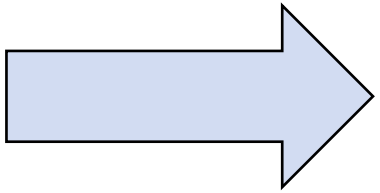
Collaborative Git

INTERSECT Bootcamp

Lauren Milechin

Outline



- Introduction
- Branches
- Pull Requests
- Git Workflows

Challenges of Collaborative Development

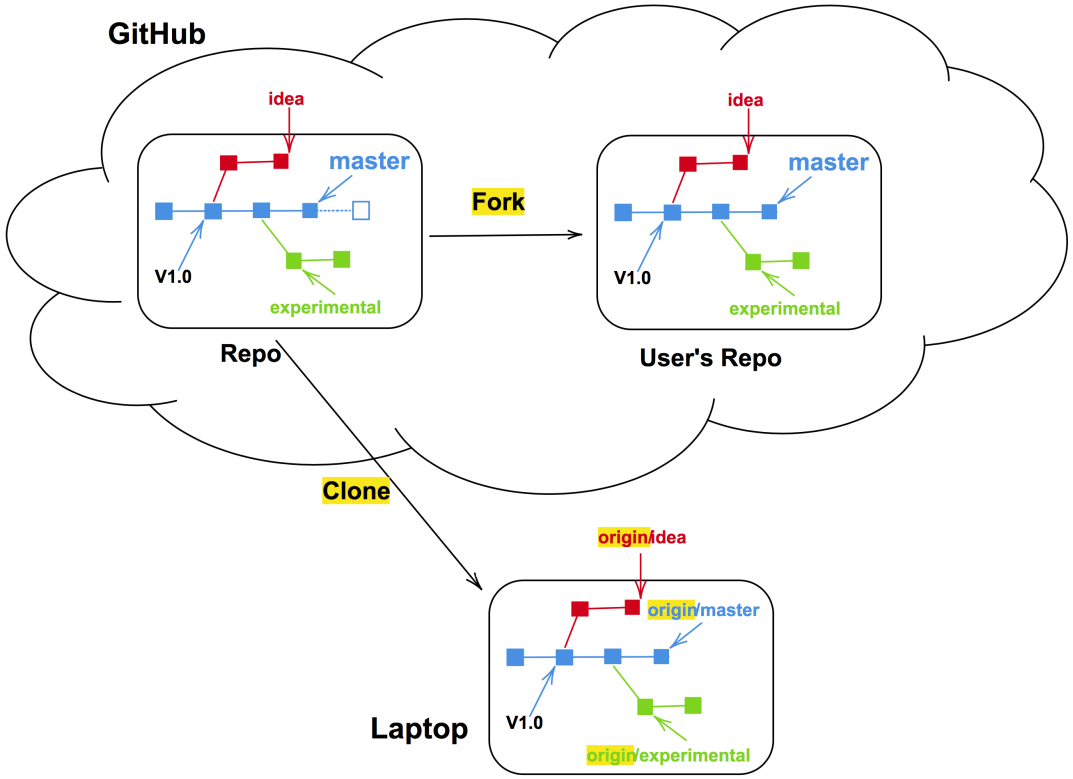
- **Sharing code such that**
  - **Everyone has the latest version**
  - **Everyone can contribute equally**
- **Keeping organized within the team**
  - **Everyone is on the same page**
  - **Communication**

Challenges of Collaborative Development

- **Sharing code such that**
  - **Everyone has the latest version**
  - **Everyone can contribute equally**
- **Keeping organized within the team**
  - **Everyone is on the same page**
  - **Communication**

Review

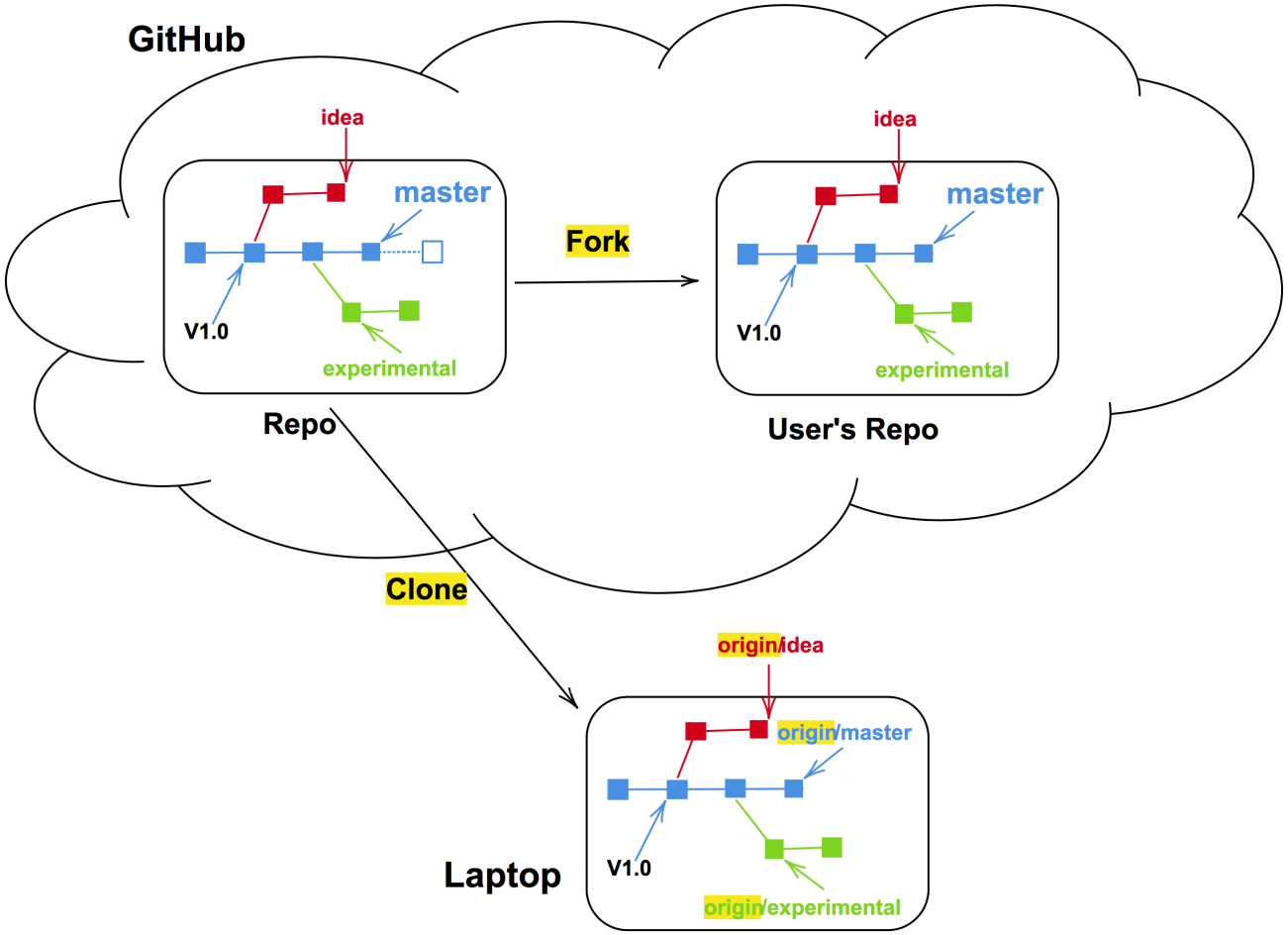
- **Repository: The project, contains all data and history (commits, branches, tags).**
  - **remote (repository): The copy of the repository on a remote server, such as GitHub. Accessible to anyone who has access to the repository on the server. Also referred to as the "central" repository.**
  - **local (repository): A copy of the repository in another place, such as a laptop or your home directory on a cluster. Generally accessible to one person.**
- **Commit: Snapshot of the project, gets a unique identifier (e.g. c7f0e8bfc718be04525847fc7ac237f470add76e).**
- **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).**



Definitions and Figure from Code Refinery: https://coderefinery.github.io/git-collaborative/remotes/

Review

- **Branch: Independent development line. The main development line is often called main or master.**
- **Cloning: Copying the whole repository to your laptop - the first time. This creates a local copy of the repository.**
- **Forking: Taking a copy of a repository (which is typically not yours) into your GitHub/GitLab account - your copy (fork) stays on GitHub/GitLab and you can make changes to your copy.**
- **Pull: Bring changes from a remote repository to your local repository.**
- **Push: Bring changes from your local repository to the remote repository.**

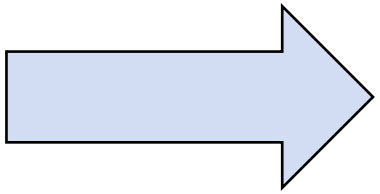


Definitions and Figure from Code Refinery: https://coderefinery.github.io/git-collaborative/remotes/

Activity

- We will work in small groups for this module
- Follow along
- One person in each group
  - Create a repository from the template
    - https://github.com/INTERSECT-training/intersect-training-day2
    - Use this Template -> Create a New Repository
  - Give access to each member in your group
- Everyone
  - Accept invite to collaborate
  - Clone the repository to your computer with `git clone`

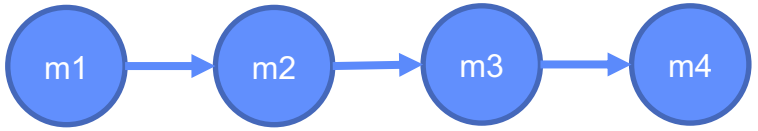
Outline



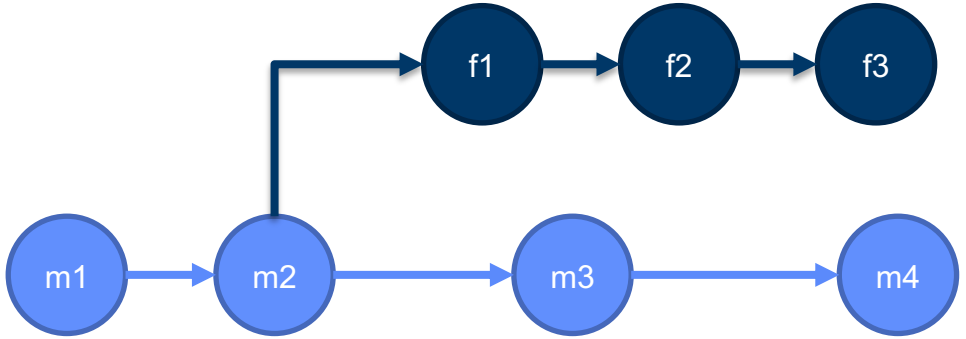
- Introduction
- Branches
- Pull Requests
- Git Workflows

Branches

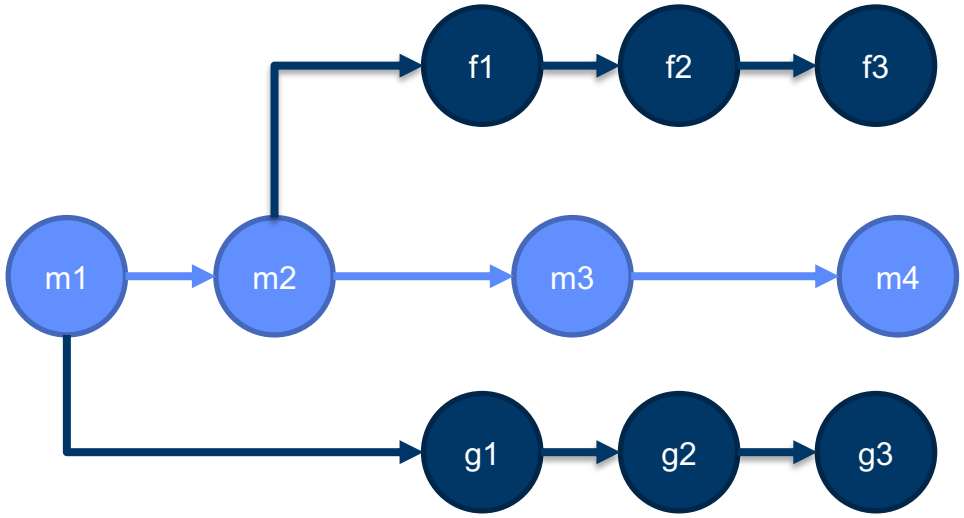
- **Independent development lines**
- **Different purposes (feature, development, production)**
- **Varying lifetimes (short vs long-lived)**
- **Commits from one branch can be merged into another**



