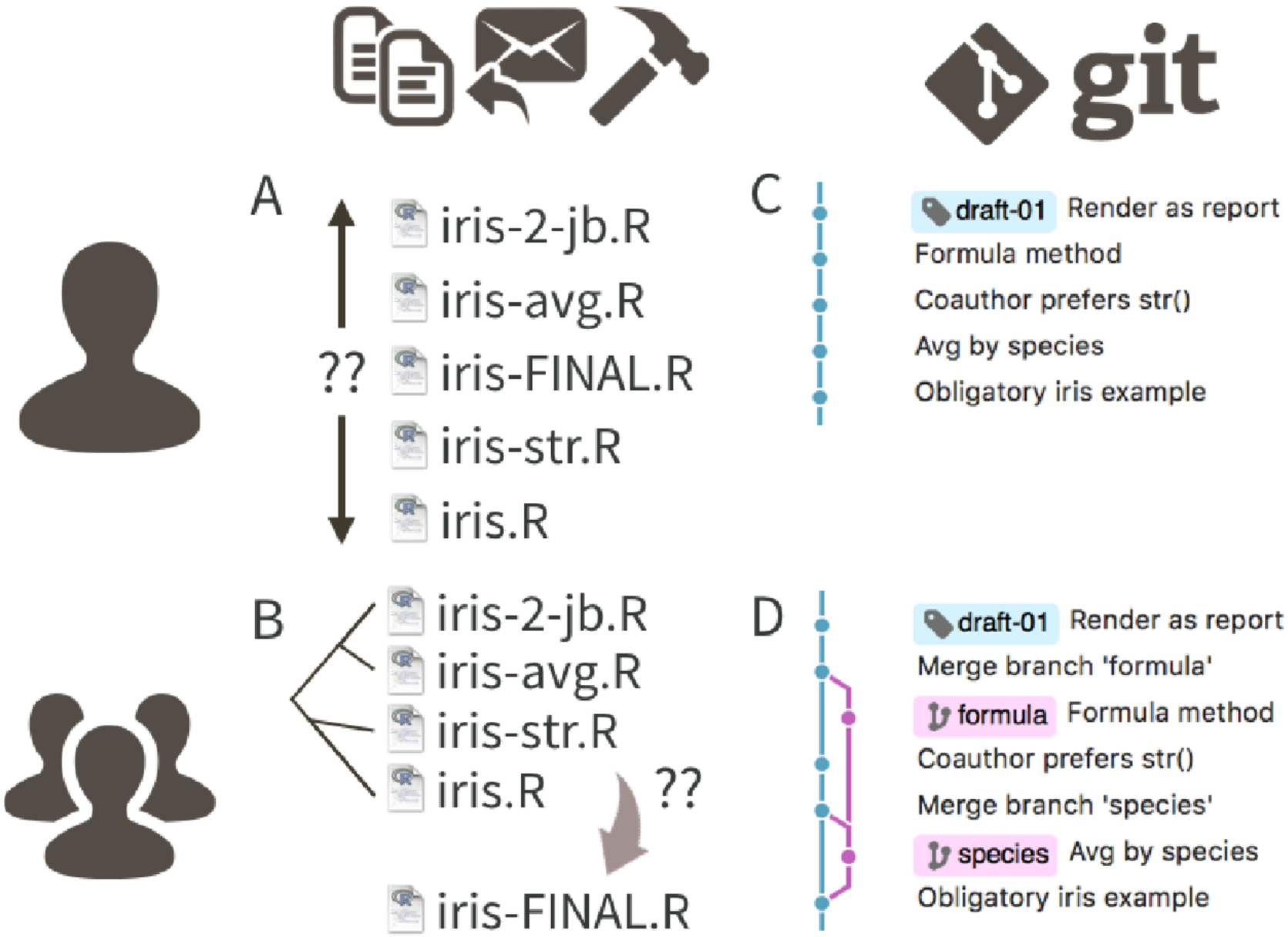
Git and GitHub

Iegor Rudnytskyi

Motivation



Source: Bryan, Jennifer. "Excuse me, do you have a moment to talk about version control?." The American Statistician 72.1 (2018): 20-27.

Motivation

Source: R Memes For Statistical Fiends Facebook page

Version-control systems

A version-control systems is a tool that tracks changes to your files in a special kind of database and shares those changes with others.

Why it is useful:

- Records the entire history of each file (makes possible to revert back to a previous version and helps to avoid duplication of files)
- Enables multiple people to simultaneously work on a single project (or one person to use multiple computers to work on a project)

Popular VCS:

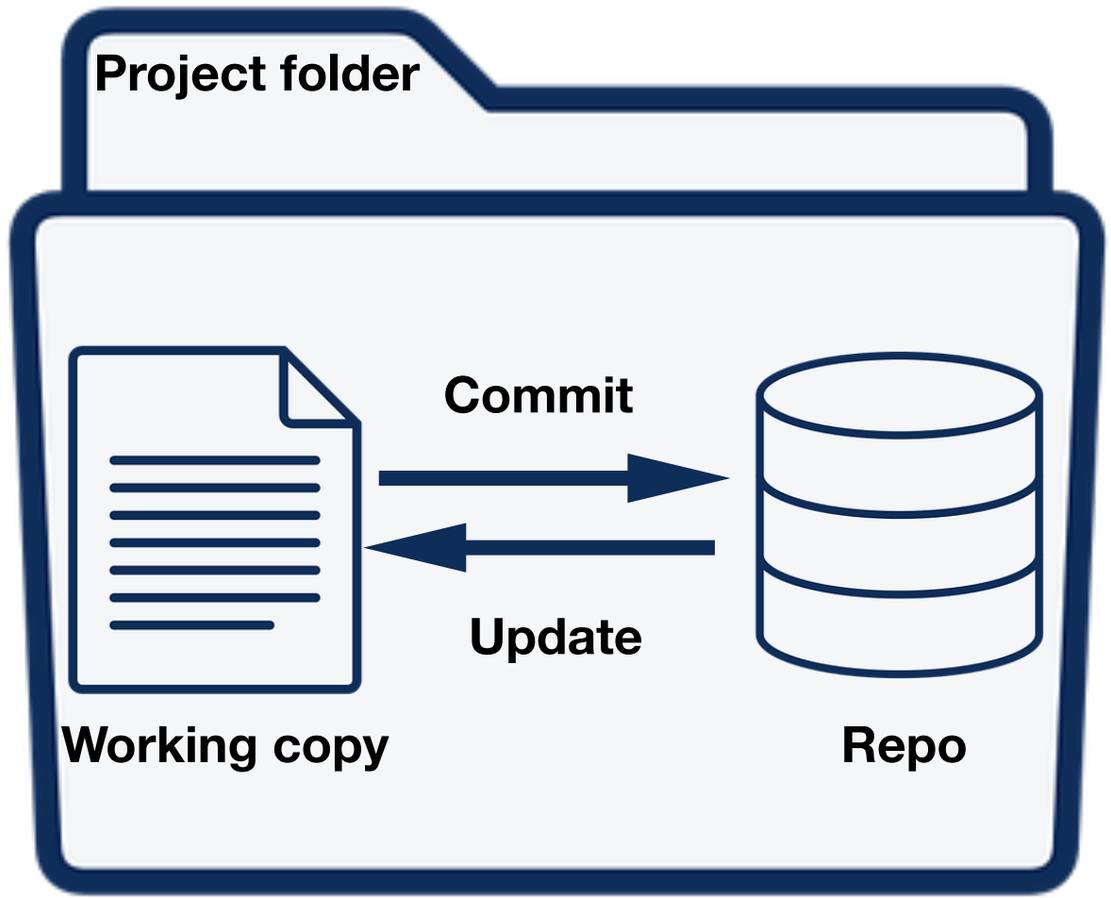
- **git**
- SVN (Subversion)
- Mercurial

Version-control systems

Software projects are typically organized in files that are stored in a folder.

A **working copy** is then your personal copy of all the files in the project.

