Collaborative distributed version control

We have learned how to work with a Git repository used by a single person. What about sharing and collaborating?

- Share the folder using email or using some file sharing service: This would lead to many back and forth emails and would be difficult to keep all copies synchronized.
- One person's repository on the web: Allows one person to keep track of more projects, gain visibility, feedback, and recognition.
- **Common repository for a group**: Everyone can directly update the same repository. Good for small groups.
- Forks or copies with different owners: Anyone can suggest changes, even without advance permission. Maintainers approve what they agree with.

Being able to share more easily (going down the above list) is *transformative* (easier to change something, that is you are not the sole owner) because it allows projects to scale to a new level.

We will **discover and exercise the most typical workflows when collaborating** using services like GitHub.

Prerequisites

1. Basic understanding of Git.
2. You need a GitHub account.

We will do this exercise on GitHub but also GitLab and Bitbucket allow similar workflows and basically everything that we will discuss is transferable. With this material and these exercises we do not endorse the company GitHub. We have chosen to demonstrate a number of concepts using examples with GitHub because it is currently the most popular web platform for hosting Git repositories and the chance is high that you will interact with GitHub-based repositories even if you choose to host your Git repository on another platform.

Concepts around collaboration

Objectives

- Be able to decide whether to divide work at the branch level or at the repository level.

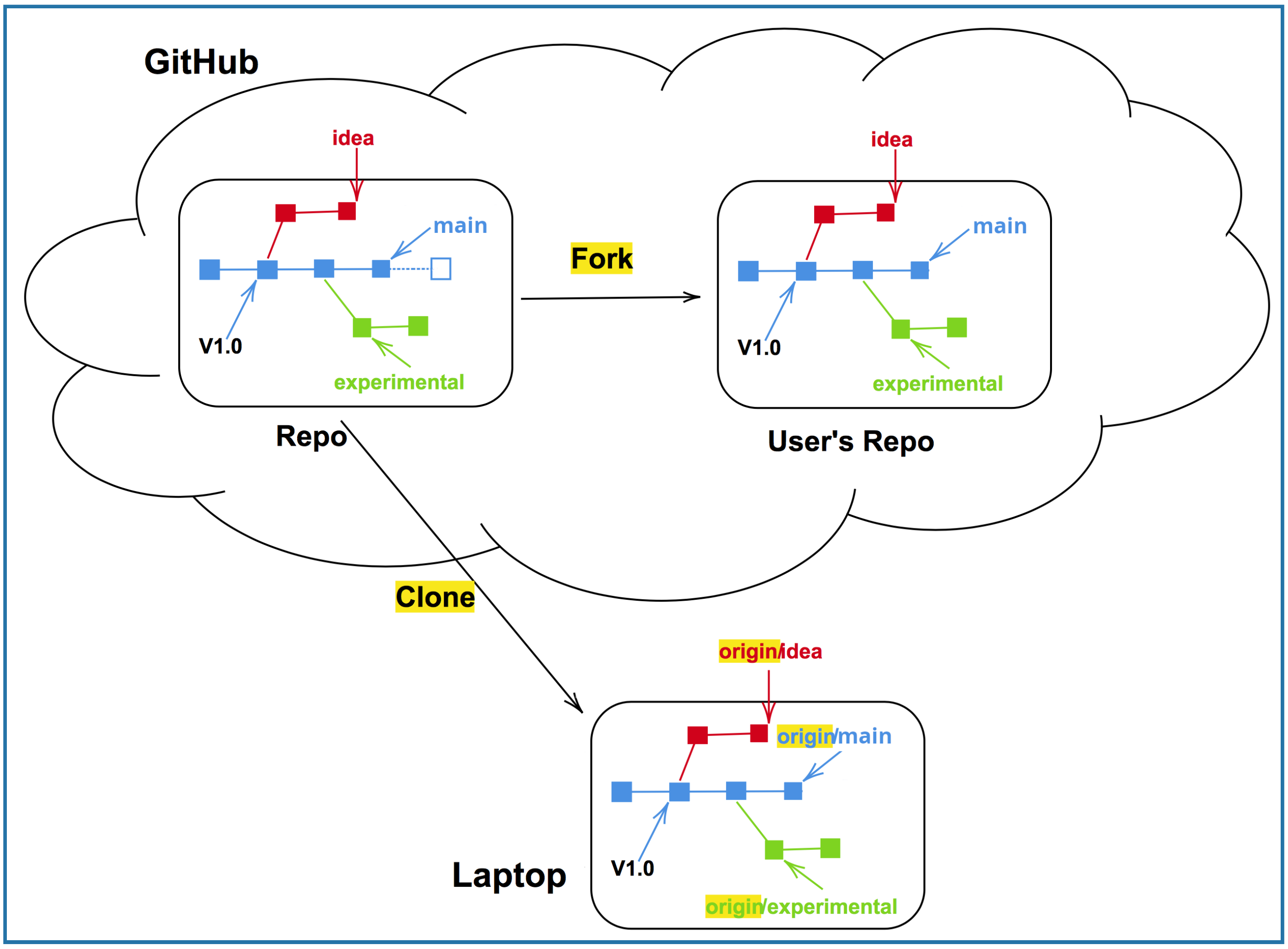
Commits, branches, repositories, forks, clones

- **repository**: The project, contains all data and history (commits, branches, tags).
- **commit**: Snapshot of the project, gets a unique identifier (e.g. `c7f0e8bfc718be04525847fc7ac237f470add76e`).
- **branch**: Independent development line. The main development line is often called `main`. Technically, a branch in Git is implemented as a pointer to a commit (imagine a sticky note with a branch name on it).
- **tag**: A pointer to one commit, to be able to refer to it later. Like a commemorative plaque that you attach to a particular commit (e.g. `phd-printed` or `paper-submitted`).
- **cloning**: Copying the whole repository to your laptop - the first time. It is not necessary to download each file one by one.
- **forking**: Taking a copy of a repository (which is typically not yours) - your copy (fork) stays on GitHub/GitLab and you can make changes to your copy.

Cloning a repository

In order to make a complete copy a whole repository, the `git clone` command can be used. When cloning, all the files, of all or selected branches, of a repository are copied in one operation. Cloning of a repository is of relevance in a few different situations:

- Working on your own, cloning is the operation that you can use to create multiple instances of a repository on, for instance, a personal computer, a server, and a supercomputer.
- The parent repository could be a repository that you or your colleague own. A common use case for cloning is when working together within a smaller team where everyone has read and write access to the same Git repository.
- Alternatively, cloning can be made from a public repository of a code that you would like to use. Perhaps you have no intention to change the code, but would like to stay in tune with the latest developments, also in-between releases of new versions of the code.



Forking and cloning

Forking a repository

When a fork is made on GitHub/GitLab a complete copy, of all or selected branches, of the repository is made. The copy will reside under a different account on GitHub/GitLab. Forking of a repository is of high relevance when working with a Git repository to which you do not have write access.

- In the fork repository commits can be made to the default branch (`main` or `master`), and to other branches.
- The commits that are made within the branches of the fork repository can be contributed back to the parent repository by means of **pull or merge requests**.

Synchronizing changes between repositories

- Repositories that are forked or cloned **do not automatically synchronize themselves**.
- We need a mechanism to communicate changes between the repositories.
- We will **pull** or **fetch** updates **from** remote repositories (we will soon discuss the difference between pull and fetch).
- We will **push** updates **to** remote repositories.
- We will learn how to suggest changes within repositories on GitHub and across repositories (**pull requests**).
- A main difference between cloning a repository and forking a repository is that the former is a general operation for generating copies of a repository to different computers, whereas forking is a particular operation implemented on GitHub/GitLab.

Should we clone or fork?

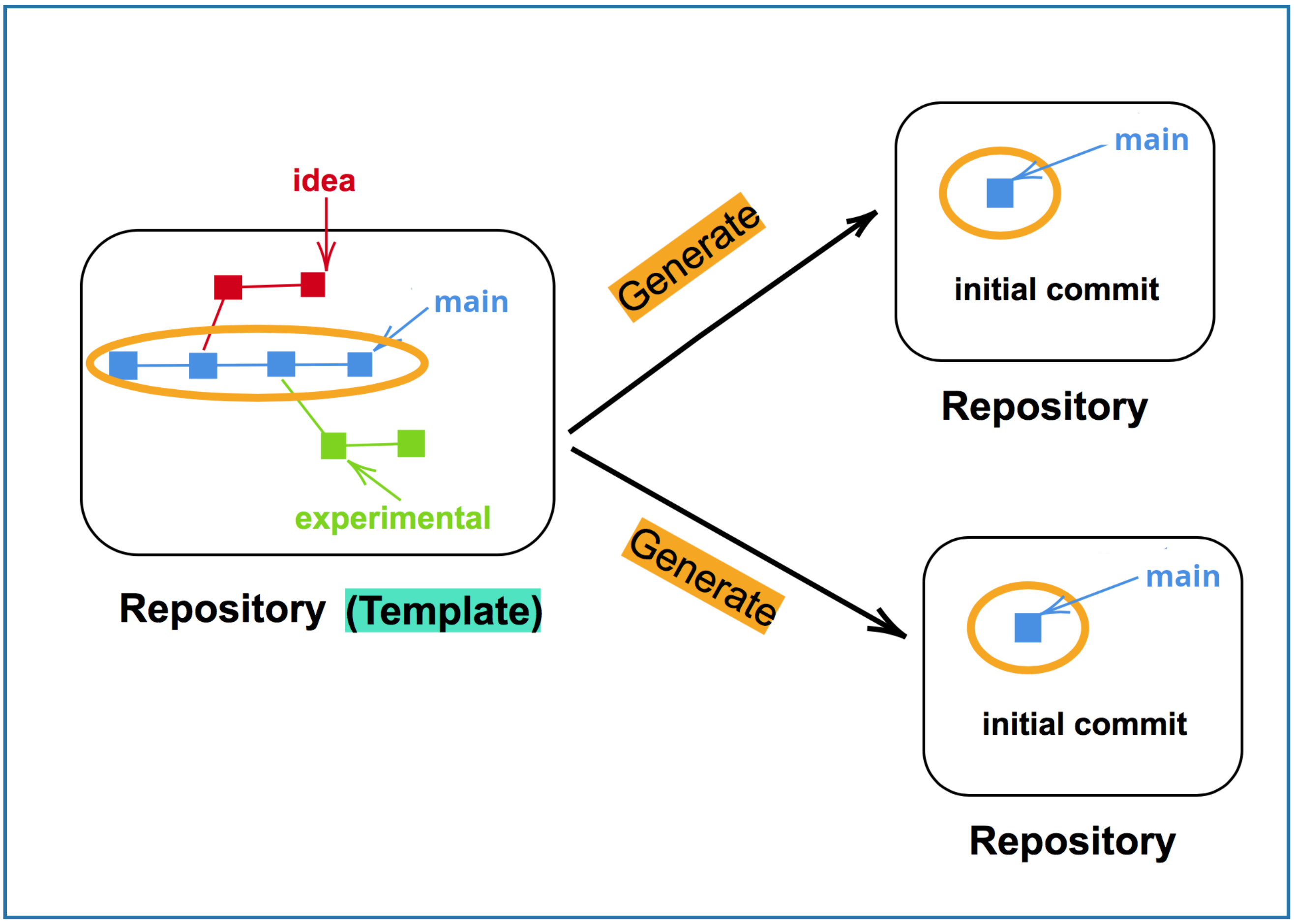
Should we all work inside one repository and clone? Or should we all work on forks?

