Institute For Molecular Science
Department Of Photo-molecular Science
OHMORI GROUP

Git @ Ohmori Group - Lesson 2

Branching, remote use with GitHub and collaboration

Giorgio Micaglio

Italian Association of Physics Students
Institute for Molecular Science - Ohmori Group

July 25, 2025

Overview

Branch Management

Giorgio Micaglio, Git @ Ohmori Group 2025

What are Branches?

Branches are a powerful feature of Git that allow us to develop new features in isolation, without affecting the project's main branch.
This allows us to experiment and test new ideas without compromising the stability of the main codebase.

In practice

A branch is like having a parallel universe where you can make all your changes to the project without modifying the original.
It's very useful when developing a new feature that should only be added to the codebase once it's stable.

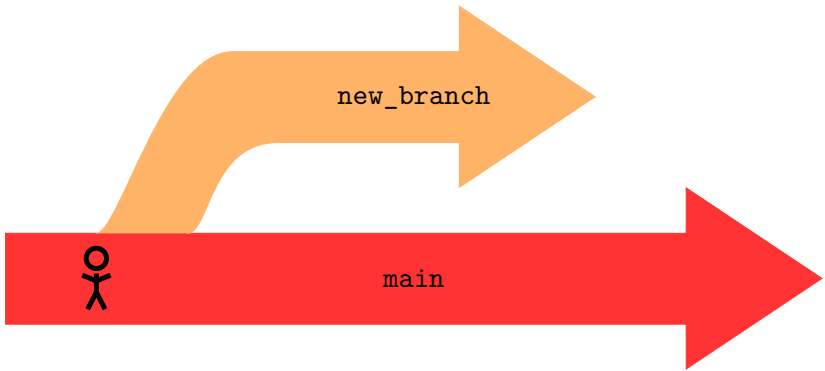
Initializing a Branch

To initialize a new branch, you need to run the command `git branch new_branch_name`.

Note:

When you initialize a Git repository, the *master* branch is created. GitHub, however, uses `main` as the default branch. In both cases, it's possible to change the default branch if needed.

```
git branch new_branch
```

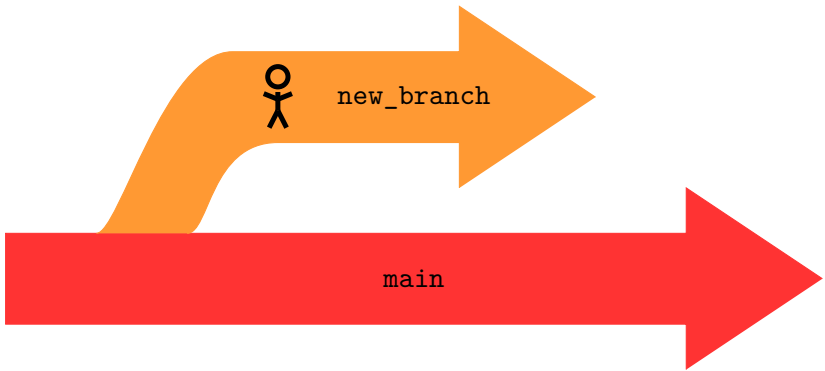


How to move between branches

The `git checkout new_branch` command moves the project to the `new_branch` branch.

This will change the files in the folder if the two branches have differences (they are likely different, otherwise what are you doing with them?).

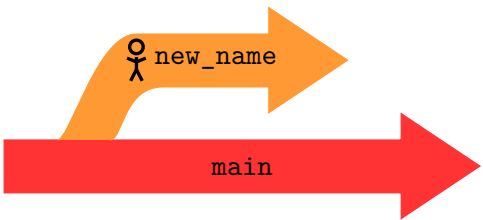
```
git checkout new_branch
```



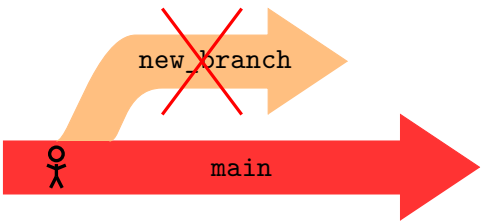
Branch Operations

- `git branch -m new_project` will change the name of the brach you are in to `new_project`;
- `git branch -m old_name new_name` will instead change the name of the branch `old_name`;
- `git branch -d new_branch` will delete the branch `new_name`.