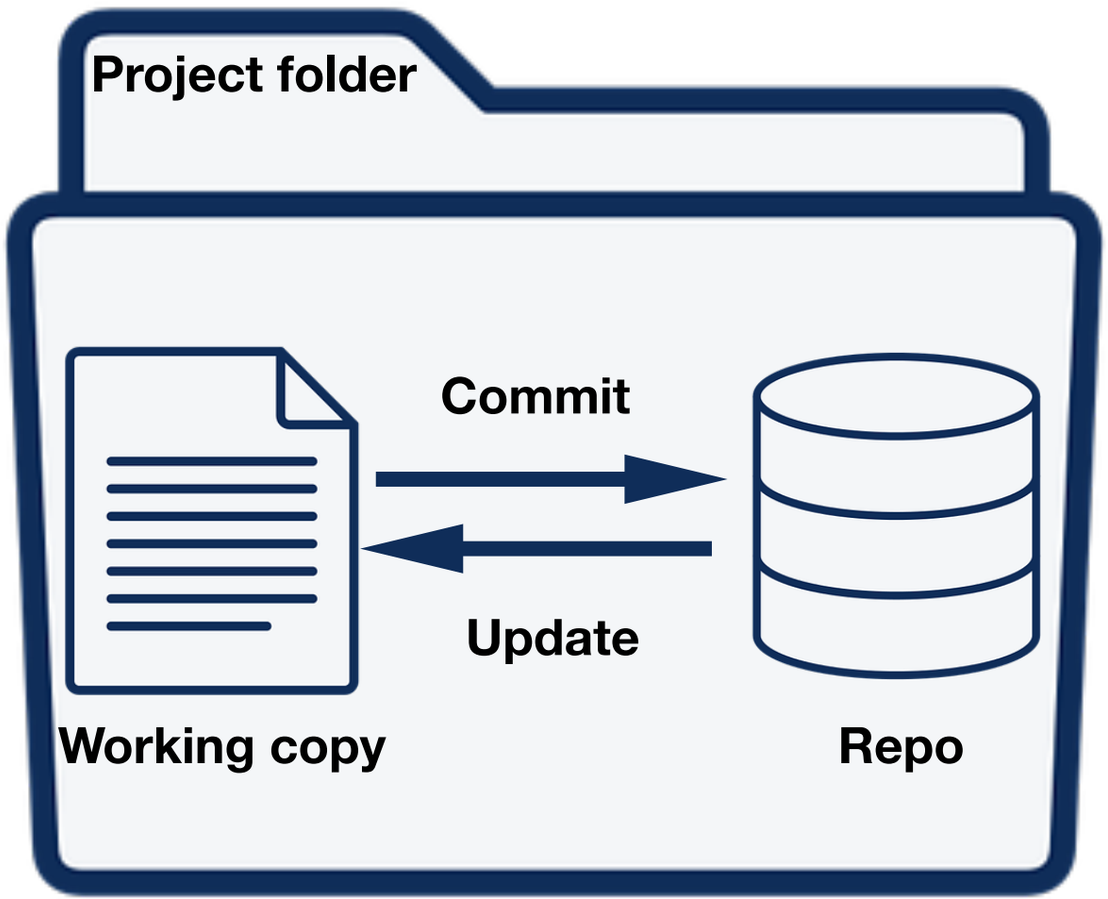
A repository (**repo** for short) is a sort of a database that records changes (historical versions or snapshots) to your files.



Version-control systems

After you have changed your files (the working copy), one needs to declare these changes to the repo. It is called to **commit**, that is to take a snapshot of all the files in the entire project, assign to it a message, and record it in repo. **Commit** is a fundamental unit of work.

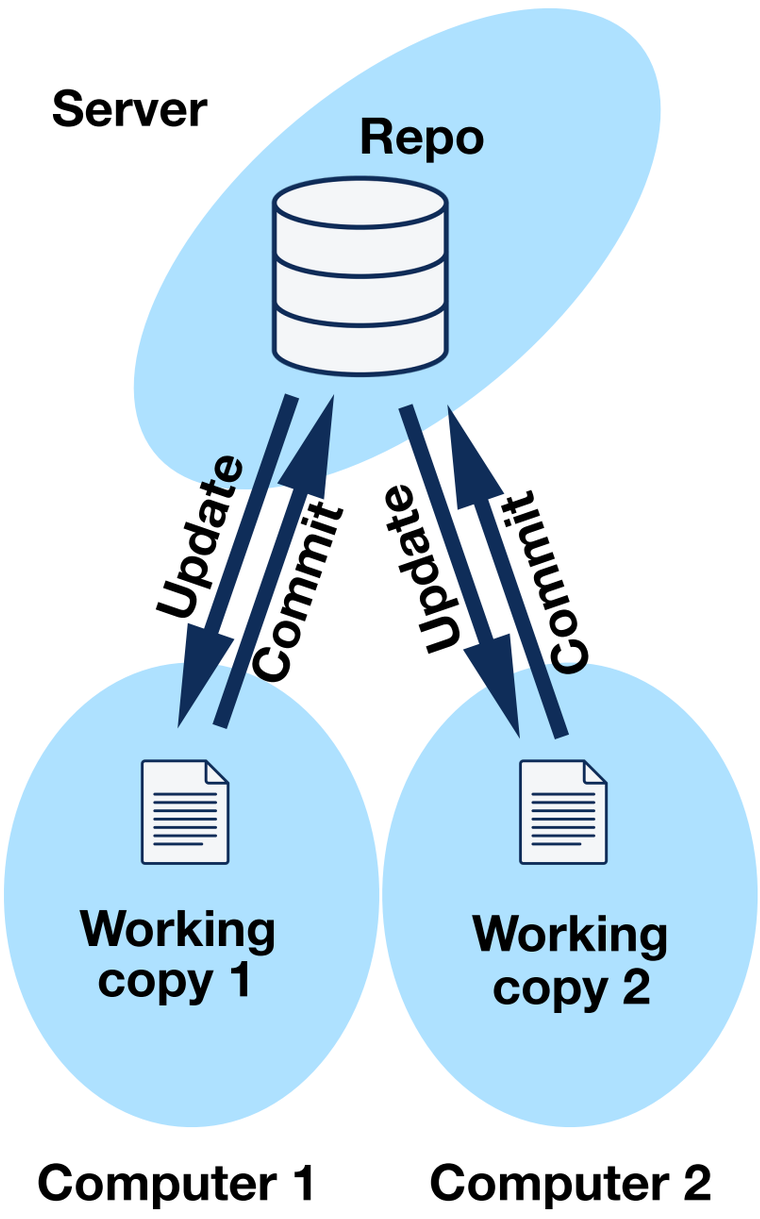
It is possible for the repository to contain edits that have not yet been applied to your working copy. Then, one needs to **update.**



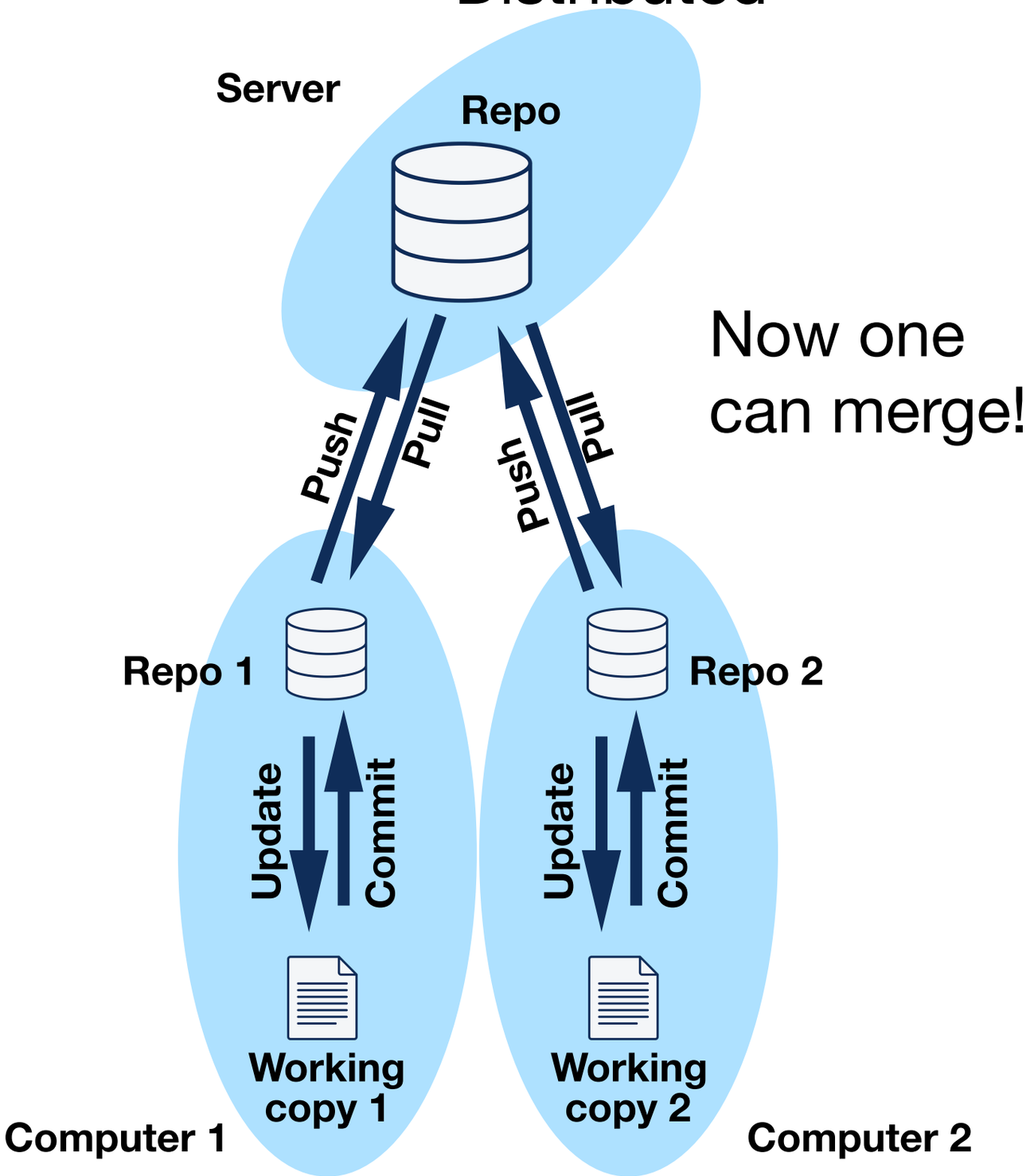
Centralized vs distributed version control system