- For most small or medium-sized research projects you probably want to **work by cloning a repository** that all research group has write access to (we will exercise this in the next section).
- It's only when you want to contribute to a project that you don't have write access to that you need to use **forks** (we will practice this as well).

Generating from templates

A repository can be marked as **template** and new repositories can be **generated** from it, like using a cookie-cutter.

The newly created repository will start with a new ("flattened") history, only one commit, and not inherit the history of the template.



Generating from a template.

Collaborating within the same repository

In this episode, we will learn how to collaborate within the same repository. We will learn how to cross-reference issues and pull requests, how to review pull requests, and how to use draft pull requests.

This exercise will form a good basis for collaboration that is suitable for most research groups.

❗ Note

When you read or hear **pull request**, please think of a **change proposal**.

Exercise

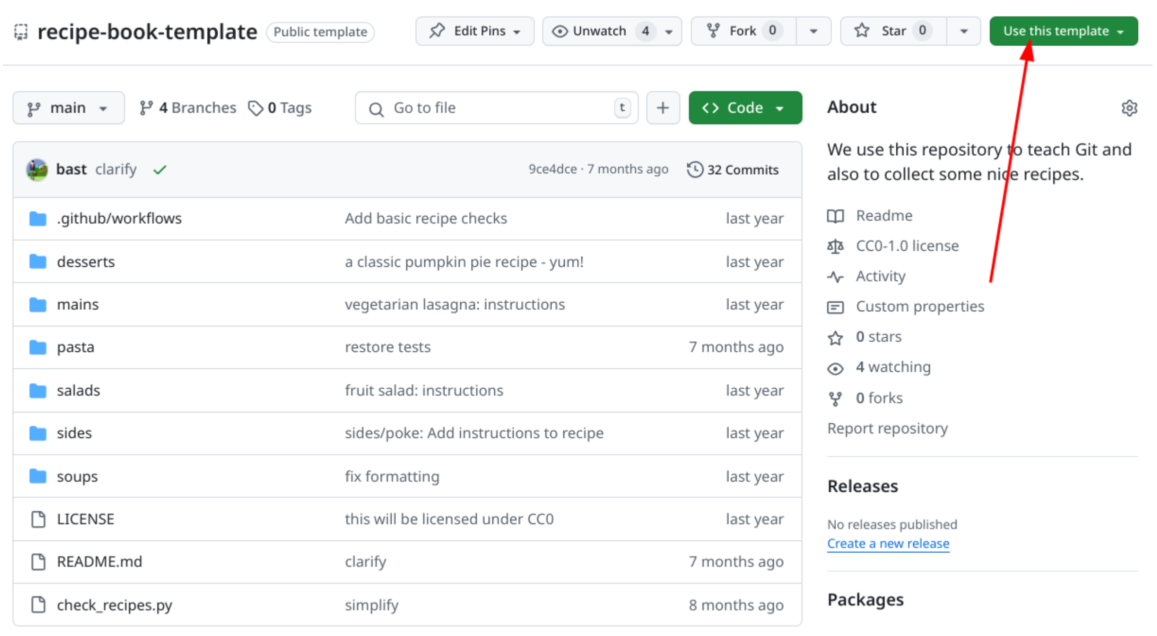
In this exercise, we will contribute to a repository via a **pull request**. This means that you propose some change, and then it can be discussed and accepted (sometimes after requesting further improvements).

⚙ Exercise preparation

First, we need to get access to some repository to which we will contribute.

Part of team/exercise room | Following on your own

- Form not too large groups (4-5 persons).
- Each group needs to appoint someone who will host the shared GitHub repository: the *maintainer*. This is typically the exercise lead (if available). Everyone else is a *collaborator*.
- The **maintainer** (one person per group) generates a new repository called `centralized-workflow-exercise` from the template https://github.com/coderefinery/recipe-book-template (There is no need to tick "*Include all branches*" for this exercise):



We use this repository to teach Git and also to collect some nice recipes.

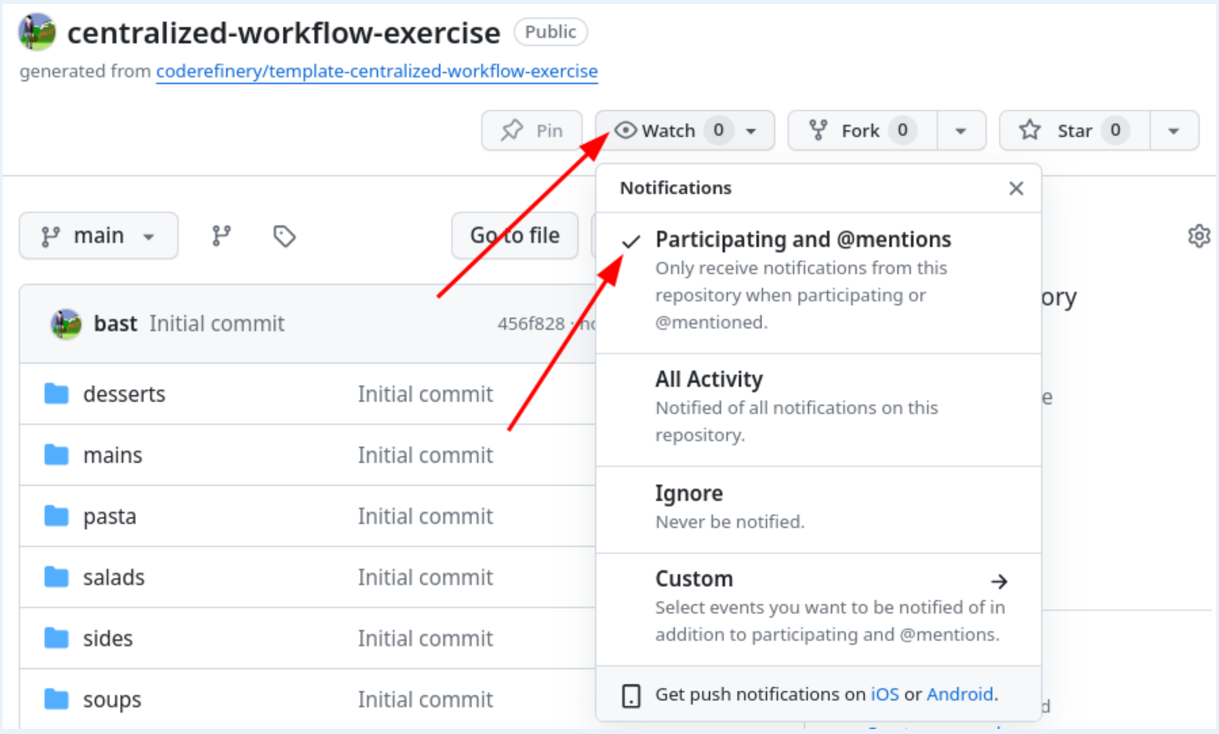
bast clarify ✓ 9ce4dce · 7 months ago 32 Commits

.github/workflows	Add basic recipe checks	last year
desserts	a classic pumpkin pie recipe - yum!	last year
mains	vegetarian lasagna: instructions	last year
pasta	restore tests	7 months ago
salads	fruit salad: instructions	last year
sides	sides/poke: Add instructions to recipe	last year
soups	fix formatting	last year
LICENSE	this will be licensed under CC0	last year
README.md	clarify	7 months ago
check_recipes.py	simplify	8 months ago

- Then **everyone in your group** needs their GitHub account to be added as collaborator to the exercise repository:

- Collaborators give their GitHub usernames to their chosen maintainer.
- Maintainer gives the other group members the newly created GitHub repository URL.
- Maintainer adds participants as collaborators to their project (Settings -> Collaborators -> Manage access -> Add people).

- **Don't forget to accept the invitation**
  - Check https://github.com/settings/organizations/
  - Alternatively check the inbox for the email account you registered with GitHub. GitHub emails you an invitation link, but if you don't receive it you can go to your GitHub notifications in the top right corner. The maintainer can also "copy invite link" and share it within the group.
- **Watching and unwatching repositories**
  - Now that you are a collaborator, you get notified about new issues and pull requests via email.
  - If you do not wish this, you can "unwatch" a repository (top of the project page).
  - However, we recommend watching repositories you are interested in. You can learn things from experts just by watching the activity that come through a popular project.


centralized-workflow-exercise
Public
generated from coderefinery/template-centralized-workflow-exercise
Pin
Watch 0
Fork 0
Star 0
main
Go to file
bast Initial commit
456f828
desserts
mains
pasta
salads
sides
soups
Initial commit
Notifications
Participating and @mentions
Only receive notifications from this repository when participating or @mentioned.
All Activity
Notified of all notifications on this repository.
Ignore
Never be notified.
Custom
Select events you want to be notified of in addition to participating and @mentions.
Get push notifications on iOS or Android.

Unwatch a repository by clicking "Watch" in the repository view, then "Participating and @mentions" - this way, you will get notifications about your own interactions.

✍️ Exercise: Collaborating within the same repository (25 min)

Technical requirements (from installation instructions):

- If you create the commits locally: Being able to authenticate to GitHub

What is familiar from the previous workshop days (not repeated here):

- Cloning a repository (previous lesson)
- Creating a branch (previous lesson)
- Committing a change on the new branch (previous lesson)
- Submit a pull request towards the main branch (previous lesson)

What will be new in this exercise:

- If you create the changes locally, you will need to **push** them to the remote repository.
- Learning what a protected branch is and how to modify a protected branch: using a pull request.
- Cross-referencing issues and pull requests.
- Practice to review a pull request.
- Learn about the value of draft pull requests.