`git branch -m new_name`



`git branch -d new_branch`



Working with more branches

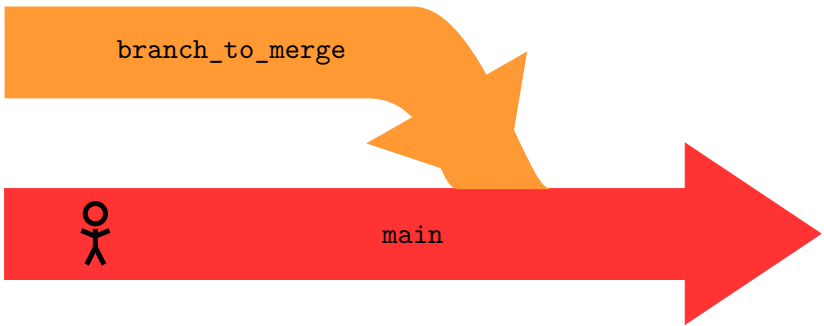
By using branches, we can develop new features in isolation from the main branch of the project. This allows us to work on multiple aspects of the software simultaneously without interference.

When we have completed the development of a new feature on a separate branch, we can merge the changes to the main branch of the project using the command

```
git merge branch_name
```

from the main branch.

```
git merge branch_to_merge
```



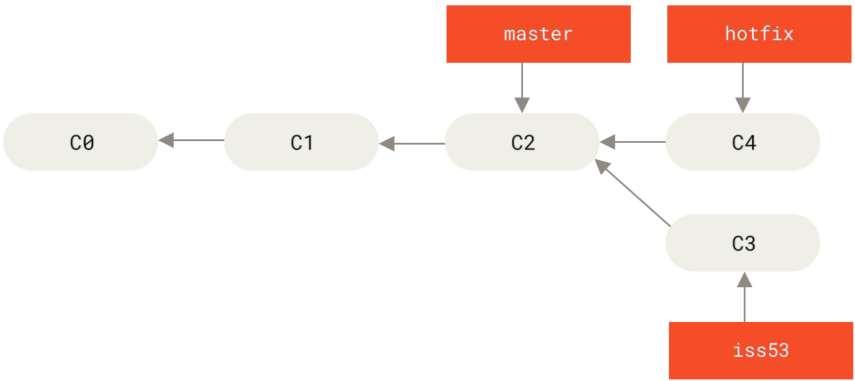
Types of Merging

There are various types of merging:

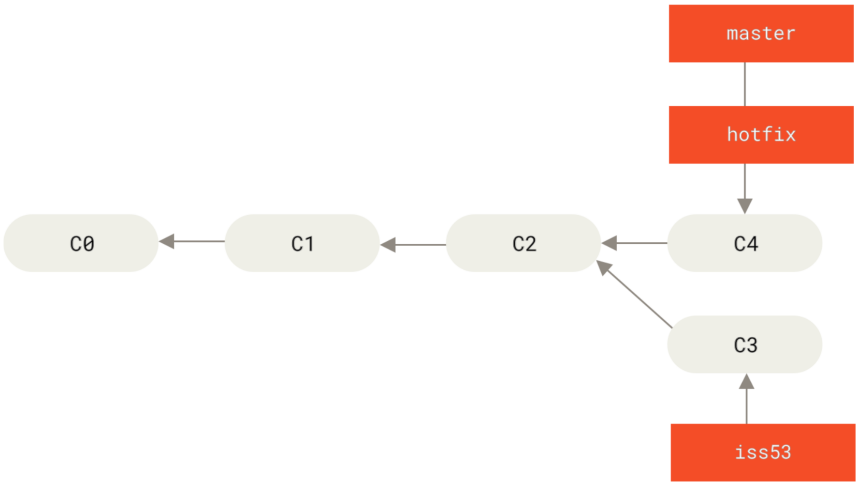
- fast-forward
- 3-way (for this type Git uses several strategies, such as `ort` and `recursive`)
- rebasing (which is not a merge)

