Git & GitHub training session

https://github.com/EESSI/meetings/wiki/Git-and-GitHub-training-session

July 3rd 2020

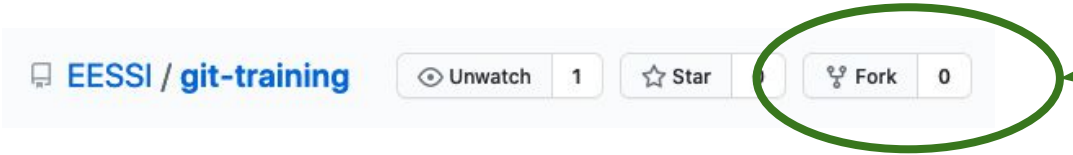
Topics

- GitHub basics
  - repos, forks, issues, pull requests
  - wiki, online editor, projects
  - GitHub Actions, CI
- Git basics
  - clone, fetch, branch, checkout, remote, pull, push, status, diff, add, commit, log, reset
- **Hands-on with a toy repository**

GitHub: repositories & forks

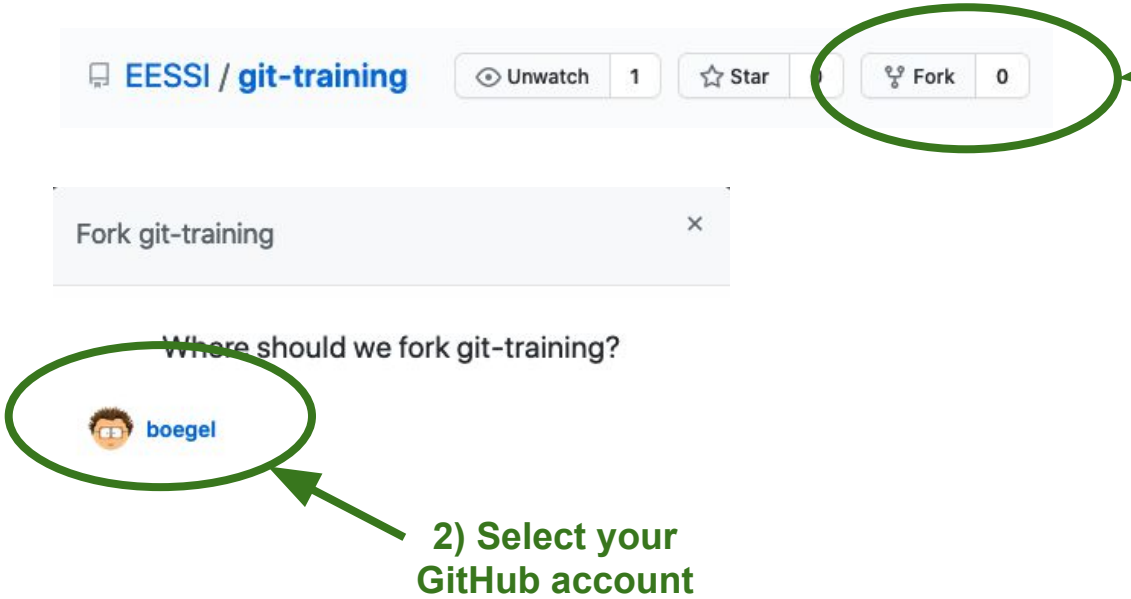
- You can create a new repository via https://github.com/new
  - Initialize with README file to you can clone it & get started (recommended)
  - Or make a totally empty repository and use “git init” and “git push”
- You can “fork” an existing repository into your GitHub account
  - Basically your own copy of that repository, in GitHub
  - Useful for making local changes, preparing contributions, testing in CI, ...

