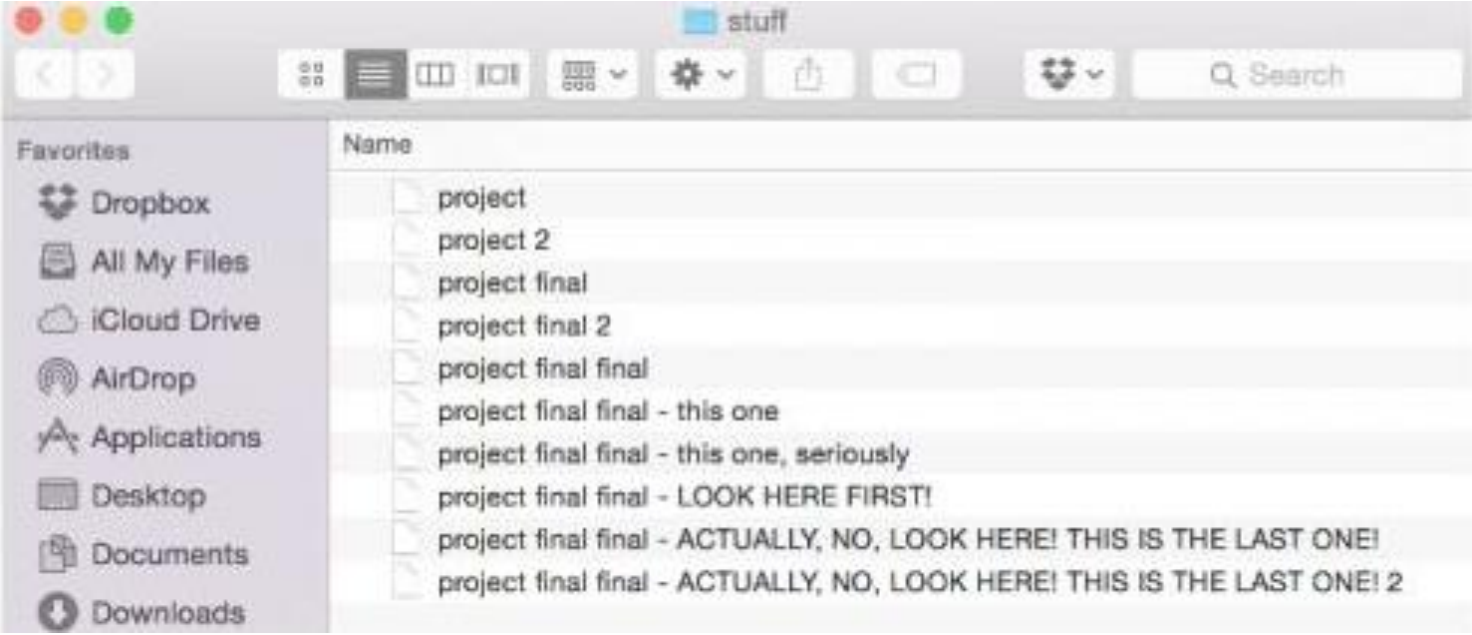
# Git

Boston University CS 506 - Lance Galletti


stuff
Search
Favorites
Dropbox
All My Files
iCloud Drive
AirDrop
Applications
Desktop
Documents
Downloads
Name
project
project 2
project final
project final 2
project final final
project final final - this one
project final final - this one, seriously
project final final - LOOK HERE FIRST!
project final final - ACTUALLY, NO, LOOK HERE! THIS IS THE LAST ONE!
project final final - ACTUALLY, NO, LOOK HERE! THIS IS THE LAST ONE! 2

# Motivation

For each codebase (repository) I own, I want to write code where:

1. Progress loss is minimized
2. Iterating on different versions of the code is easy
3. Collaboration is productive

# GitHub vs Git

**GitHub** --> [browser] a **website** to **backup of your files online**

**Git** --> [terminal] a **version control system**

# Minimal Progress Loss

This is achieved by effectively “backing up your work”.

By creating regular save points (called **commits**) and **pushing** them up to GitHub (from your laptop), if your laptop is destroyed you will only lose the commits you did not upload to GitHub.

# Minimal Progress Loss

This is achieved by effectively “backing up your work”.

By creating regular save points (called **commits**) and pushing them up to GitHub (from your laptop), if your laptop is destroyed you will only lose the commits you did not upload to GitHub.

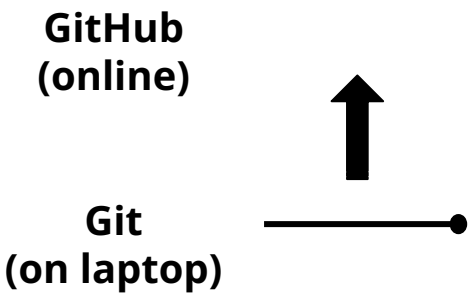
**GitHub**
**(online)**

**Git**
**(on laptop)**

# Minimal Progress Loss

This is achieved by effectively “backing up your work”.

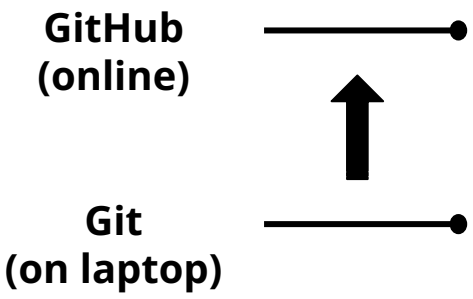
By creating regular save points (called **commits**) and pushing them up to GitHub (from your laptop), if your laptop is destroyed you will only lose the commits you did not upload to GitHub.



# Minimal Progress Loss

This is achieved by effectively “backing up your work”.

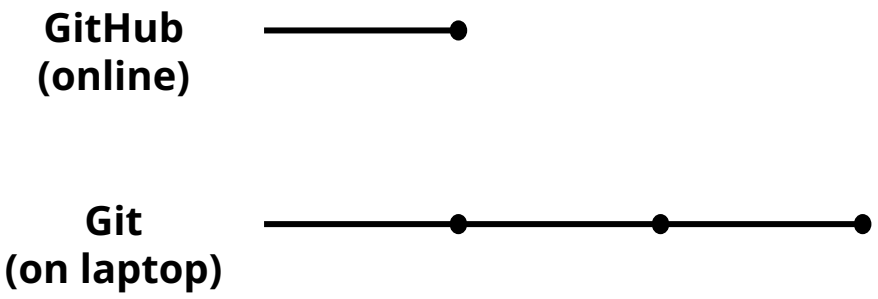
By creating regular save points (called **commits**) and pushing them up to GitHub (from your laptop), if your laptop is destroyed you will only lose the commits you did not upload to GitHub.



# Minimal Progress Loss

This is achieved by effectively “backing up your work”.

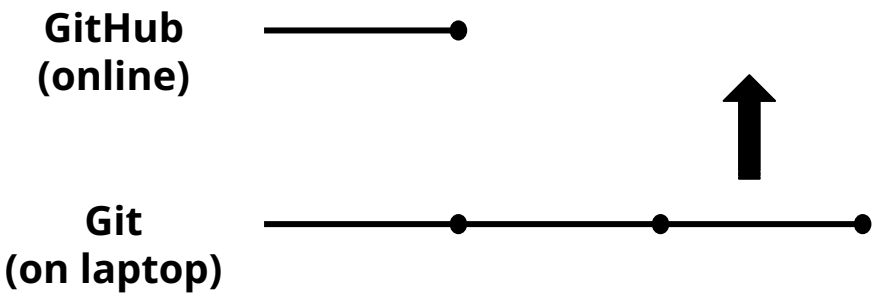
By creating regular save points (called **commits**) and pushing them up to GitHub (from your laptop), if your laptop is destroyed you will only lose the commits you did not upload to GitHub.



# Minimal Progress Loss

This is achieved by effectively “backing up your work”.

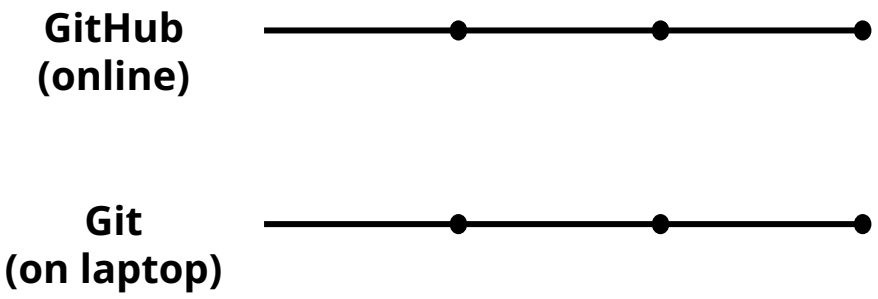
By creating regular save points (called **commits**) and pushing them up to GitHub (from your laptop), if your laptop is destroyed you will only lose the commits you did not upload to GitHub.



# Minimal Progress Loss

This is achieved by effectively “backing up your work”.

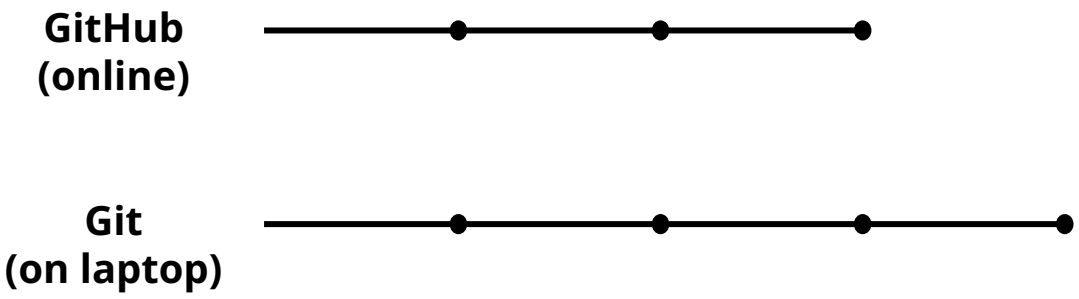
By creating regular save points (called **commits**) and pushing them up to GitHub (from your laptop), if your laptop is destroyed you will only lose the commits you did not upload to GitHub.



# Minimal Progress Loss

This is achieved by effectively “backing up your work”.

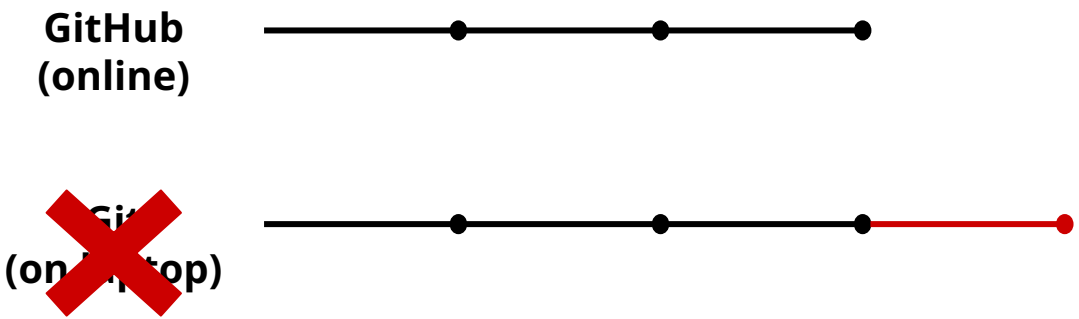
By creating regular save points (called **commits**) and pushing them up to GitHub (from your laptop), if your laptop is destroyed you will only lose the commits you did not upload to GitHub.



