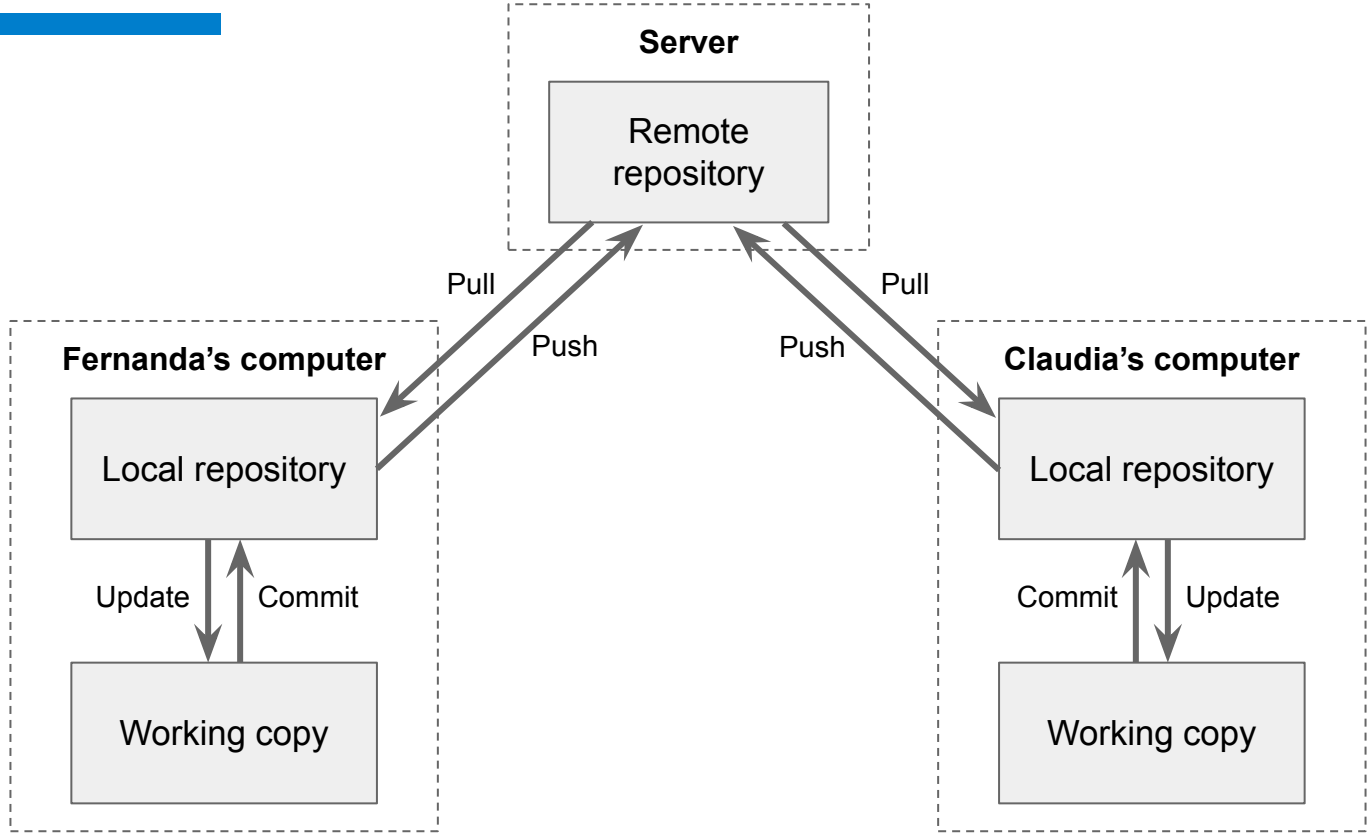
Releases

No releases published

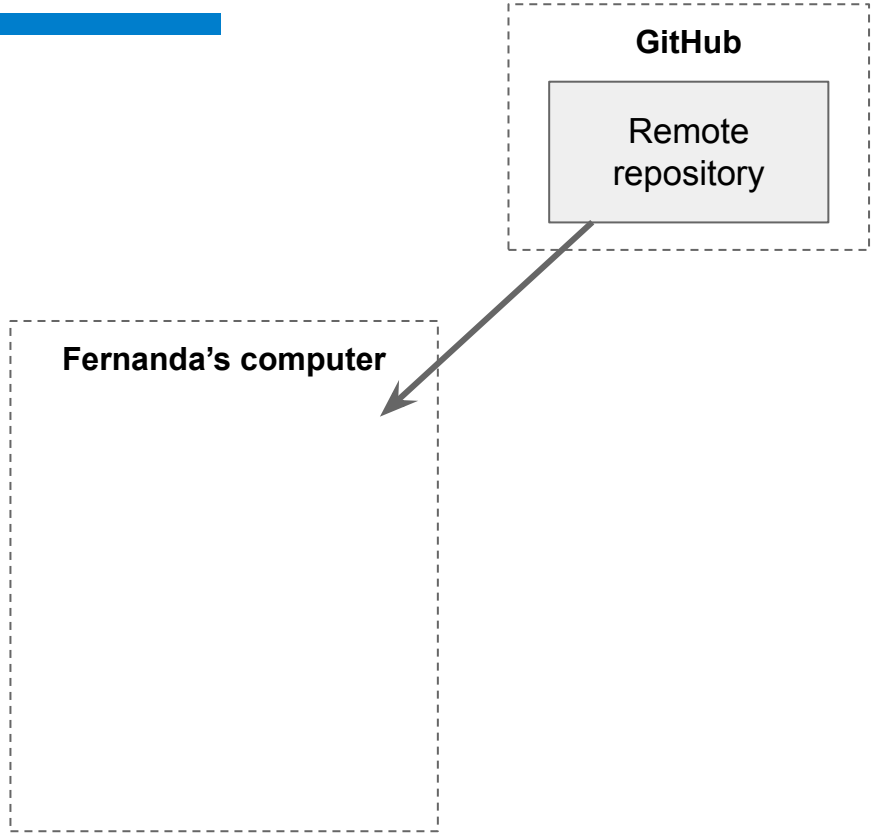
Create a new release

Repository URL: https://github.com/fermadeiral/example

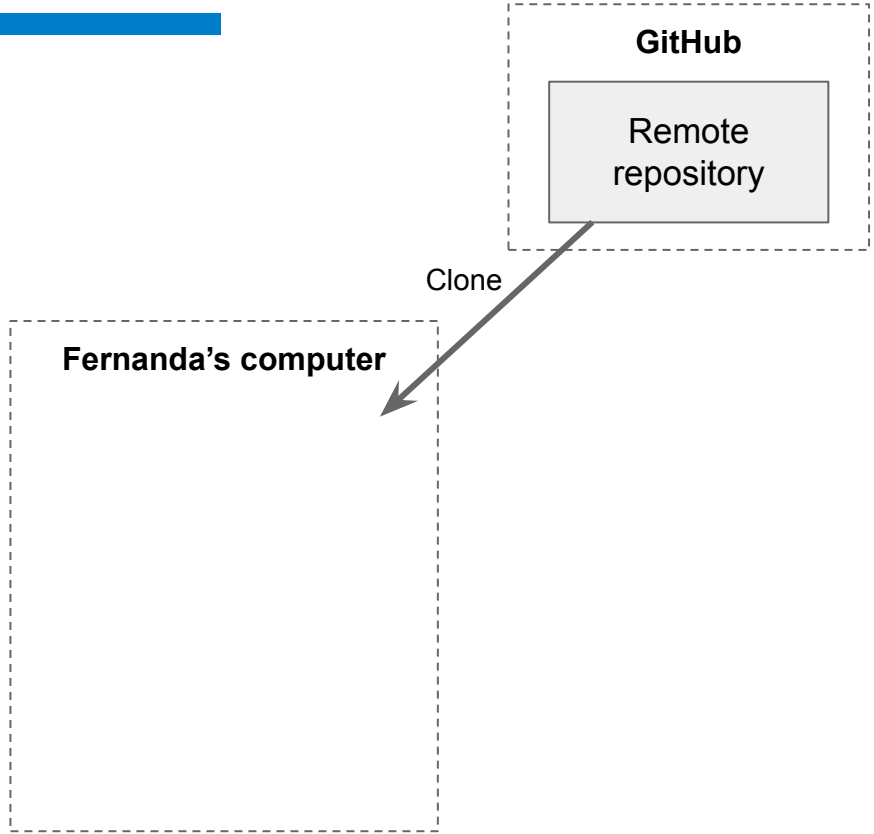
Distributed version control



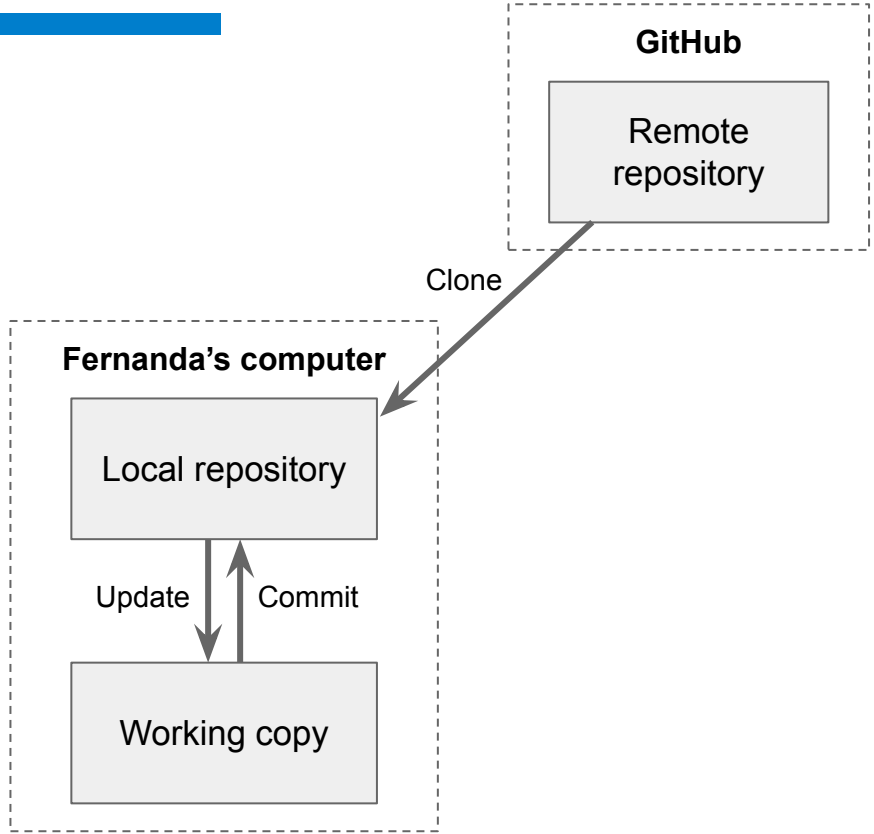
How to start collaborating in a project



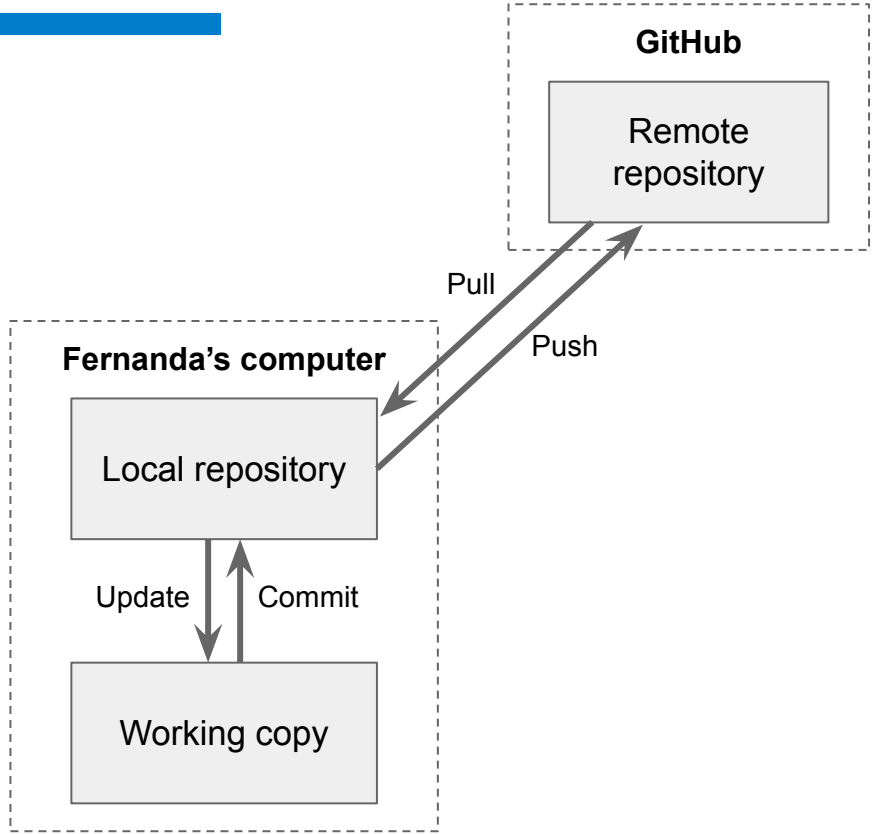
How to start collaborating in a project



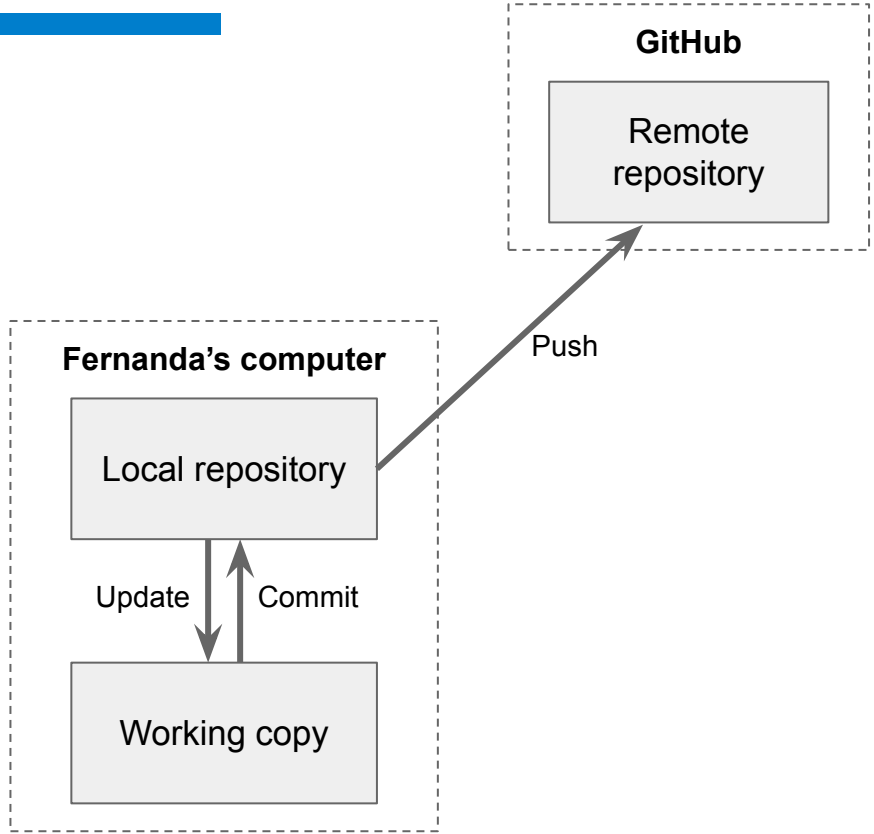
How to start collaborating in a project



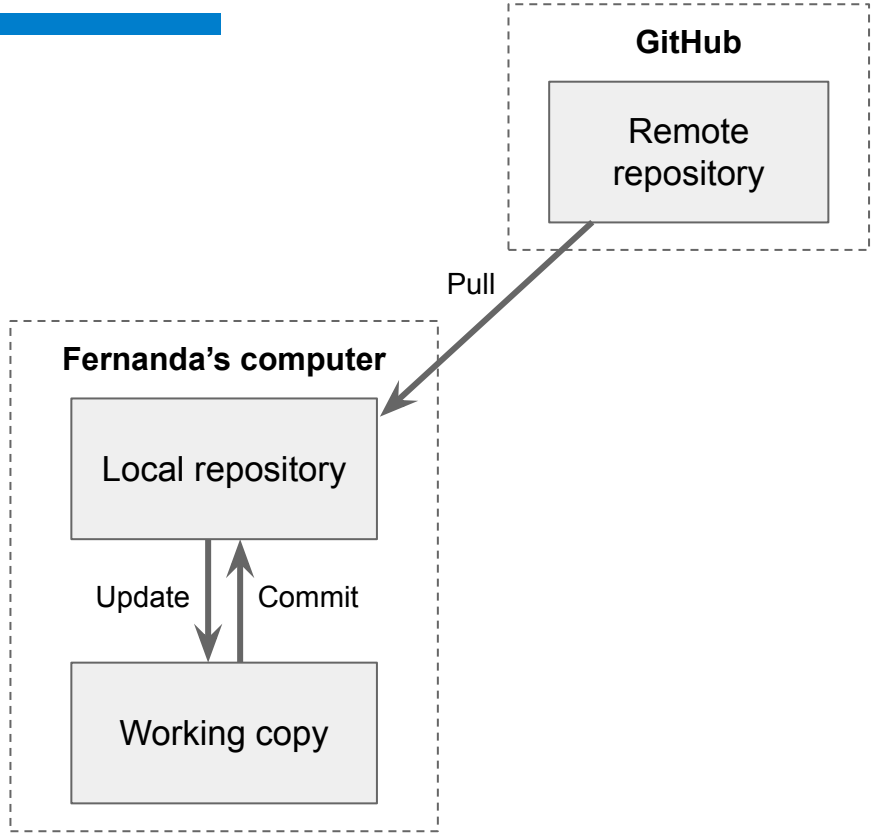
How to start collaborating in a project



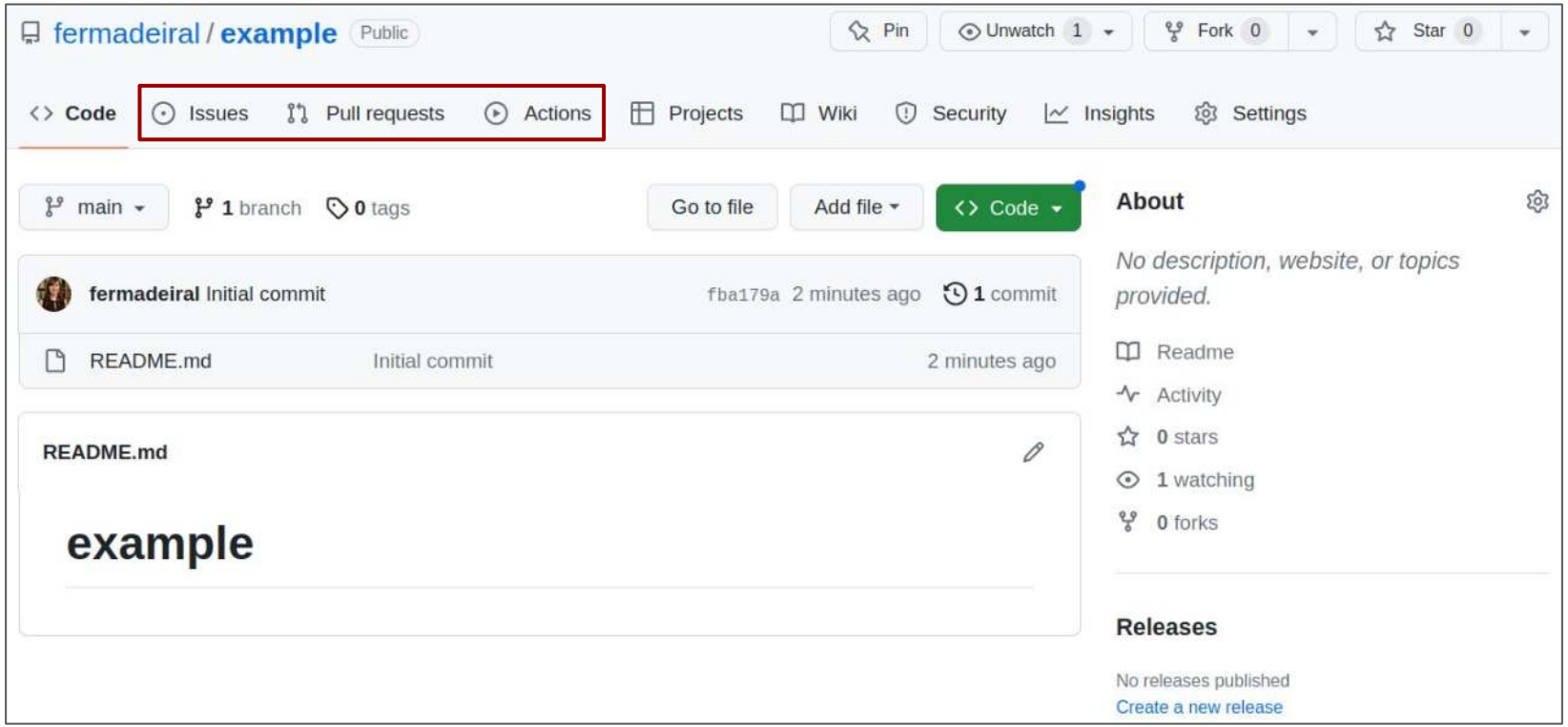
How to start collaborating in a project (assuming you have permission)



How to update your local version after changes in the remote by others



GitHub features



Issues

Issues

To track:
– Ideas
– Feedback
– Tasks
– Bugs
…

Code | Issues 379 | Pull requests 24 | Discussions | Actions | Projects | Wiki | Security | ···

Filters | is:issue is:open | Labels 24 | Milestones 5 | New issue