Problem: where to store the repo if you are collaborating?

Centralized



Distributed



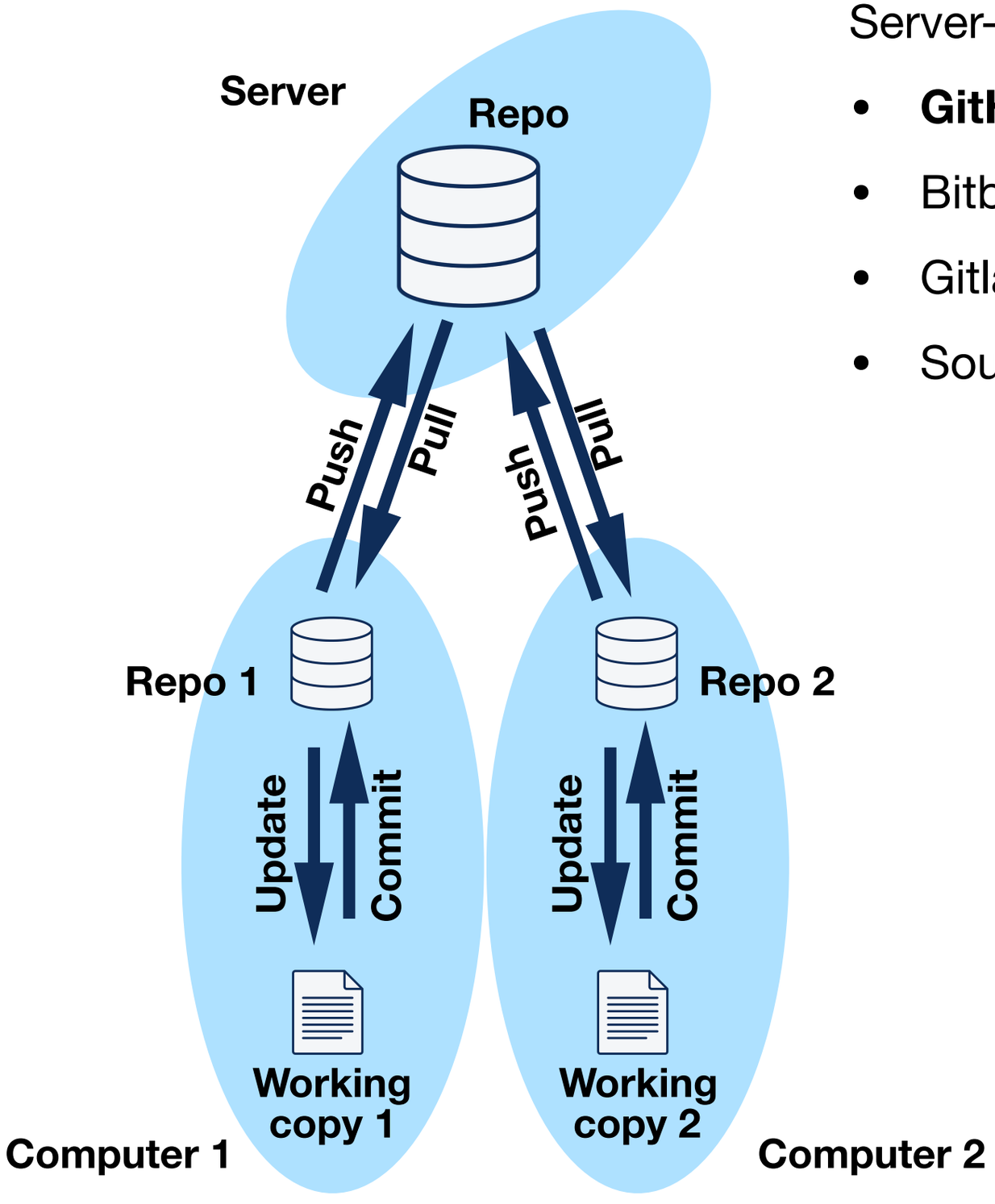
Now one can merge!

Distributed vs centralized version control system

After you commit, others have no access to your changes until you **push** your changes to the central repository. When you update, you do not get others' changes unless you have first **pull**ed those changes into your repository. For others to see your changes, 4 things must happen:

- You **commit**
- You **push**
- They **pull**
- They update (in git it is done during pulling, no needs of separate command)

Web-based version-control repository hosting services



Server-side:

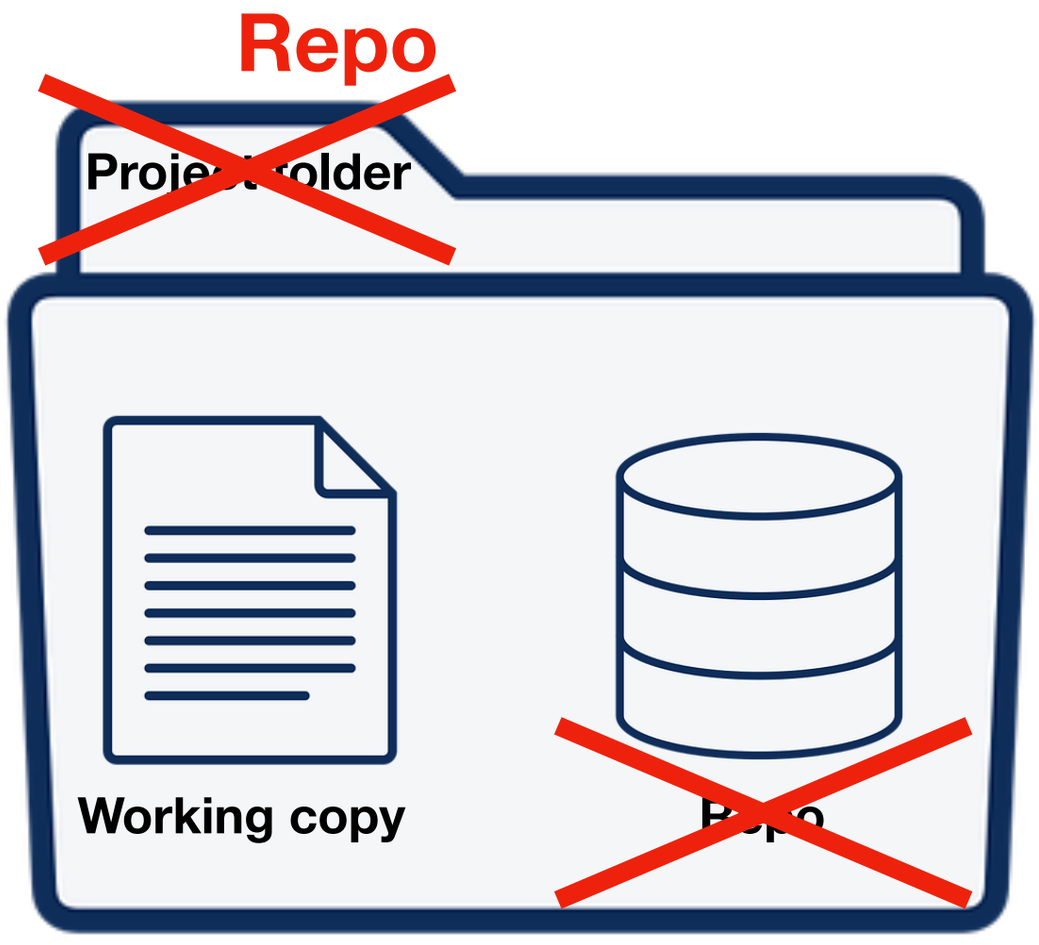
- **GitHub** (**git**, SVN)
- Bitbucket (git, Mercurial)
- Gitlab (git)
- SourceForge (git, SVN)