Exercise tasks:

1. In the GitHub repository, open an issue where you describe the change you want to make. Note down the issue number since you will need it later.
2. If you work locally and not on GitHub, you need to clone the repository to your computer before you can create a new branch (more details in “Solution and hints” below).
3. Create a new branch.
4. Make a change to the recipe book on the new branch and in the commit cross-reference the issue you opened (see the walk-through below for how to do that).
5. If you work locally, push your new branch (with the new commit) to the repository you are working on.
6. Open a pull request towards the main branch.
7. Review somebody else’s pull request and give constructive feedback. Merge their pull request.
8. Try to create a new branch with some half-finished work and open a draft pull request. Verify that the draft pull request cannot be merged since it is not meant to be merged yet.

Solution and hints

(1) Opening an issue

This is done through the GitHub web interface. For example, you could give the name of the recipe you want to add (so that others don’t add the same one). It is the “Issues” tab.

(2) Create a new branch.

If on GitHub, you can make the branch in the web interface (Recording changes). If working locally, you need to follow Cloning a Git repository and working locally.

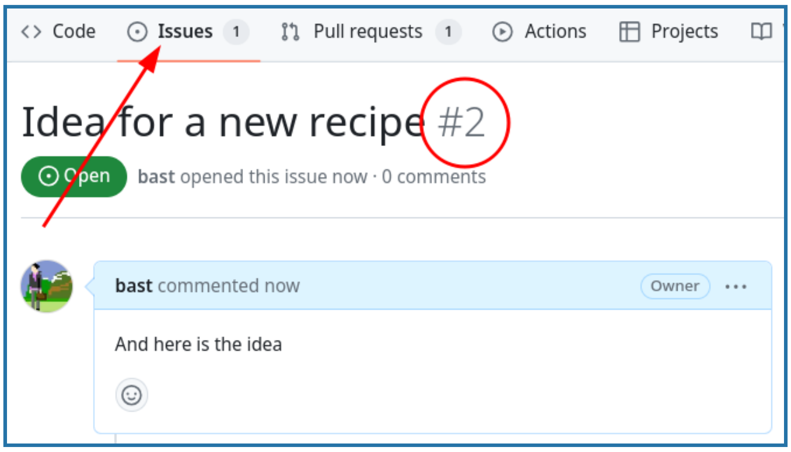
(3) Make a change adding the recipe

Add a new file with the recipe in it. Commit the file. In the commit message, include the note about the issue number, saying that this will close that issue (right below here we show how).

Cross-referencing issues and pull requests

Each issue and each pull request gets a number and you can cross-reference them.

When you open an issue, note down the issue number (in this case it is `#2`):



Idea for a new recipe #2

bast opened this issue now · 0 comments

bast commented now

And here is the idea

You can reference this issue number in a commit message or in a pull request, like in this commit message:

```
this is the new recipe; fixes #2
```

If you forget to do that in your commit message, you can also reference the issue in the pull request description. And instead of `fixes` you can also use `closes` or `resolves` or `fix` or `close` or `resolve` (case insensitive).

Here are all the keywords that GitHub recognizes:
https://help.github.com/en/articles/closing-issues-using-keywords

Then observe what happens in the issue once your commit gets merged: it will automatically close the issue and create a link between the issue and the commit. This is very useful for tracking what changes were made in response to which issue and to know from when **until when precisely** the issue was open.

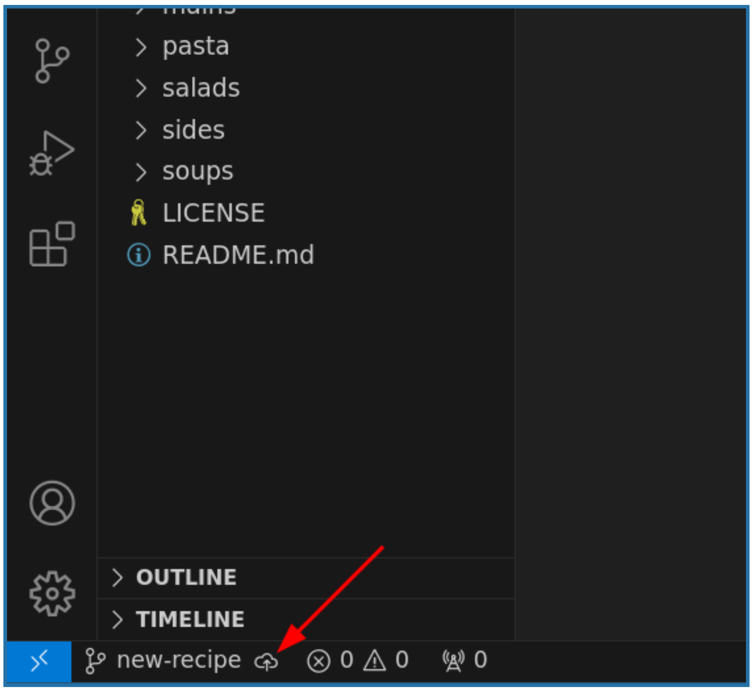
(4) Push to GitHub as a new branch

This is only necessary if you created the changes locally. If you created the changes directly on GitHub, you can skip this step.

Push the branch to the repository. You should end up with a branch visible in the GitHub web view (if you are unsure how, see Cloning a Git repository and working locally).

VS Code | Command line

In VS Code, you can "publish the branch" to the remote repository by clicking the cloud icon in the bottom left corner of the window:



(5) Open a pull request towards the main branch

This is done through the GitHub web interface. We saw this in a previous lesson.

(6) Reviewing pull requests

You review through the GitHub web interface.

Checklist for reviewing a pull request:

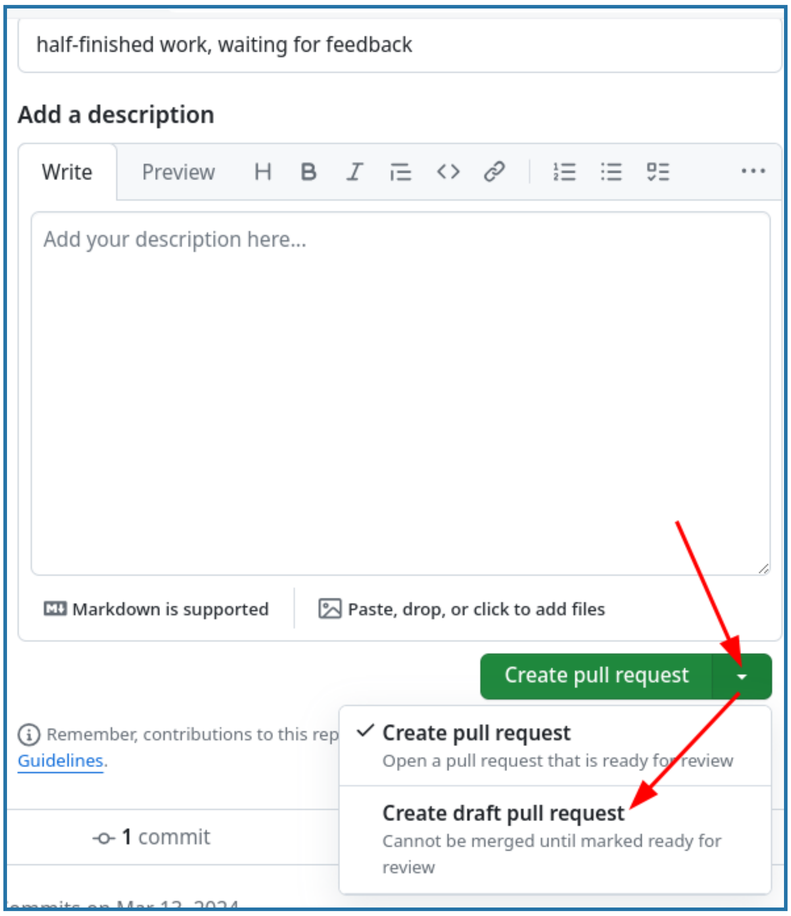
- Be kind, on the other side is a human who has put effort into this.
- Be constructive: if you see a problem, suggest a solution.
- Towards which branch is this directed?
- Is the title descriptive?

- Is the description informative?
- Scroll down to see commits.
- Scroll down to see the changes.
- If you get incredibly many changes, also consider the license or copyright and ask where all that code is coming from.
- Again, be kind and constructive.
- Later we will learn how to suggest changes directly in the pull request.

If someone is new, it's often nice to say something encouraging in the comments before merging (even if it's just "thanks"). If all is good and there's not much else to say, you could merge directly.

(7) Draft pull requests

Try to create a draft pull request:



half-finished work, waiting for feedback

Add a description

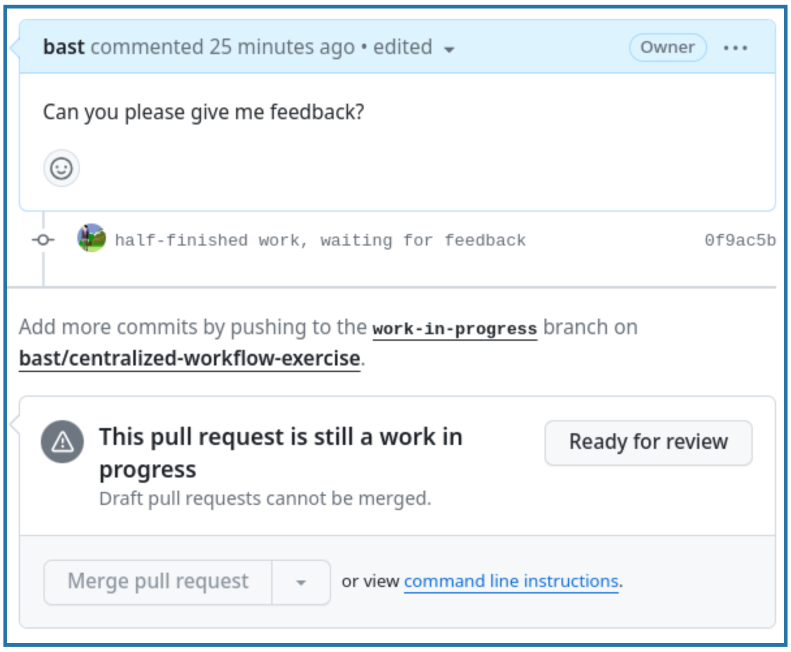
Add your description here...