Main Branch Only



Main Branch with Feature Branch



Main Branch with 2 Feature Branches

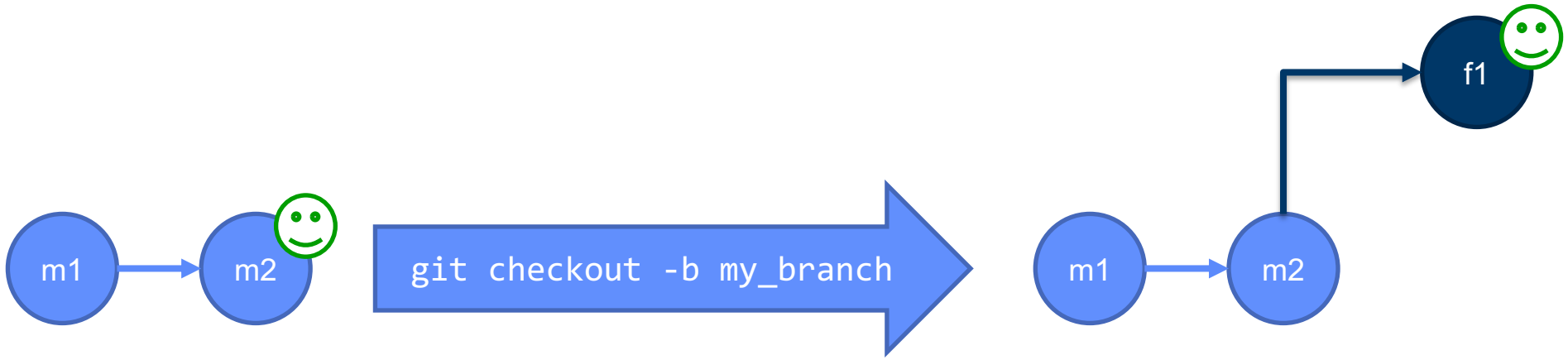
Creating Branches

- **Good practice to first pull any recent changes to the repository**

```
git checkout main
git pull
```

- **Create a branch**

```
git checkout -b my_new_branch
```



Creating Branches

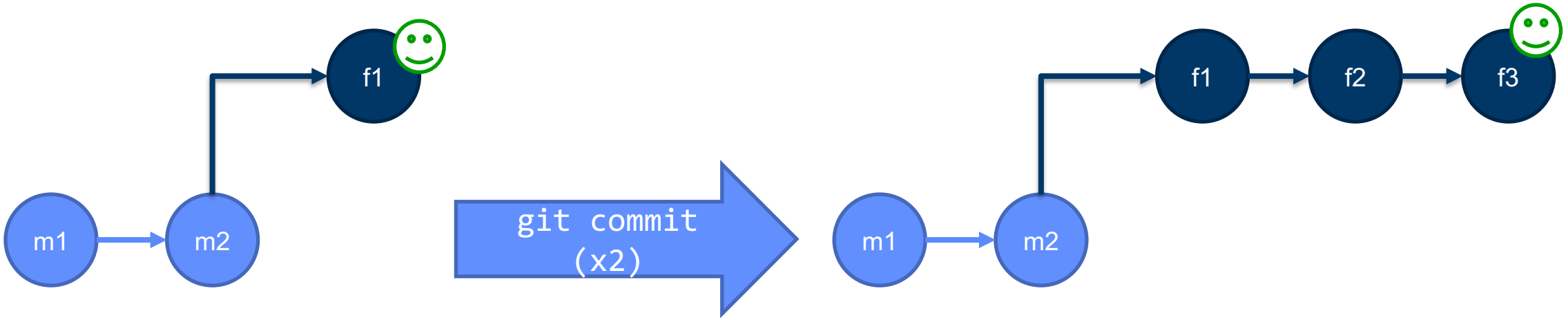
- **Good practice to first pull any recent changes to the repository**

```
git checkout main
git pull
```

- **Create a branch**

```
git checkout -b my_new_branch
```

- **New commits will be added to this branch**



Creating Branches

- **Good practice to first pull any recent changes to the repository**

```
git checkout main
git pull
```

- **Create a branch**

```
git checkout -b my_new_branch
```

- **New commits will be added to this branch**
- **Publish your branch**

```
git push --set-upstream origin my_new_branch
```

- **Add new commits to the remote repository**

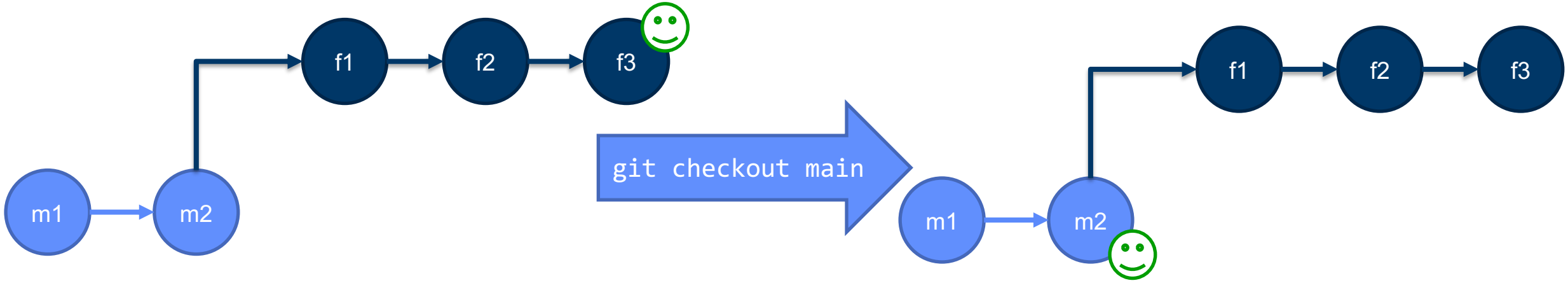
```
git push
```

Changing Branches

- Changing branches using the command git checkout
- For example, to change to the main branch, run

```
git checkout main
```

- The `git status` command will display which branch you are on



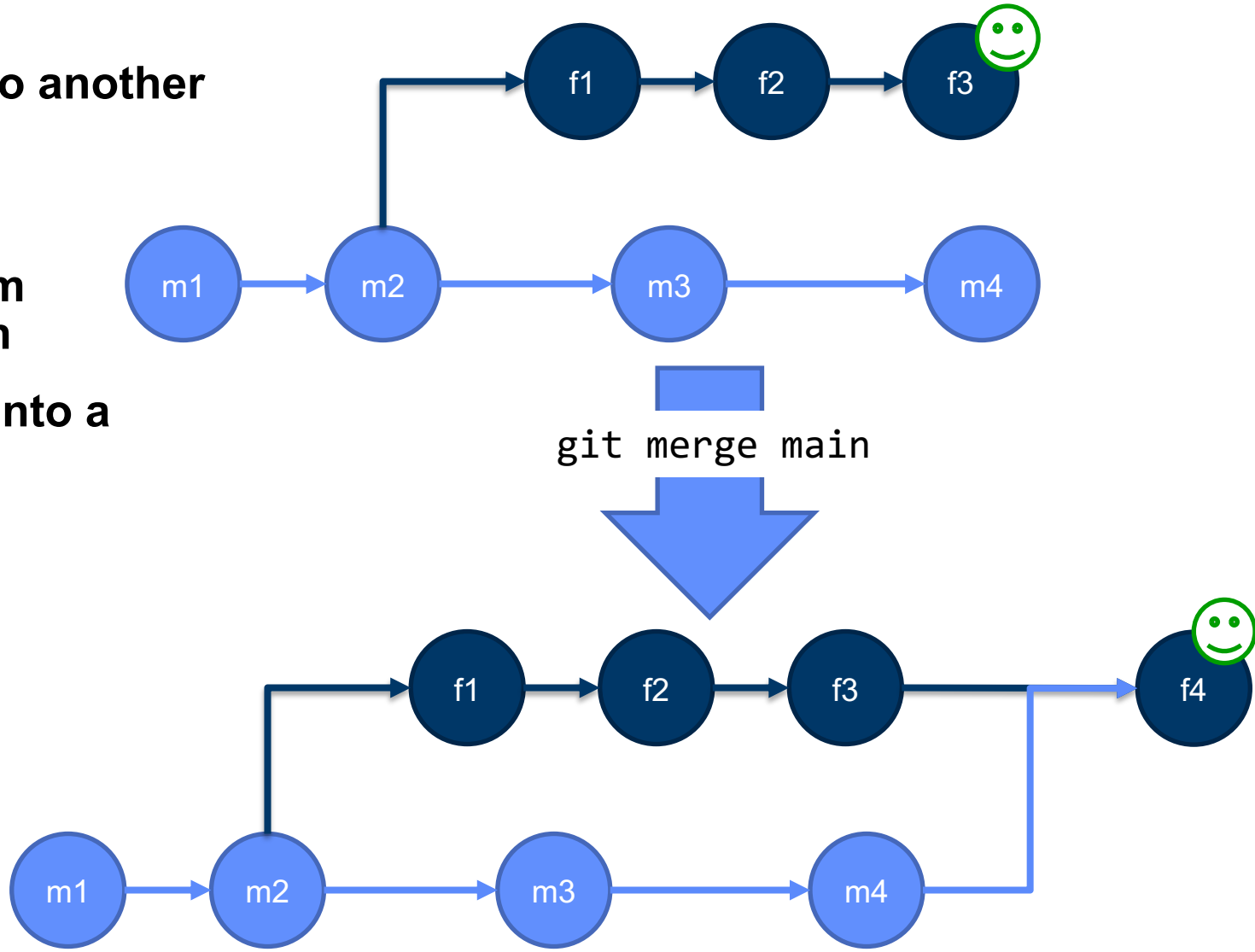
Merging Branches

- **Bring changes from one branch into another with a merge**

```
git merge branch_name
```

- **This command brings changes from `branch_name` to the current branch**
- **To merge new changes from main into a my_new_branch**

```
git checkout my_new_branch
git merge main
```