Why GitHub:

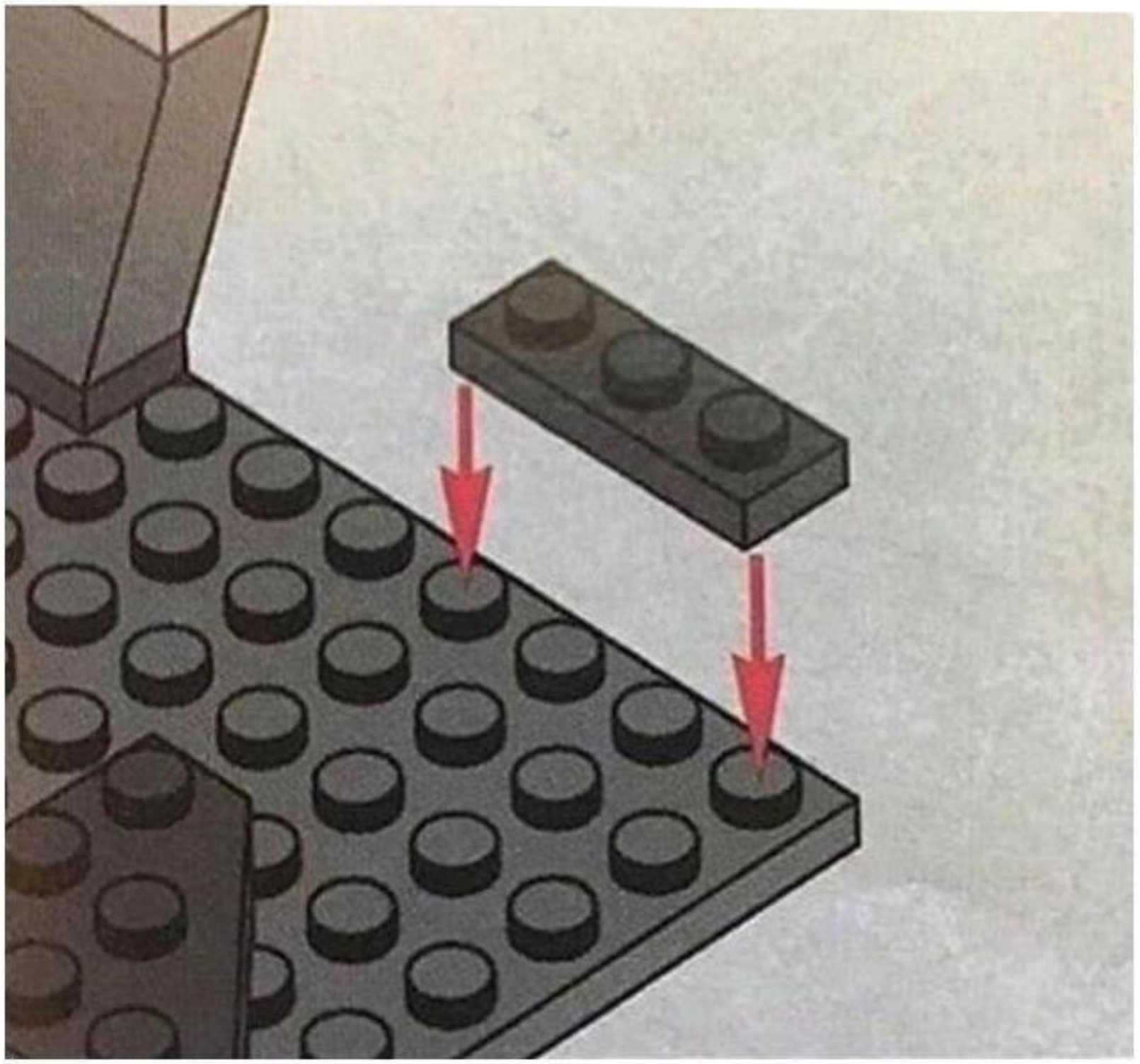
- Makes sharing easy
- Serves as Dropbox for backups
- Bug reporting system of GitHub Issues
- Barebone web-site
- GitHub Desktop
- Commenting commits, etc.

Again, Git is a version-control system, while GitHub is a hosting service for Git repos.

Version-control systems



Read tutorial they said, everything is clear they said…



How to create a new repository in GitHub?

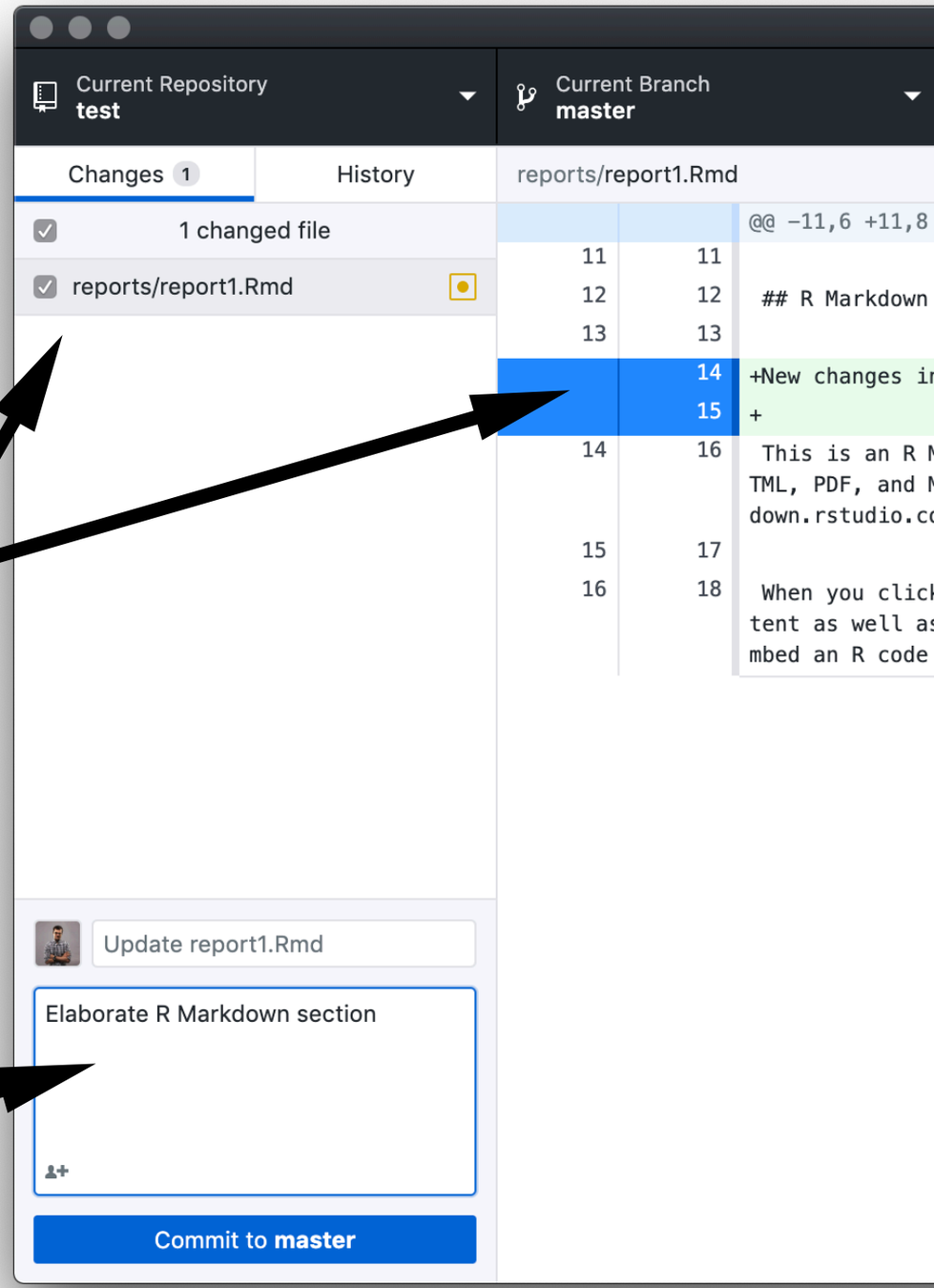
1. Pick a good name (**note:** names are defined for homeworks)
2. Open https://github.com/new
3. Fill in **Repository name** and **Description**. Choose the type of repo (**Public** vs **Private**) and hit **Create repository**.
4. Voilà! An empty repo has been created on a server. Its address it https://github.com/username/reponame

How to synchronize/connect GitHub repo with GitHub Desktop (clone)?