379 Open | 1,741 Closed | Author | Label | Projects | Milestones | Assignee | Sort

- @Condition badly implemented bug — 1
  #3270 opened on May 10 by fraf 1.6.0
- WriteAccessors prefers 'later' candidates to earlier ones? — 1
  #3258 opened on Apr 28 by facboy
- Support for Java 21 Sequenced Collections feature
  #3240 opened on Apr 16 by filiphr 1.7.0
- MapStruct Support idea快捷键shift+f6 替换错误 bug — 1
  #3233 opened on Apr 11 by yuchen-zheng
- Implement `InjectionStrategy.SETTER` enhancement good first issue up-for-grabs — 3
  #3229 opened on Apr 4 by MLNW
- Disable source presence checking for a single field using annotations — 2
  #3222 opened on Mar 31 by mjustin
- @InheritConfiguration not support non-abstract method — 6
  #3221 opened on Mar 30 by blackc716
- kapt fails with java.lang.reflect.InvocationTargetException (no error message) bug kotlin — 2

Issues – bug report

Code | Issues 379 | Pull requests 24 | Discussions | Actions | Projects | Wiki | Security | Insights

@Condition badly implemented #3270

New issue

Open **fraf** opened this issue on May 10 · 1 comment

fraf commented on May 10

Expected behavior

@condition should be a wrapper of the existing generated code (generated without this annotation). Not an overwritter of existing generated code.

Actual behavior

When using **@condition**, the generated code is rewritten which is bad (in comparison when not present). It needs to wrap it, not rewrite it.

Steps to reproduce the problem

1. Create a classical mapper class :

```
@Mapper(componentModel = "spring", uses = {TaxeMapper.class})
public interface FormuleMapper {

        Formule formuleEntityToDTO(FormuleOption formule);
```

Assignees

No one assigned

Labels

bug

Projects

None yet

Milestone

1.6.0

Development

No branches or pull requests

Issues

Organization:
– Labels
– Milestones
– Assignee
– Traceability

Code | Issues 379 | Pull requests 24 | Discussions | Actions | Projects | Wiki | Security | ...