Merging Branches

- **Bring commits from one branch into another with a merge**

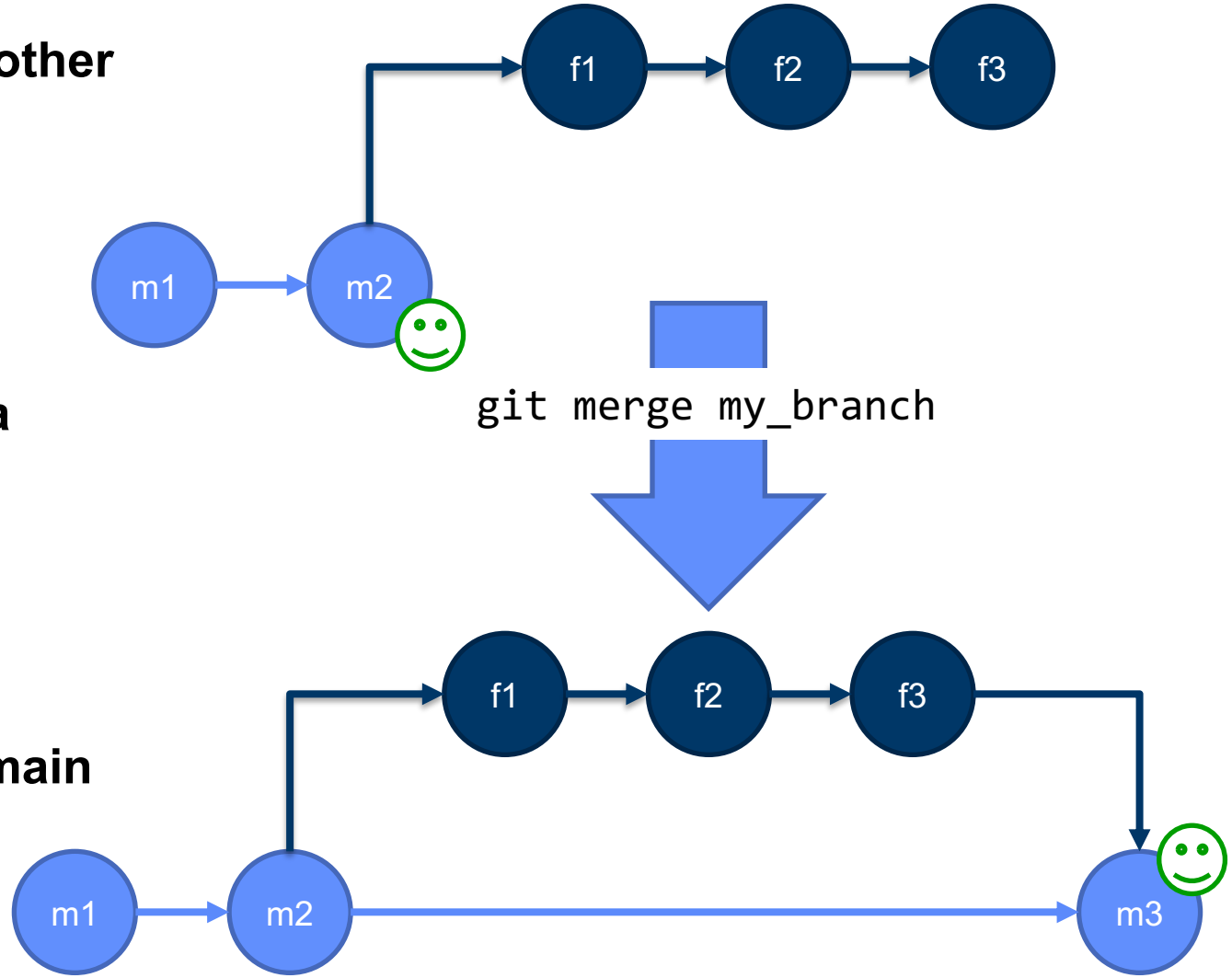
```
git merge branch_name
```

- **This command brings commits from `branch_name` to the current branch**
- **To merge new changes from main into a my_new_branch**

```
git checkout my_new_branch
git merge main
```

- **When development is complete on my_new_branch it can be merged into main with**

```
git checkout main
git merge my_new_branch
```



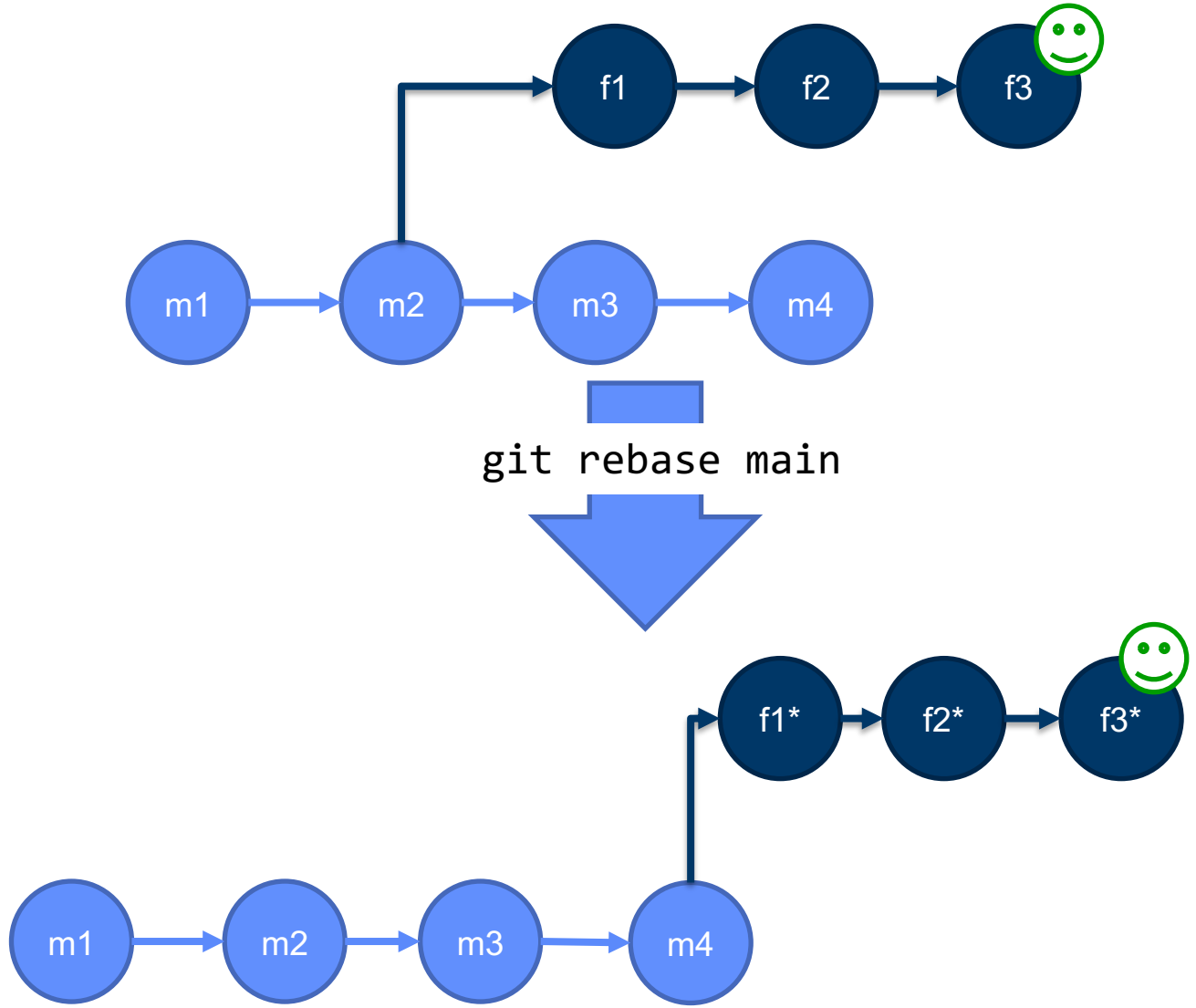
Merge Conflicts

- Git will do its best to merge automatically
- Merge conflicts occur when when git can't do this
  - Usually when the same line is changed in both branches
- Merge conflicts must be resolved before completing the merge
  - Manually using a text editor
  - Using a visual mergetool
  - Communicate with collaborators if necessary
- Ways to avoid:
  - Modular code- work on one feature less likely to impact another
  - Regularly pull any changes from main and merge into feature branch
  - When development of a feature is complete, check for and merge any changes from main into the feature branch before merging the feature branch into main
- https://coderefinery.github.io/git-intro/conflicts/
- https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts

Merge vs Rebase

- Rebase sometimes an alternative to merging
- For example
  - Two new main branch commits after branch is created
  - Rebase your feature branch onto the main branch with

    ```
    git rebase main
    ```
  - This “replays” your branch commits at the end of the main branch
- Rewrites history
- Should never be done on a public branch or commit



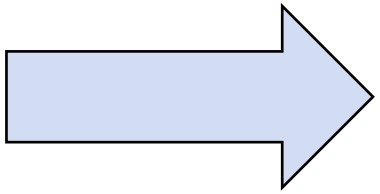
Activity

- **Everyone:**
  - **Create a new branch in your local clone:** `git checkout -b branch_name`
  - **Publish the branch to the remote repository:** `git push --set-upstream origin branch_name`
  - **Make a small change, commit, and push**
  - **Look at the remote repository, look at your branch and your team's branches**
- **All but one team member:**
  - **Merge the branch**

```
git pull
git checkout main
git merge branch_name
git push
```

- **Did anyone run into merge conflicts? How might this have been prevented?**
- **There should be one unmerged branch at the end of this activity**

Outline



- Introduction
- Branches
- Pull Requests
- Git Workflows

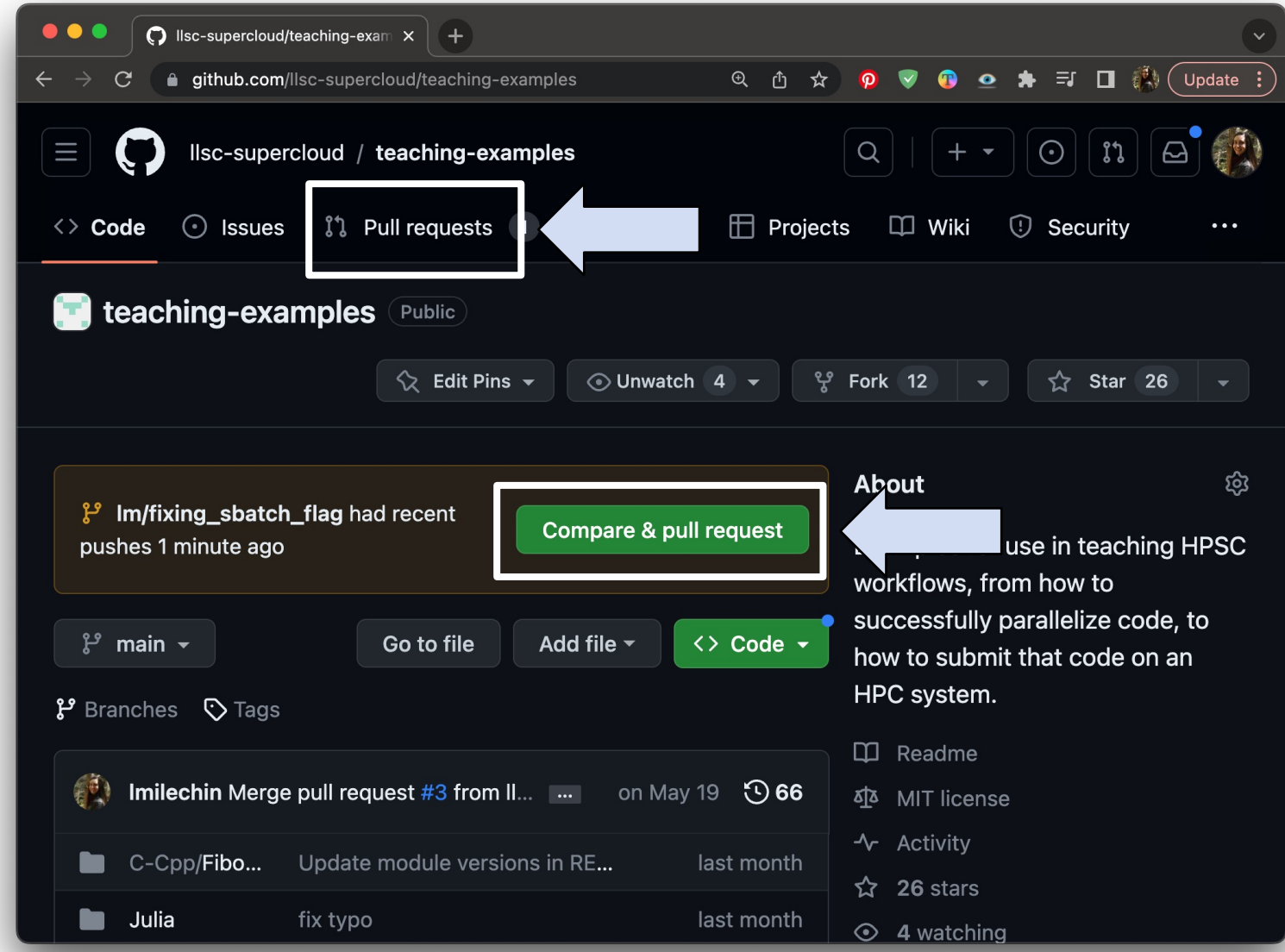
Pull Requests

- A process on GitHub to request a branch be merged into another branch
  - You are asking for your branch to be “pulled” into another branch
- GitLab uses the term “Merge Request”
- Allows code review and testing before merge
  - Alerts team and others about changes in branch before merge
  - Discussions ensue with possible follow up commits
  - Can request reviewer (may want particular expertise, may want to give someone knowledge)
- Set policies for merge
  - Enforce rules such as coding standards
  - Minimum number of reviewers
  - Protected branches – who can merge to certain branches etc.