1. Open GitHub Desktop
2. Click **Clone Repository…**
3. Type the location of the repo in the following way: **username/reponame**
4. Voilà! You create a local copy of the repo. It is located in …/GitHub folder.

How to record changes of files locally (commit)?

1. Change files at …/GitHub/reponame
2. Open GitHub Desktop and navigate to a corresponding repo
3. Observe changes: they are highlighted in green (added lines) or in red (removed lines)
4. Check checkboxes of files that has been changed
5. Type the commit message (see next slide)
6. Hit **Commit**



How to write a good commit message?

Ideally, each commit should be *minimal* but *complete*:

- **Minimal**: A commit should only contain changes related to a single problem. This will make it easier to understand the commit at a glance, and to describe it with a simple message. If you should discover a new problem, you should do a separate commit.
- **Complete**: A commit should solve the problem that it claims to solve. If you think you've fixed a bug, the commit should contain a unit test that confirms you are right.

Source: Hadley Wickham, R packages

How to write a good commit message?

Each commit message should:

- **Be concise, yet evocative**. At a glance, you should be able to see what a commit does. But there should be enough detail so you can remember (and understand) what was done.
- **Describe the why, not the what**. Since you can always retrieve the diff associated with a commit, the message doesn’t need to say exactly what changed. Instead it should provide a high-level summary that focuses on the reasons for the change.

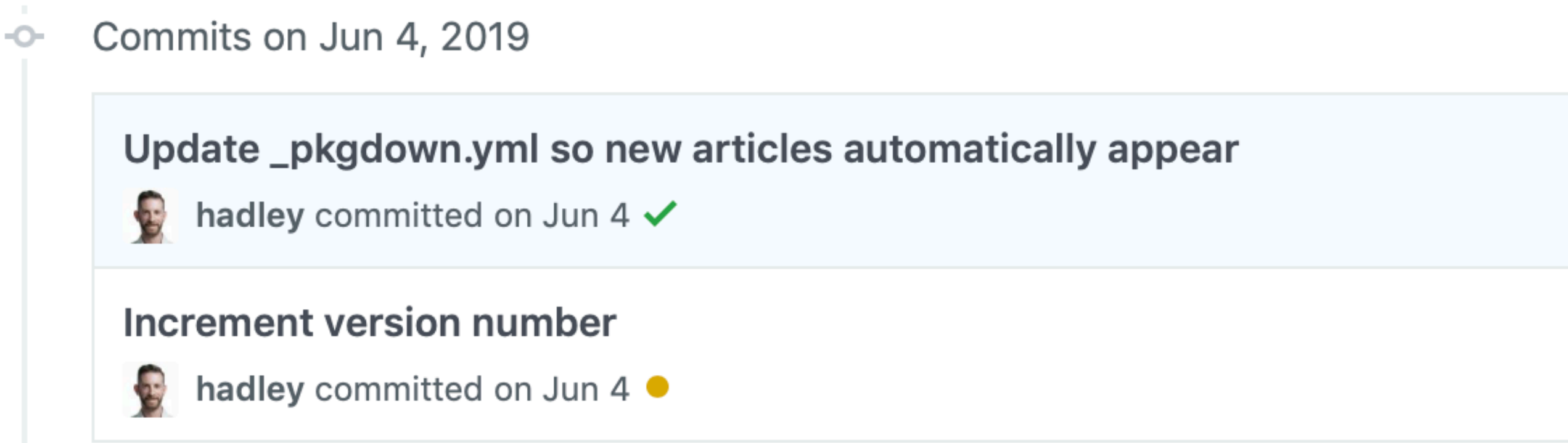
Source: Hadley Wickham, R packages

The seven rules of a great Git commit message

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

Source: How to Write a Git Commit Message, https://chris.beams.io/posts/git-commit/

Commits (inspiration)



Commits (inspiration)

Commits on Feb 9, 2017

Add plot summary API (#2034) ···

wch committed on Feb 9, 2017 ✓

Commits on Feb 2, 2017

Fix partial argument match

wch committed on Feb 2, 2017 ✓

Commits on Aug 12, 2016

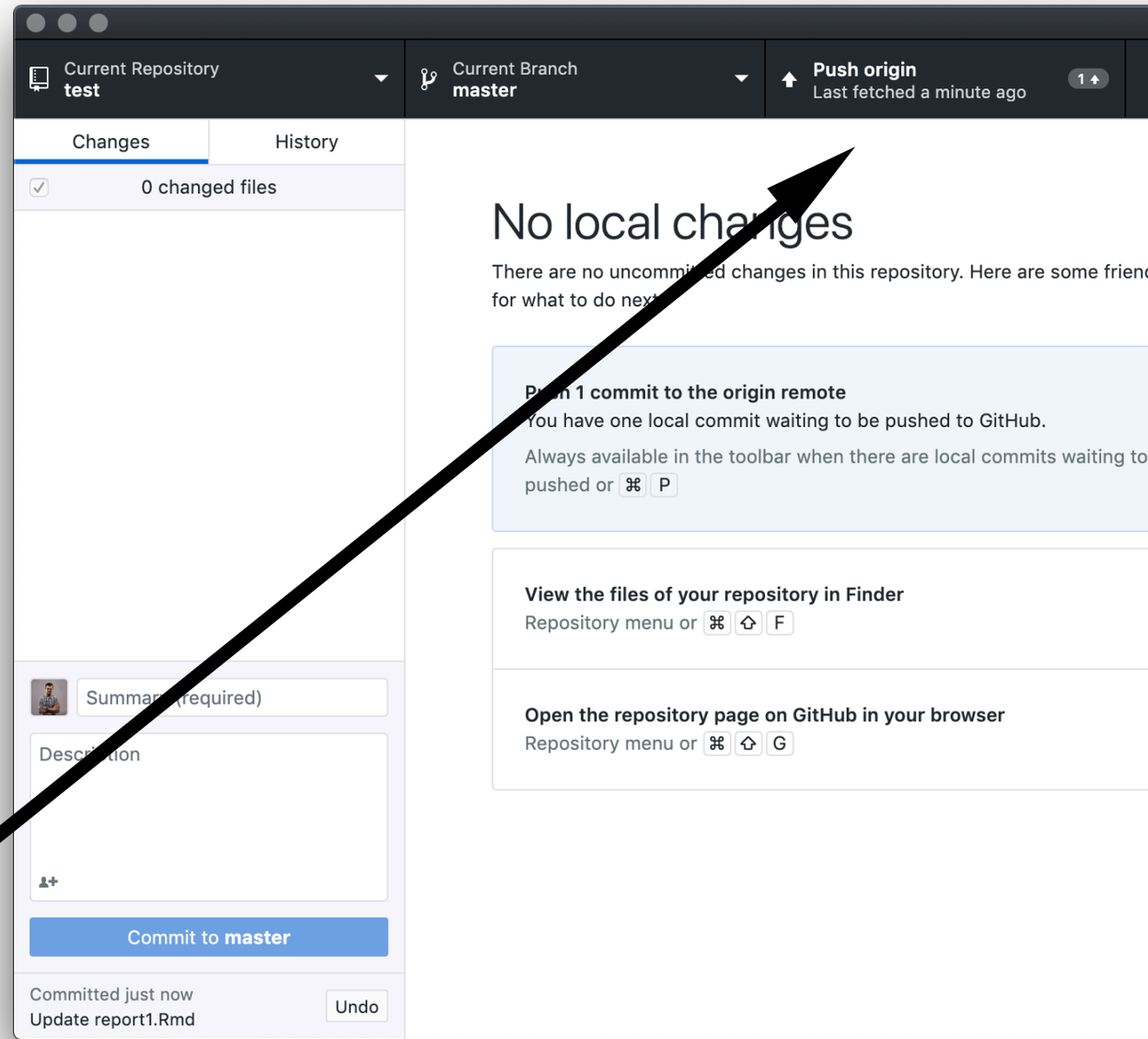
Preserve class when adding uneval objects (#1624) ···

wch authored and **hadley** committed on Aug 12, 2016 ✓

How to send committed changes to GitHub (push)?