# Minimal Progress Loss

This is achieved by effectively “backing up your work”.

By creating regular save points (called **commits**) and pushing them up to GitHub (from your laptop), if your laptop is destroyed you will only lose the commits you did not upload to GitHub.



# Demo

# Iterating on Different Versions

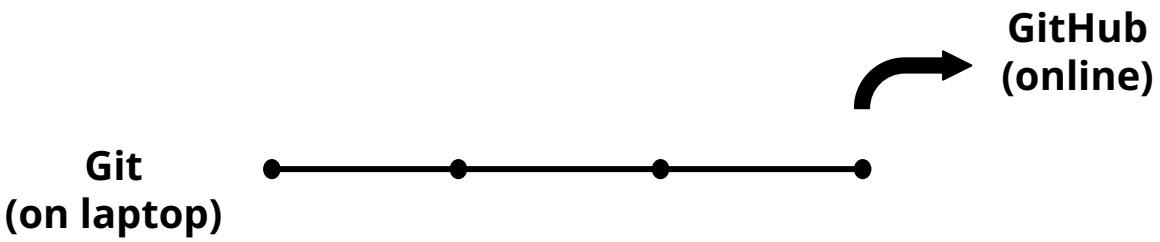
The ease or difficulty of adding a new feature to the code base may depend on the state / version of the codebase.

It may be easiest to add this feature at a specific commit.

# Iterating on Different Versions

The ease or difficulty of adding a new feature to the code base may depend on the state / version of the codebase.

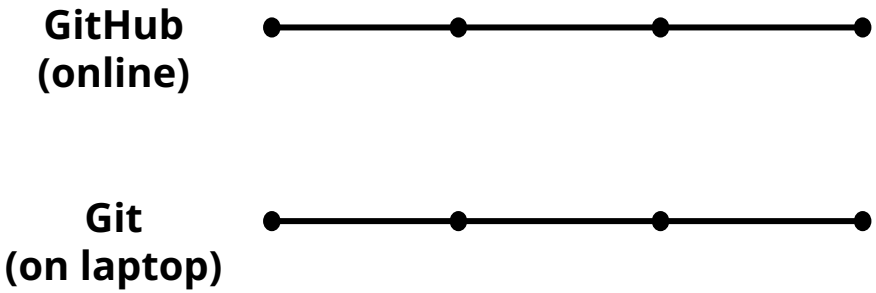
It may be easiest to add this feature at a specific commit.



# Iterating on Different Versions

The ease or difficulty of adding a new feature to the code base may depend on the state / version of the codebase.

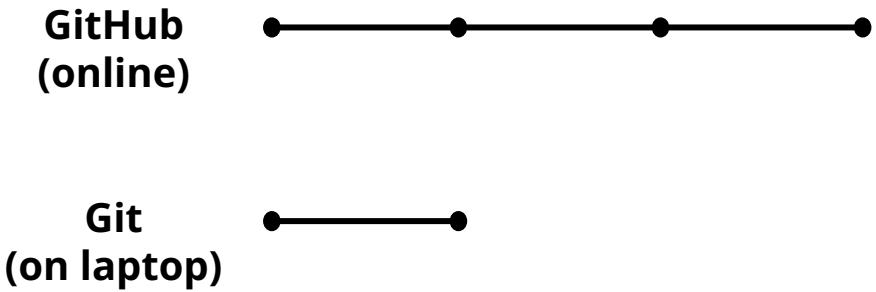
It may be easiest to add this feature at a specific commit.



# Iterating on Different Versions

The ease or difficulty of adding a new feature to the code base may depend on the state / version of the codebase.

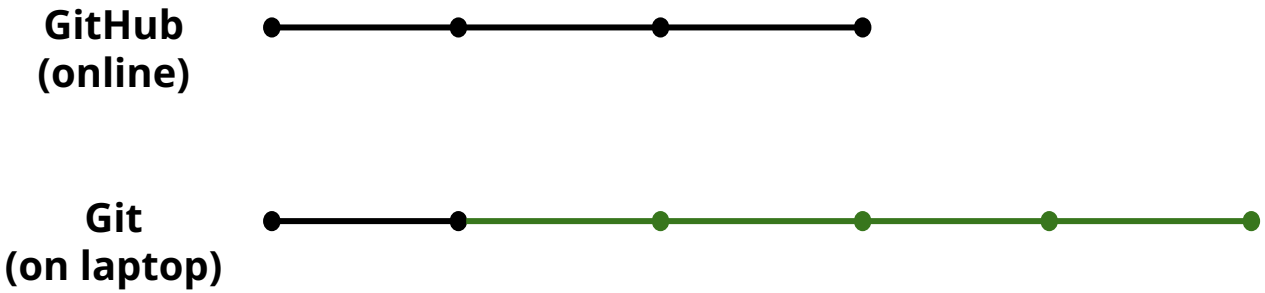
It may be easiest to add this feature at a specific commit.



# Iterating on Different Versions

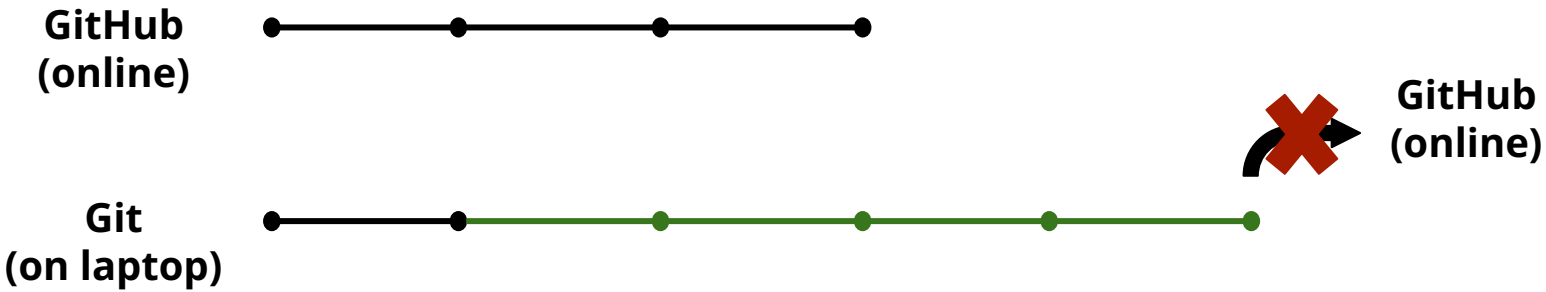
The ease or difficulty of adding a new feature to the code base may depend on the state / version of the codebase.

It may be easiest to add this feature at a specific commit.



# Iterating on Different Versions

What happens now?



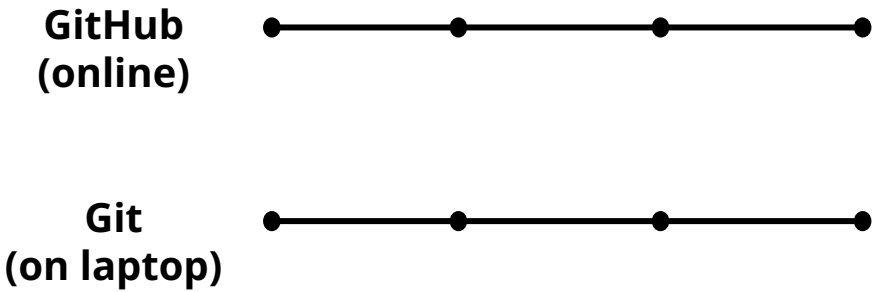
# Iterating on Different Versions

Looks like we need:

1. A way to preserve both versions of history
2. A way to overwrite history if we choose (this is dangerous as we will lose that history)

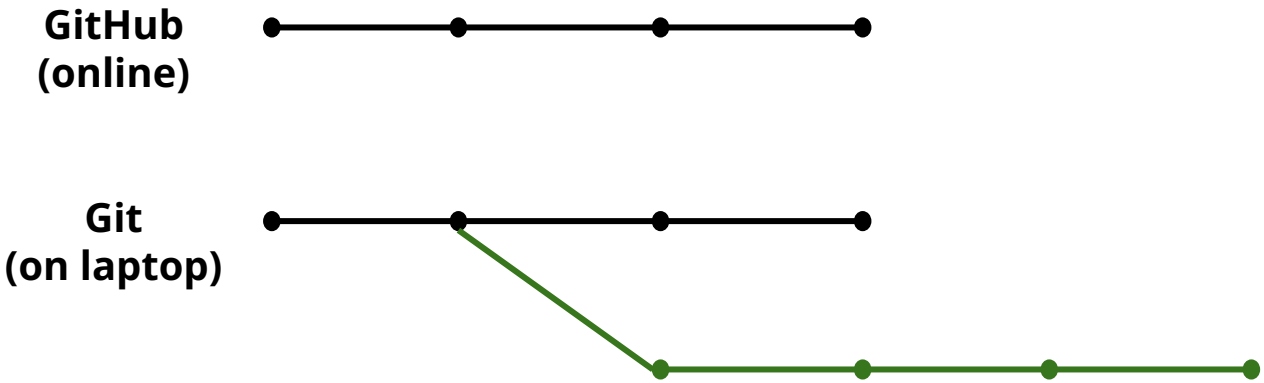
# Iterating on Different Versions

Let's try that again!