For more details: `https://git-scm.com/docs/git-merge`

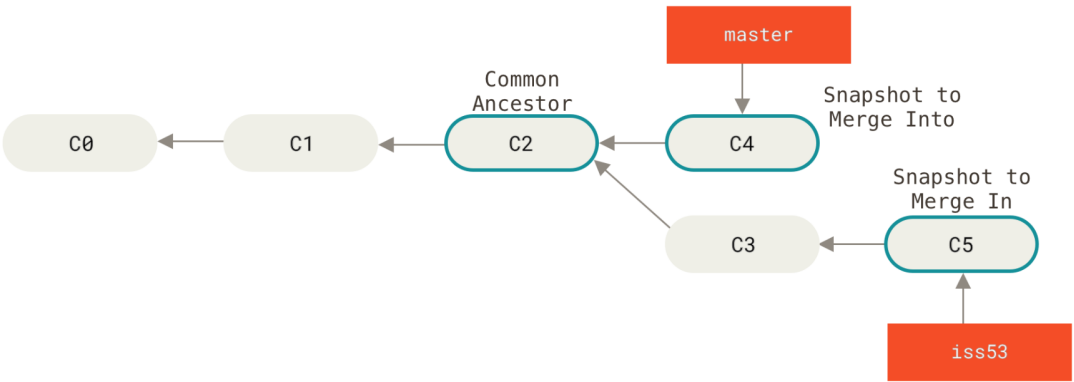
Types of Merging: fast-forward



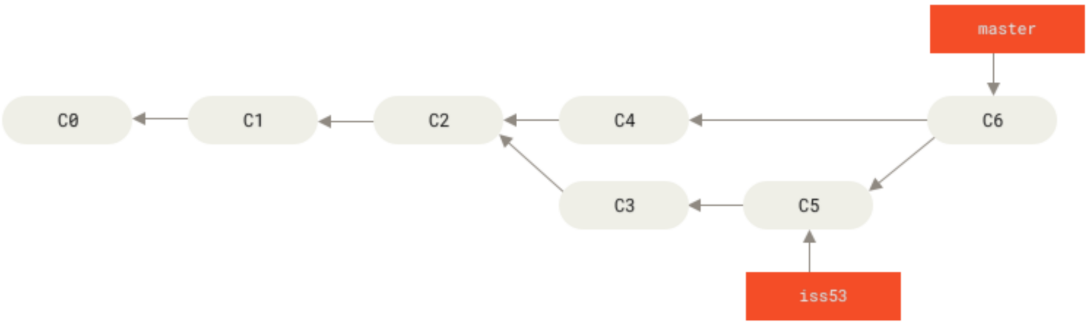
Types of Merging: fast-forward



Types of Merging: 3-way merge



Types of Merging: 3-way merge



Basic merge conflicts

During a branch merge, conflicts may occur if the same lines of code are modified differently in different branches. We have to resolve these conflicts manually before we can complete the merge.

Git will highlight such changes so that the developer can detect them and thus resolve them more easily.
The `git status` command shows which files are in conflict and each file needs to be edited manually.

Basic merge conflicts

Conflicts are referred to as:

```
     <<<<<<< HEAD:index.html
contact : email.support@github.com
=======
please contact us at support@github.com
>>>>>>> iss53:index.html
```

Once the conflicting files have been modified, staging and committing of these files must be performed and finally merged again.

Interesting

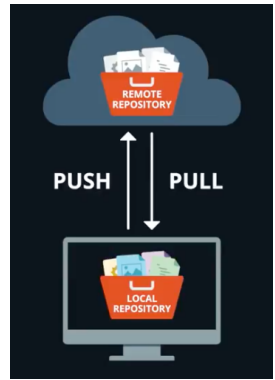
The commands `git diff` and `git log` can also be used between various branches.