David M. Rogers, Collaborative Software Development in Better Scientific Software tutorial, in ISC High Performance, Hamburg, Germany and online, 2023. DOI: 10.6084/m9.figshare.22790762.

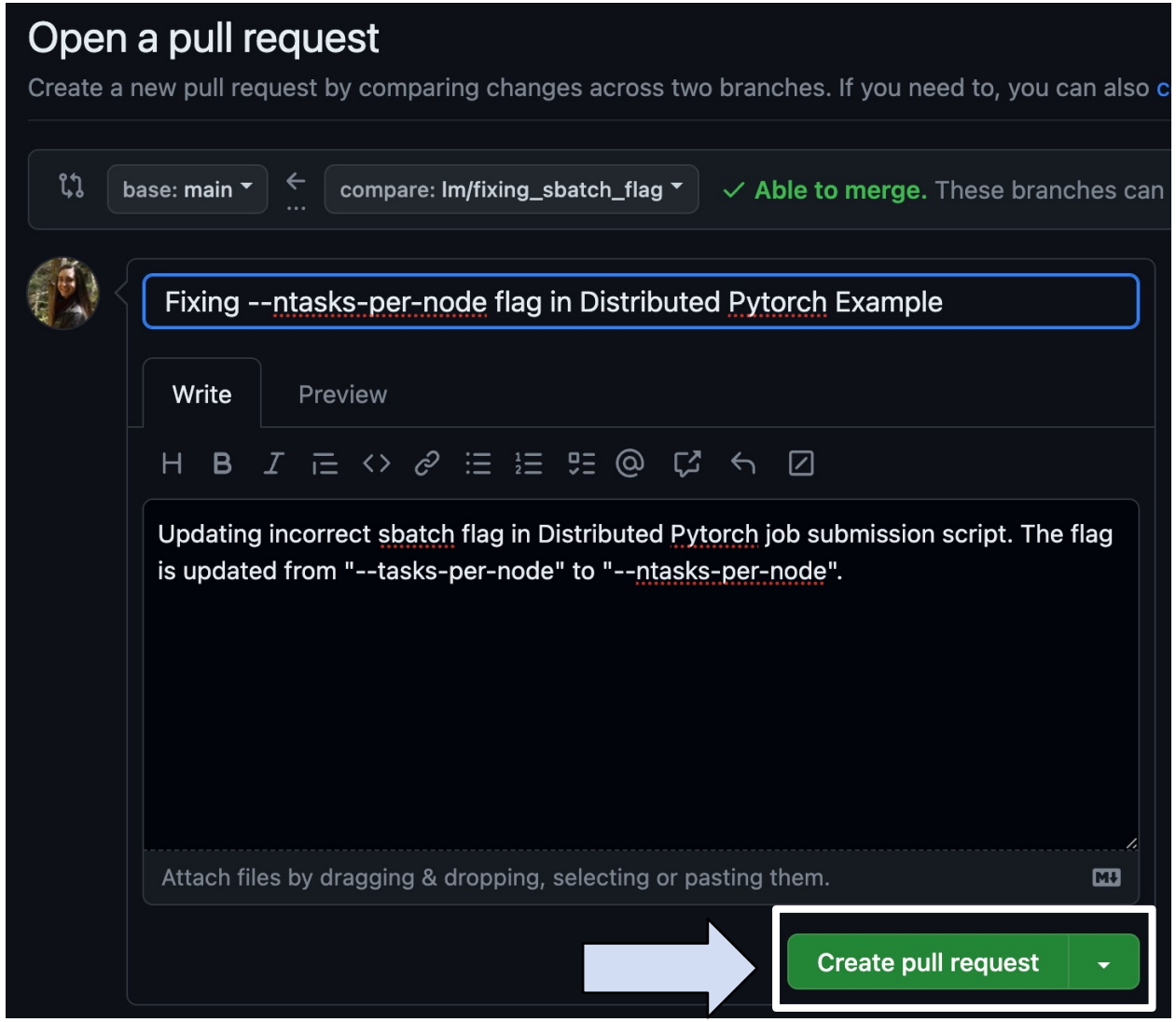
Creating Pull Request

- First make sure you've pushed all commits with `git status` and `git push`
- Go to your branch in GitHub
- If you pushed changes recently you may see a button that says "Compare and Pull Request", click if it is there
- If not click "Pull Requests"
  - If you click "Pull Requests" select "New Pull Request" and change the "compare" dropdown to your branch name



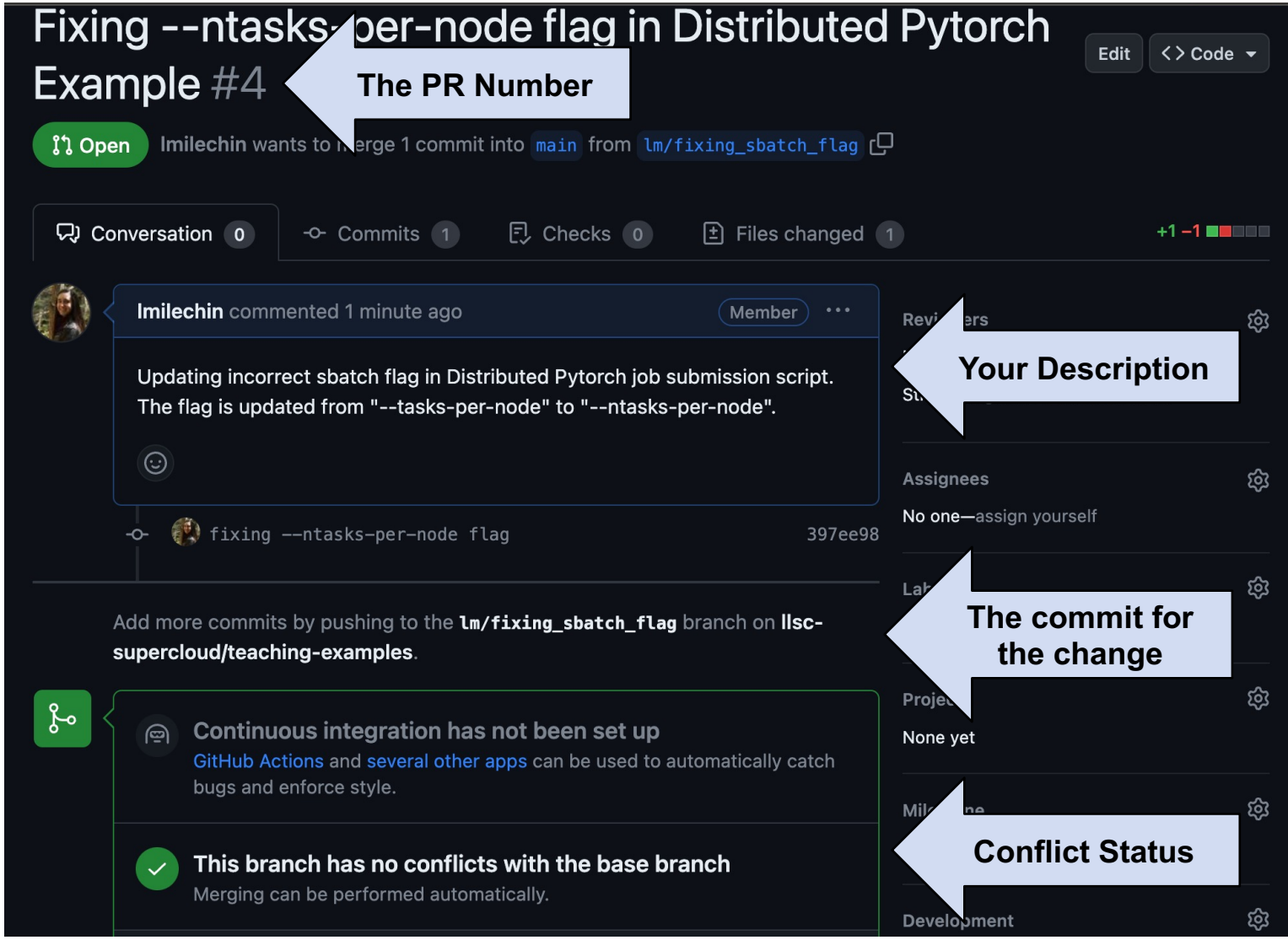
Creating a Pull Request

- **Fill out the form with**
  - **A title**
  - **A description of the changes you made and why you made them, include any other relevant information**
- **When you are done click “Create pull request”**



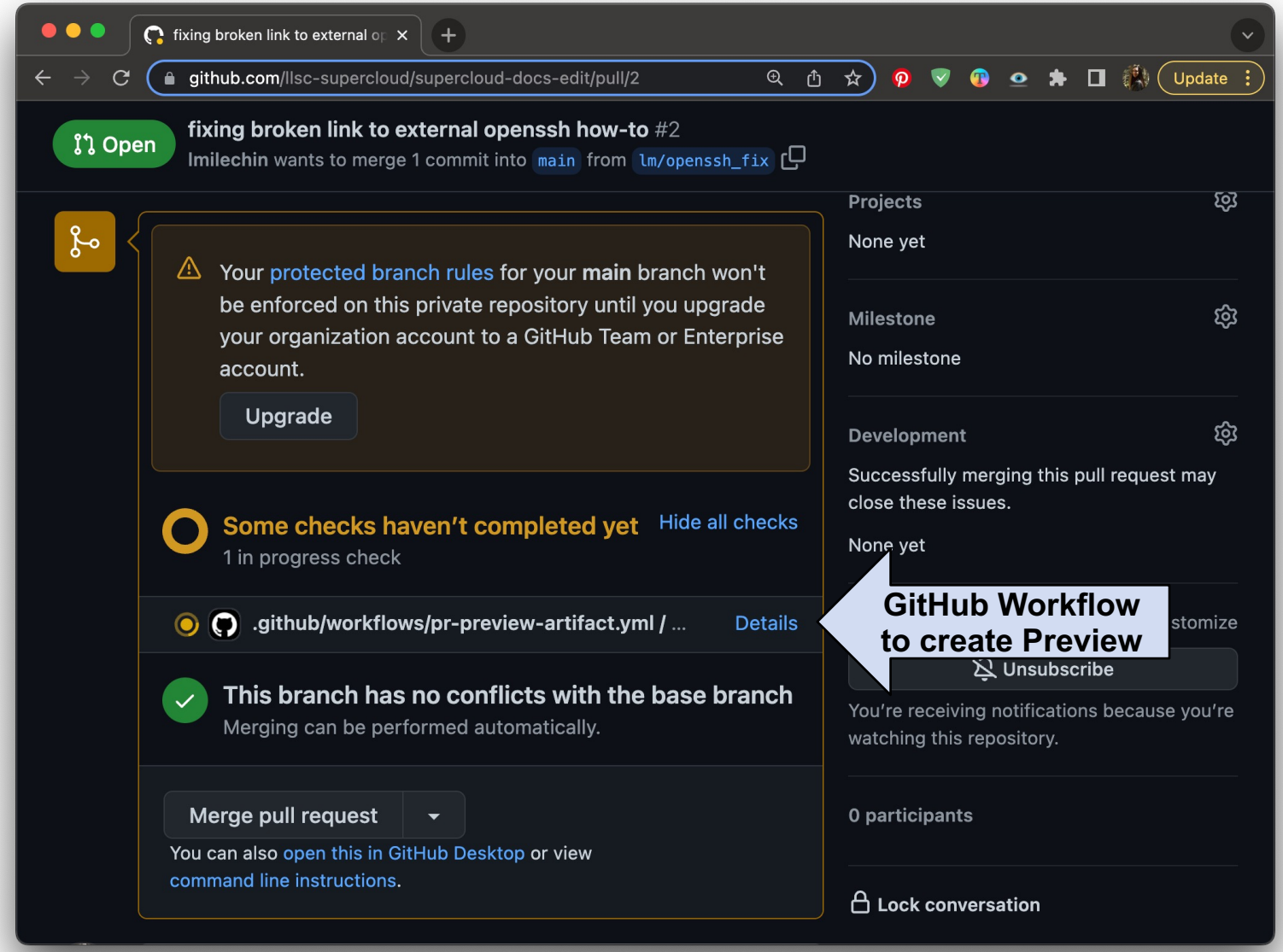
Anatomy of a Pull Request

- The PR number is displayed next to the PR title
- In the conversation tab you'll see a timeline of comments and commits
  - The first "comment" is the initial description
  - Scroll to the bottom of the to add more comments
- It will also show whether there are any merge conflicts
  - Resolve any conflicts locally and push the commit to update