GitHub: forking a repository



1) Click “Fork” button

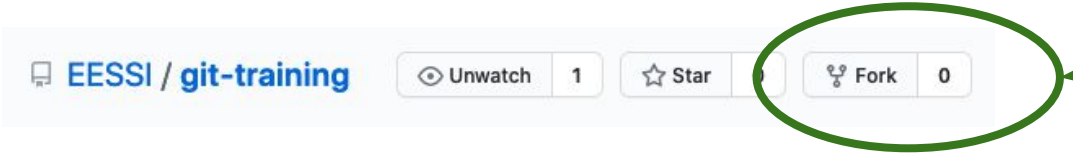
GitHub: forking a repository



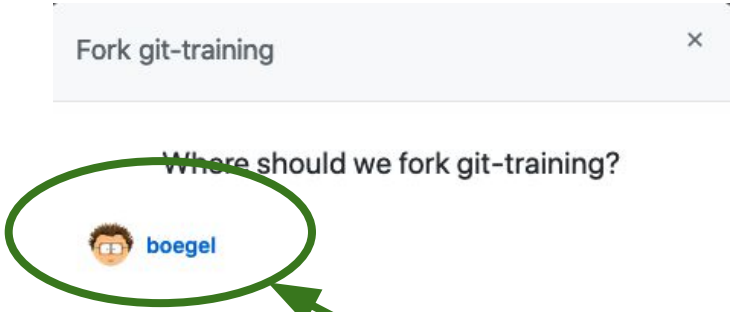
1) Click “Fork” button

2) Select your GitHub account

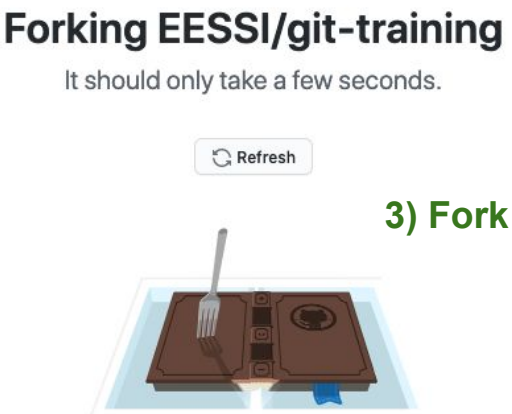
GitHub: forking a repository



1) Click “Fork” button

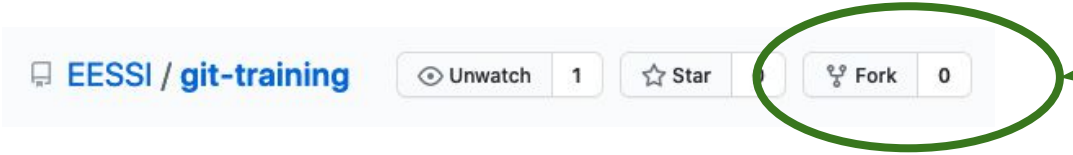


2) Select your GitHub account

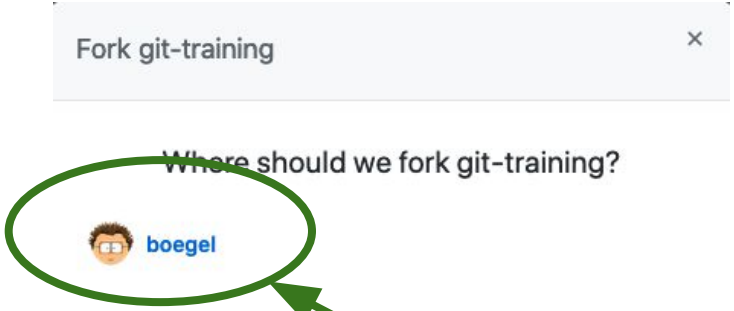


3) Fork is being created...