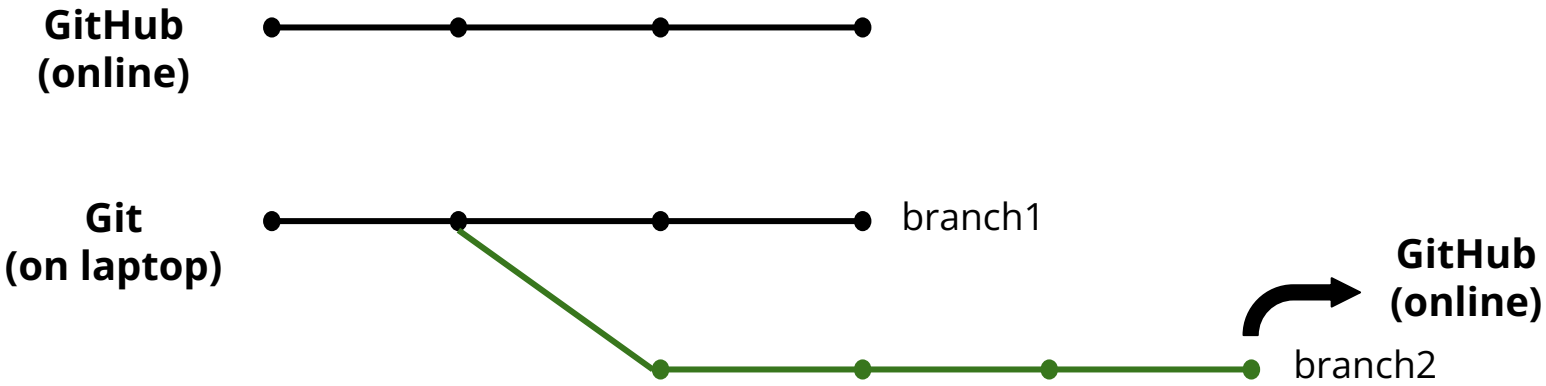
# Iterating on Different Versions

We will **branch** off of that particular commit



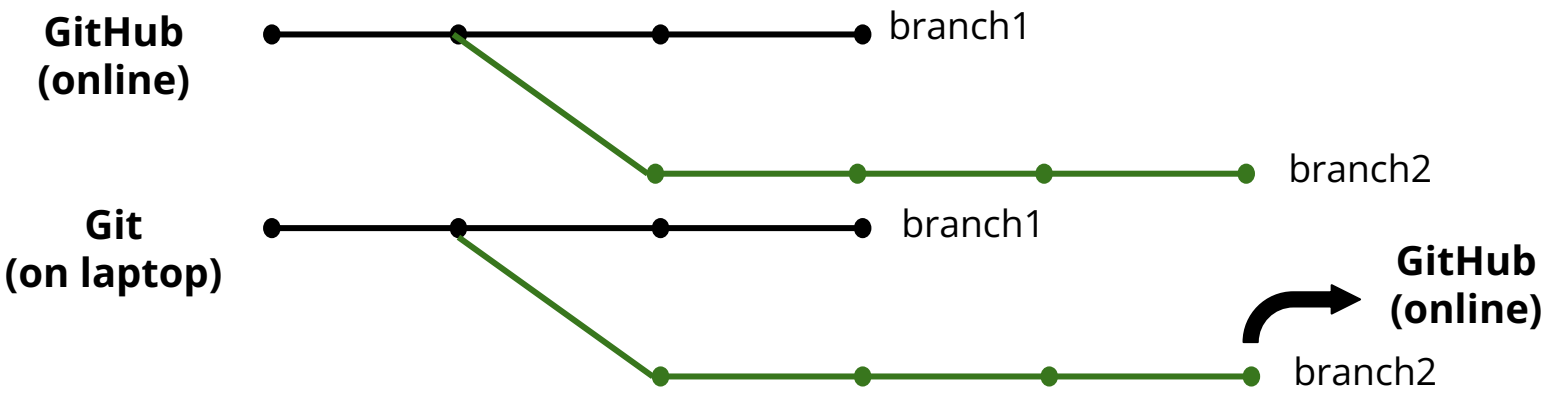
# Iterating on Different Versions

We can push **commits** per **branch**



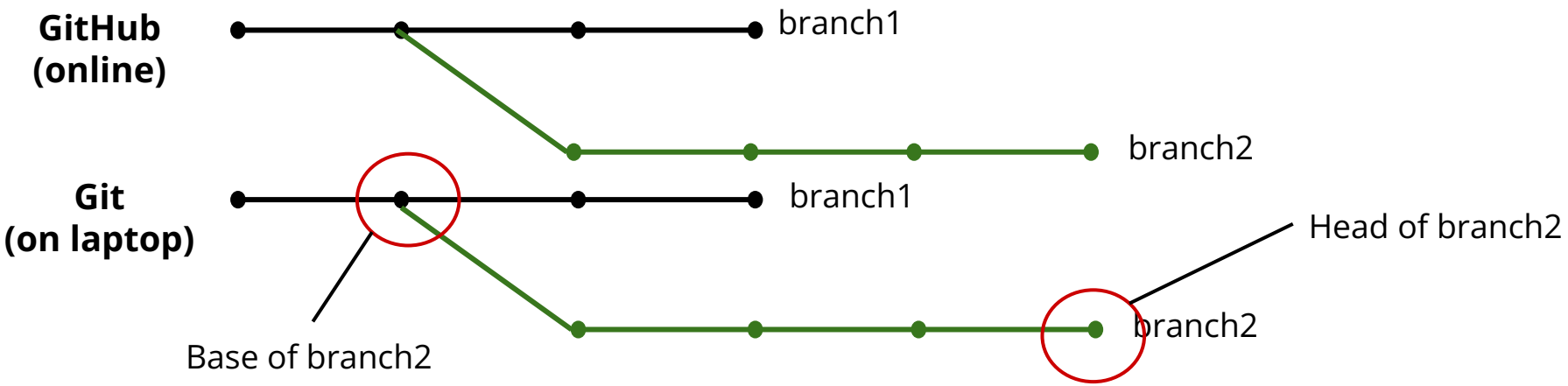
# Iterating on Different Versions

We can push **commits** per **branch**



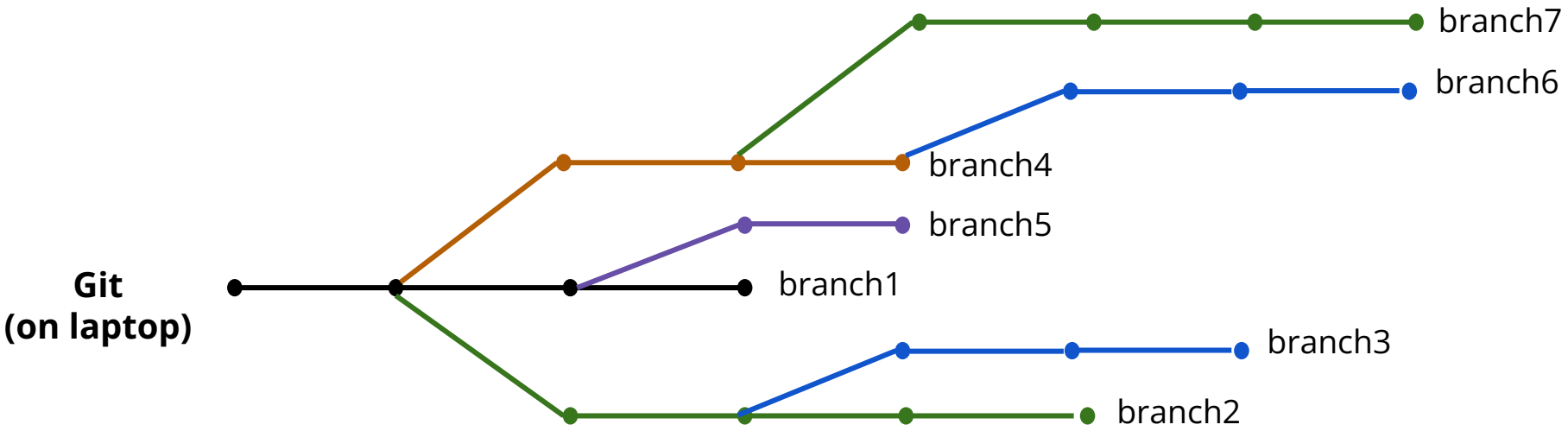
# Iterating on Different Versions

We can push **commits** per **branch**



# Iterating on Different Versions

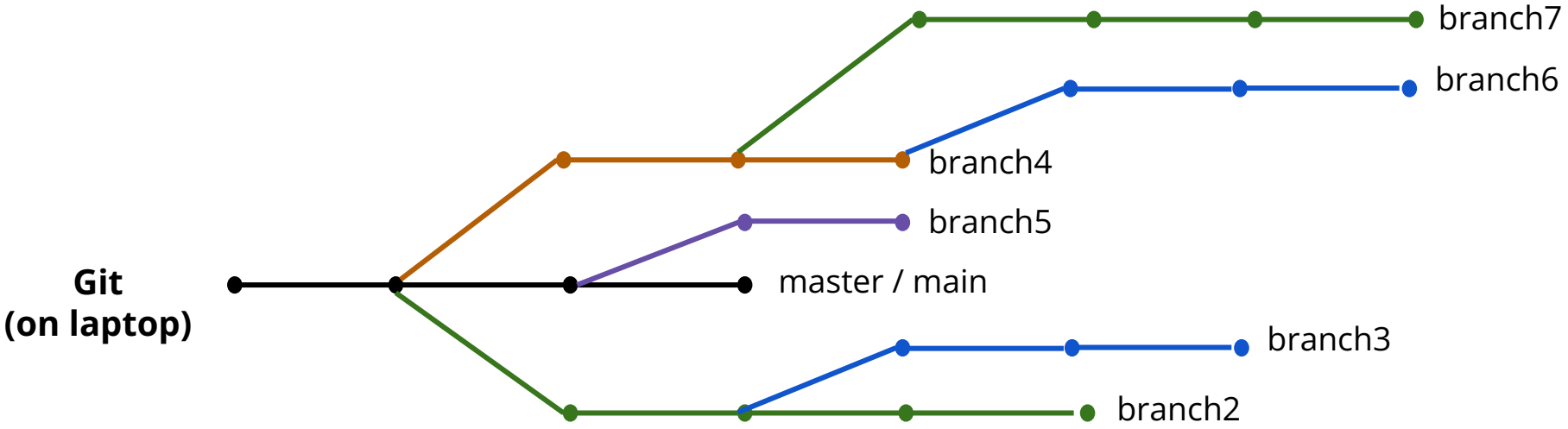
We can create lots of **branches**



# Iterating on Different Versions

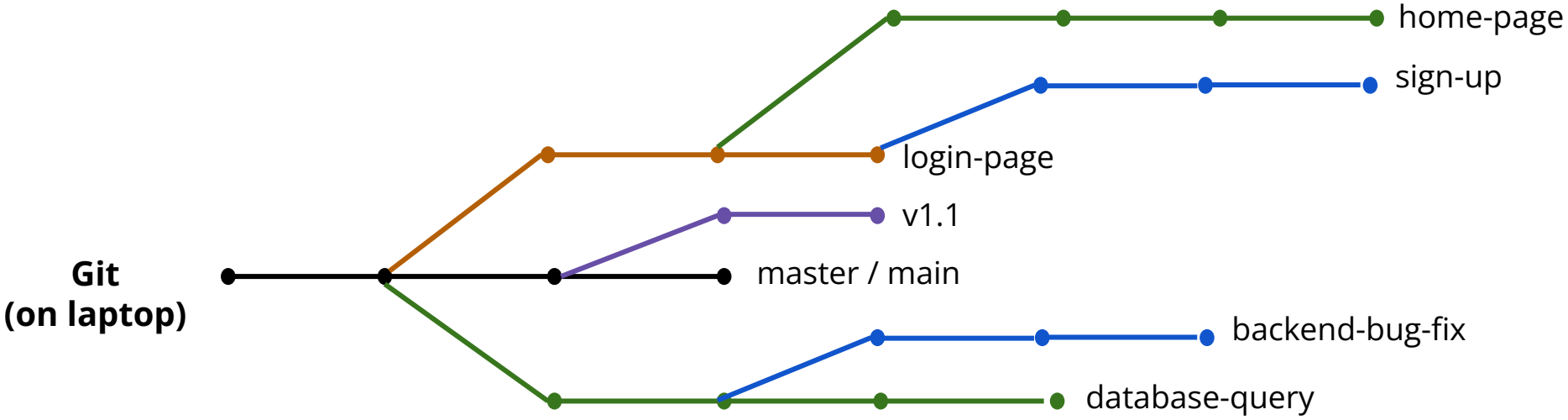
But one branch needs to chosen as the primary, stable branch