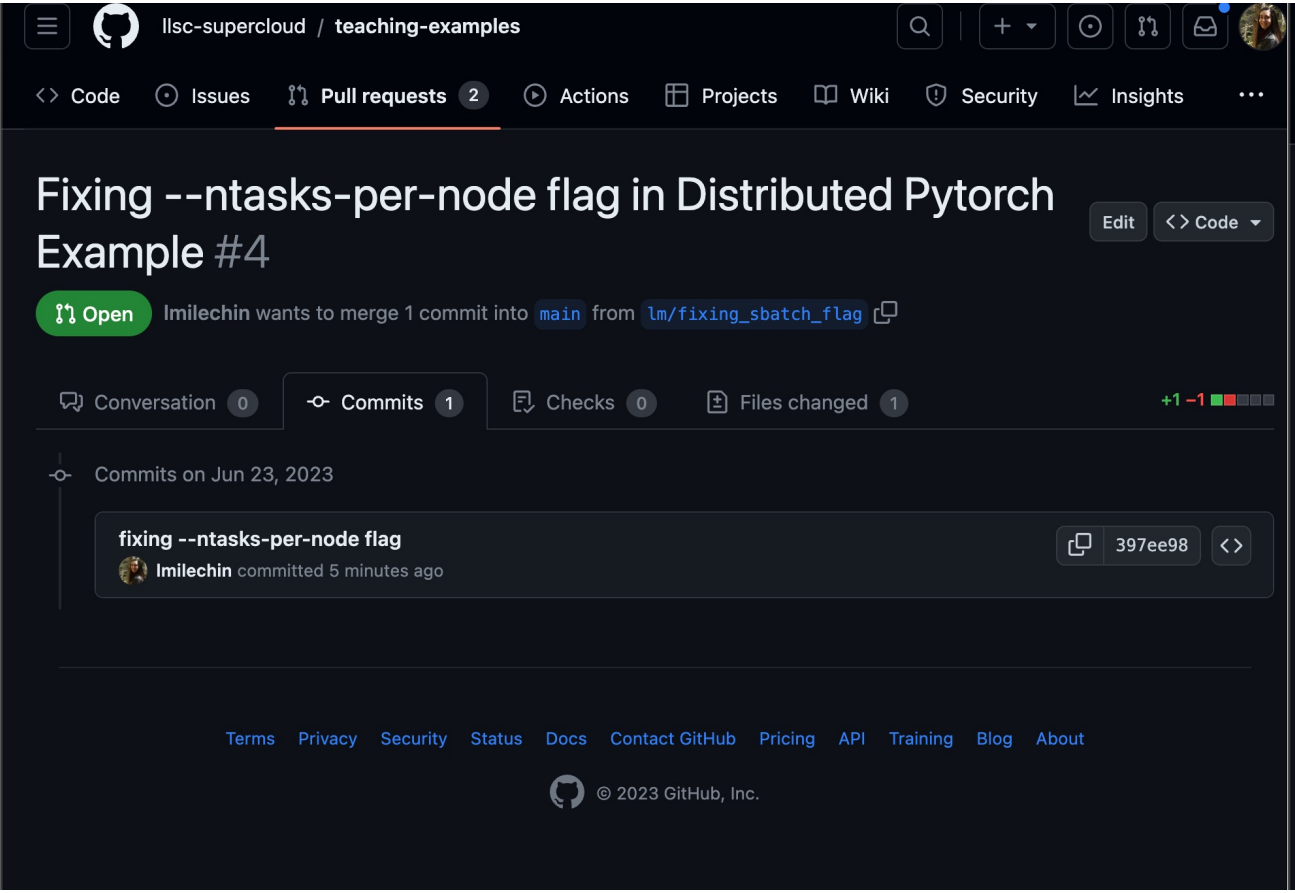


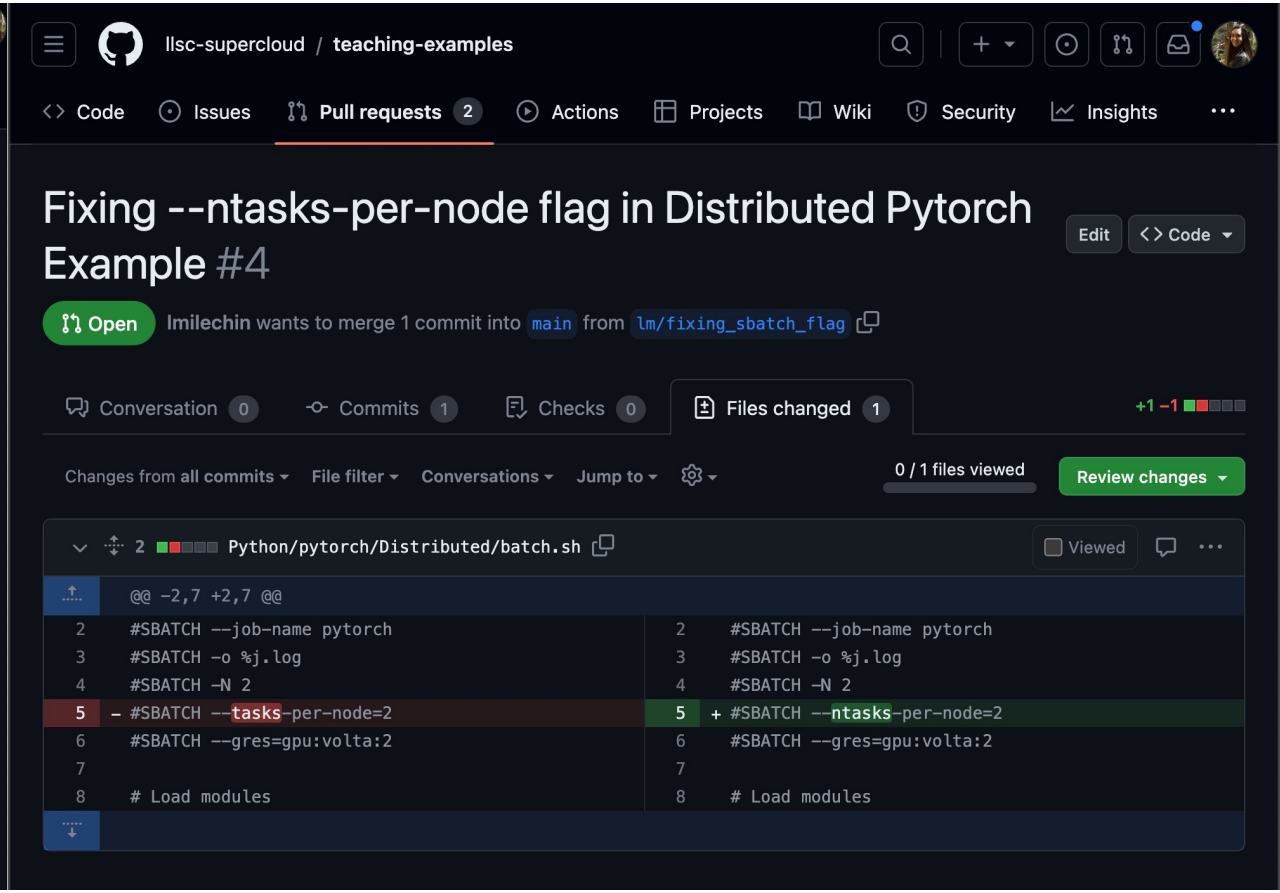
Anatomy of a Pull Request

- The PR number is displayed next to the PR title
- In the conversation tab you'll see a timeline of comments and commits
  - The first "comment" is the initial description
  - Scroll to the bottom of the to add more comments
- It will also show whether there are any merge conflicts
  - Resolve any conflicts locally and push the commit to update
- Continuous Integration