Remote Work

Giorgio Micaglio, Git @ Ohmori Group 2025

Remote Repositories

- These are versions of a Git project hosted on the Internet or on a network
- You can configure multiple remote repos per project
- Collaborating with others means sharing your work via one of these remotes
- This is done through **push** (uploading) and **pull** (downloading) actions
- In this course, we will use remote repos hosted on GitHub



GitHub

- One of the largest Git repo hosts, a central collaboration point for millions of developers
- Used by many open-source projects for hosting, issue tracking, code review, and more

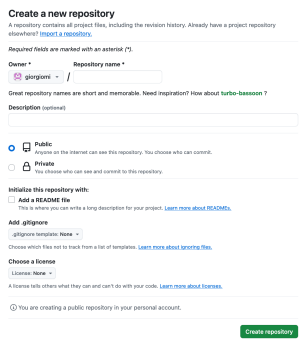
Important

Registration and SSH configuration (lesson 1) are required. You can check with:

```
ssh -T git@github.com
```

Creating a Remote Repository

To start working remotely, we need to create a remote repository on GitHub.
Go to GitHub's homepage (`https://github.com`) and click on **New** at the top left.
A screen like this will appear:



Creating a Remote Repository

- Enter the repository name
- Choose between public (visible to everyone, read-only) and private repo
- DO NOT initialize the repo with `README.md` (leave the box unchecked)
- DO NOT initialize with `.gitignore` (choose `.gitignore template: None`)
- DO NOT initialize with a license (choose `License: None`)

Creating a Remote Repository

- Enter the repository name
- Choose between public (visible to everyone, read-only) and private repo
- DO NOT initialize the repo with `README.md` (leave the box unchecked)
- DO NOT initialize with `.gitignore` (choose `.gitignore template: None`)
- DO NOT initialize with a license (choose `License: None`)

No worries!

These actions can be done later

- Click the green button at the bottom right
- The new remote repo will be at: `https://github.com/user-name/repo-name`

Initializing a Remote Repository

Now that you've created the remote repo, you can proceed in two ways:

1. Initialize the remote repo from a Git local repo you already have
2. Clone the (empty) remote repo locally

Let's look at the first method (**initializing**). Run these commands:

- `git branch -M main`
- `git remote add origin git@github.com:user-name/repo-name.git`
- `git push -u origin main`

The first renames your main branch to `main`, the second adds a remote connection to the repo you just created, and the third pushes your local contents to the server.

Cloning a Remote Repository

Now let's look at how to **clone** a remote repo from GitHub to your local machine.

Important

Use this method only if you haven't already initialized a local project

To do so, use:

```
git clone git@github.com:user-name/repo-name.git
```

This command clones the remote repo into a new directory named `repo-name`. You can clone any public repository on GitHub!

Workflow - remote

Let's now look in detail at how to work with remote repos. First, move into the repo directory using `cd`.

The command

```
git remote
```

shows the remote servers configured for the local repo. In both methods above, this should return `origin`, the default name Git gives to the cloned server.

Workflow - fetch

To fetch data from the server, run:

```
git fetch <remote>
```

This command downloads changes from the remote repo (added since your last fetch) to your local machine.

Workflow - fetch

To fetch data from the server, run:

```
git fetch <remote>
```

This command downloads changes from the remote repo (added since your last fetch) to your local machine.

Important

`fetch` only downloads–no merge is done. You must run `git merge` separately, or else…

Workflow – pull

The command

```
git pull
```

automatically fetches and merges from the remote repo into the local one.

Workflow - pull

The command

```
git pull
```

automatically fetches and merges from the remote repo into the local one.

Important

The first time you run this command, you'll see a warning telling you to configure `pull.rebase`
To set Git's default behavior (fast-forward if possible, otherwise merge commit), run:
`git config --global pull.rebase "false"`

Workflow – push

To send data to the server, use:

```
git push <remote> <branch>
```

- Use this when your project is in a state worth sharing
- Every committed change is pushed to the server, and the server's branch is updated (moved to your latest commit)

Workflow - push

To send data to the server, use:

```
git push <remote> <branch>
```

- Use this when your project is in a state worth sharing
- Every committed change is pushed to the server, and the server's branch is updated (moved to your latest commit)

Important

`git push` only works if no collaborator has pushed in the meantime! We'll see how to resolve this later...