Remember, contributions to this rep Guidelines.

1 commit

Write
Preview
Markdown is supported
Paste, drop, or click to add files
Create pull request
Create pull request
Open a pull request that is ready for review
Create draft pull request
Cannot be merged until marked ready for review

Verify that the draft pull request cannot be merged until it is marked as ready for review:


bast commented 25 minutes ago • edited

Owner

Can you please give me feedback?

half-finished work, waiting for feedback 0f9ac5b

Add more commits by pushing to the work-in-progress branch on bast/centralized-workflow-exercise.

This pull request is still a work in progress

Draft pull requests cannot be merged.

Draft pull requests can be useful for:

- **Feedback**: You can open a pull request early to get feedback on your work without signaling that it is ready to merge.
- **Information**: They can help communicating to others that a change is coming up and in progress.

What is a protected branch? And how to modify it?

A protected branch on GitHub or GitLab is a branch that cannot (accidentally) deleted or force-pushed to. It is also possible to require that a branch cannot be directly pushed to or modified, but that changes must be submitted via a pull request.

To protect a branch in your own repository, go to “Settings” -> “Branches”.

Summary

- We used all the same pieces that we’ve learned previously.
- But we successfully contributed to a **collaborative project**!
- The pull request allowed us to contribute without changing directly: this is very good when it’s not mainly our project.

💬 **Let’s clarify typical questions**

What is the difference between `git pull` and a pull request?

- `git pull` is a command that fetches changes from a remote repository and merges them into the current branch.
- Pull request: change proposal. It might have been named this way because after you accept a pull request, internally it git pulls the changes from the branch containing the change proposal.

What is the difference between a pull request and an issue?

- Pull request is a mechanism to suggest and review changes.
- An issue is a place where we note and discuss problems or ideas.
- Both get a number and they can reference each other but that's all they have in common.
- Linking a pull request to an issue

What is the practical difference between branch + pull and fork + pull?

- The practical difference between branch + pull and fork + pull lies in how collaboration is structured in a Git-based workflow.
- Branch + Pull (Single Repository Contribution) : Used when contributing to a repository where you have direct write access.
- Process:
  1. Clone the main repository.
  2. Create a new branch in the same repository.
  3. Make changes and commit them.
  4. Push the branch to the same repository.
  5. Open a Pull Request from your branch to the main branch.
- Fork + Pull (External Contribution): Used when contributing to a repository where you do not have write access.
- Process:
  1. Fork the repository (create your own copy under your GitHub account).
  2. Clone your forked repository locally.
  3. Create a new branch in your fork.
  4. Make changes and commit them.
  5. Push the branch to your fork.
  6. Open a Pull Request from your fork to the original repository.

Practicing code review

In this episode we will practice the **code review process**. We will learn how to ask for changes in a pull request, how to suggest a change in a pull request, and how to modify a pull request.

This will enable research groups to work more collaboratively and to not only improve the code quality but also to **learn from each other**.

Exercise

⚙ Exercise preparation

We can continue in the same exercise repository which we have used in the previous episode.

✍️ **Exercise: Practicing code review (25 min)**

Technical requirements:

- If you create the commits locally: Being able to authenticate to GitHub

What is familiar from previous episodes:

- Creating a branch (previous lesson)
- Committing a change on the new branch (previous lesson)
- Opening and merging pull requests (previous lesson)

What will be new in this exercise:

- As a reviewer, we will learn how to ask for changes in a pull request.
- As a reviewer, we will learn how to suggest a change in a pull request.
- As a submitter, we will learn how to modify a pull request without closing the incomplete one and opening a new one.

Exercise tasks:

1. Create a new branch and one or few commits: in these improve something but also deliberately introduce a typo and also a larger mistake which we will want to fix during the code review.
2. Open a pull request towards the main branch.
3. As a reviewer to somebody else's pull request, ask for an improvement and also directly suggest a change for the small typo. (Hint: suggestions are possible through the GitHub web interface, view of a pull request, "Files changed" view, after selecting some lines. Look for the "±" button.)
4. As the submitter, learn how to accept the suggested change. (Hint: GitHub web interface, "Files Changed" view.)
5. As the submitter, improve the pull request without having to close and open a new one: by adding a new commit to the same branch. (Hint: push to the branch again.)
6. Once the changes are addressed, merge the pull request.

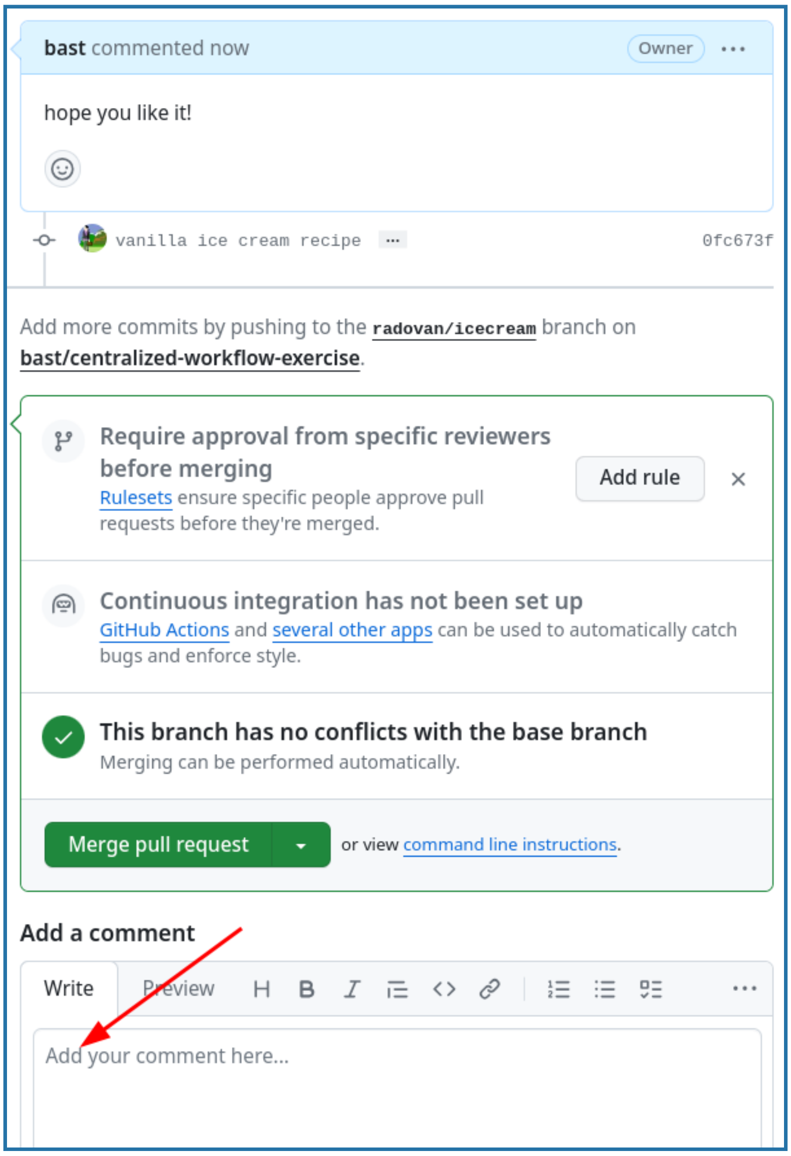
Help and discussion

From here on out, we don't give detailed steps to the solution. You need to combine what you know, and the extra info below, in order to solve the above.

How to ask for changes in a pull request

Technically, there are at least two common ways to ask for changes in a pull request.

Either in the comment field of the pull request:


bast commented now Owner

hope you like it!

vanilla ice cream recipe 0fc673f

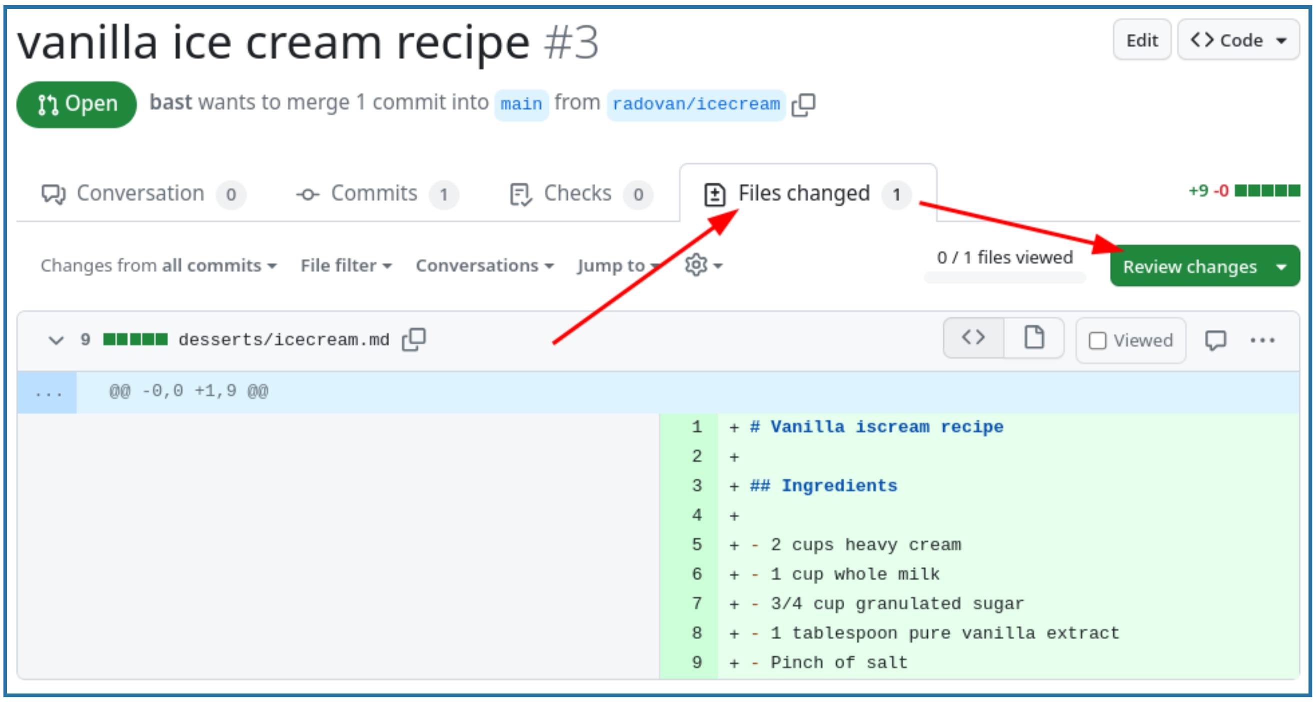
Add more commits by pushing to the **radovan/icecream** branch on **bast/centralized-workflow-exercise**.