Anatomy of a Pull Request



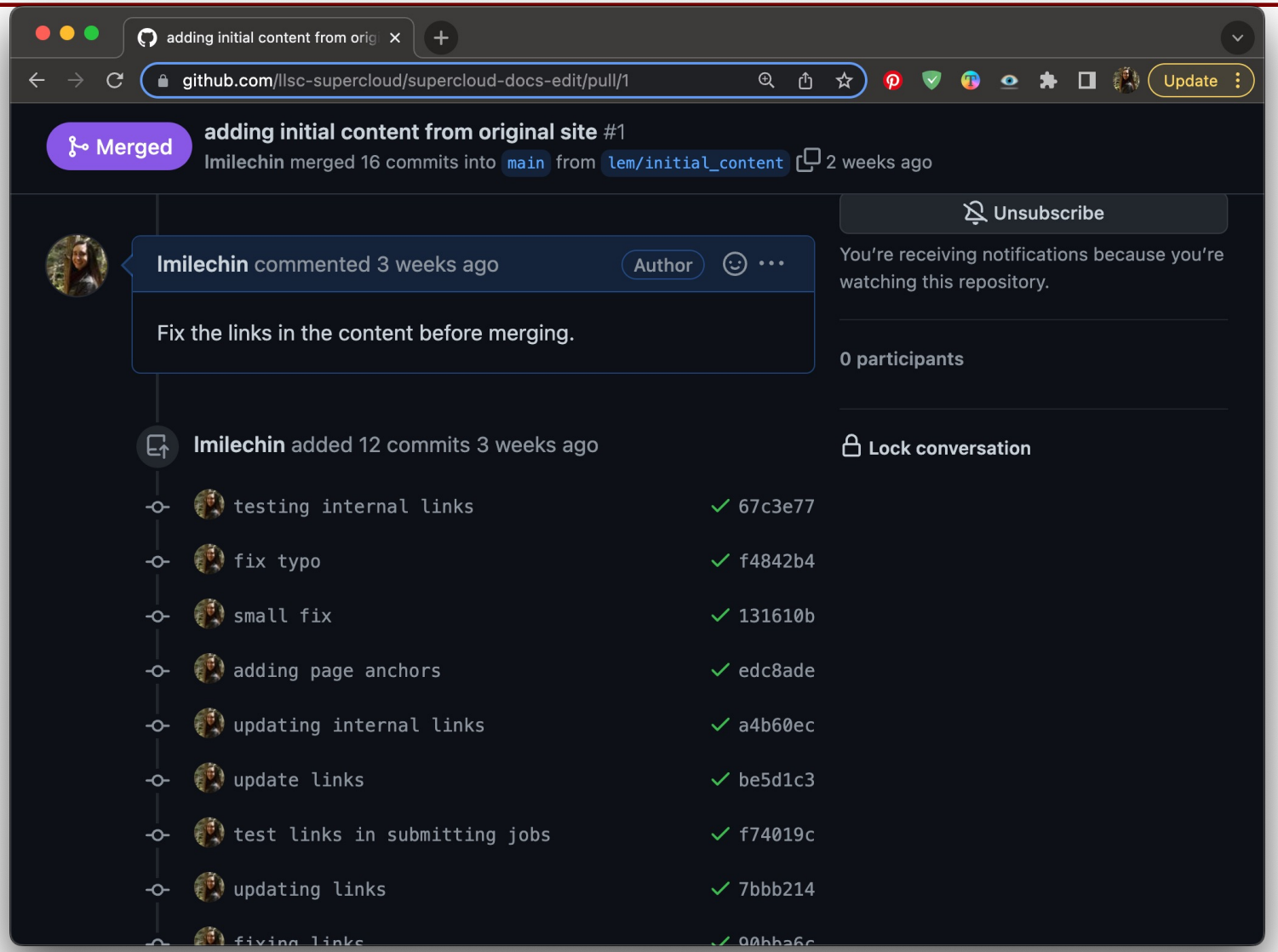


The Commits tab shows all the commits in the PR

Click “Files Changed” to review changes

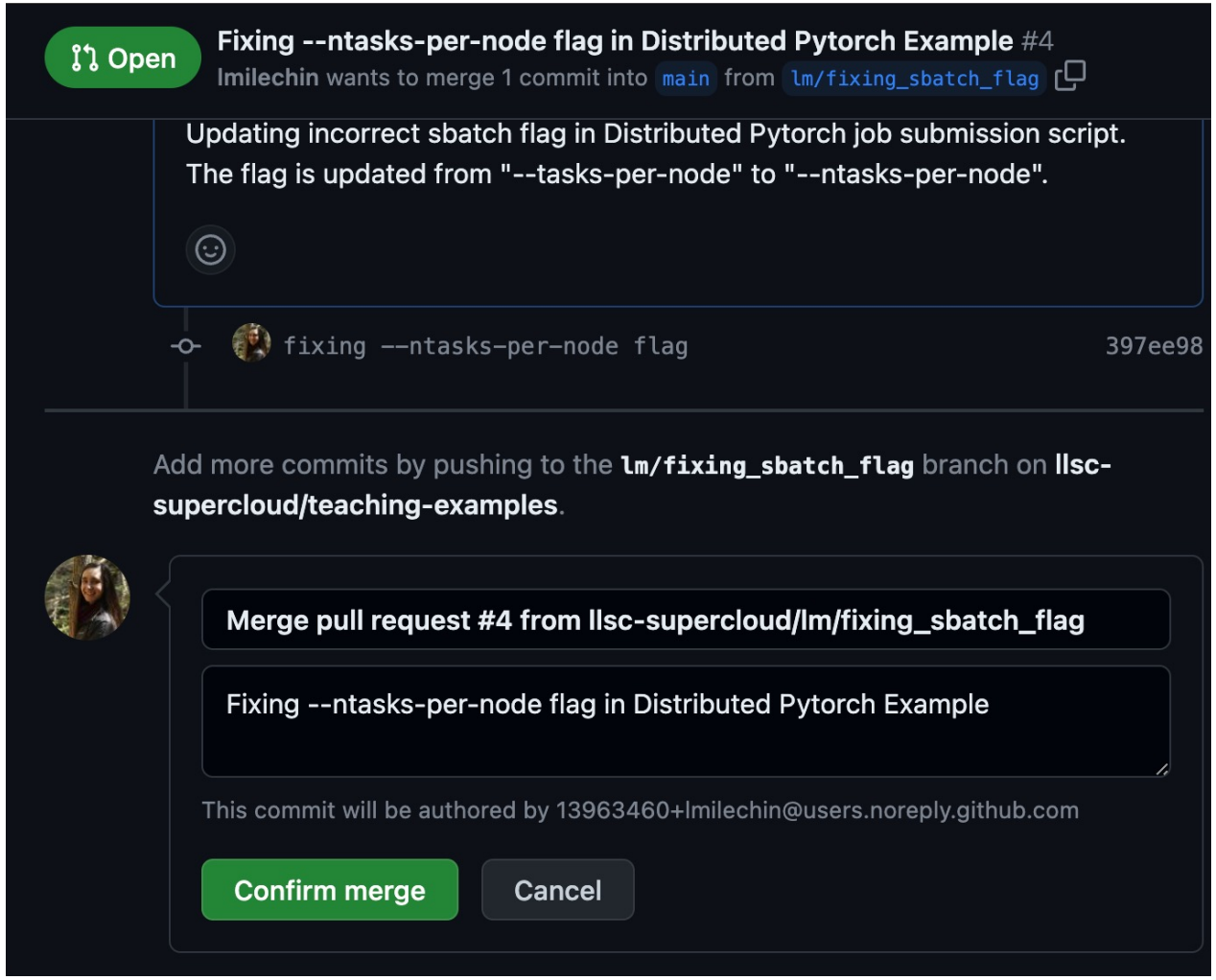
Updating a Pull Request

- Others on the team should review the changes
- Feedback can be given using the comment field at the bottom of the Conversation Tab
- To update the PR, make updates in your branch and push the commits
- New commits will be added to the PR automatically



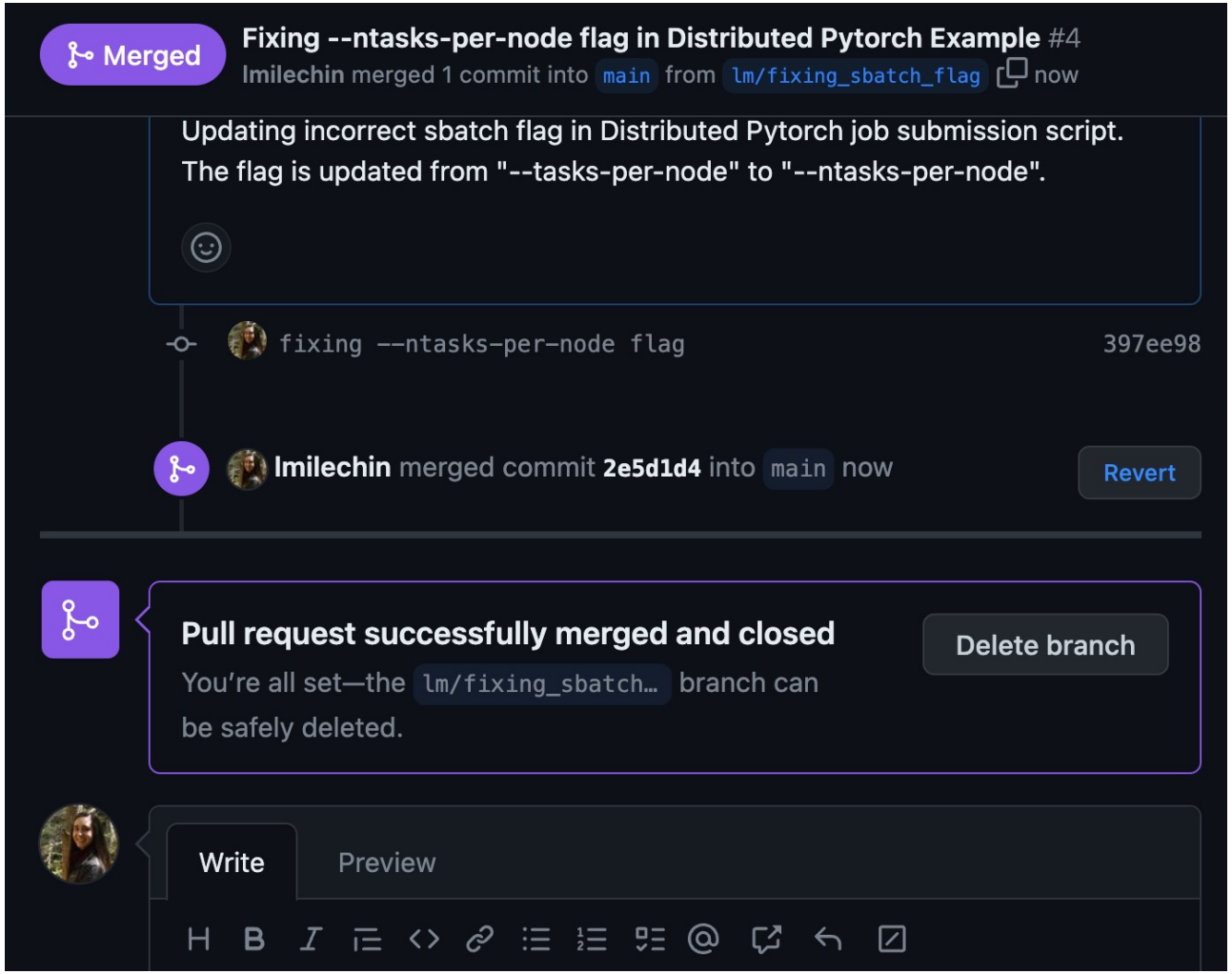
Accepting and Merging a Pull Request

- Once the PR has been reviewed it is ready to merge
- Click “Merge Pull Request” to merge, then click “Confirm merge”
- After merging, you can click “delete branch” if you’d like, new material from here should be in a new branch
- When the PR is merged the web page will be updated



Accepting and Merging a Pull Request

- Once the PR has been reviewed it is ready to merge
- Click “Merge Pull Request” to merge, then click “Confirm merge”
- After merging, you can click “delete branch” if you’d like, new material from here should be in a new branch
- When the PR is merged the web page will be updated



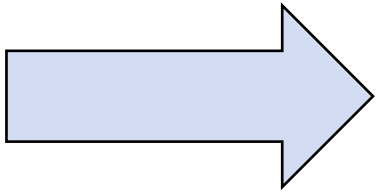
Activity

- Your team's repository should have one unmerged branch, if not, create one now and publish the branch
- Everyone: Log into GitHub
- Branch owner: create a Pull Request
- Team: Review the Pull Request
  - Look at the commits and files changed
  - Do the changes look okay? Comment if not
  - Are there any merge conflicts?
- Branch owner: address any merge conflicts or comments
- Team: comment when it looks good to merge
- Once approved merge the pull request

- In both this activity and the last one your branch was eventually merged into the main branch. How does this experience compare to the previous activity?
- If you have extra time, the remaining team members can create their own Pull Requests with new branches.