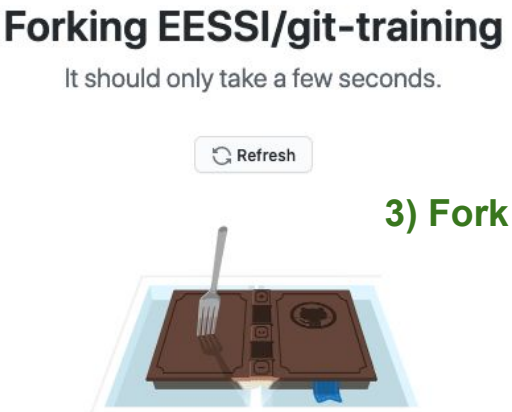
GitHub: forking a repository



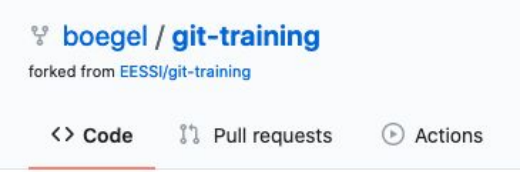
1) Click “Fork” button



2) Select your GitHub account

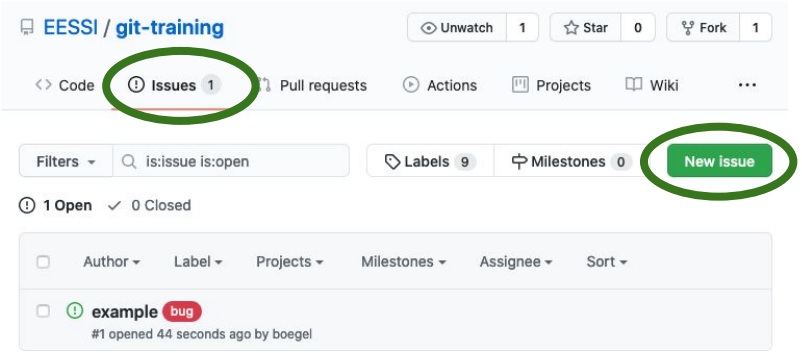


3) Fork is being created...



4) Done!

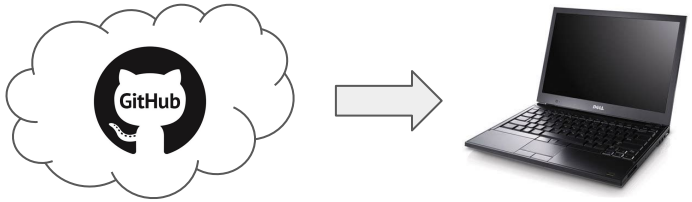
GitHub: issues

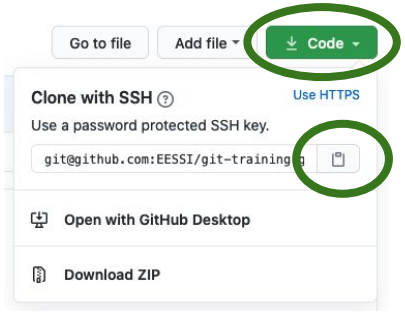


- Bugs, feature requests, questions, …
- Each issue has a unique ID (#123)
- Can have labels or milestones, assigned to someone, part of a project, …
- Use “New issue” button to create new issues
- **Should be used for “documenting” stuff for future reference**

  (TODOs, errors, problems, design decisions, …)

git clone





- Use “git clone” to make a local working copy of a repository

```
git clone git@github.com:EESSI/git-training.git
```

- Can be done via “git@” or “https://” URL
  - Using “git@” URL is recommended (easier for read/write access)
  - Cloning via “git@” requires having SSH public key in GitHub account! see https://github.com/settings/keys
  - “https://” URL also works, but requires using GitHub password for write access

git remote

- Use “git remote” to create/list set of tracked repository (central + forks)

```
$ cd git-training

$ git remote add YOU git@github.com:YOU/git-training.git
```

Change “YOU” to your GitHub account!

- ‘origin’ usually points to central repository (sometimes named “upstream”)

```
$ git remote -v

YOU     git@github.com:YOU/git-training.git (fetch)
YOU     git@github.com:YOU/git-training.git (push)
origin    git@github.com:EESSI/git-training.git (fetch)
origin    git@github.com:EESSI/git-training.git (push)
```

Git: commits

- Git using “blockchain” mechanism (Merkle trees)
- Each set of changes in repository corresponds to a “commit”
- Each commit has a parent commit
- Full commit ID is 40 character SHA-1 hash:

  c246bb12f089d15849672a4044eb2ba75baf37b5

- Shorthands (first 7 chars) are commonly used: c246bb1

git branch

- A “branch” in Git is little more than a **label** for a specific commit
- Branches can be “updated”: label moves to another commit
- Default branch is usually ‘master’ (maybe soon ‘main’)
- Listing all branches: `git branch`
- Creating a branch: `git branch example`
- Deleting a branch: `git branch -d example`
- HEAD is a special “branch”: refers to *current* branch

git checkout

- To switch to a specific branch, use `git checkout`
- To create a new branch & switch to it:

```
git checkout -b example
```

git status

- To check current status of your working copy, use `git status`
- See which files were changed, added, removed, ...