Require approval from specific reviewers before merging
Rulesets ensure specific people approve pull requests before they're merged.

Continuous integration has not been set up
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch
Merging can be performed automatically.

Or by using the “Review changes”:


vanilla ice cream recipe #3

Open bast wants to merge 1 commit into `main` from `radovan/icecream`

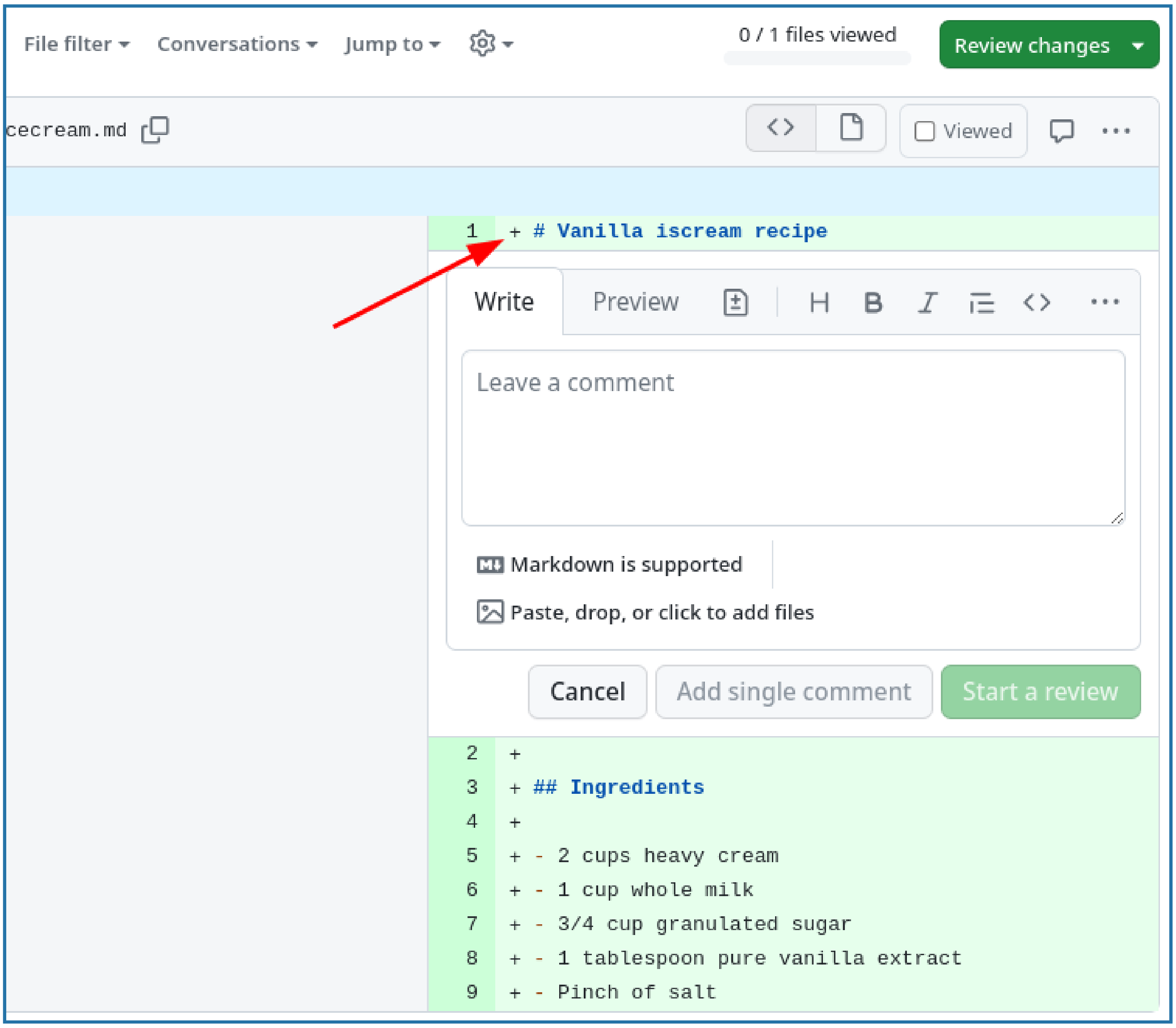
```
@@ -0,0 +1,9 @@
+ # Vanilla iscream recipe
+
+ ## Ingredients
+
+ - 2 cups heavy cream
+ - 1 cup whole milk
+ - 3/4 cup granulated sugar
+ - 1 tablespoon pure vanilla extract
+ - Pinch of salt
```

And always please be kind and constructive in your comments. Remember that the goal is not gate-keeping but **collaborative learning**.

How to suggest a change in a pull request as a reviewer

If you see a very small problem that is easy to fix, you can suggest a change as a reviewer.

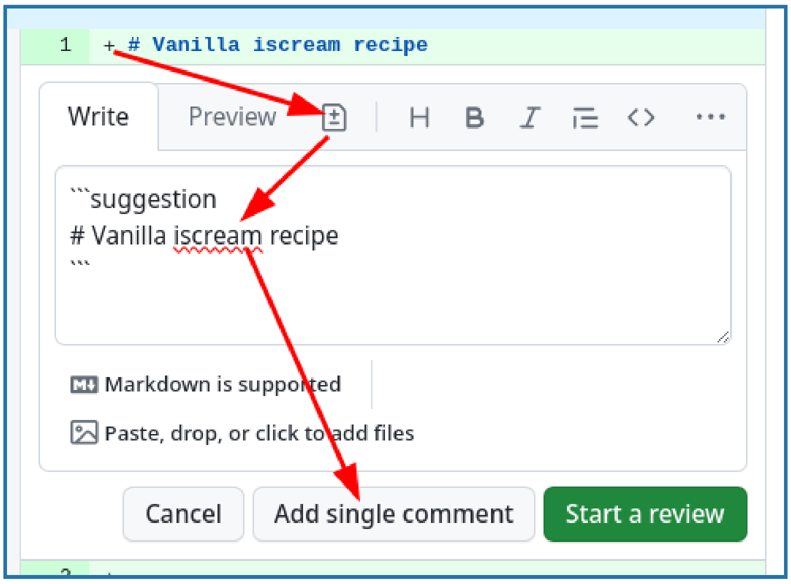
Instead of asking the submitter to fix the tiny problem, you can suggest a change by clicking on the plus sign next to the line number in the “Files changed” tab:



```
+ # Vanilla iscream recipe
+
+ ## Ingredients
+
+ - 2 cups heavy cream
+ - 1 cup whole milk
+ - 3/4 cup granulated sugar
+ - 1 tablespoon pure vanilla extract
+ - Pinch of salt
```

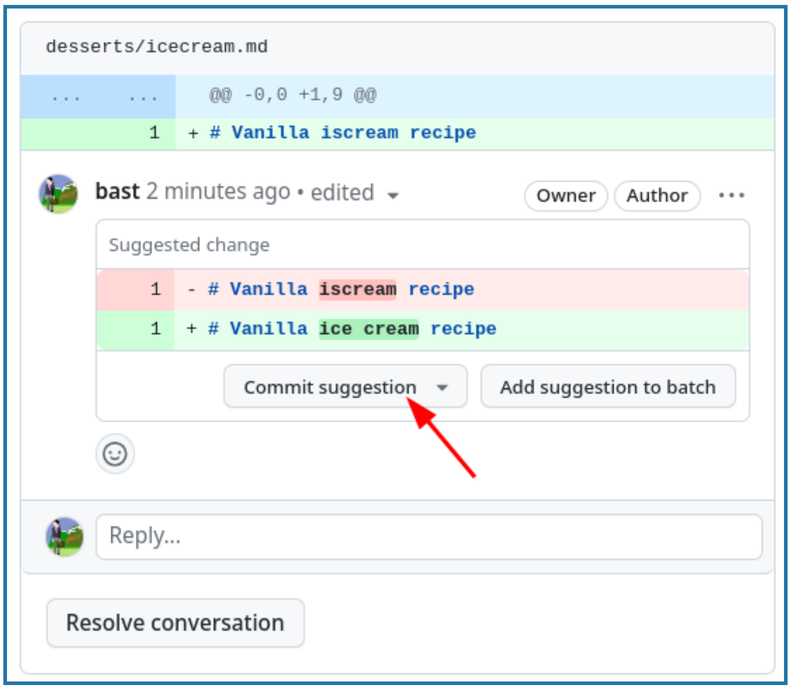
Here you can comment on specific lines or even line ranges.

But now the interesting part is to click on the “Add a suggestion” symbol (the one that looks like plus and minus). Now you can fix the tiny problem (in this case a typo) and then click on the “Add single comment” button:


```
1  + # Vanilla iscream recipe
```

~~~
```suggestion
# Vanilla iscream recipe
```
~~~

The result is this and the submitter can accept the change with a single click:


```
desserts/icecream.md
...   ...   @@ -0,0 +1,9 @@
         1  + # Vanilla iscream recipe
```

```
1  - # Vanilla iscream recipe
1  + # Vanilla ice cream recipe
```

After accepting with “Commit suggestion”, the improvement gets added to the pull request as a new commit.

How to modify a pull request to address the review comments

If the reviewer asks for changes, it is not necessary to close the pull request and later open a new one. It can even be counter-productive to do so: This can fragment the discussion and the history of the pull request and can make it harder to understand the context of the changes.

A much better mechanism is to recognize that pull requests are not implemented from a specific commit to a specific branch, but **always from a branch to a branch**.

This means that you can make amendments to the pull request by adding new commits to the same source branch. This way the pull request will be updated automatically and the reviewer can see the new changes and comment on them.

The fact that pull requests are from branch to branch also strongly suggests that it is a good practice to **create a new branch for each pull request**. Otherwise you could accidentally modify an open pull request by adding new commits to the source branch.