This branch is typically called the “master” or “main” branch



# Iterating on Different Versions

Other branches are usually named after either the feature that is being developed on or the major or minor version of the software / product

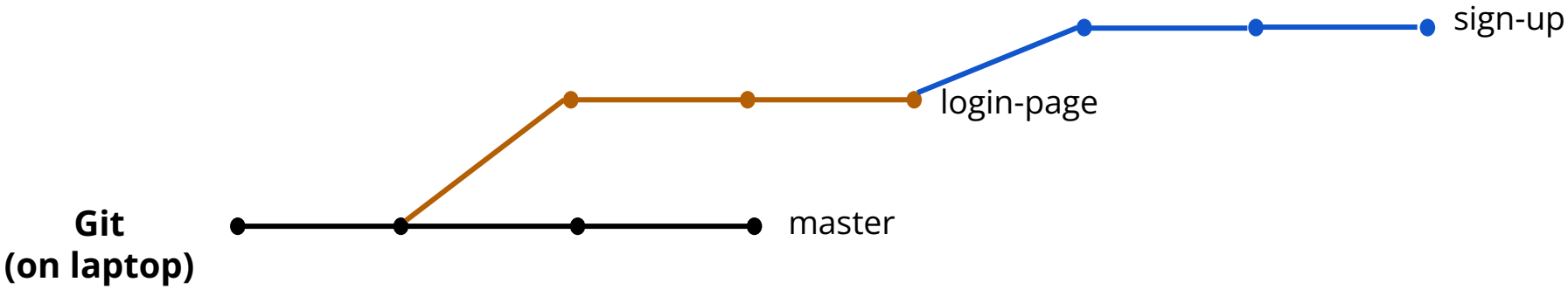


# Iterating on Different Versions

At some point we will want to clean up certain branches by **merging** them with the master / main branch or with each other.

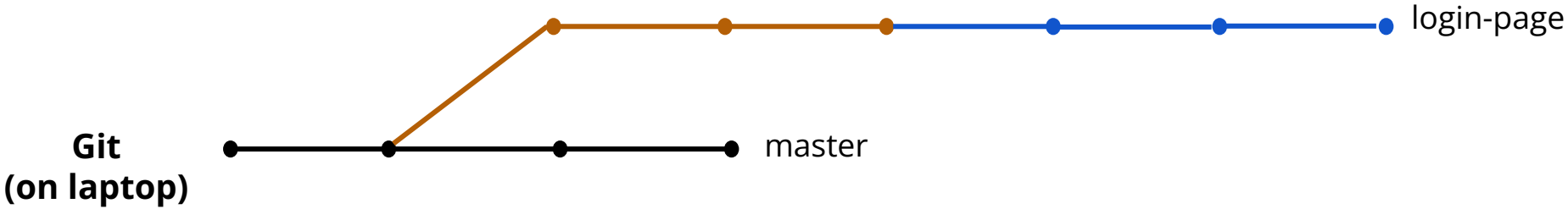
# Iterating on Different Versions

At some point we will want to clean up certain branches by **merging** them with the master / main branch or with each other.



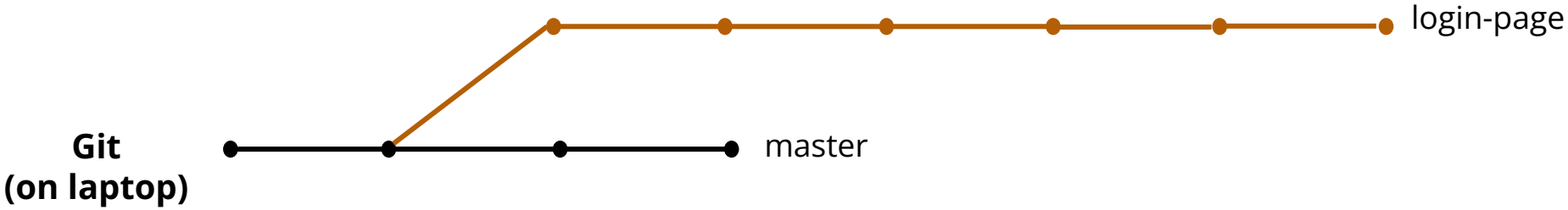
# Iterating on Different Versions

Merging is trivial if the **base** of one branch is the **head** of the other - the changes are “simply” appended.



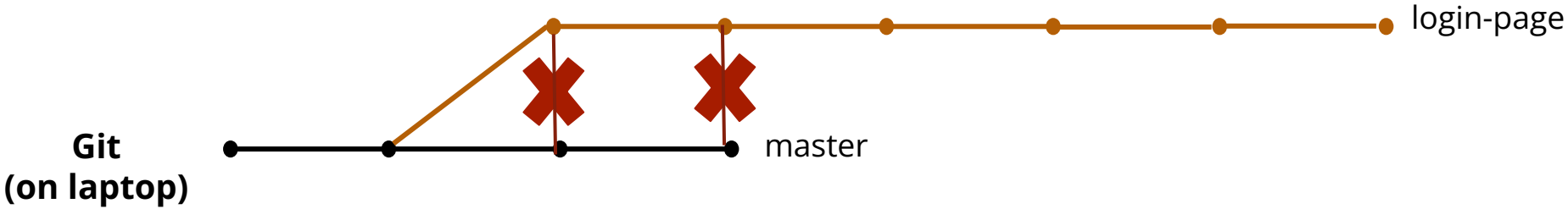
# Iterating on Different Versions

When this is not the case, commits can conflict with each other



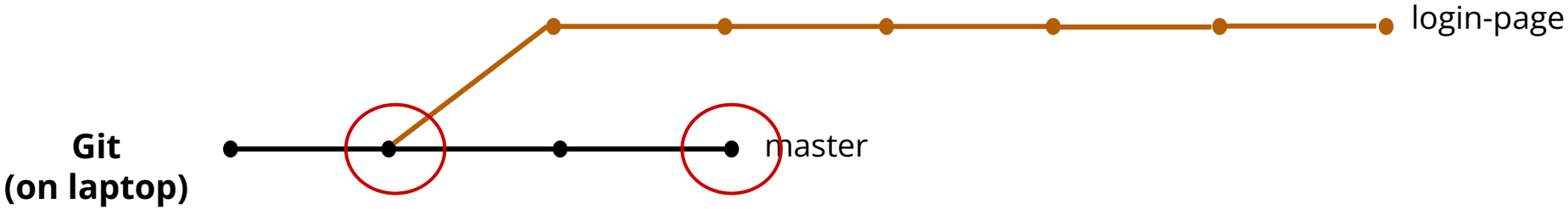
# Iterating on Different Versions

When this is not the case, commits can conflict with each other



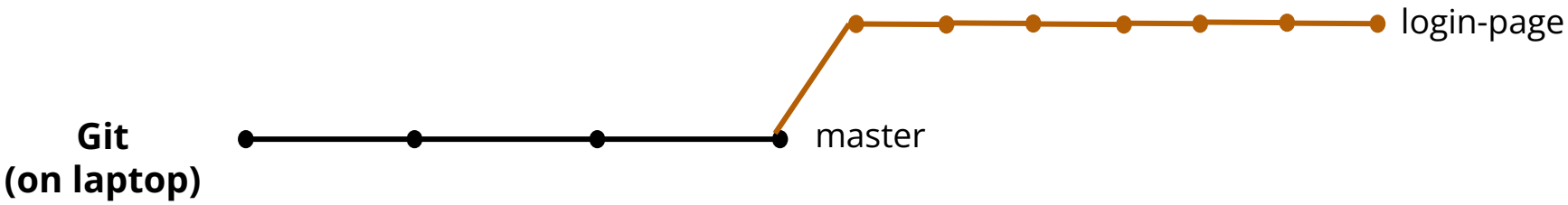
# Iterating on Different Versions

We need to change the **base** of the login-page branch (**rebase**) to be at the **head** of the master branch



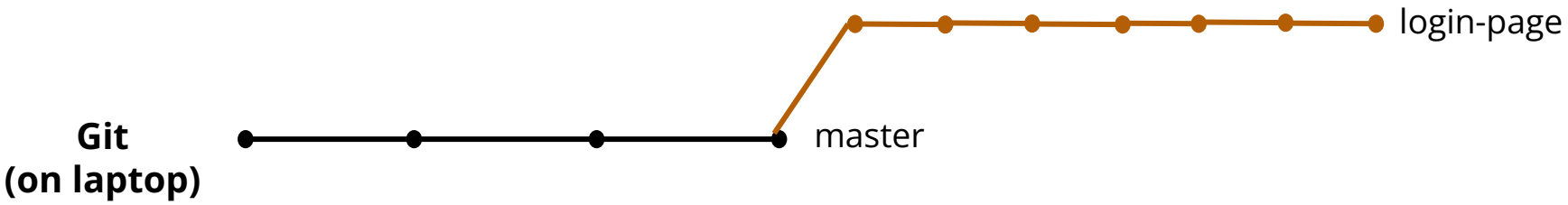
# Iterating on Different Versions

We need to change the base of the login-page branch (**rebase**) to be at the head of the master branch



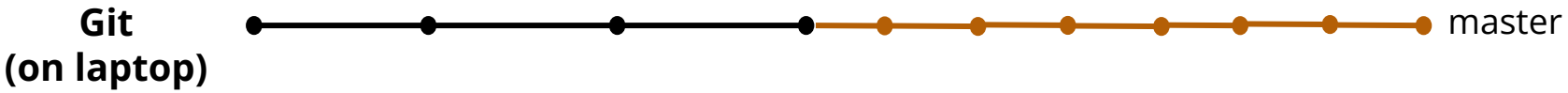
# Iterating on Different Versions

This is not a simple operation! It will often require **manual intervention** to resolve the conflicts.



# Iterating on Different Versions

This is not a simple operation! It will often require **manual intervention** to resolve the conflicts.