Outline

- Introduction
- Branches
- Pull Requests
- Git Workflows

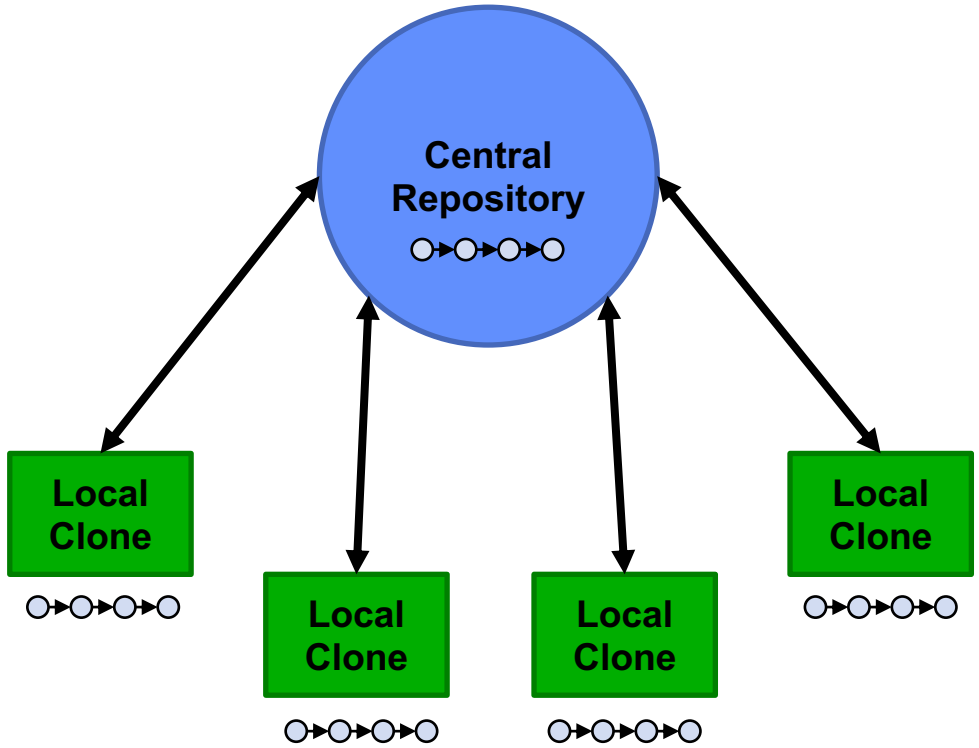


Git Workflows

- A process for how your team is going to work with a repository, including
  - How you are going to use branches (branching strategy)
    - Feature branches, ”lifetime” branches (production, development, release branches)
    - What is considered the "latest version", or what is "production" vs "in development"
    - How branches are merged
  - When you are making pull requests and into what
  - Testing
  - Whether you are working in a single, central repository, or separate forked repositories
- Help everyone stay on the same page about the state of the code
- Enable communication
- Tell us
  - What branches are protected
  - What is tested and when
  - What branches are stable

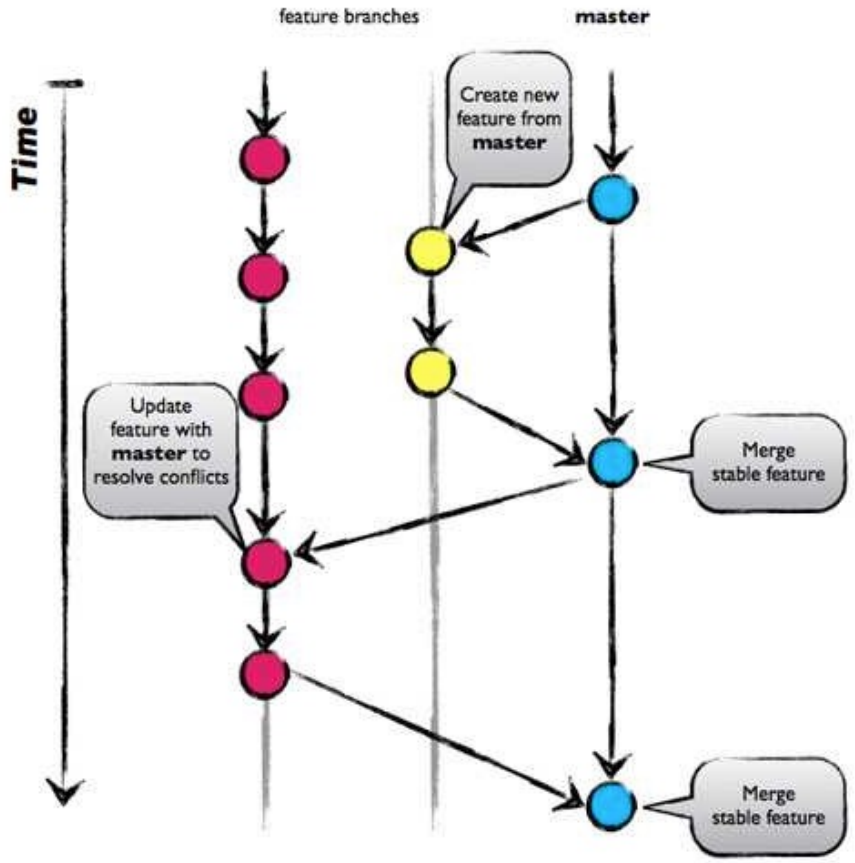
Centralized or Trunk-Based Workflow

- Central repository with one main branch
- Developers work in their local clone and push less frequently
- Central repository meant to be stable
- Resolve conflicts locally before pushing
- Not the best for collaborative development



GitHub Flow

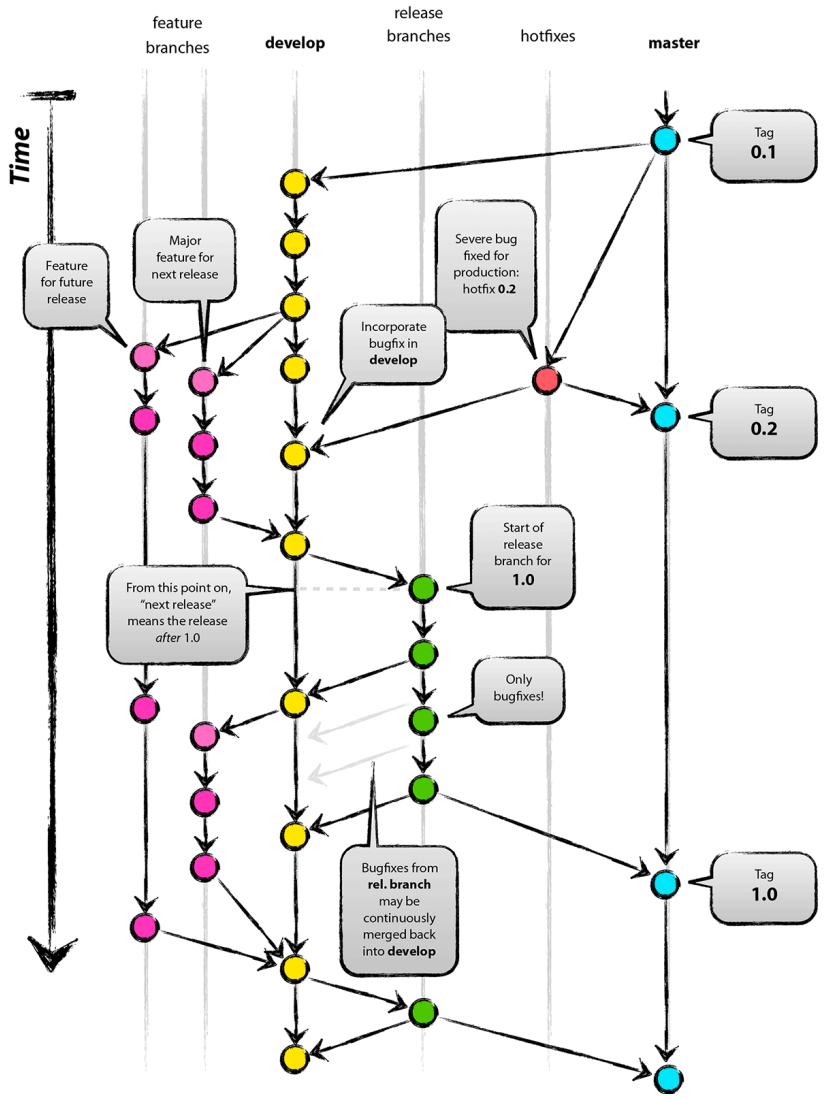
- A feature branching workflow
- Published as viable alternative to Git Flow (we'll see this next)
- No structured release schedule
- Continuous deployment & continuous integration
- Simpler workflow
- Key Ideas
  - All commits in the main branch are deployable
  - Base feature branches off main
  - Push local repository to remote constantly
  - Open Pull Requests early to start dialogue
  - Merge into main after Pull Request review



http://scottchacon.com/2011/08/31/github-flow.html

David M. Rogers, Collaborative Software Development in Better Scientific Software tutorial, in ISC High Performance, Hamburg, Germany and online, 2023. DOI: 10.6084/m9.figshare.22790762.

Git Flow



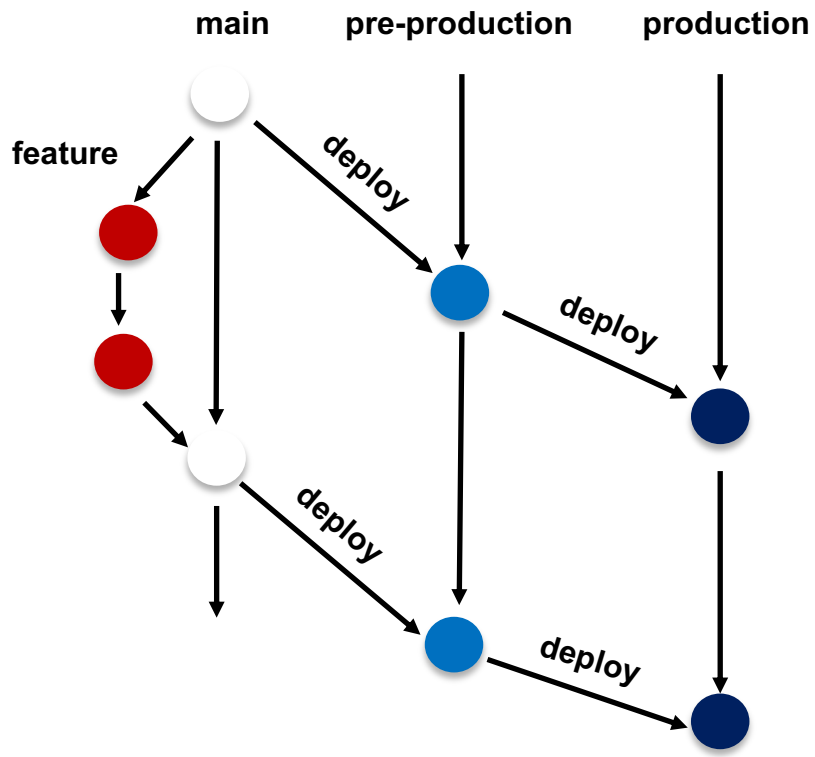
- **Full-featured workflow**
- **Increased complexity**
- **Designed for Software with official releases**
- **Feature branches based off of develop**
- **Git extensions to enforce policy**
- **How are develop and main synchronized?**
- **Where do merge conflicts occur and how are they resolved?**

Author: Vincent Driessen
Original Blog: https://nvie.com/posts/a-successful-git-branching-model/
License: Creative Commons

David M. Rogers, Collaborative Software Development in Better Scientific Software tutorial, in ISC High Performance, Hamburg, Germany and online, 2023. DOI: 10.6084/m9.figshare.22790762.

GitLab Flow

- **Published as viable alternative to Git Flow & GitHub Flow**
- **Semi-structured release schedule**
- **Simplifies difficulties in synchronizing infinite lifetime branches**
- **Key Ideas**
  - **Main branch is staging area**
  - **Mature code in main flows downstream into**
    - **pre-production & production infinite lifetime branches**
  - **Allow for release branches with downstream flow**
    - **Fixes made upstream & merged into main.**
    - **Fixes cherry picked into release branch**



David M. Rogers, Collaborative Software Development in Better Scientific Software tutorial, in ISC High Performance, Hamburg, Germany and online, 2023. DOI: 10.6084/m9.figshare.22790762.