Filters | is:issue is:open | Labels 24 | Milestones 5 | New issue

379 Open | 1,741 Closed | Author | Label | Projects | Milestones | Assignee | Sort

- @Condition badly implemented bug — 1
  #3270 opened on May 10 by fraf 1.6.0
- WriteAccessors prefers 'later' candidates to earlier ones? — 1
  #3258 opened on Apr 28 by facboy
- Support for Java 21 Sequenced Collections feature
  #3240 opened on Apr 16 by filiphr 1.7.0
- MapStruct Support idea快捷键shift+f6 替换错误 bug — 1
  #3233 opened on Apr 11 by yuchen-zheng
- Implement `InjectionStrategy.SETTER` enhancement good first issue up-for-grabs — 3
  #3229 opened on Apr 4 by MLNW
- Disable source presence checking for a single field using annotations — 2
  #3222 opened on Mar 31 by mjustin
- @InheritConfiguration not support non-abstract method — 6
  #3221 opened on Mar 30 by blackc716
- kapt fails with java.lang.reflect.InvocationTargetException (no error message) bug kotlin — 2

Pull requests

Pull requests

Main goals:

- To allow **code review** and
- To perform **automated checks** on changes

…before changes are merged in the main branch

Pull requests

Create a new local branch

Perform changes in your working directory and commit them to the local repository, as many times as you want (i.e., multiple commits)

Push the branch to the remote repository

The branch will appear on GitHub

Pull requests

branch_for_pr had recent pushes less than a minute ago

Compare & pull request

main

Go to file

Add file

Code

fermadeiral Update README.md ···

3 hours ago

2

README.md Update README.md 3 hours ago

README.md

example

Adding changes...

About

No description, website, or topics provided.