Workflow - push

Collaboration

Centralized Workflow

- This is the most common workflow for simple collaboration, although there are others (integration manager, dictator-lieutenants, etc.)
- It's convenient for small teams (2–4 people), but not limited to that
- It's familiar to people who have never used Git before
- It consists of:
  1. A **central hub** (or **repository**): accepts code and allows developers to sync with it
  2. **Nodes**: the developers themselves, who interact only with the hub

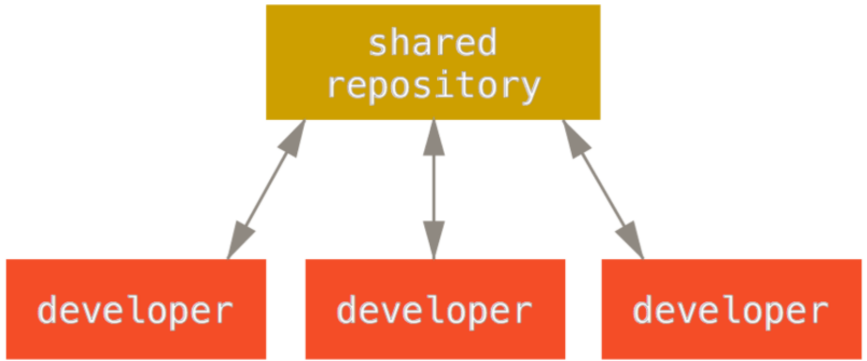
Centralized Workflow

- This is the most common workflow for simple collaboration, although there are others (integration manager, dictator-lieutenants, etc.)
- It's convenient for small teams (2–4 people), but not limited to that
- It's familiar to people who have never used Git before
- It consists of:
  1. A **central hub** (or **repository**): accepts code and allows developers to sync with it
  2. **Nodes**: the developers themselves, who interact only with the hub

Note

In this workflow, nodes only interact with the hub—not with each other.

Centralized Workflow



Workflow Example

- Alice and Bob both clone the central hub and make changes; Alice pushes first and has no issues
- Bob, however, must:
  1. Fetch the remote repository, which now includes Alice's work
  2. Merge it locally
  3. Resolve any conflicts without overwriting Alice's changes

Workflow Example

- Alice and Bob both clone the central hub and make changes; Alice pushes first and has no issues
- Bob, however, must:
  1. Fetch the remote repository, which now includes Alice's work
  2. Merge it locally
  3. Resolve any conflicts without overwriting Alice's changes

Note

If Bob tries to push before fetching and merging, the server will reject it.

Workflow Example

- Alice and Bob both clone the central hub and make changes; Alice pushes first and has no issues
- Bob, however, must:
  1. Fetch the remote repository, which now includes Alice's work
  2. Merge it locally
  3. Resolve any conflicts without overwriting Alice's changes

Note

If Bob tries to push before fetching and merging, the server will reject it.

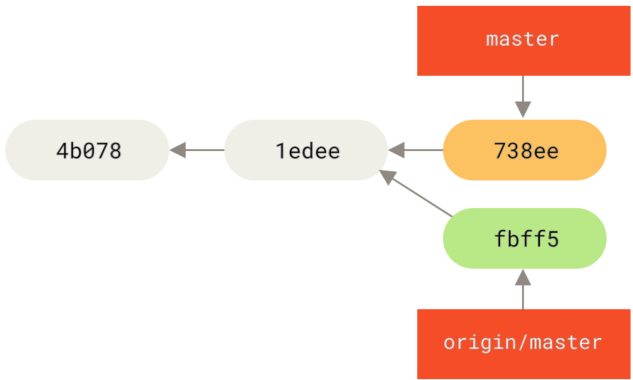
Very Important

DO NOT use `git push --force`, or you'll delete your collaborators' commits!

Workflow Example

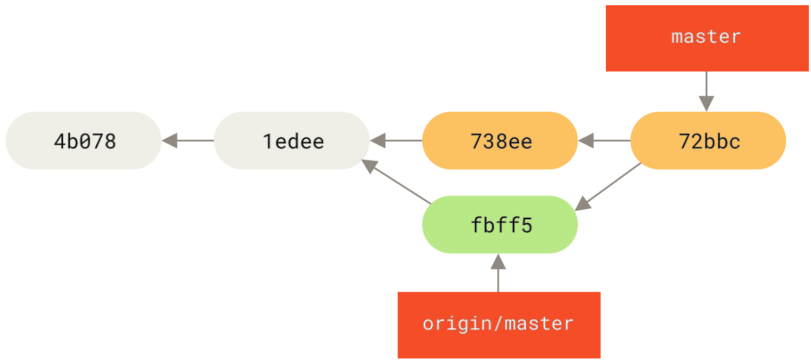
Bob's Repo - fetch

Bob performs a fetch from the remote repository, which contains Alice's commit (`fbff5`)