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.
nothing to commit, working tree clean
```

```
$ echo test > test.txt; git status
On branch master
Your branch is up to date with 'origin/master'.
Untracked files:
    test.txt
```

git add (“staging”)

- To add changes to a branch, you have to “stage” them first: **`git add`**
- This is basically preparing to make a commit

```
$ git add test.txt
$ git status
On branch master
Your branch is up to date with 'origin/master'.
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   test.txt
```

git reset

- To undo staging of changes, you can use `git reset`
- If you're in trouble `git reset --hard` can be an escape hatch
- **BE VERY CAREFUL** with `--hard` (best way to lose your work)

```
$ git status
Changes to be committed:
    modified:   test.txt
```

```
$ git reset
Unstaged changes after reset:
M    test.txt

$ git status
Changes not staged for commit:
    modified:   test.txt
```

git commit

- To create a new commit, use… `git commit`
- By default: use staged changes + open editor to create commit message
- Using meaningful commit message is strongly recommended
  (be kind to your future self)
- Use `git commit -m "..."` to specify commit message directly
- Use `git commit -a` to include changes to *all* *<u>tracked</u>* files

git log

To see overview of commits in current branch, use `git log`

```
$ git log
commit 9f702a56e7367534e8b20691534610930ee7f595 (HEAD -> example)
Author: Kenneth Hoste <kenneth.hoste@ugent.be>
Date:   Fri Jul 3 12:48:02 2020 +0200
        add new file: test.txt

commit c246bb12f089d15849672a4044eb2ba75baf37b5 (origin/master, origin/HEAD, master)
Author: Kenneth Hoste <kenneth.hoste@ugent.be>
Date:   Fri Jul 3 09:57:00 2020 +0200
        Initial commit
```

git checkout (revisited)

- To undo (unstaged) changes, you can use `git checkout`
- Resets files to last commit **(final, no going back!)**
- `git checkout .` **to undo *all* changes (make sure you want to!)**

```
$ echo 123 > test.txt
$ git status
...
    modified:   test.txt
```

```
$ git checkout test.txt
Updated 1 path from the index

$ cat test.txt
test
```

git rm

- To stage a file or directory for removal, use **`git rm`**
- Just using rm only removes the file, doesn't do staging!

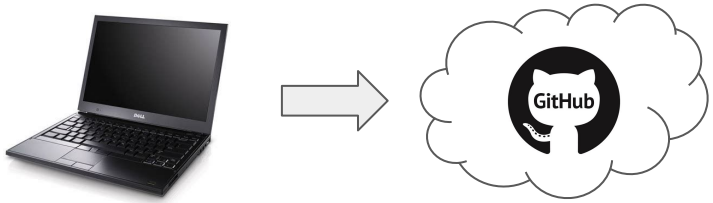
```
$ git rm test.txt
rm 'test.txt'
$ git status
On branch example
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:   test.txt
```

git diff

- To see overview of current (unstaged) changes, use `git diff`
- To see overview of staged changes, use `git diff --cached`

```
$ git diff
diff --git a/test.txt b/test.txt
index 9daeafb..190a180 100644
--- a/test.txt
+++ b/test.txt
@@ -1 +1 @@
-test
+123
```

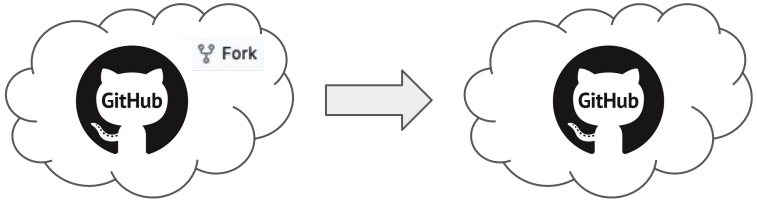
git push



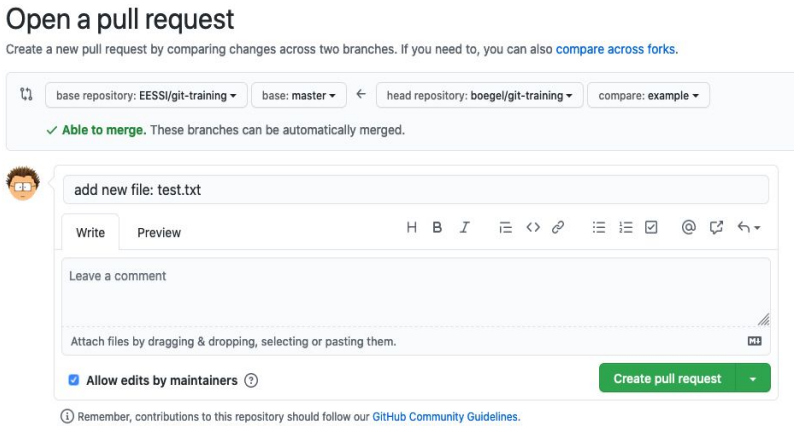
- Use **`git push`** to push a branch to GitHub (usually to *your* fork)
- Same procedure to update an existing branch