Pushing refers to sending your committed changes to a remote repository, such as a repository hosted on GitHub. For instance, if you change something locally, you'd want to then **push** those changes so that others may access them.

1. Hit the **Push origin** button

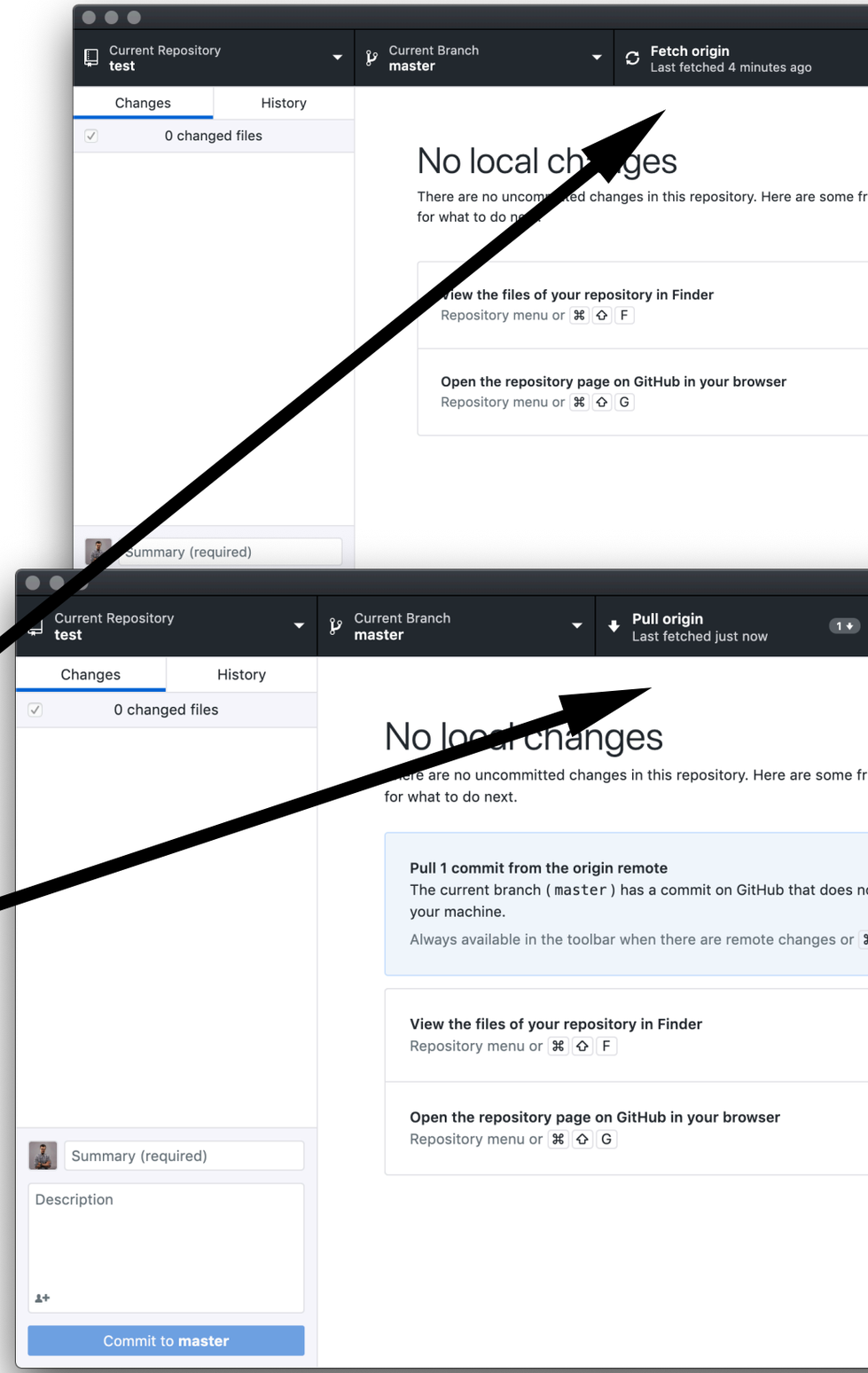


Source: GitHub glossary, https://help.github.com/en/articles/github-glossary

How to update your local repo with changes made by others (**pull**)?

Pull refers to when you are fetching in changes. For instance, if someone has edited the remote file you're both working on, you'll want to **pull** in those changes to your local copy so that it's up to date.

1. Hit the **Fetch origin** button
2. Hit the **Pull origin** button



Source: GitHub glossary, https://help.github.com/en/articles/github-glossary

Your usual workflow…

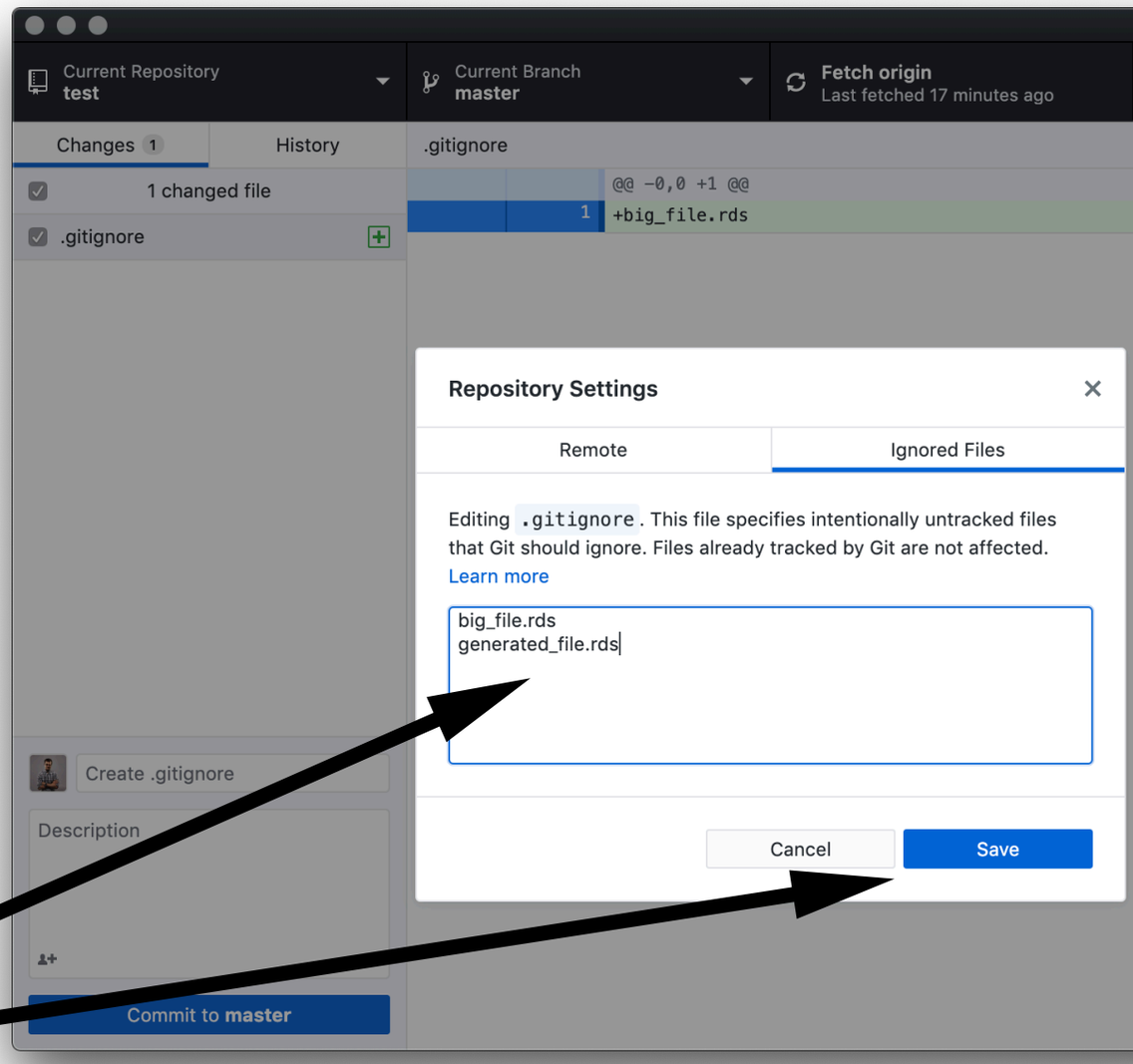
1. Open GitHub Desktop
2. **Pull** changes introduced by your collaborators
3. Add changes
4. **Commit** changes
5. **Push**

How to resolve a merge conflict?

What is .gitignore?