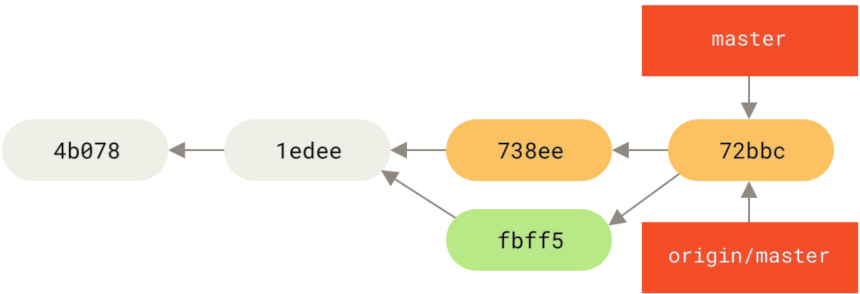
Bob's Repo - merge

Bob merges locally (`72bbc`)



Bob's Repo - push

Bob pushes his changes

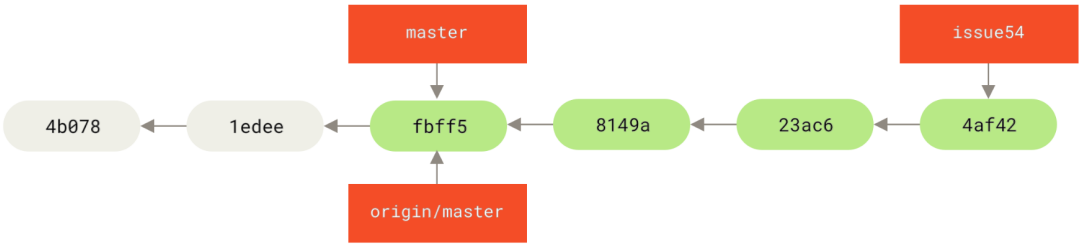


Workflow Example

- Meanwhile, Alice has created a new branch `issue` and made 3 commits
- Knowing Bob has pushed his work, she fetches it to take a look
- Alice switches to the `master` branch
- Merges `issue` into `master` (fast-forward)
- Merges `origin/master` (Bob's commit)
- Pushes everything