Summary

- Our process isn't just about code now. It's about discussion and working together to make the whole process better.
- GitHub (or GitLab) discussions and reviewing are quite powerful and can make small changes easy.

How to contribute changes to repositories that belong to others

In this episode we prepare you to suggest and contribute changes to repositories that belong to others. These might be open source projects that you use in your work.

We will see how Git and services like GitHub or GitLab can be used to suggest modification without having to ask for write access to the repository and accept modifications without having to grant write access to others.

Exercise

⚙ Exercise preparation

Part of team/exercise room | Following on your own

Maintainer (team lead):

- Create an exercise repository called `forking-workflow-exercise` by generating from a template using this template: https://github.com/coderefinery/recipe-book-template.
- In this case we **do not add collaborators** to the repository (this is the point of this example).
- Share the link to the newly created repository with your group.

Learners in exercise team: Fork the newly created repository (not the "coderefinery" one) and then **clone your fork** (if you wish to work locally).

✍ Exercise: Collaborating within the same repository (25 min)

Technical requirements:

- If you create the commits locally: Being able to authenticate to GitHub

What is familiar from the previous episodes:

- Forking a repository (previous lesson)
- Creating a branch (previous lesson)
- Committing a change on the new branch (previous lesson)
- Opening and merging pull requests (previous lesson)

What will be new in this exercise:

- Opening a pull request towards the upstream repository.
- Pull requests can be coupled with automated testing.
- Learning that your fork can get out of date.
- After the pull requests are merged, updating your fork with the changes.
- Learn how to approach other people's repositories with ideas, changes, and requests.

Exercise tasks:

1. Open an issue in the upstream exercise repository where you describe the change you want to make. Take note of the issue number.
2. Create a new branch in your fork of the repository.
3. Make a change to the recipe book on the new branch and in the commit cross-reference the issue you opened. See the walk-through below for how to do this.
4. Open a pull request towards the upstream repository.
5. Team leaders will merge the pull requests. For individual participants, the instructors and workshop organizers will review and merge the pull requests. During the review, pay attention to the automated test step (here for demonstration purposes, we test whether the recipe contains an ingredients and an instructions sections).
6. After few pull requests are merged, update your fork with the changes.
7. Check that in your fork you can see changes from other people's pull requests.

Help and discussion

Help! I don't have permissions to push my local changes

Maybe you see an error like this one:

```
Please make sure you have the correct access rights
and the repository exists.
```

Or like this one:

```
failed to push some refs to cr-workshop-exercises/forking-workflow-exercise.git
```

In this case you probably try to push the changes not to your fork but to the original repository and in this exercise you do not have write access to the original repository.

The simpler solution is to clone again but this time your fork.

✔ Recovery

But if you want to keep your local changes, you can change the remote URL to point to your fork. Check where your remote points to with `git remote --verbose`.

It should look like this (replace `USER` with your GitHub username):

```
$ git remote --verbose

origin  git@github.com:USER/forking-workflow-exercise.git (fetch)
origin  git@github.com:USER/forking-workflow-exercise.git (push)
```

It should **not** look like this:

```
$ git remote --verbose

origin  git@github.com:cr-workshop-exercises/forking-workflow-exercise.git (fetch)
origin  git@github.com:cr-workshop-exercises/forking-workflow-exercise.git (push)
```

In this case you can adjust “origin” to point to your fork with:

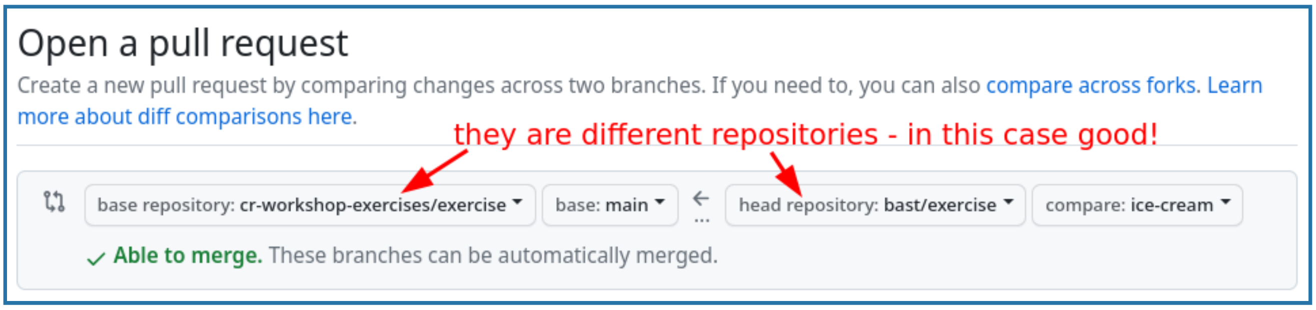
```
$ git remote set-url origin git@github.com:USER/forking-workflow-exercise.git
```

Opening a pull request towards the upstream repository

We have learned in the previous episode that pull requests are always from branch to branch. But the branch can be in a different repository.

When you open a pull request in a fork, by default GitHub will suggest to direct it towards the default branch of the upstream repository.

This can be changed and it should always be verified, but in this case this is exactly what we want to do, from fork towards upstream:


Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more about diff comparisons here.

they are different repositories - in this case good!

base repository: cr-workshop-exercises/exercise ▾ base: main ▾ ← head repository: bast/exercise ▾ compare: ice-cream ▾

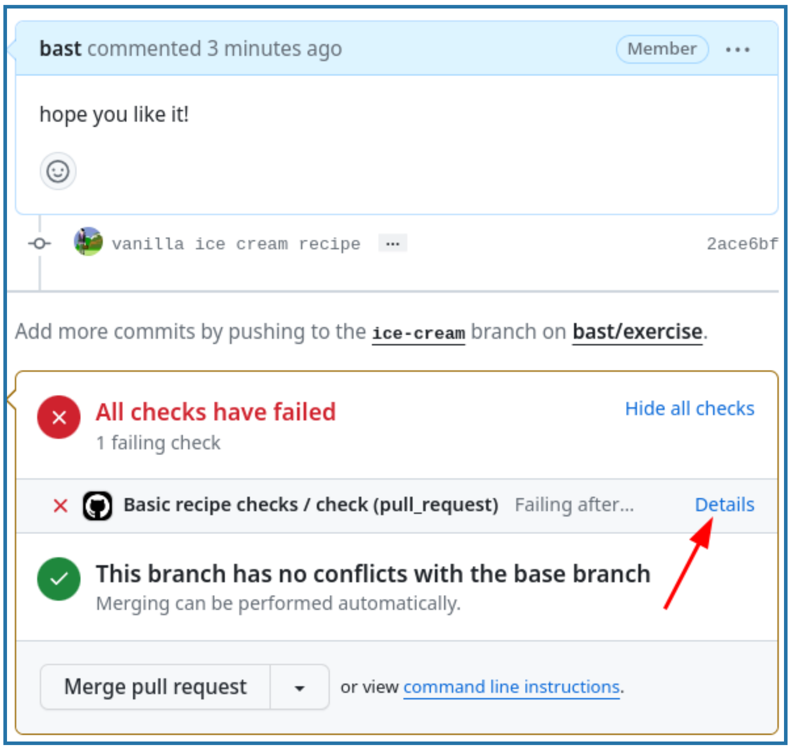
✓ Able to merge. These branches can be automatically merged.

Pull requests can be coupled with automated testing

We added an automated test here just for fun and so that you see that this is possible to do.

In this exercise, the test is silly. It will check whether the recipe contains both an ingredients and an instructions section.

In this example the test failed:


bast commented 3 minutes ago · Member

hope you like it!

vanilla ice cream recipe ··· 2ace6bf

Add more commits by pushing to the `ice-cream` branch on bast/exercise.

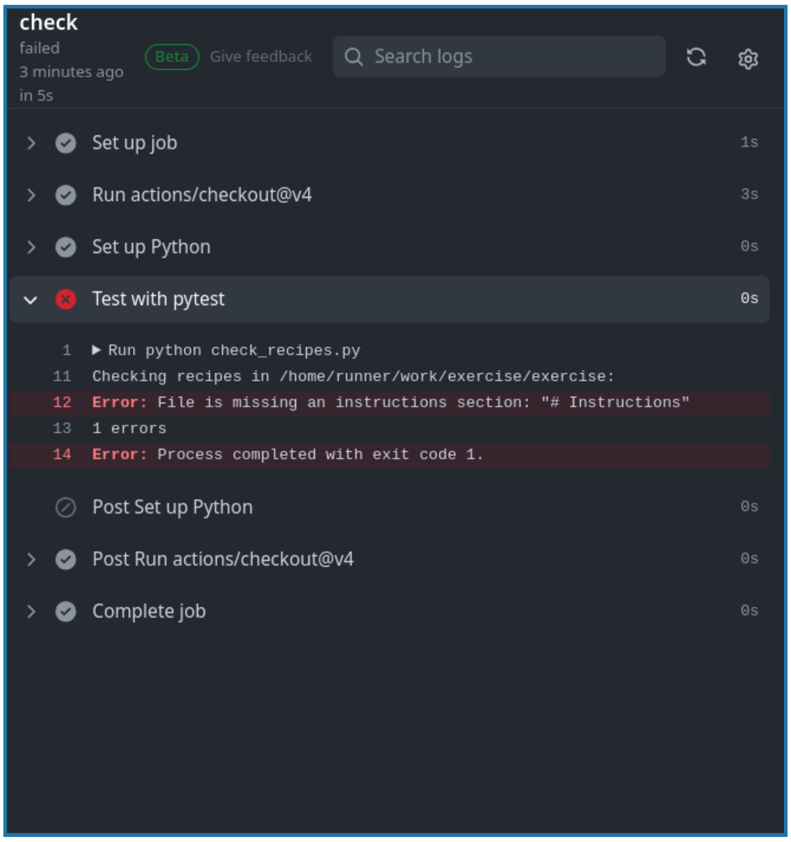
All checks have failed
1 failing check
Hide all checks

✕ Basic recipe checks / check (pull_request) Failing after... Details

✓ This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request ▾ or view command line instructions.

Click on the “Details” link to see the details of the failed test:



```
Set up job                         1s
Run actions/checkout@v4            3s
Set up Python                      0s
Test with pytest                   0s
▶ Run python check_recipes.py
Checking recipes in /home/runner/work/exercise/exercise:
Error: File is missing an instructions section: "# Instructions"
1 errors
Error: Process completed with exit code 1.
Post Set up Python                 0s
Post Run actions/checkout@v4       0s
Complete job                       0s
```

How can this be useful?