Often, there are files that you don't want to include in the repository. They might be transient (like LaTeX or C build artifacts), very large, or generated on demand. You should add them to `.gitignore`. This will prevent them from accidentally being added.

1. Repository -> Repository Settings -> Ignored Files
2. Add the file name to the list
3. Hit **Save**
4. **Commit** your changes and **push**



How to use GitHub Issues?

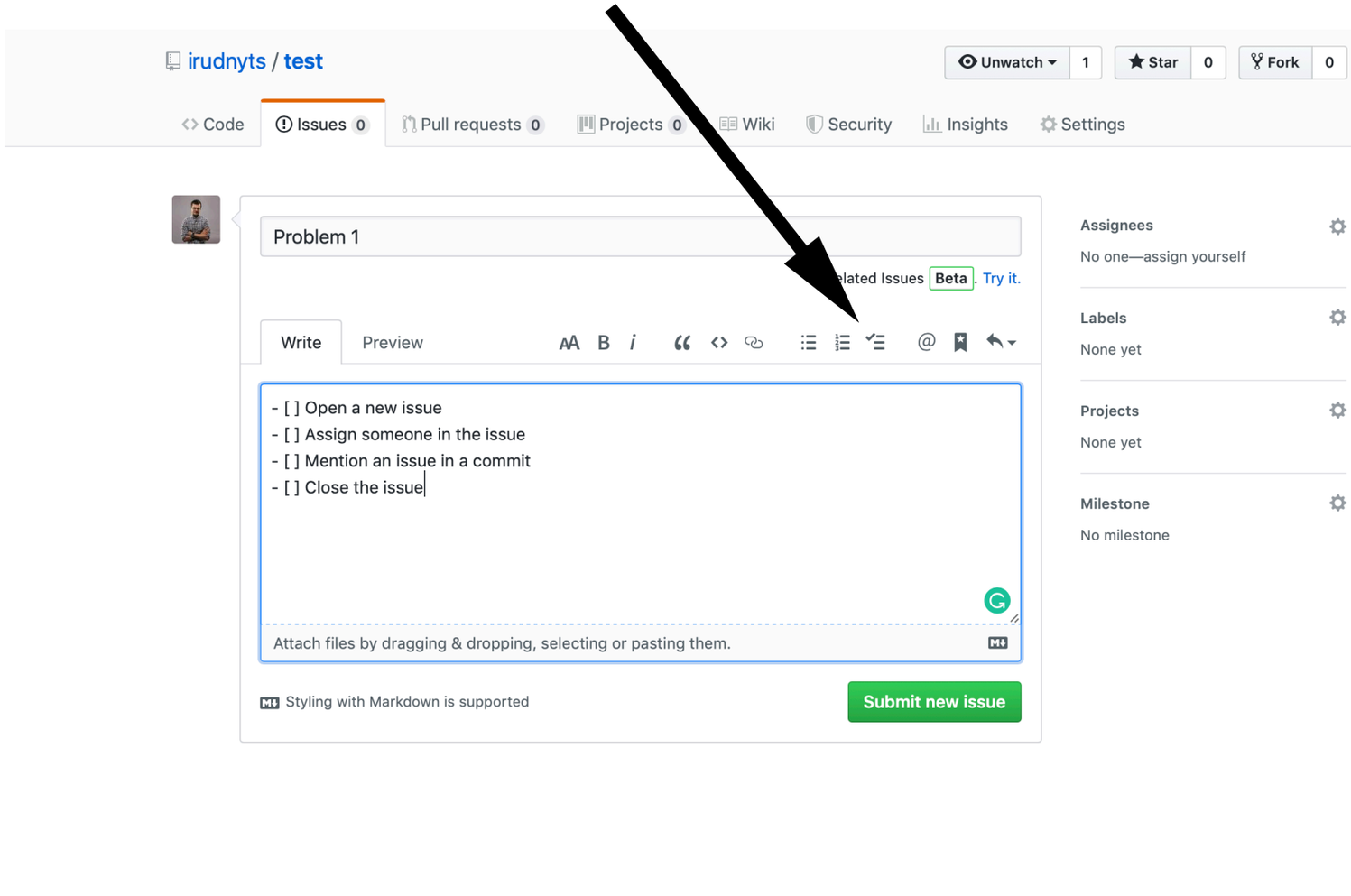
GitHub Issues: not only a bug tracker, but also a task organizer (i.e. to-do check lists):

- You can assign an issue to a collaborator.
- Task lists also have a summary.
- Issues are numbered and can be referenced in commit messages. Therefore, it is better to separate to-do items into separate issues.
- Do not forget to close an issue once it is done.

Source: Managing your work with issues, https://help.github.com/articles/managing-your-work-with-issues/

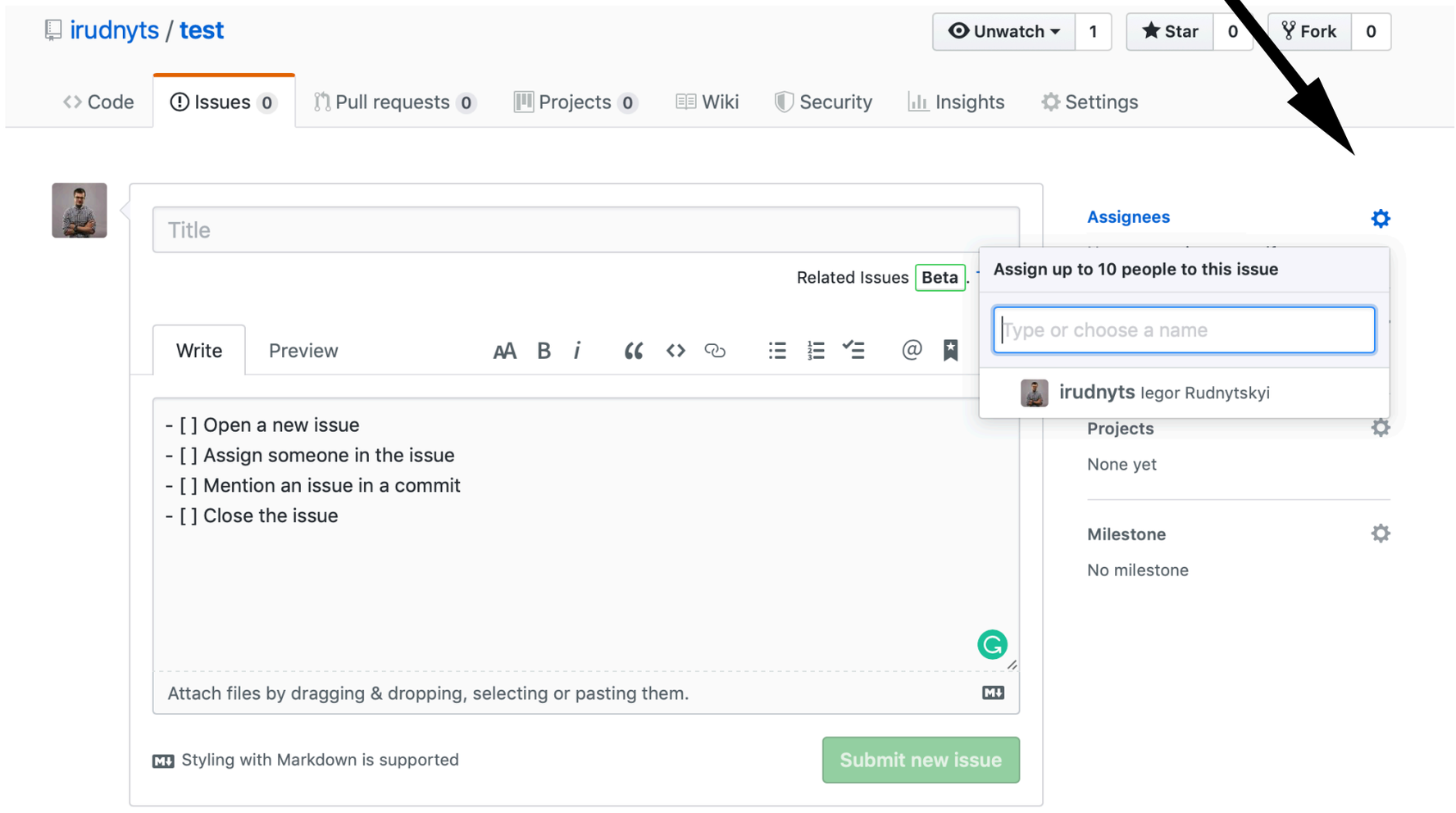
How to create a new issue?