Readme

Activity

0 stars

2 watching

0 forks

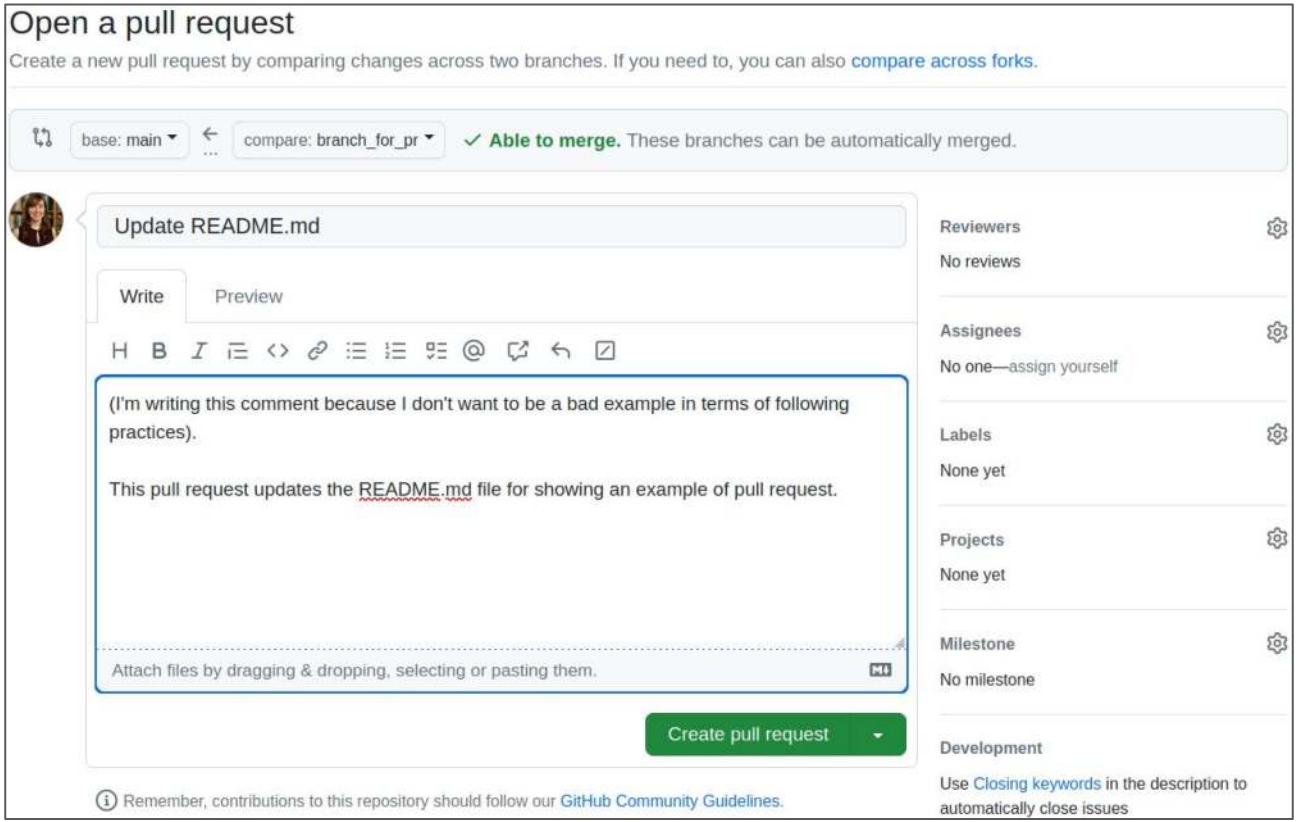
Releases

No releases published

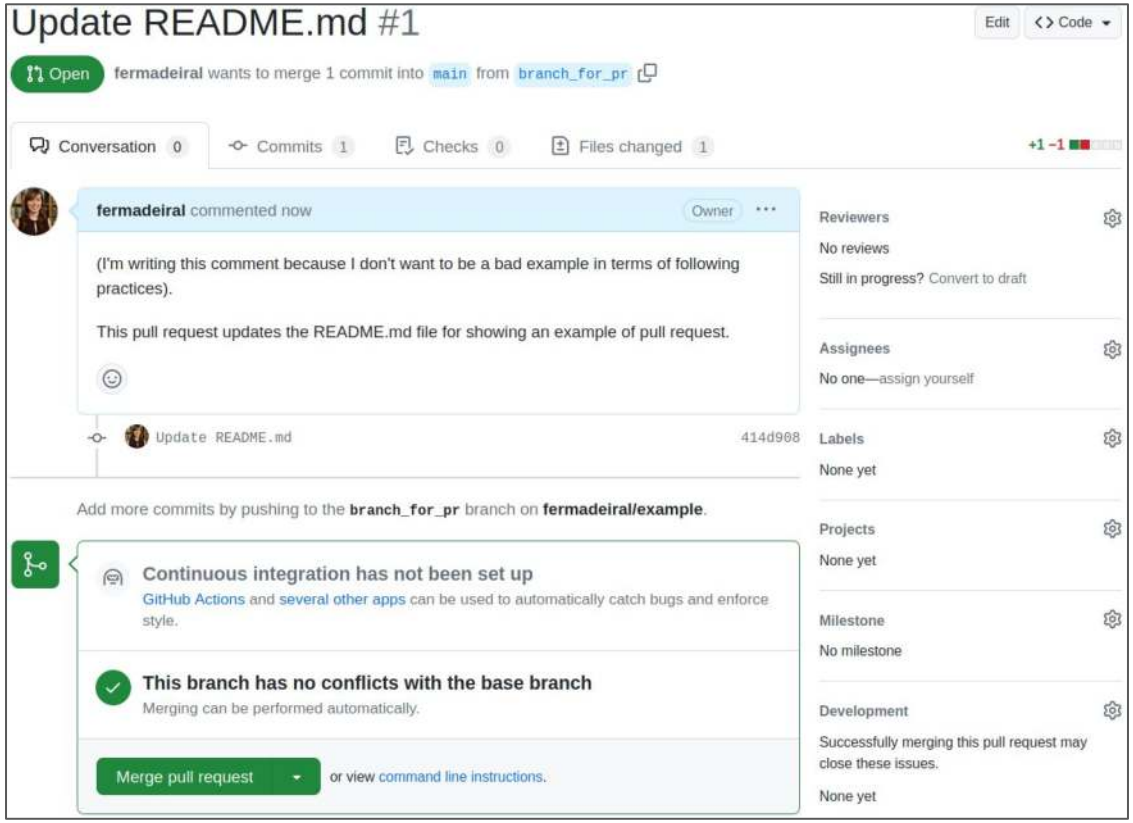
Create a new release

Packages

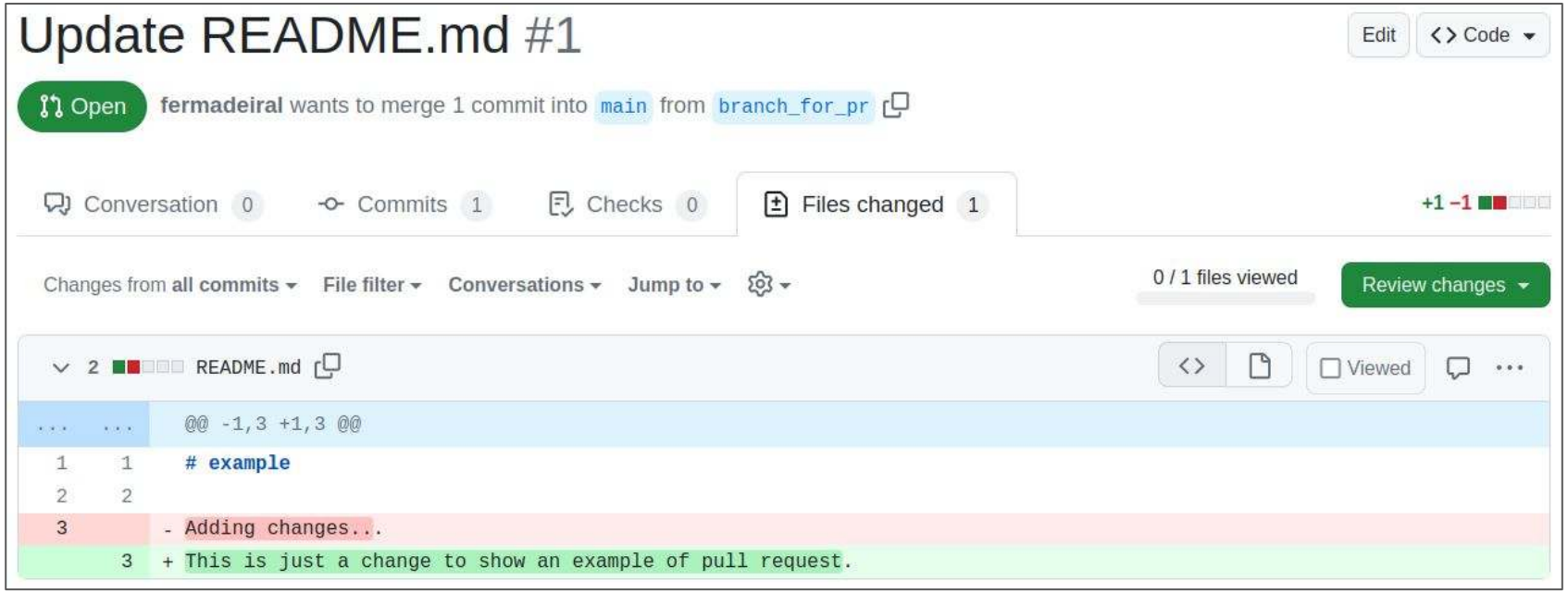
Pull requests



Pull requests

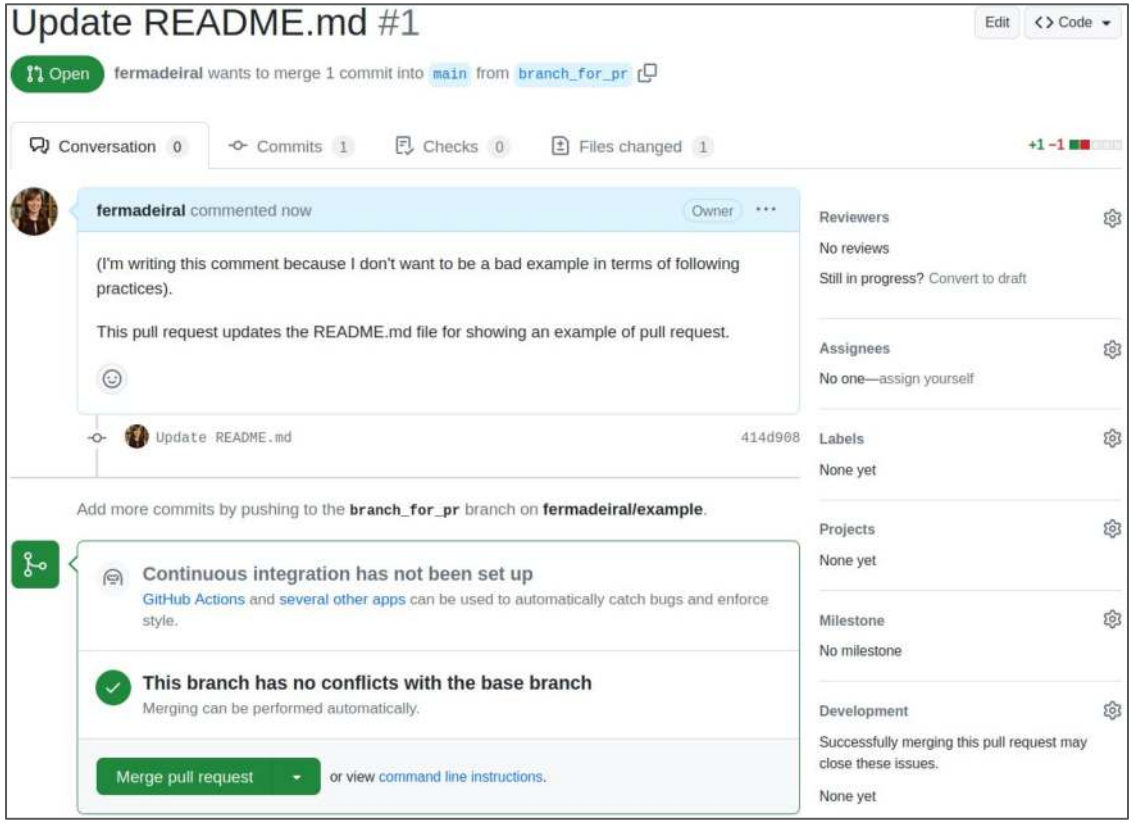


Pull requests



Pull requests

If the button “Merge pull request” is pressed, what will happen?



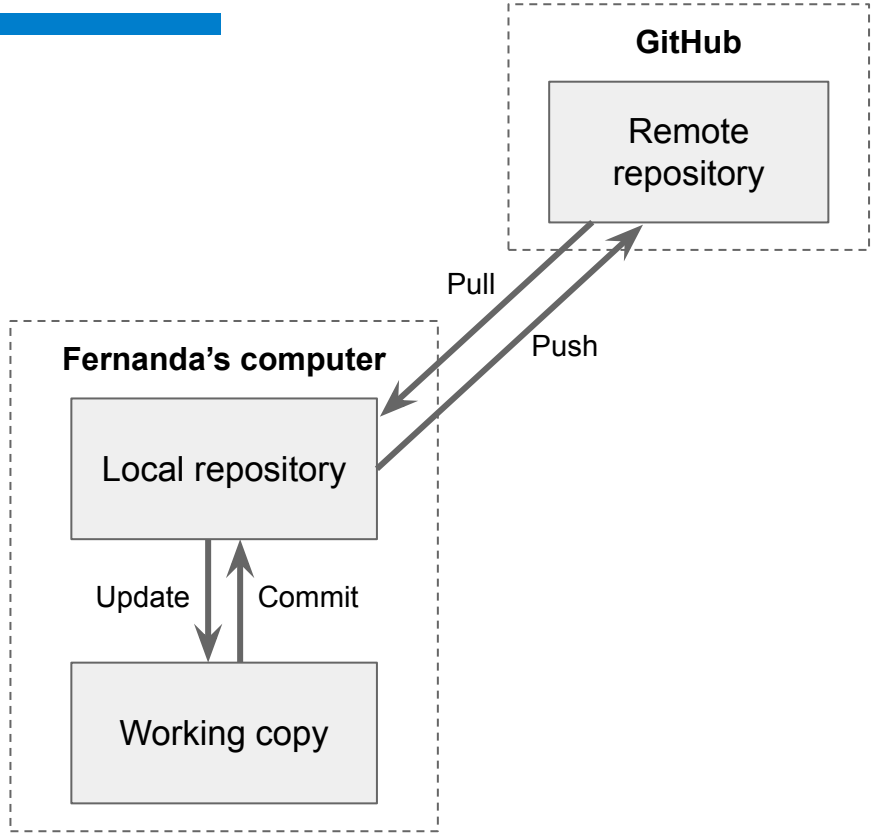
Pull-based development

Pull-based development

Main goal:

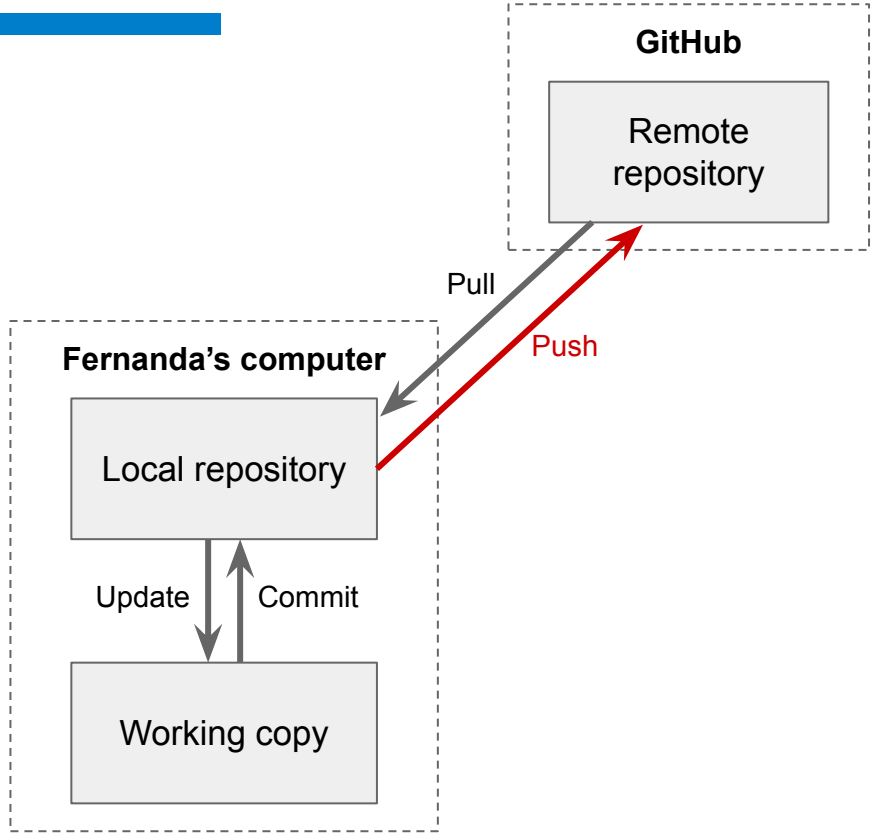
- To allow **external contributions in the repository**

Let’s go back…



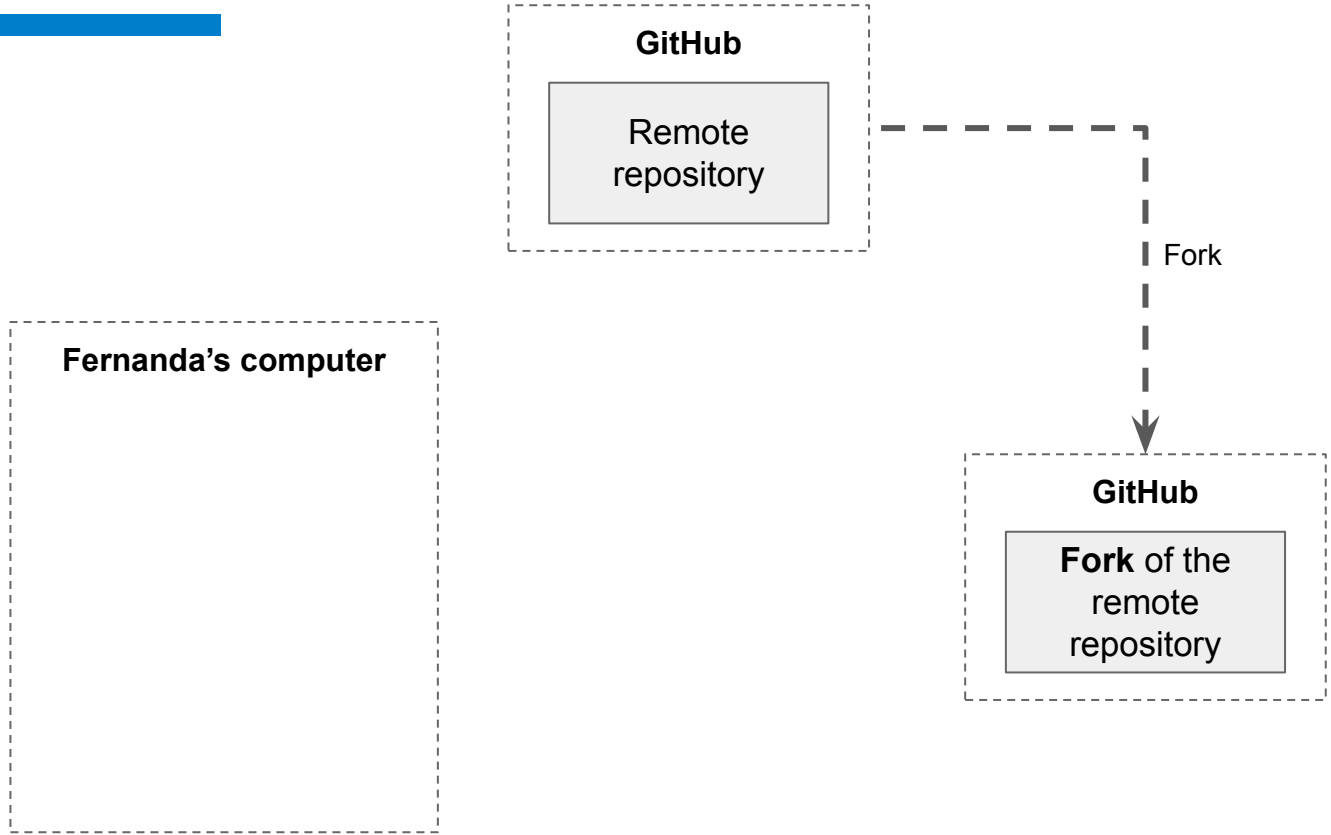
If you don’t have permissions on the remote repository, what can’t you do here?