- The project can define what kind of tests are expected to pass before a pull request can be merged.
- The reviewer can see the results of the tests, without having to run them locally.

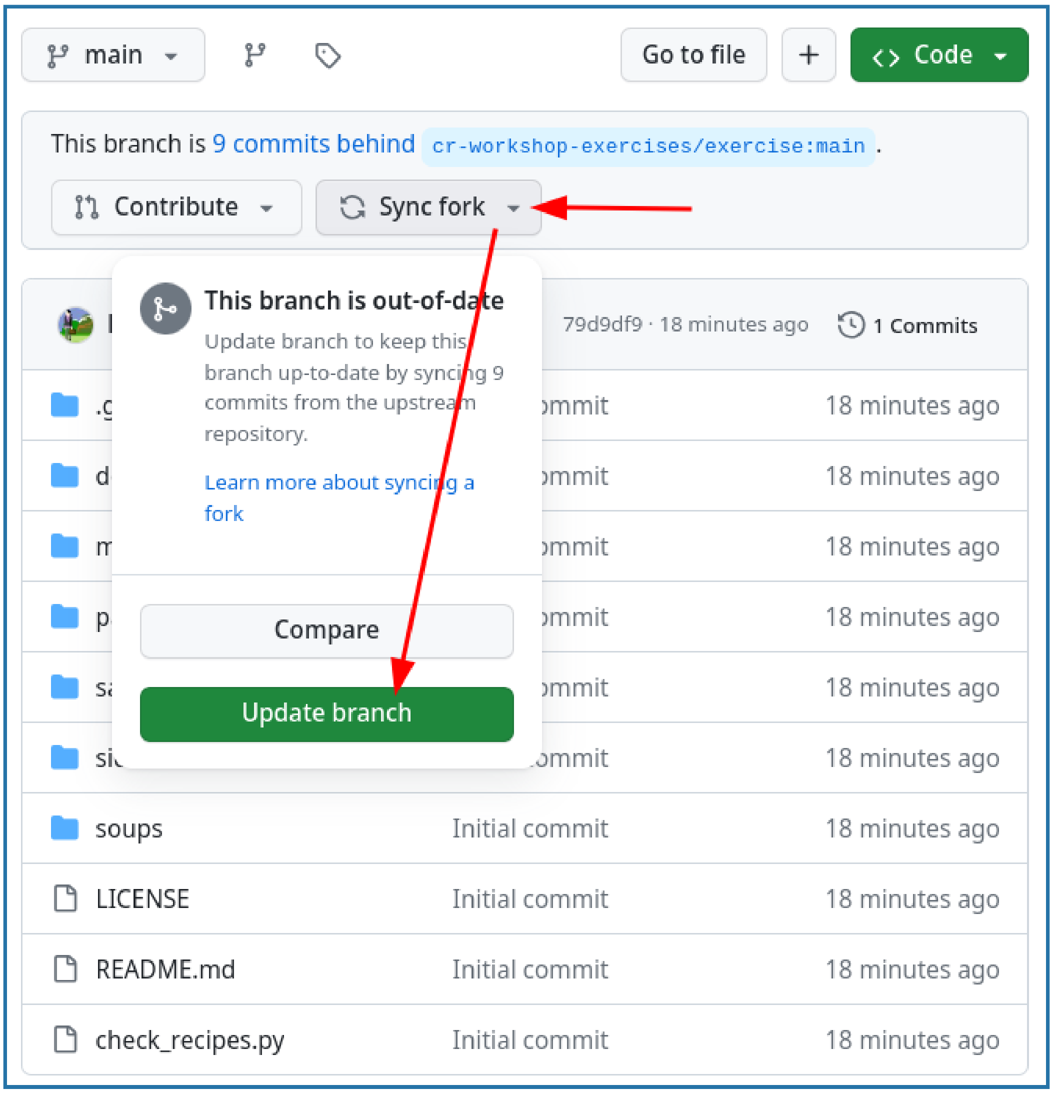
How does it work?

- We added a GitHub Actions workflow to automatically run on each push or pull request towards the `main` branch.

What tests or steps can you image for your project to run automatically with each pull request?

How to update your fork with changes from upstream

This used to be difficult but now it is two mouse clicks: Navigate to your fork and notice how GitHub tells you that your fork is behind. In my case, it is 9 commits behind upstream. To fix this, click on “Sync fork” and then “Update branch”:



This branch is 9 commits behind cr-workshop-exercises/exercise:main.

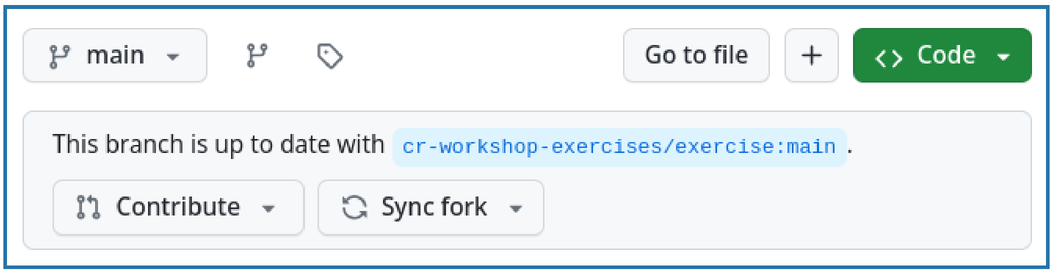
This branch is out-of-date

Update branch to keep this branch up-to-date by syncing 9 commits from the upstream repository.

79d9df9 · 18 minutes ago · 1 Commits

.g	ommit	18 minutes ago
d	ommit	18 minutes ago
m	ommit	18 minutes ago
p	ommit	18 minutes ago
sa	ommit	18 minutes ago
si	ommit	18 minutes ago
soups	Initial commit	18 minutes ago
LICENSE	Initial commit	18 minutes ago
README.md	Initial commit	18 minutes ago
check_recipes.py	Initial commit	18 minutes ago

After the update my “branch is up to date” with the upstream repository:



This branch is up to date with cr-workshop-exercises/exercise:main.

How to approach other people’s repositories with ideas, changes, and requests

Contributing very minor changes

- Clone or fork+clone repository
- Create a branch
- Commit and push change
- Open a pull request or merge request

If you observe an issue and have an idea how to fix it

- Open an issue in the repository you wish to contribute to
- Describe the problem

- If you have a suggestion on how to fix it, describe your suggestion
- Possibly **discuss and get feedback**
- If you are working on the fix, indicate it in the issue so that others know that somebody is working on it and who is working on it
- Submit your fix as pull request or merge request which references/closes the issue

Motivation

- **Inform others about an observed problem**
- Make it clear whether this issue is up for grabs or already being worked on

If you have an idea for a new feature

- Open an issue in the repository you wish to contribute to
- In the issue, write a short proposal for your suggested change or new feature
- Motivate why and how you wish to do this
- Also indicate where you are unsure and where you would like feedback
- **Discuss and get feedback before you code**
- Once you start coding, indicate that you are working on it
- Once you are done, submit your new feature as pull request or merge request which references/closes the issue/proposal

Motivation

- **Get agreement and feedback before writing 5000 lines of code** which might be rejected
- If we later wonder why something was done, we have the issue/proposal as reference and can read up on the reasoning behind a code change

Summary

- This forking workflow lets you propose changes to repositories for which **you have no write access**.
- This is the way that much modern open-source software works.
- You can now contribute to any project you can view.

Hooks

Objectives

- Learn how to couple scripts to Git repository events.

Instructor note

- 10 min teaching/demonstration

Sometimes you would like Git events (commits, pushes, etc.) to **trigger scripts** which take care of some tasks. **Hooks are scripts that are executed before/after certain Git events**. They can be used to enforce nearly any kind of policy for your project. There are client-side and server-side hooks.

Client-side hooks

You can find and edit them here:

```
$ ls -l .git/hooks/
```

- `pre-commit`: before commit message editor (example: make sure tests pass)
- `prepare-commit-msg`: before commit message editor (example: modify default messages)
- `commit-msg`: after commit message editor (example: validate commit message pattern)
- `post-commit`: after commit process (example: notification)
- `pre-rebase`: before rebase anything (example: disallow rebasing published commits)
- `post-rewrite`: run by commands that rewrite commits
- `post-checkout`: after successful `git checkout` (example: generating documentation)
- `post-merge`: after successful merge
- `pre-push`: runs during `git push` before any objects have been transferred
- `pre-auto-gc`: invoked just before the garbage collection takes place

See also https://pre-commit.com, a framework for managing and maintaining multi-language pre-commit hooks.

Example for a `pre-commit` hook which checks whether a Python code is PEP 8-compliant using pycodestyle:

```
#!/usr/bin/env bash

# ignore errors due to too long lines
pycodestyle --ignore=E501 myproject/
```

Server-side hooks

You can typically edit them through a web interface on GitHub/GitLab.

- `pre-receive`: before accepting any references
- `update`: like `pre-receive` but runs once per pushed branch
- `post-receive`: after entire process is completed
- Typical use:

    - Maintenance work
    - Automated tests
    - Refreshing of documentation/website
    - Sanity checks
    - Code style checks
    - Email notification
    - Rebuilding software packages

Actions, workflows, and continuous integration services

GitHub and GitLab let you define workflows/actions/recipes which are triggered by e.g. `git push` or by a release (tag creation). They can be customized and almost any automation you can think of becomes possible.

These services use hooks under the hood. These days, project are more likely to use these higher-level services rather than Git hooks directly.

You can read more about these services here:

- GitHub Actions
- GitLab CI

In our projects we use these services to:

- Build websites
- Build documentation
- Run tests
- Create containers
- Package and upload packages
- Spellchecking

Non-bare and bare repositories

Objectives

- Understanding the difference between non-bare and bare repositories.
- Being able to create a common repository for a group on our local computer or server.

Instructor note

- 10 min teaching/demonstration

Non-bare repository

- A **non-bare repository** contains `.git/` as well as a snapshot of your tracked files that you can directly edit called **the working tree** (the actual files you can edit).

- **This is where we edit and commit changes.**
- When we create a repository with `git init`, it is a non-bare, “normal”, repository.