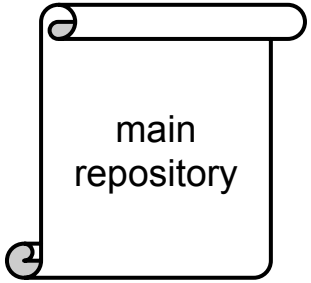
# Demo

# Collaboration

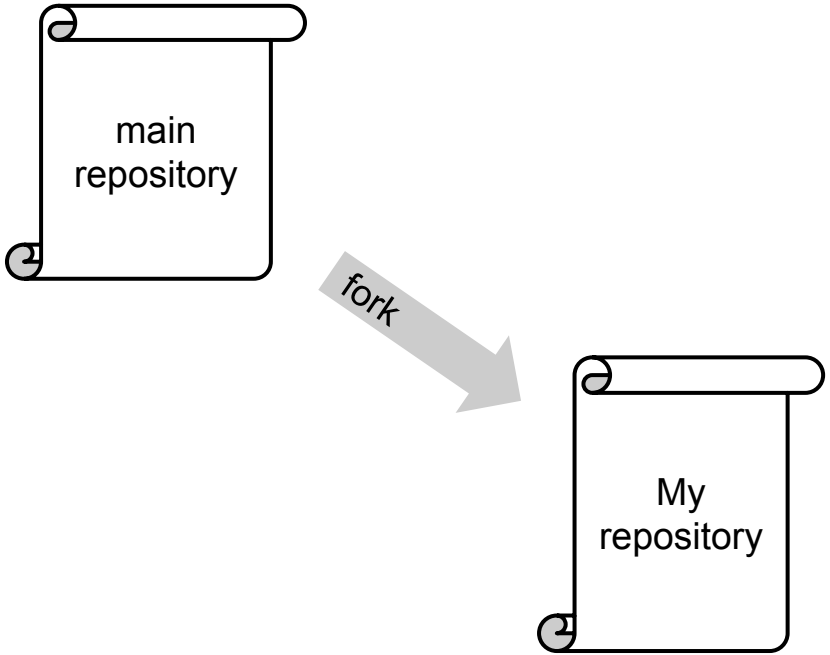
In order to contribute code, collaborators must:

1. Make a copy of (**fork**) the main repository
2. Make all the changes they want to this copy
3. Request that part of their copy be merged into the main repository via a **Pull Request** (PR)

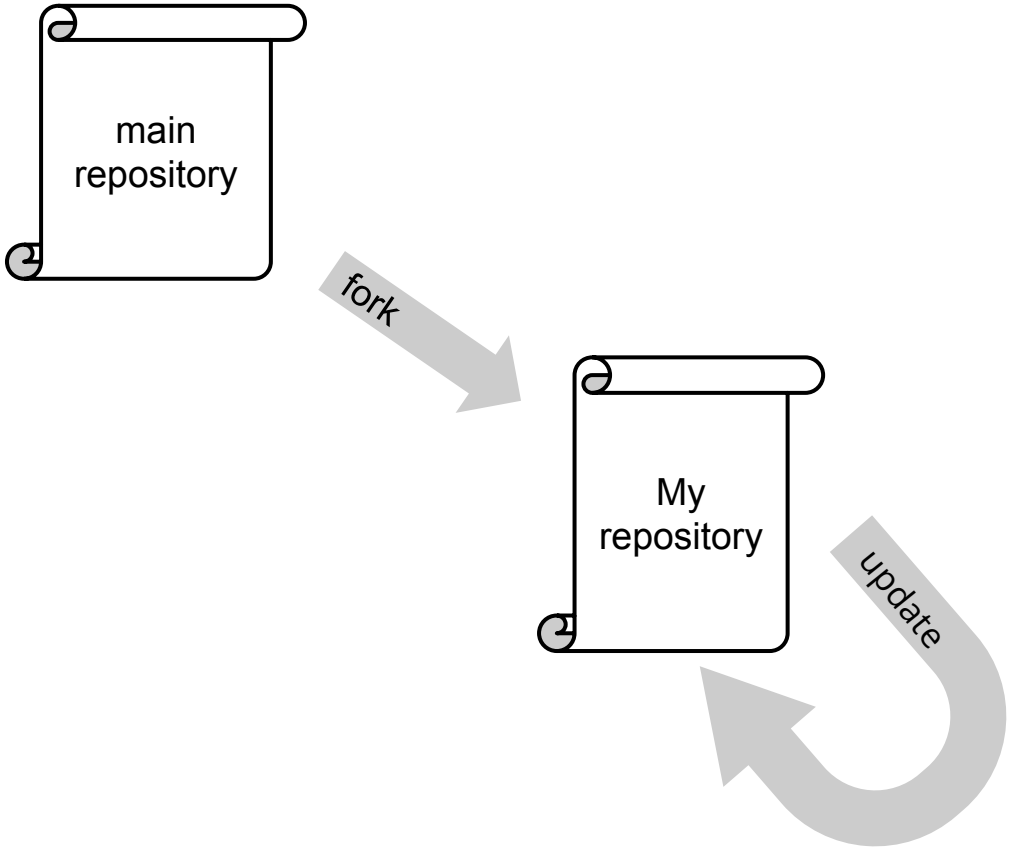
# Collaboration



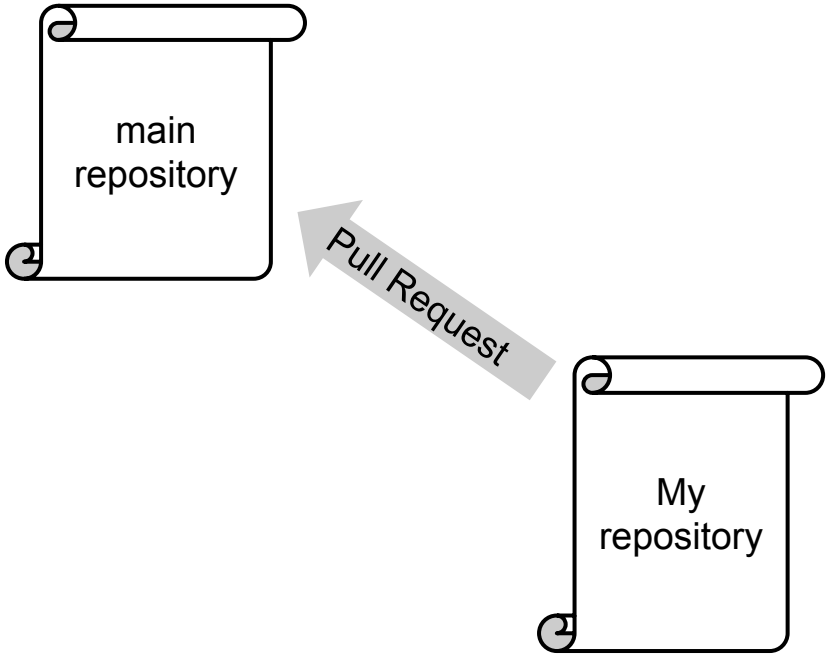
# Collaboration



# Collaboration



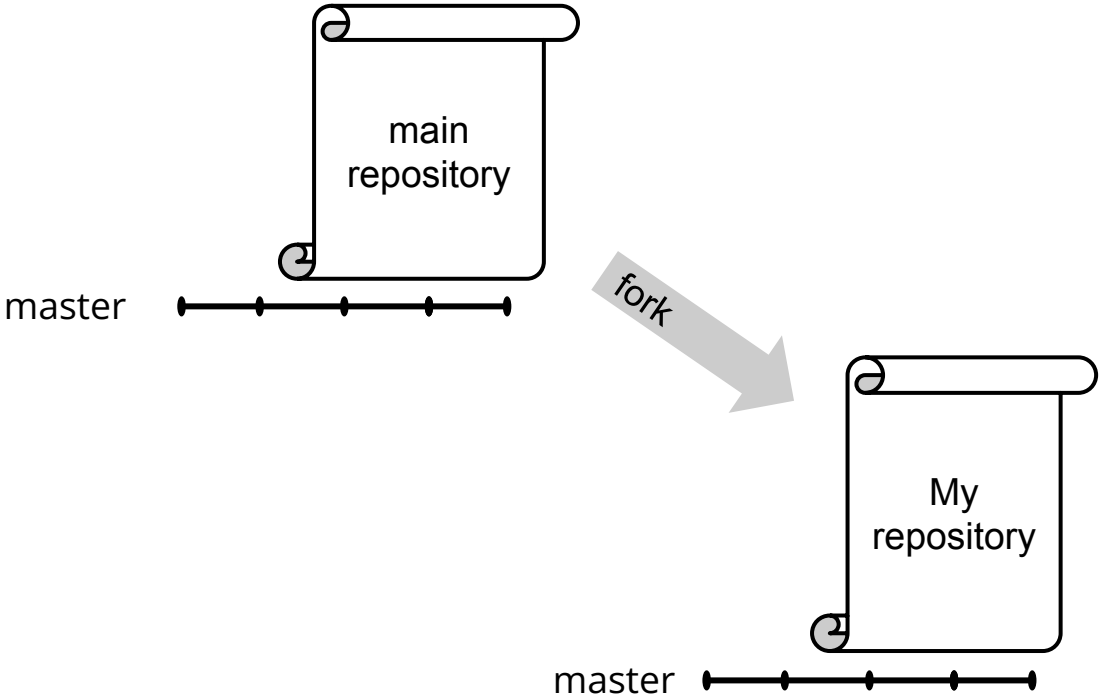
# Collaboration



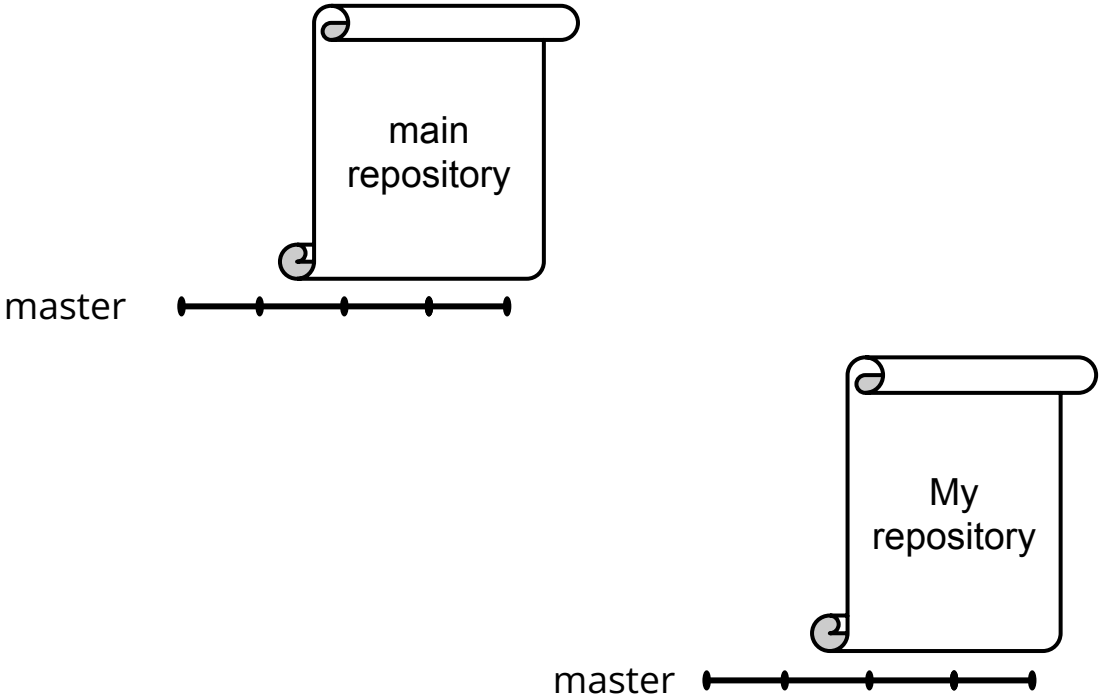
# Collaboration

How do you keep your copy up to date?