Let’s go back…

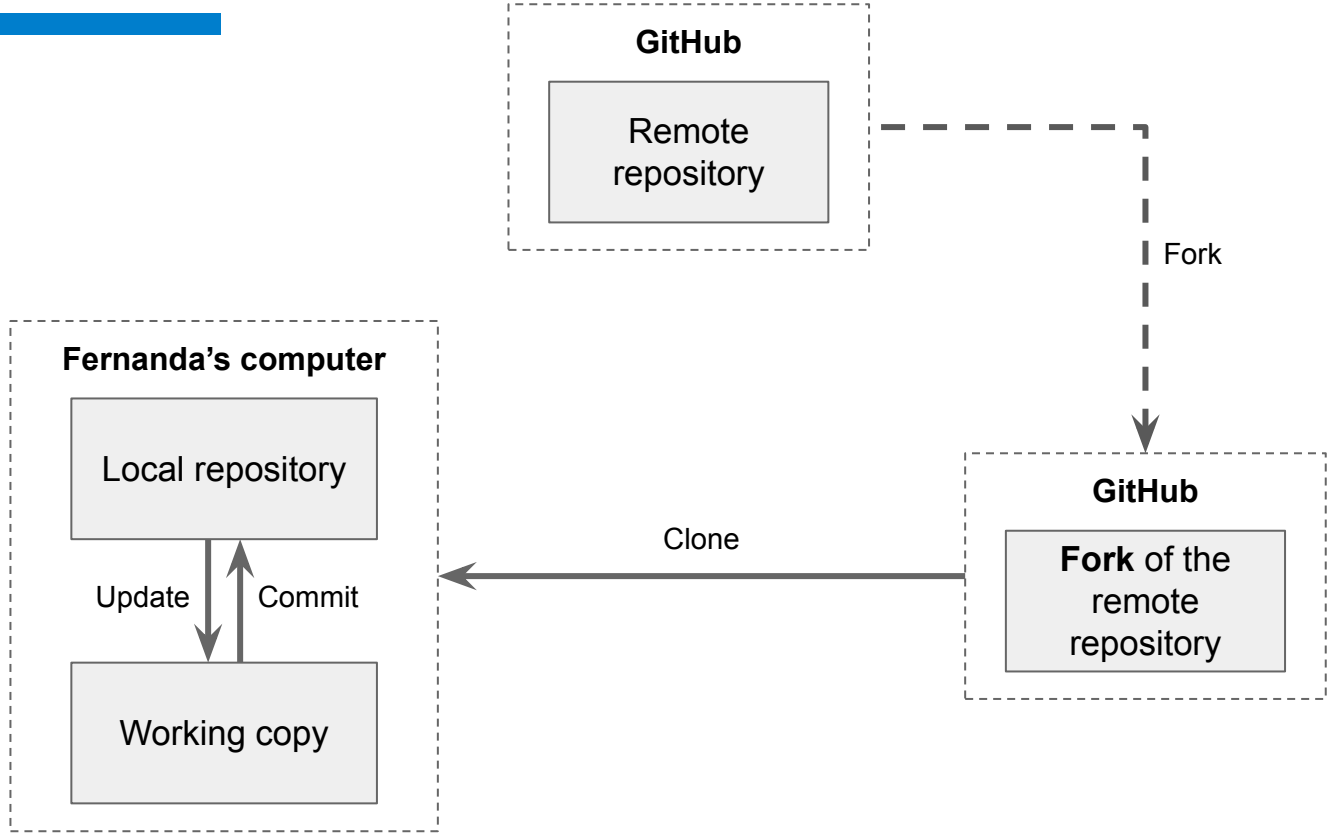


If you don’t have permissions on the remote repository, what can’t you do here?

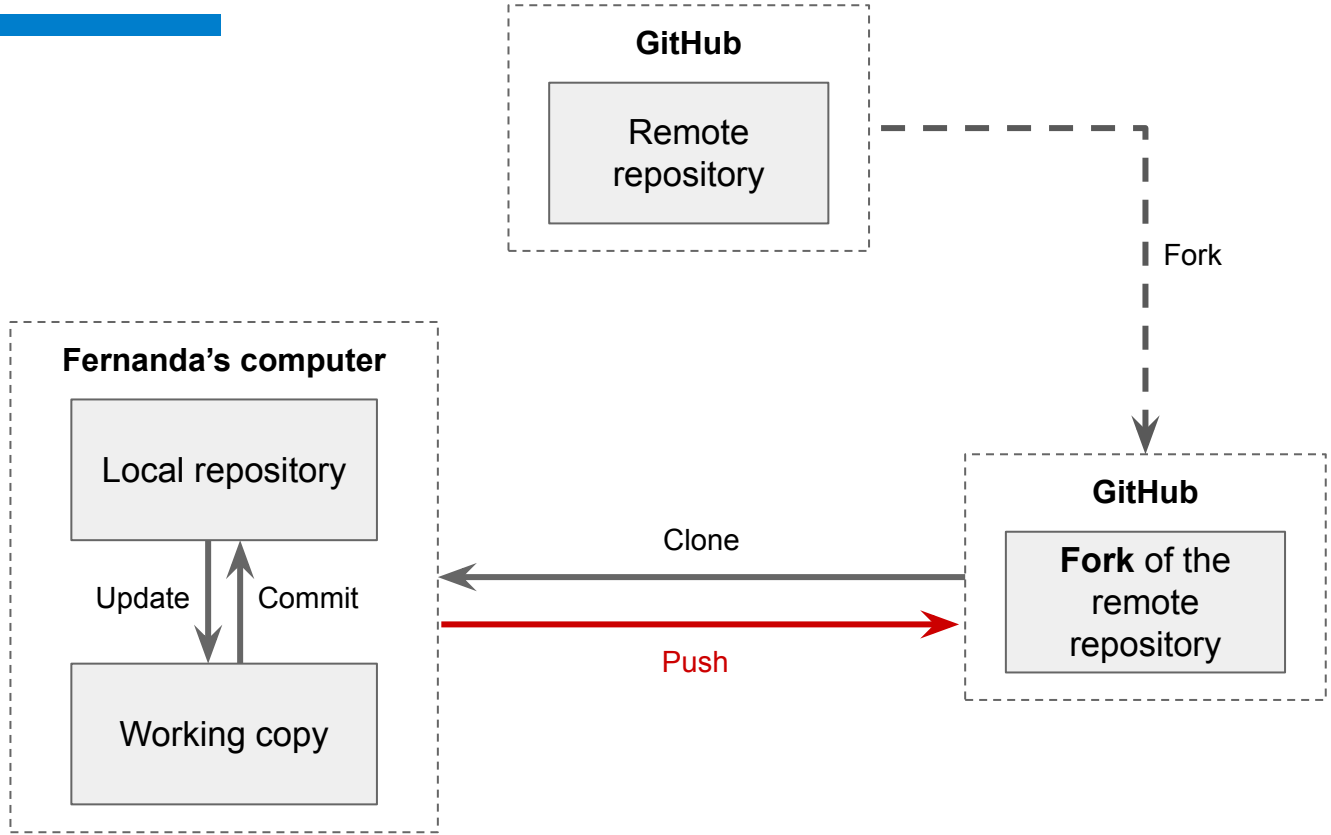
Pull-based development



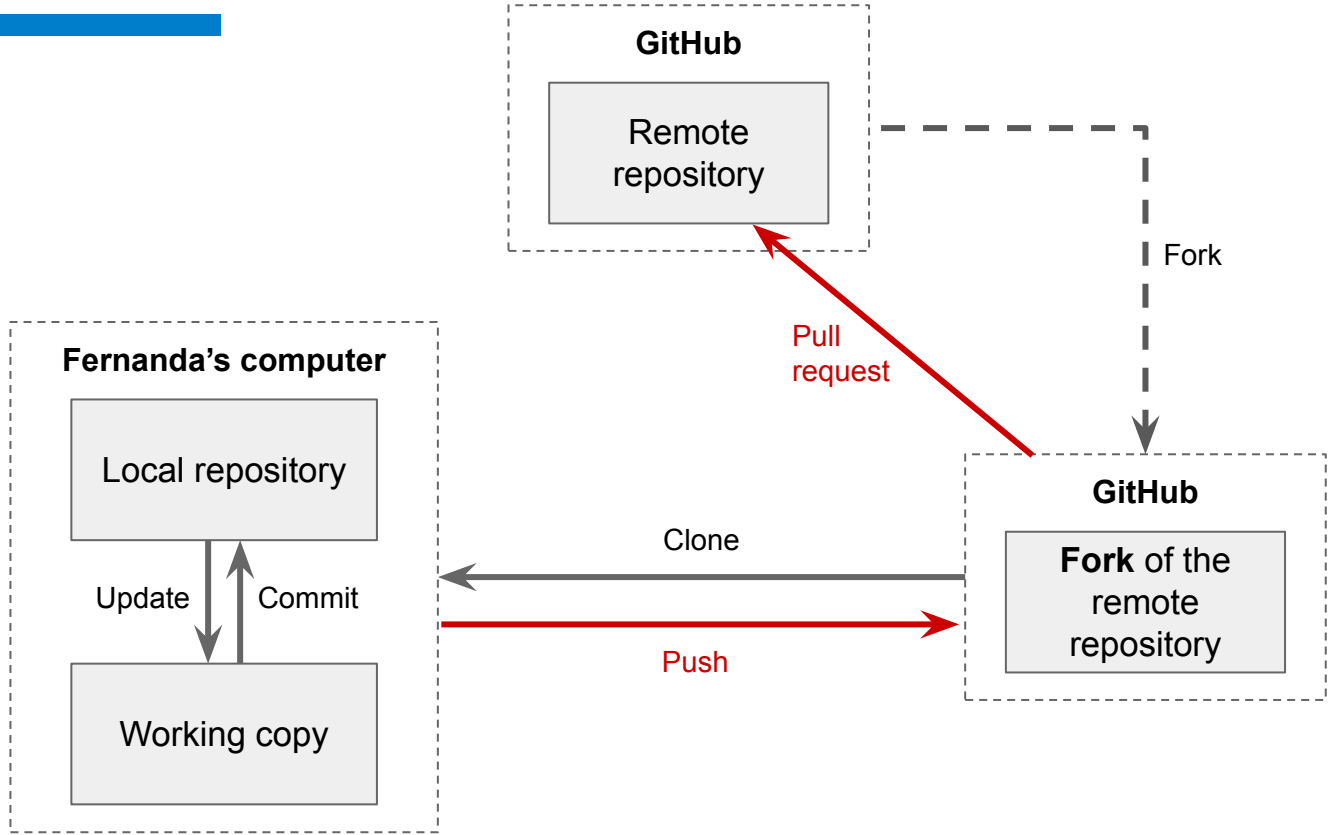
Pull-based development



Pull-based development



Pull-based development



Pull-based development

Main goal:

- To allow **external contributions in the repository**

What other goal do you see with this approach?