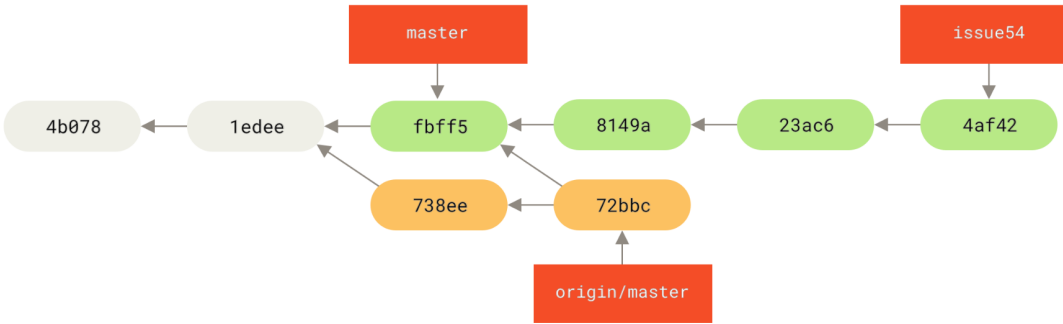
Alice's Repo

Alice has made 3 commits on the `issue54` branch



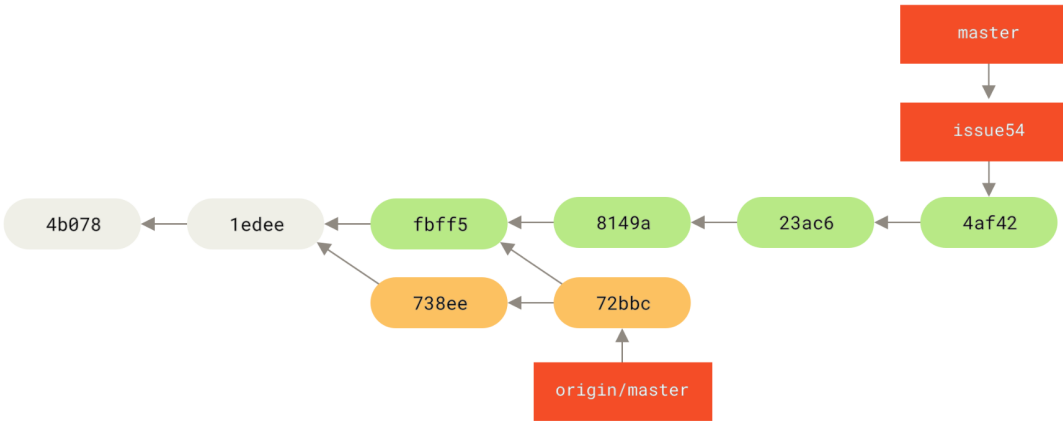
Alice's Repo - fetch

Alice fetches Bob's work



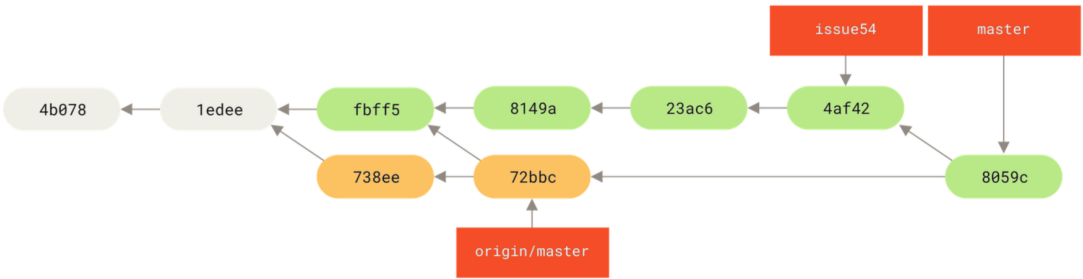
Alice's Repo - fast-forward merge

Alice performs a fast-forward merge of `issue54` into `master`



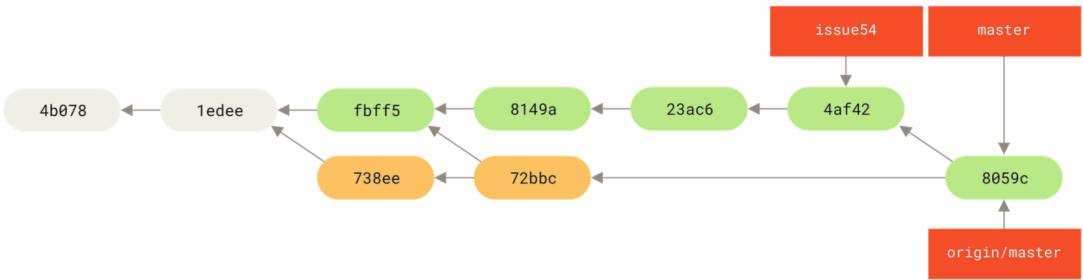
Alice's Repo - 3-way merge

Alice performs a 3-way merge of `origin/master` into `master`

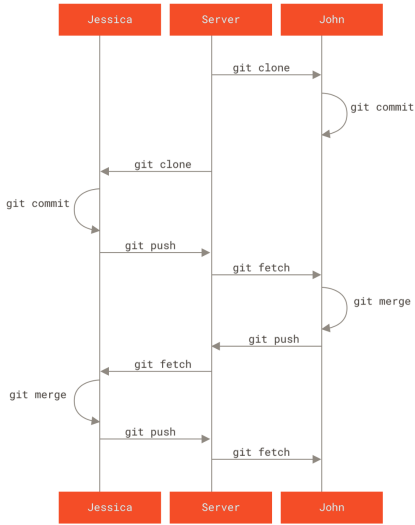


Alice's Repo - push

Alice pushes `master`



Workflow Example (Summary)



Alternatives

- So far, we've only seen local merges
- Alternatively, you can push the branch to the server
- To merge remotely, GitHub offers **pull requests**, which collaborators can review, approve, or reject

Caution

If there are conflicts remotely, you'll need to resolve them remotely!

Exercises

https://ai-sf.it/trento/downloads/git/en_es2.pdf

The End

Giorgio Micaglio

Italian Association of Physics Students
Institute for Molecular Science - Ohmori Group

July 25, 2025