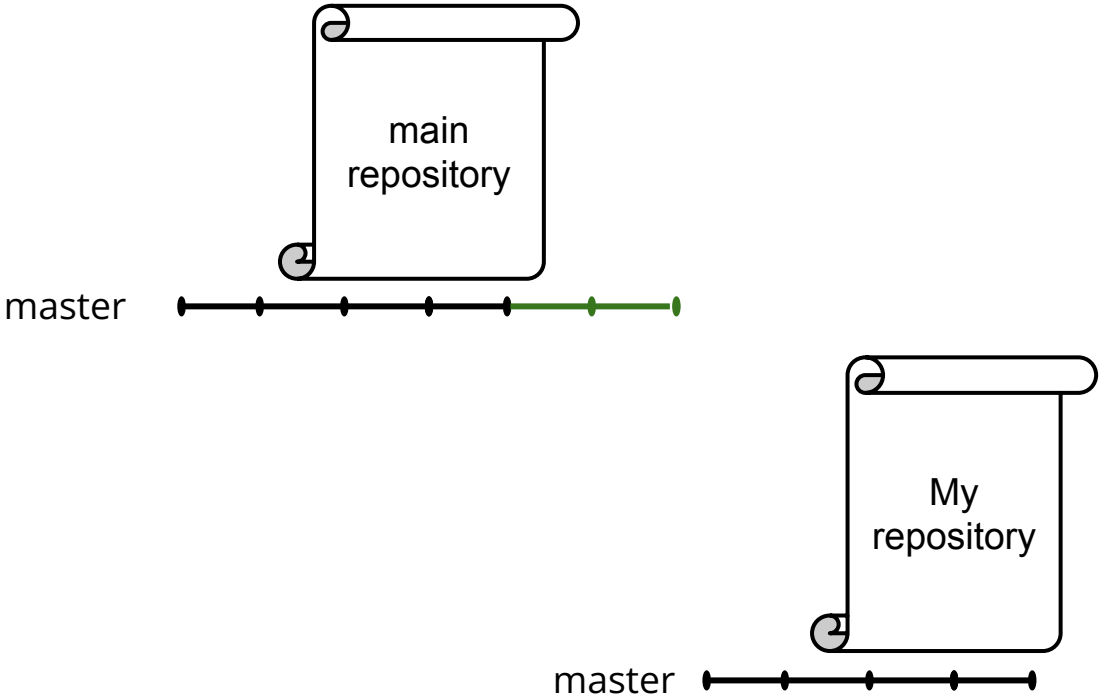
# Collaboration



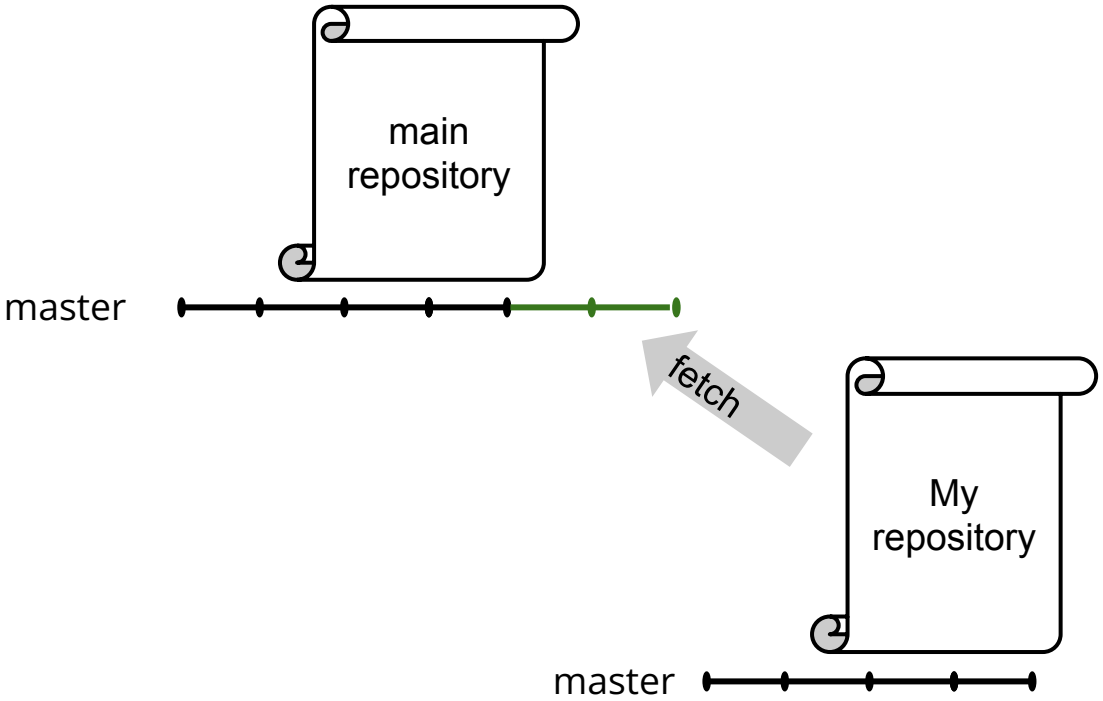
# Collaboration



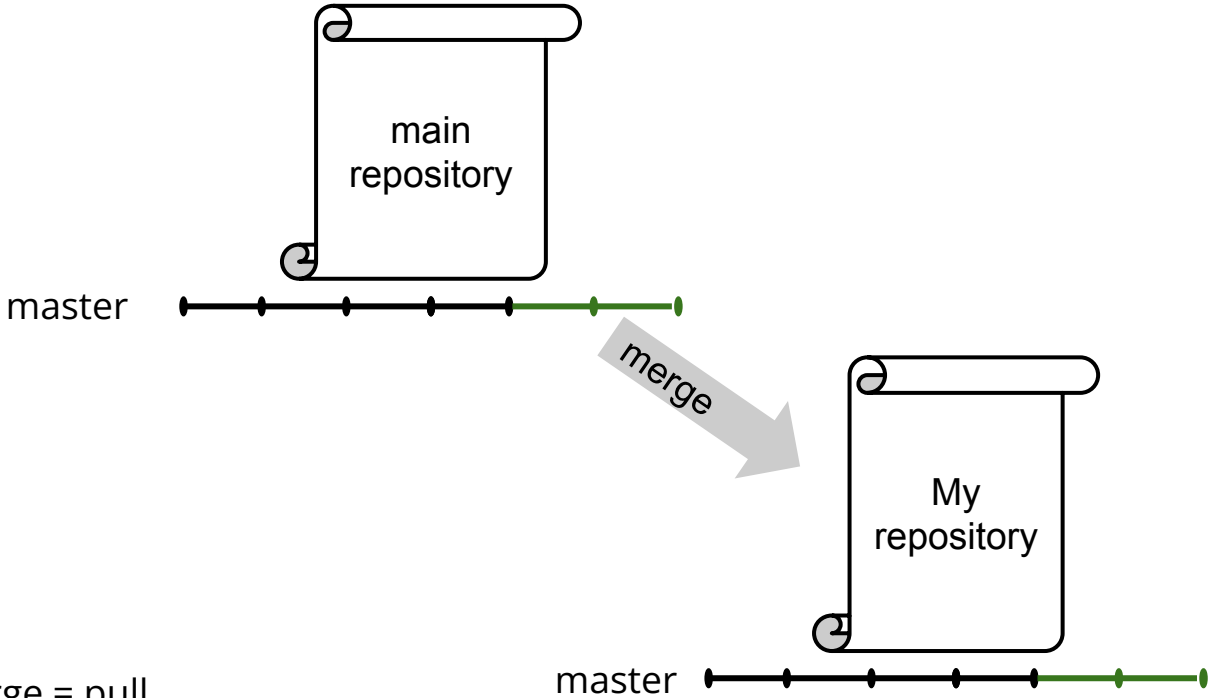
# Collaboration



# Collaboration



# Collaboration

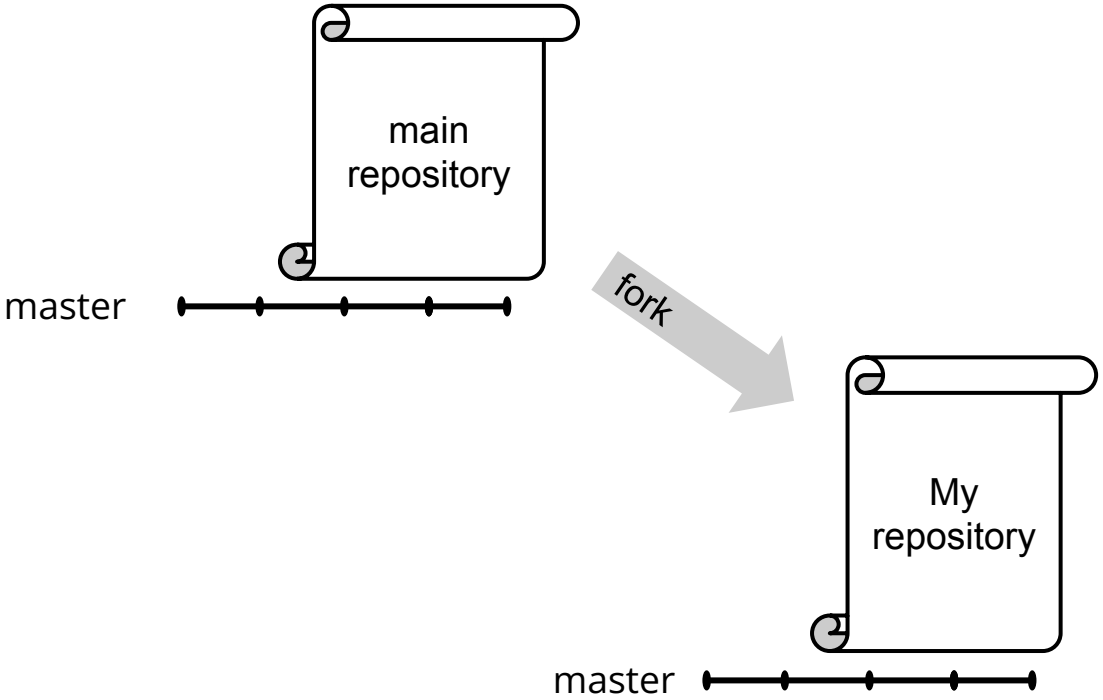



Note: fetch + merge = pull

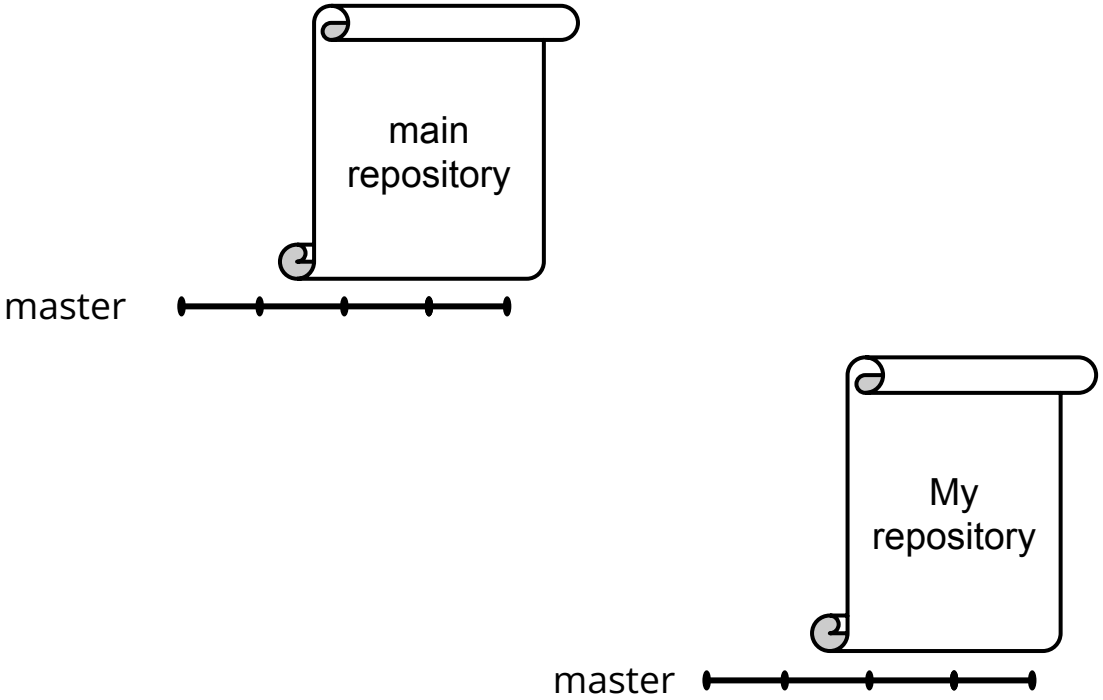
# Collaboration

This is trivial when the **base** of one branch matches the **head** of the other.