GitHub Actions

GitHub Actions

Main goal:

- To **automate** software development tasks and processes

GitHub Actions: workflow

A workflow is a configurable automated process that will run one or more jobs

Workflows are defined in a .yml file checked in to your repository

Workflows are defined in the “.github/workflows” directory in the repository

Workflow basics

A workflow must contain the following basic components:

- One or more **events** that will trigger the workflow
- One or more **jobs**, each of which will execute on a runner machine and run a series of one or more **steps**
- Each step can either **run** a script defined by the developer or a predefined action

GitHub Actions: example

```yaml
name: hello-world
on: push
jobs:
  my-job:
    runs-on: ubuntu-latest
    steps:
      - name: My step
        run: echo "Hello World!"
```

GitHub Actions: example

```yaml
name: hello-world
on: push
jobs:
  my-job:
    runs-on: ubuntu-latest
    steps:
      - name: My step
        run: echo "Hello World!"
```

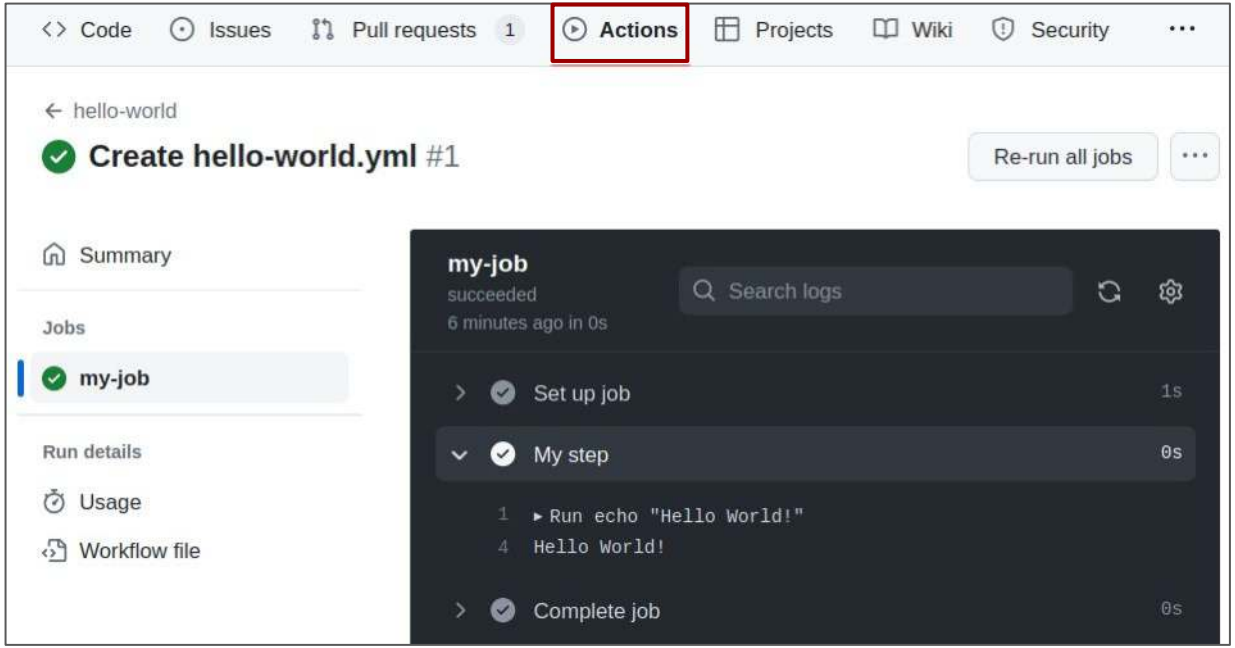
Event

Job

Step

GitHub Actions: example

```yaml
name: hello-world
on: push
jobs:
  my-job:
    runs-on: ubuntu-latest
    steps:
      - name: My step
        run: echo "Hello World!"
```



GitHub Actions: example

```yaml
name: Java CI
on: push
jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: '17'
          distribution: 'temurin'
      - name: Build with Maven
        run: mvn --batch-mode --update-snapshots package
```

Build and test
with Maven
on push (CI)

More of this on the CI/CD
lecture!

GitHub Actions: example

```yaml
on:
  issues:
    types: opened

jobs:
  comment:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/github-script@v6
        with:
          script: |
            github.rest.issues.createComment({
              issue_number: context.issue.number,
              owner: context.repo.owner,
              repo: context.repo.repo,
              body: '👋 Thanks for reporting!'
            })
```

Comment on an issue with the predefined action “actions/github-script”

Basic collaboration practices

Basic collaboration practices

- Each commit is an atomic change

Each commit does one, and only one simple thing, that can be summed up in a simple sentence

Basic collaboration practices

- Each commit message is clear, follows a pattern, and is traceable

Pattern: For example, using a prefix to state the kind of change in the commit (e.g., "fix:", "feature:")

Traceable: For example, using a suffix with the number of issue or pull request that is linked to an issue

Basic collaboration practices

- Creation of issues to organize small tasks
- Issues should be updated, i.e., closed once addressed, and contain information about the commit(s) in which they were addressed

Basic collaboration practices

- Every commit on the master branch should only be done through the merge of a pull request
- Every pull request should be merged by a different person than who opened it

Basic collaboration practices

In open-source repositories, you can find instructions for collaborations/contributions:

- Sometimes in the README.md
- Sometimes there is a file named CONTRIBUTING.md

Open Source

Free Software

Freedom 0: The freedom to run the program as you wish, for any purpose.

Freedom 1: The freedom to study how the program works, and change it, so it does your computing as you wish. Access to the source code is a precondition for this.

Freedom 2: The freedom to redistribute copies so you can help others in your community.

Freedom 3: The freedom to distribute copies of your modified versions to others. By doing this, you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

Open Source

Open source stands for criteria a little looser than those of Free Software

Free Software vs. Open Source

They stand for views based on fundamentally different values:

- For the free software movement, free software is an ethical imperative, essential **respect for the users' freedom**.
- By contrast, the philosophy of open source considers issues in terms of **how to make software “better”**, in a practical sense only.

Licenses

They outline the terms and conditions under which the software can be used, modified, and distributed

Licenses

Free		Proprietary
Permissive	**Copyleft (protective or restrictive)**	
E.g., MIT	E.g., GPL	No copying, no modification

Licenses (only the ones related to free and open source)

	Permissive	**Copyleft (protective or restrictive)**
Right to use	Yes	Yes
Right to modify	Yes	Yes
Right to distribute original	Yes, with the same license	Yes, with the same license
Right to distribute modified	Yes, with any license	Yes, with the same license
Right to proprietization	Yes	No

Final remarks

Free and open-source software are similar, but have different values/focus

Licenses are important:

- They specify how software can be used, modified, and distributed
- Unlicensed software cannot be used

TODO

Reading

Exam material:

- Slides “SEP2024-Lecture5-git-full.pdf”
- Slides “SEP2024-Lecture5-GitHub-full.pdf”
- Ian Sommerville, “Engineering Software Products”, 2020: Chapter 10 - “DevOps and Code Management”, Section 10.1 - “Code management[1]”

Recommended material:

- https://git-scm.com/
- GitHub Docs: https://docs.github.com/en
- Book: Gustavo Pinto, “Open Source Licensing 101”, 2020

[1] “Code management systems” is the same as “version control systems”

Takeaways?

?