Bare repository

- A **bare repository** contains only the `.git/` part, no files you can directly edit.
- By convention the names of bare repositories end with `.git` to emphasize this.
- We never do actual editing work inside a bare repository.
- GitHub, GitLab, etc. store a bare repository.
- You can also create a bare repository on your computer/server to store your private repository.

If we have enough time, the instructor demonstrates how to create a bare repository on the local computer:

✍ Bare-1: Create and use a bare repository

- Create a new local repository with `git init`.

```
$ cd /path/to/example
$ git init
```

- Populate it with a file and a commit or two.
- Create one or two branches.
- Clone this repository on the same computer with either `--bare` or `--mirror`:

```
$ git clone --bare /path/to/example /path/to/example-bare
```

- Inspect the bare repository.
- Clone the bare repository:

```
$ git clone /path/to/example-bare /path/to/example-clone
$ cd /path/to/example-clone
```

- Inside the clone inspect `git remote -v`.
- Inside the clone create a commit and push the commit to `origin`.
- The bare repository can be cloned several times and one can exercise pushing and pulling changes.

❗ Keypoints

- We do programming work inside non-bare repositories.
- We can create a local common bare repository where we can push to and pull from.

Quick reference

Other cheatsheets

See the git-intro cheatsheet for the basics.

- Interactive git cheatsheet
- Very detailed 2-page git cheatsheet

Glossary

remote

Roughly, another git repository on another computer. A repository can be linked to several other remotes.

push

Send a branch from your current repository to another repository

fetch

Update your view of another repository

pull

Fetch (above) and then merge

origin

Default name for a remote repository.

origin/NAME

A branch name which represents a remote branch.

main

Default name for main branch.

merge

Combine the changes on two branches.

conflict

When a merge has changes that affect the same lines, git can not automatically figure out what to do. It presents the conflict to the user to resolve.

issue

Feature of web repositories that allows discussion related to a repository.

pull request

A GitHub/Gitlab feature that allows you to send a code suggestion using a branch, which allows one-button merging. In Gitlab, called "**merge request**".

git hook

Code that can run before or after certain actions, for example to do tests before allowing you to commit.

bare repository

A copy of a repository that only is only the `.git` directory: there are no files actually checked out. Directory names usually like `something.git`

Commands we use

This excludes most introduced in the git-intro cheatsheet.

Setup:

- `git clone URL [TARGET-DIRECTORY]`: Make a copy of existing repository at <url>, containing all history.

Status:

- `git status`: Same as in basic git, list status
- `git remote [-v]`: List all remotes
- `git graph`: see a detailed graph of commits. Create this command with `git config --global alias.graph "log --all --graph --decorate --oneline"`

General work:

- `git switch BRANCH-NAME`: Make a branch active.
- `git push [REMOTE-NAME] [BRANCH:BRANCH]`: Send commits and update the branch on the remote.
- `git pull [REMOTE-NAME] [BRANCH-NAME]`: Fetch and then merge automatically. Can be convenient, but to be careful you can fetch and merge separately.
- `git fetch [REMOTE-NAME]`: Get commits from the remote. Doesn't update local branches, but updates the remote tracking branches (like origin/NAME).
- `git merge [BRANCH-NAME]`: Updates your current branch with changes from another branch. By default, merges to the branch is is tracking by default.
- `git remote add REMOTE-NAME URL`: Adds a new remote with a certain name.

List of exercises

Full list

This is a list of all exercises and solutions in this lesson, mainly as a reference for helpers and instructors. This list is automatically generated from all of the other pages in the lesson. Any single teaching event will probably cover only a subset of these, depending on their interests.

Instructor guide

Approximate schedule

Times here are in CE(S)T.

- 08:50 - 09:00 (10 min) Soft start and icebreaker question
- 09:00 - 09:15 (15 min) Recap Git, any HedgeDoc questions to highlight
- 09:15 - 09:25 (10 min) Concepts around collaboration
  - Explain terms: Pull, push, clone, fork. Focus on pull and not fetch.
  - Focus more on clone and less on generating from templates and importing.
- 09:25 - 10:00 (35 min) Collaborating within the same repository
  - Exercise (incl preparation)
- 10:00 - 10:10 (10 min) Break
- 10:10 - 10:30 (20 min) Collaborating within the same repository
  - Demo and Q/A
- 10:30 - 11:00 (30 min) Practicing code review
- 11:00 - 12:00 (60 min) Break
- 12:00 - 12:50 (50 min) Distributed version control and forking workflow
  - Concepts and what are exercise outcomes
  - Exercise
- 12:50 - 13:00 (10 min) Break
- 13:00 - 13:30 (30 min) Discussion, demonstration, Q&A, feedback, what to expect next week

Preparing exercises within exercise groups

Exercise leads typically prepare exercise repositories for the exercise group (although the material speaks about "maintainer" who can also be one of the learners). Preparing the first exercise (centralized workflow) **will take more time** than preparing the second (forking workflow). Most preparation time is not the generating part but will go into communicating the URL to the exercise group, communicating their usernames, adding them as collaborators, and waiting until everybody accepts the GitHub invitation to join the newly created exercise repository.

Preparing exercises for the live stream

What instructors need to do at least 1 day before the workshop

- This **takes 30-60 minutes** to set up. Allocate the time for this before the workshop.
- Make sure to remove all participants from a previous workshop from these two places:
  - https://github.com/orgs/cr-workshop-exercises/teams/stream-exercise-participants
  - https://github.com/orgs/cr-workshop-exercises/people

- We create the exercises **in an organization** (not under your username) so that you can give others admin access to add collaborators. Also this way you can then fork yourself if needed.
- All exercise repositories can be created from https://github.com/coderefinery/recipe-book-template by `git clone --mirror` from the template followed by `git push --mirror` towards the exercise repository.
- We have created two versions of each **a day in advance** to signal which one might end up being discussed on recording/stream:
  - `centralized-workflow-exercise-recorded`
  - `centralized-workflow-exercise`
  - `forking-workflow-exercise-recorded`
  - `forking-workflow-exercise`
- Protect the default branch of the two `centralized-*` repositories (but this can also be done on stream as the very first step if you are sure you will remember as instructor).

What to communicate to learners at least 1 day before the workshop

- Example email from a previous workshop

How should learners request access

This is also in the email template above but they need to:

- Open an issue at https://github.com/cr-workshop-exercises/access-requests/issues/new?template=access-request.md
- Accept invitation from GitHub sent to their email address (that GitHub knows about).
- "Unwatch" both these repositories by clicking the "Watch" button (top middle of the screen) and then select "Participating and mentions":
  - https://github.com/cr-workshop-exercises/centralized-workflow-exercise
  - https://github.com/cr-workshop-exercises/centralized-workflow-exercise-recorded

How to add learners to the team stream-exercise-participants

You need to be "owner" of https://github.com/orgs/cr-workshop-exercises/teams/stream-exercise-participants to be able to add people to the team.

1. Check https://github.com/cr-workshop-exercises/access-requests/issues. Any open issue means the person hasn't been added yet.
2. Assign one issue to yourself. This way other organizers know that this is being worked on.
3. Add person to https://github.com/orgs/cr-workshop-exercises/teams/stream-exercise-participants

- Click on "Add member" -> "Invite" -> "Add (username) to stream-exercise-participants"

- **Do not add/invite the person anywhere else**, not as collaborator to any exercise repo directly. Only add/invite them into the team "stream-exercise-participants". Motivation: This way we give instructors the control over when the exercise can start. Otherwise learners might merge changes before the lesson and thus change the example and confuse instructors and learners.

4. Close the issue on https://github.com/cr-workshop-exercises/access-requests/issues with the following comment (feel free to adapt it):

```
Thanks! I have added you to the collaborative exercise team.

What you should do before the exercise starts:

1) You will get an invitation from GitHub to your email address (that GitHub
   knows about). Please accept that invitation so that you can participate in
   the collaborative exercise.

2) To make sure you don't get too many emails during the exercise, don't forget
   to "unwatch" both
   https://github.com/cr-workshop-exercises/centralized-workflow-exercise and
   https://github.com/cr-workshop-exercises/centralized-workflow-exercise-recorded.
   To "unwatch", go to the repository and click the "Watch" button (top
   middle of the screen) and then select "Participating and mentions".
```

Why we teach this lesson

In order to collaborate efficiently using Git, it's essential to have a solid understanding of how remotes work, and how to contribute changes through pull requests or merge requests. The git-intro lesson teaches participants how to work efficiently with Git when there is only one developer (more precisely: how to work when there are no remote Git repositories yet in the picture). This lesson dives into the collaborative aspects of Git and focuses on the possible collaborative workflows enabled by web-based repository hosting platforms like GitHub.

This lesson is meant to directly benefit workshop participants who have prior experience with Git, enabling them to put collaborative workflows involving code review directly into practice when they return to their normal work. For novice Git users (who may have learned a lot in the git-intro lesson) this lesson is somewhat challenging, but the lesson aims to introduce them to the concepts and give them confidence to start using these workflows later when they have gained some further experience in working with Git.

Intended learning outcomes

By the end of this lesson, learners should:

- Understand the concept of remotes
- Be able to describe the difference between local and remote branches
- Be able to describe the difference between centralized and forking workflows
- Know how to use pull requests or merge requests to submit changes to another projects

- Know how to reference issues in commits or pull/merge requests and how to auto-close issues
- Know how to update a fork
- **Be able to contribute in code review as submitter or reviewer**

Interesting questions you might get

- If participants run `git graph` they might notice `origin/HEAD`. This has been omitted from the figures to not overload the presentation. This pointer represents the default branch of the remote repository.

Timing

- The centralized collaboration episode is densest and introduces many new concepts, so at least an hour is required for it.
- The forking-workflow exercise repeats familiar concepts (only introduces forking and distributed workflows), and it takes maybe half the time of the first episode.
- The “How to contribute changes to somebody else’s project” episode can be covered relatively quickly and offers room for discussion if you have time left. However, this should not be skipped as this is perhaps the key learning outcome.

Typical pitfalls

Difference between pull and pull requests

The difference between pull and pull requests can be confusing, explain clearly that pull requests or merge requests are a different mechanism specific to GitHub, GitLab, etc.

Pull requests are from branch to branch, not from commit to branch

The behavior that additional commits to a branch from which a pull request has been created get appended to the pull request needs to be explained.

Other practical aspects

- In in-person workshops participants really have to sit next to someone, so that they can see the screens. From the beginning.
- Emphasize use of `git graph` a lot, just like in the git-solo lesson.