Forks in Workflows

- So far we have assumed branches are created in the main repository- not feasible for some projects
- Contributors:
  - Fork the repository and mainly interact with their fork
  - Create branches in fork and create a pull request from the fork to the original repository to merge changes
- Anybody can propose contributions without asking for advance permission (to public projects)
- Maintainer still has full control over what is merged
- Challenge: contributors now have more than one remote to work with
- This is used by almost all large (and small) open-source projects these days

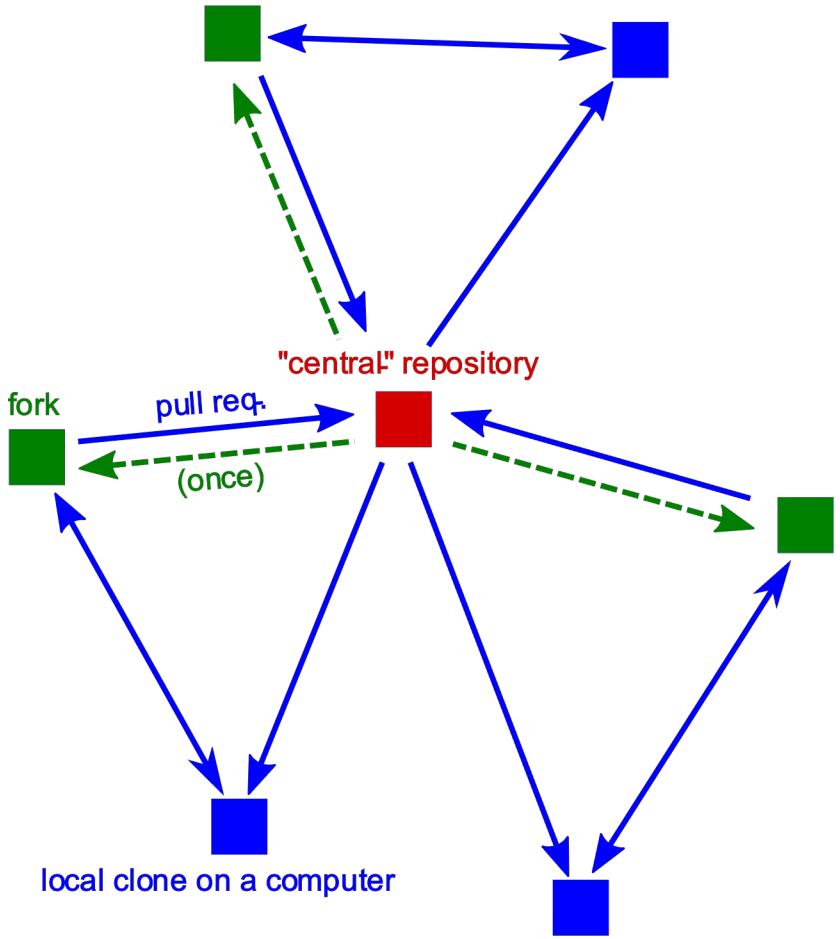


Figure from Code Refinery: https://coderefinery.github.io/git-collaborative/distributed/

Tips for Using Forks

- **Set remotes to help keep track of original and forked repositories**

```
git remote add upstream https://github.com/project_team/project.git
git remote add my-fork https://github.com/me/project.git
```

- **View remotes with**

```
git remote -v
```

- **Retrieve upstream changes with**

```
git fetch upstream
```

- **Merge upstream changes into your fork's main branch**

```
git checkout main # this is the main branch for your fork
git merge upstream/main
```

- **Retrieve any upstream changes and merge into your main and feature branch frequently, especially before you create a Pull Request**

Guidelines for establishing a Git Workflow

- **Communicating it to your collaborators**
  - **Include workflow in Contributing Guide**
  - **Establish conventions for branch naming (issues, major/minor versions)**
- **Enforce workflow**
  - **Branch protections**
  - **Limiting who can push/merge**
  - **Testing & review requirements**
- **Adopt what is good for your team**
  - **Consider team culture and project challenges**
  - **Assess what is and isn't feasible/acceptable**
  - **Start with simplest and add complexity where and when necessary**
- **Be careful in shared spaces**
  - **Each person should work within their own clone**
  - **Separate "production" clone for running**

David M. Rogers, Collaborative Software Development in Better Scientific Software tutorial, in ISC High Performance, Hamburg, Germany and online, 2023. DOI: 10.6084/m9.figshare.22790762.

Backup

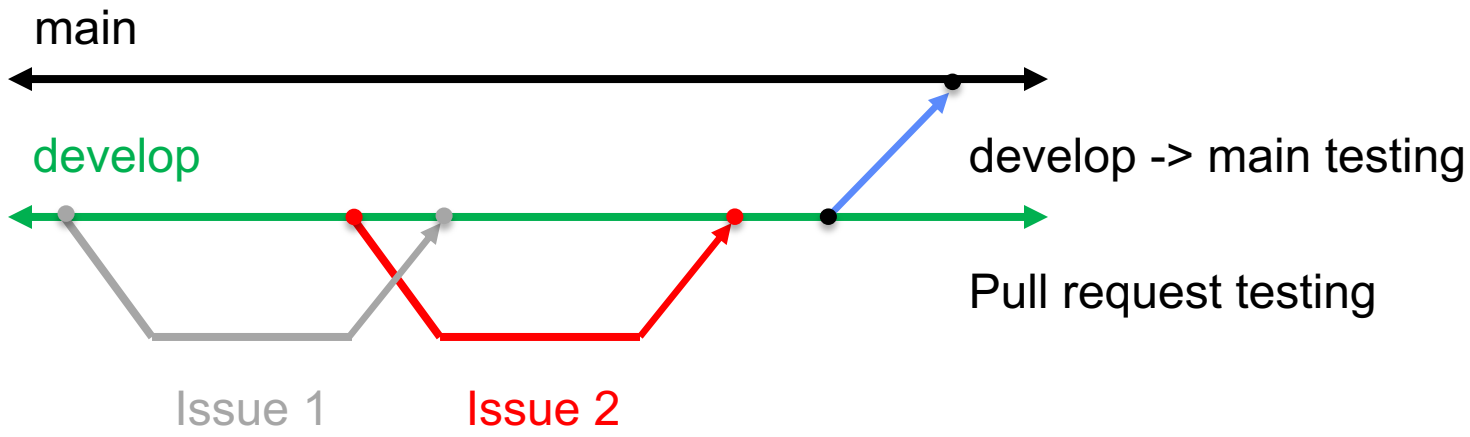
Current Trilinos Workflow

Test-driven workflow

- **Feature branches start and end with develop**
- **All changes to develop must come from GitHub pull requests**
- **Feature branches are merged into develop only after passing pull request test suite**
- **Change sets from develop are tested daily for integration into main**

Workflow designed so that

- **All commits in main are in develop**
- **Merge conflicts exposed when integrating into develop**
- **Merge conflicts never occur when promoting to main**



https://trilinos.github.io/

David M. Rogers, Collaborative Software Development in Better Scientific Software tutorial, in ISC High Performance, Hamburg, Germany and online, 2023. DOI: 10.6084/m9.figshare.22790762.

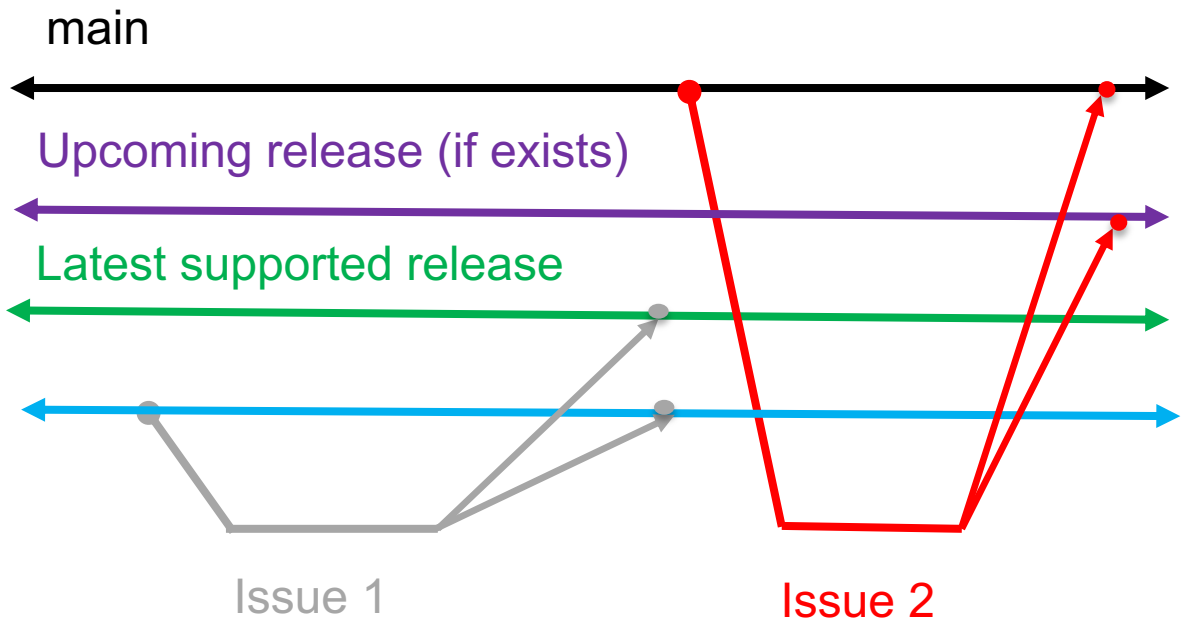
Current Open MPI Workflow

Versioning:

Major versions - break compatibility

Minor versions – visible

Releases correct issues



https://www.open-mpi.org

Workflow designed so that

- **Support two most recent releases**
- **Issues are addressed on all applicable branches**
- **All PR's reviewed by at least one core develope**
- **Main and supported branches work at all times**
- **Developers work on main or feature branches depending on complexity of the changes**

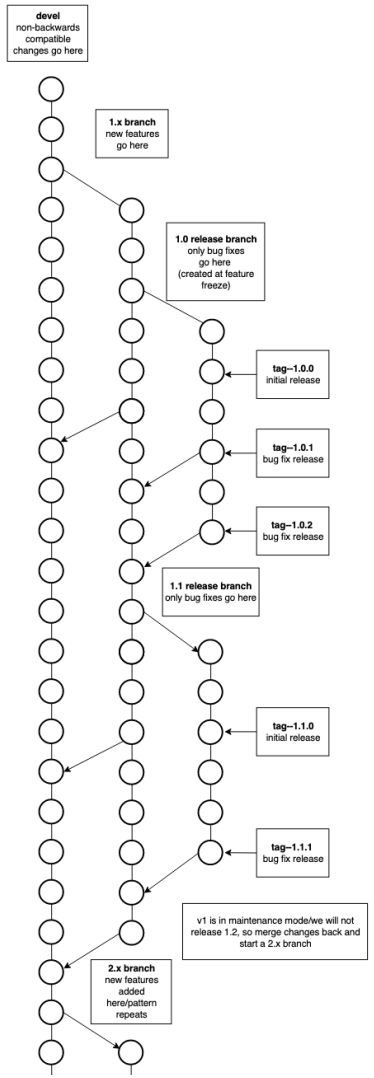
Testing

- **CI testing on PR's for any branch using Jenkins (limited set of compilers, hardware, tests)**
- **Nightly testing on all branches using community-built MTT framework (more complex set of compilers, hardware, tests)**
- **Additional testing for release candidates**

David M. Rogers, Collaborative Software Development in Better Scientific Software tutorial, in ISC High Performance, Hamburg, Germany and online, 2023. DOI: 10.6084/m9.figshare.22790762.

Current FleCSI Workflow

https://flecsi.github.io/flecsi



Versioning:

Incompatible - **devel** branch breaks compatibility with previous versions

Feature (1, 2 …) named for major version

Release - **(1.x, 2.x …)** named for major.minor version, correct issues, tags used for bug fixes.

Workflow designed so that

- All supported branches work at all times
- Merge Requests are tested and reviewed

Testing

- Customized unit-testing framework based on Google Test
- Special *gitlab-ci* branch - images and configuration files

David M. Rogers, Collaborative Software Development in Better Scientific Software tutorial, in ISC High Performance, Hamburg, Germany and online, 2023. DOI: 10.6084/m9.figshare.22790762.