```
$ git push YOU example
...
remote:
remote: Create a pull request for 'example' on GitHub by visiting:
remote:     https://github.com/boegel/git-training/pull/new/example
remote:
To github.com:boegel/git-training.git
 * [new branch]      example -> example
```

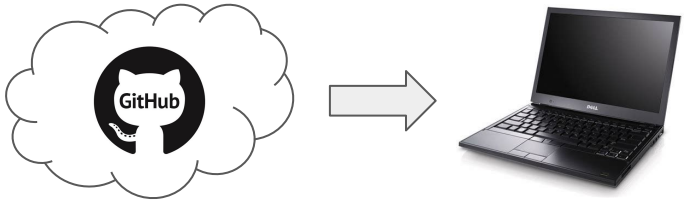
GitHub: pull requests



- To ask to include changes in a repository, you can open a **pull request (PR)**
- Typically from branch in fork to master branch in central repository
- Usually also involves a review process + adding additional changes
- Adding commits to branch used for PR is equivalent to updating the PR



git fetch



- Use “git fetch” to download changes from a repository (via a remote)
- Can be either from central repository (origin) or a fork

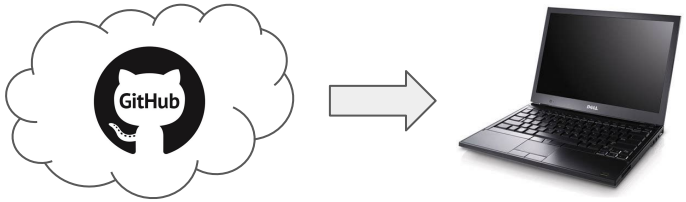
```
$ git remote add boegel git@github.com:boegel/git-training.git
$ git fetch boegel
...
From github.com:boegel/git-training
 * [new branch]      example -> boegel/example
 * [new branch]      master  -> boegel/master
```

git merge

- To include all changes from branch A, use **`git merge`**

```
$ git checkout master
$ git checkout -b my_example
$ git merge boegel/example
Updating c246bb1..9f702a5
Fast-forward
 test.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
```

git pull



- `git pull` is basically equivalent `git fetch` + `git merge`
- Download changes from specific remote + branch,

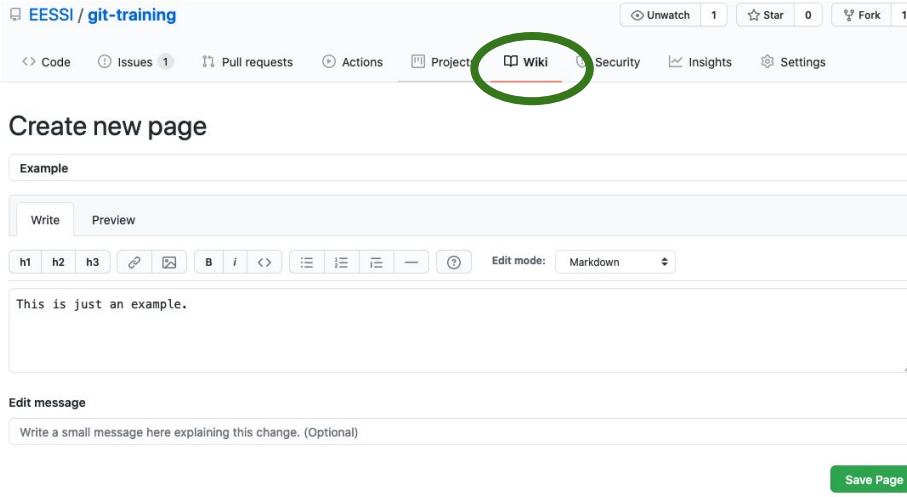
  and merge them into current branch
- To update your master branch:

```
git checkout master

git pull origin master
```

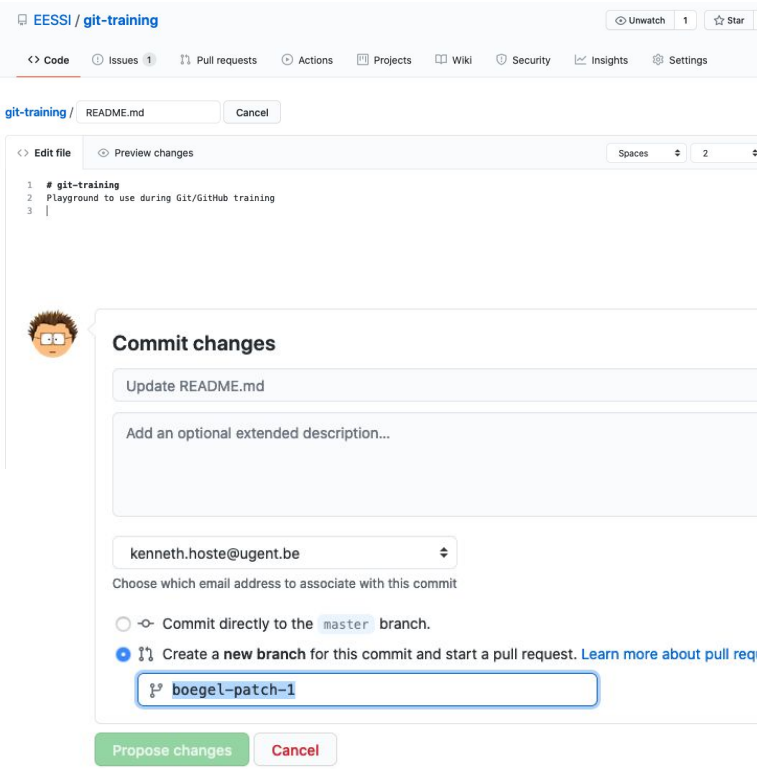
GitHub: wiki

- Each GitHub repository comes with a wiki
- Markdown format by default (but other formats also supports)
- Useful for taking notes, sharing information
- Easily goes stale…
- Used in EESSI for:
  - Meeting notes
  - Brainstorm sessions



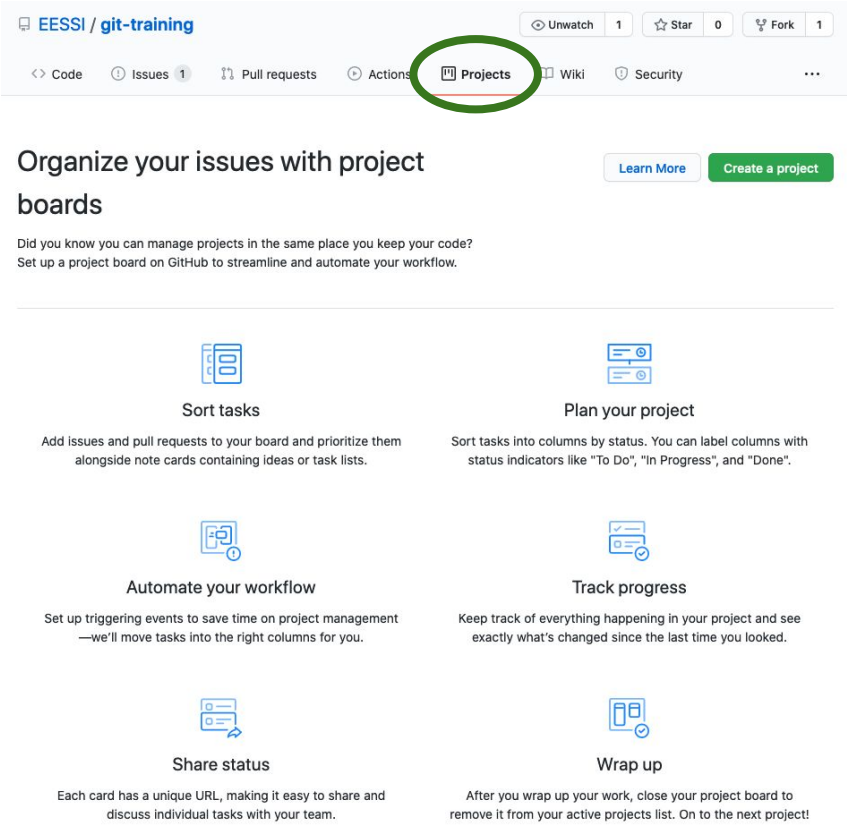
GitHub: code editor

- Editor available in GitHub web interface
- Useful for making quick changes + PR
- Pay attention to:
  - Meaningful commit message + description
  - Name of branch
  - Branch is created in central repo (not your fork)!



GitHub: projects

- Kanban board
- Useful for following up on specific goals
- Not used yet in EESSI, maybe later?



GitHub Actions (CI)

- Native CI support in GitHub
- Workflow is defined in a YAML file in .github/workflows
- Can be used for any type of automation
- Automated testing, perhaps deployment of software in EESSI stack?