1. Go to the web-page of your repo (i.e., https://github.com/username/reponame)
2. Open tab **Issues** click on **New issue**
3. Fill the body of the issue
4. One can use a check items list



How assign the issue to a collaborator?

1. Click on the gear next to the **Assignees**
2. Start typing the username of the collaborator and click on the their icon



Where to find the number of the issue?

Problem 1 #1

Edit New issue

Open irudnyts opened this issue 44 seconds ago · 0 comments

irudnyts commented 44 seconds ago Owner

- [ ] Open a new issue
- [ ] Assign someone in the issue
- [ ] Mention an issue in a commit
- [ ] Close the issue

irudnyts self-assigned this now

Write Preview

Leave a comment

Assignees

irudnyts

Labels

None yet

Projects

None yet

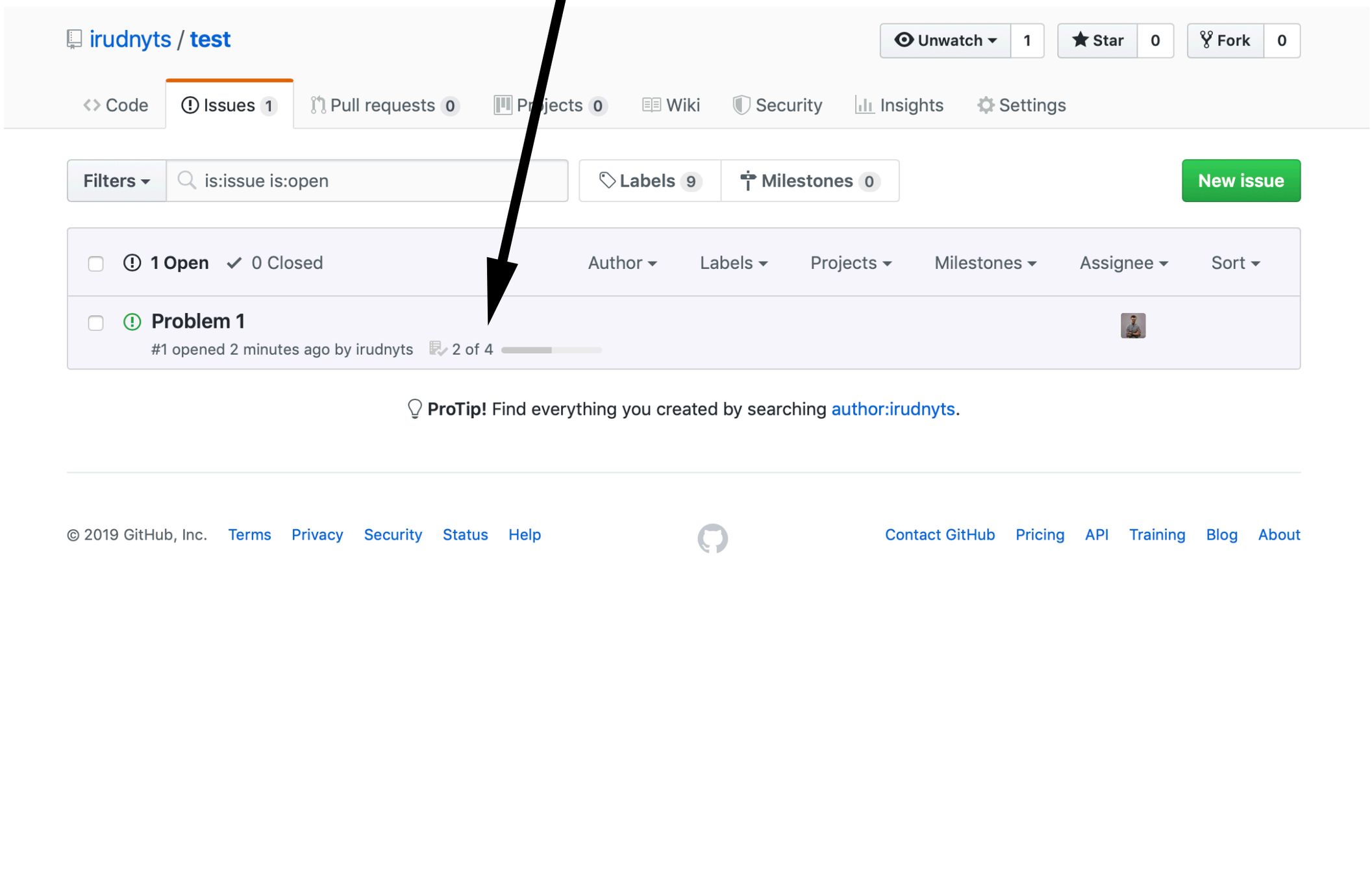
Milestone

No milestone

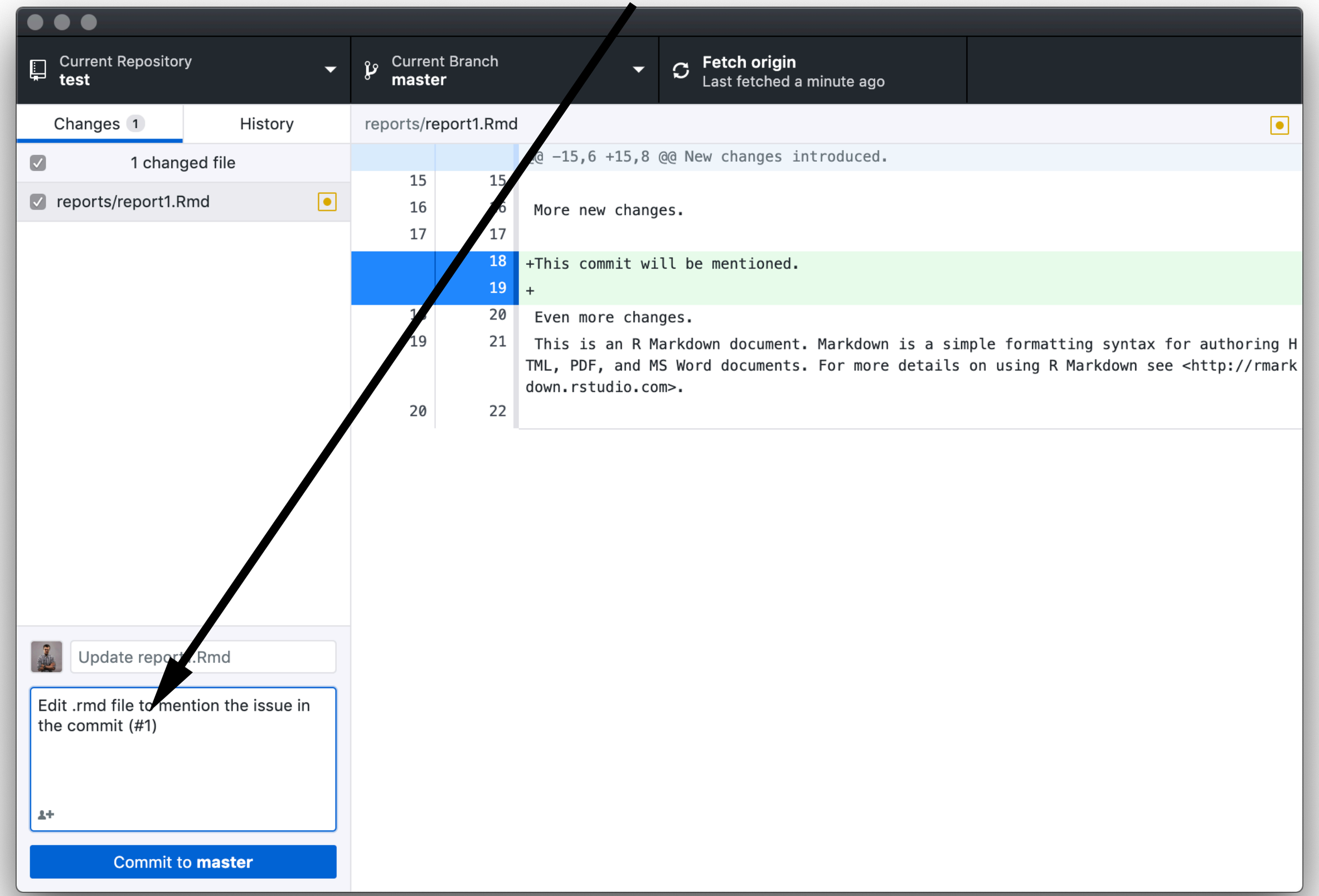
Notifications Customize

Unsubscribe

Where to find progress/summary of the issue?



How to mention the issue in a commit?



How to close the issue?

1. Click **Close issue**