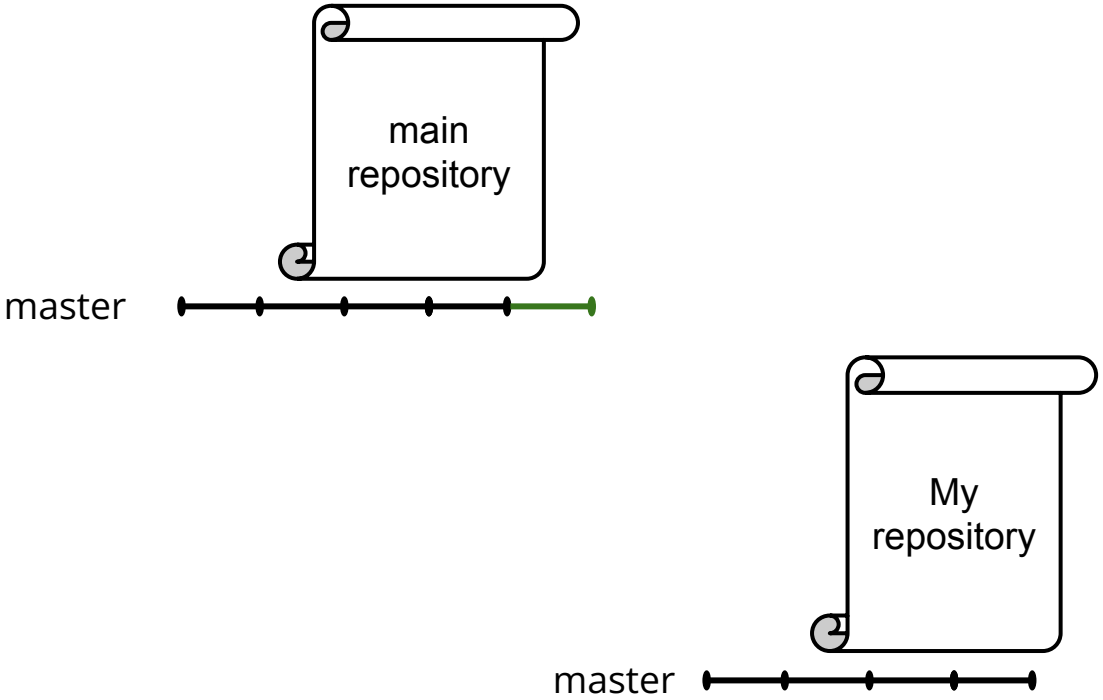
# Collaboration



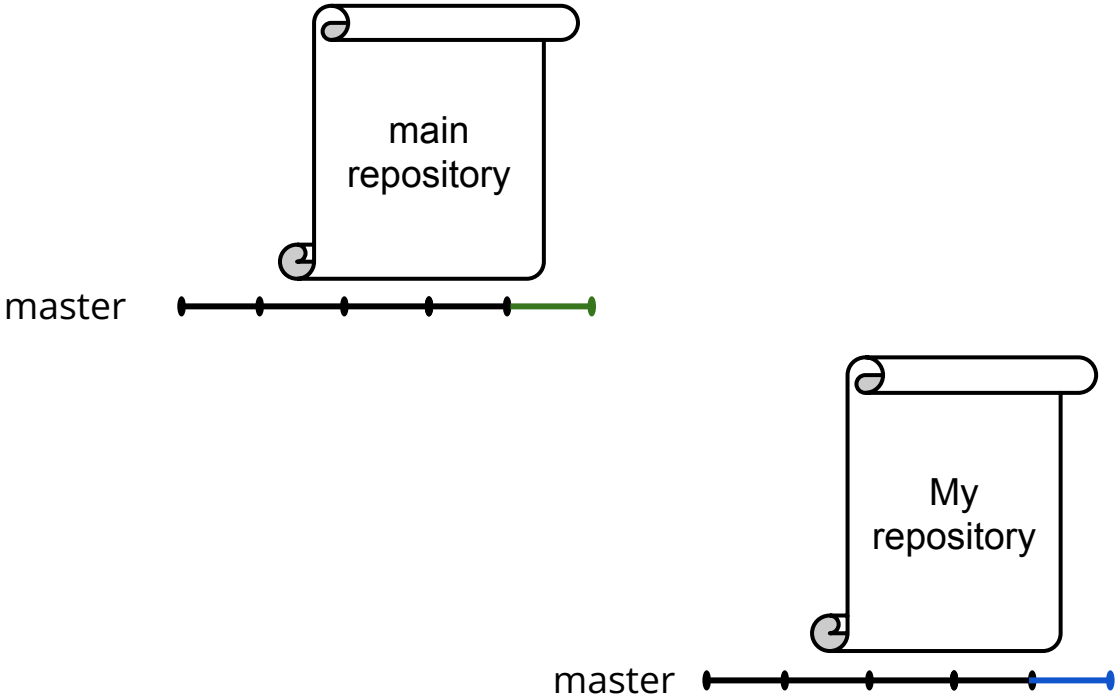
# Collaboration



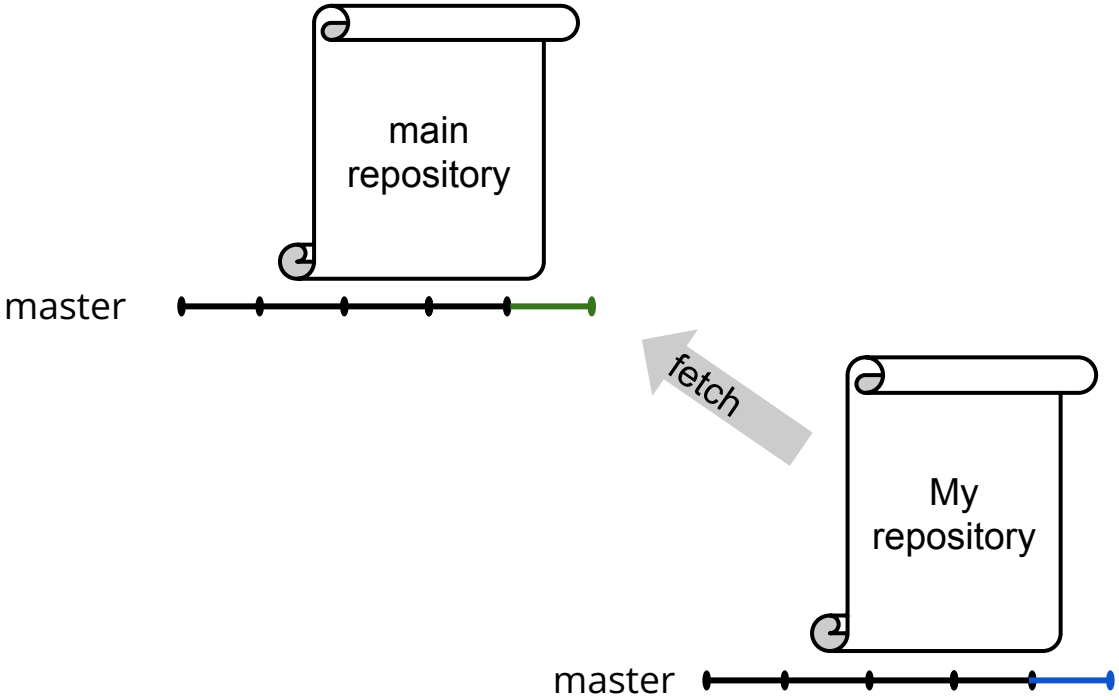
# Collaboration



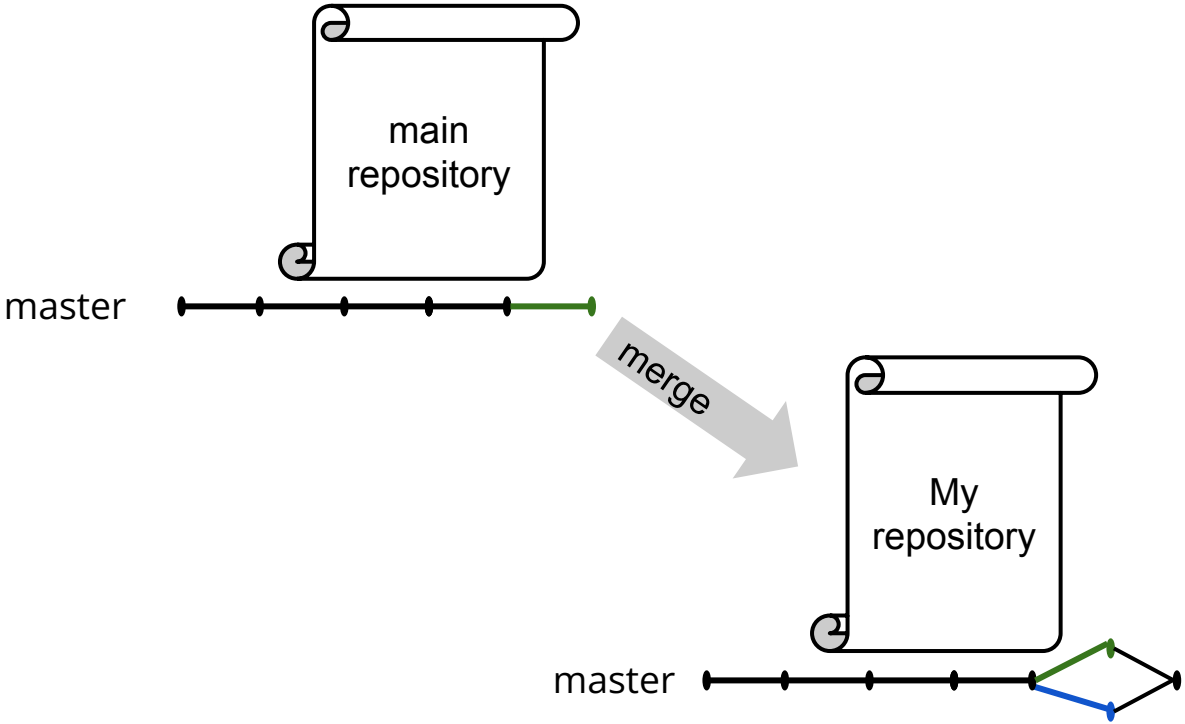
# Collaboration



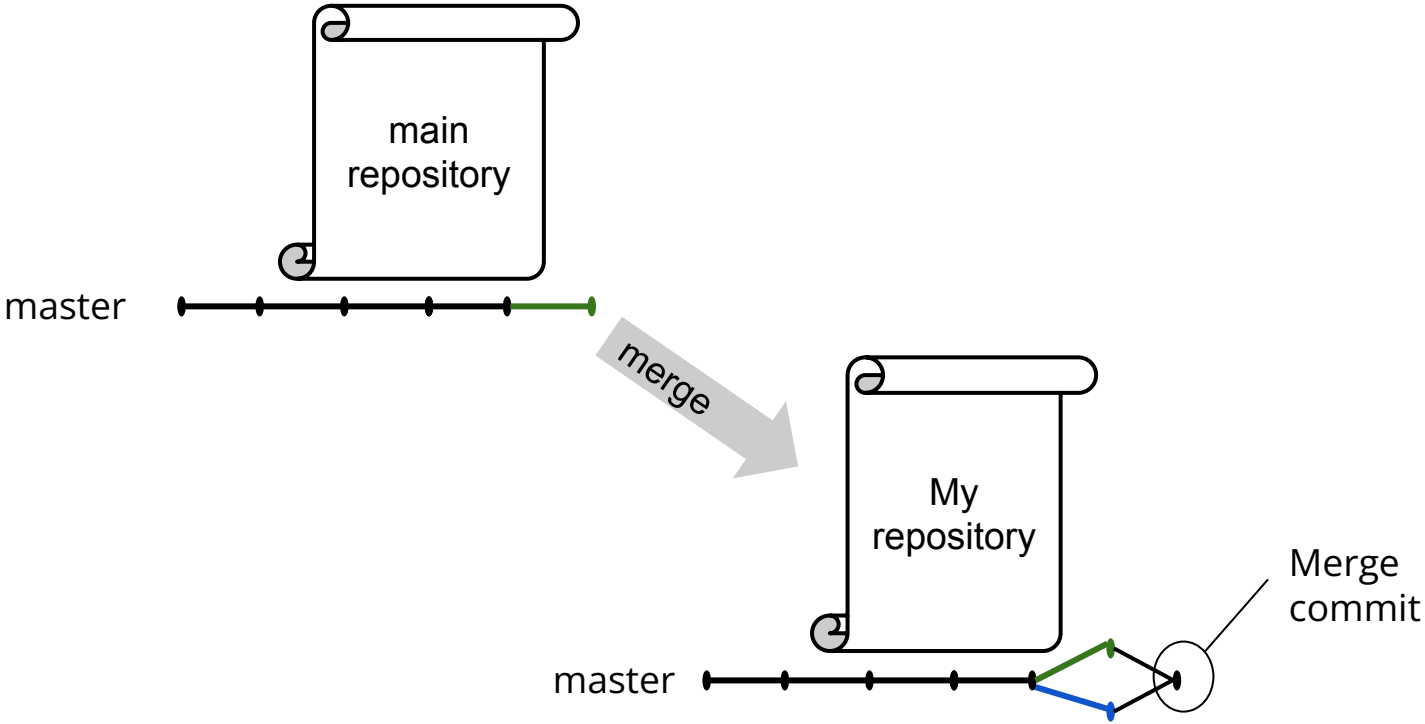
# Collaboration



# Collaboration



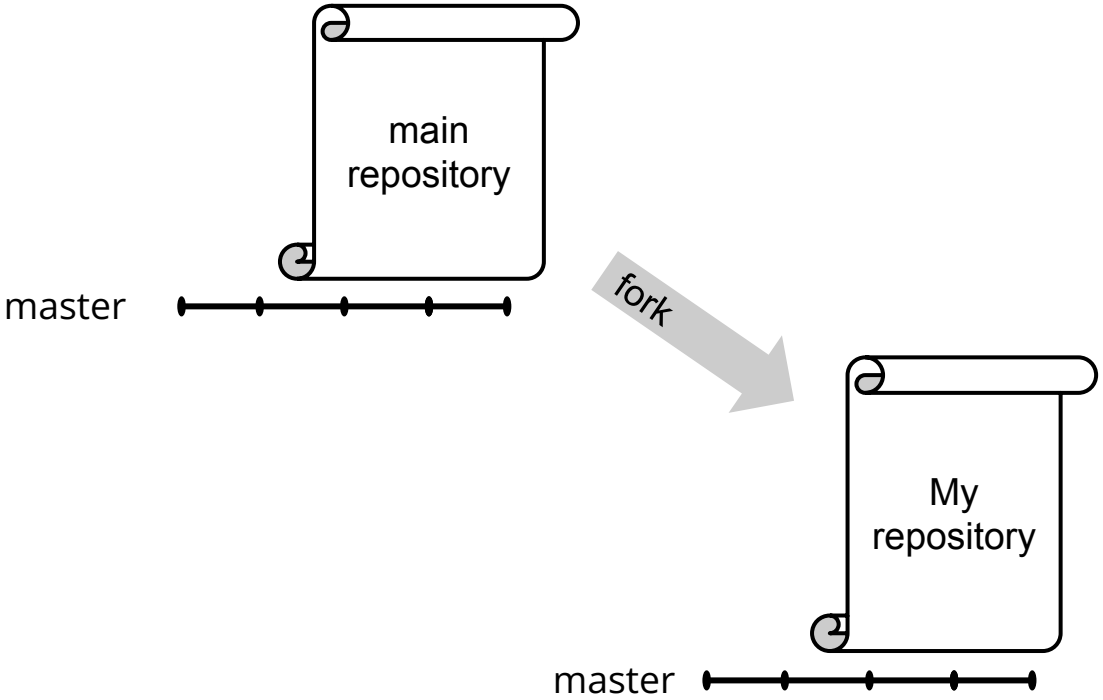
# Collaboration



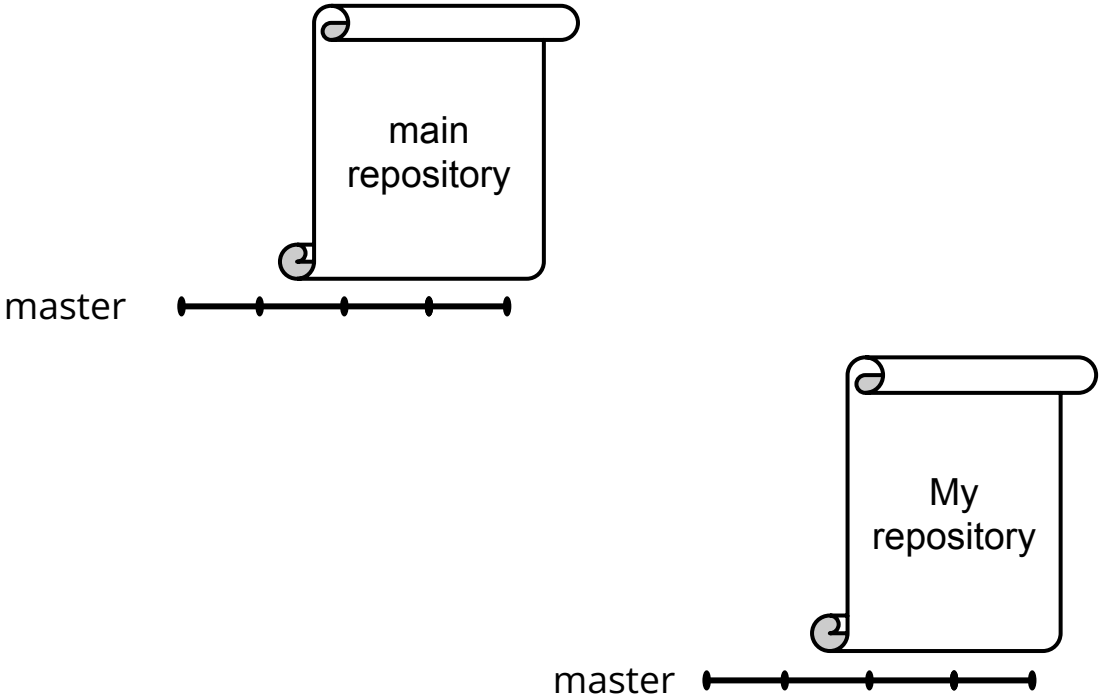
# Collaboration

- Having merge commits and diamond shapes in the version history is confusing
- But it preserves both versions exactly as they are so it’s handy for public branches that others depend on (they won’t get conflicts)
- Commits should be logical steps in the creation of a code base. A merge commit on your local development branch for the time that you decided to keep it in sync does not align with that philosophy.
- Try rebasing instead

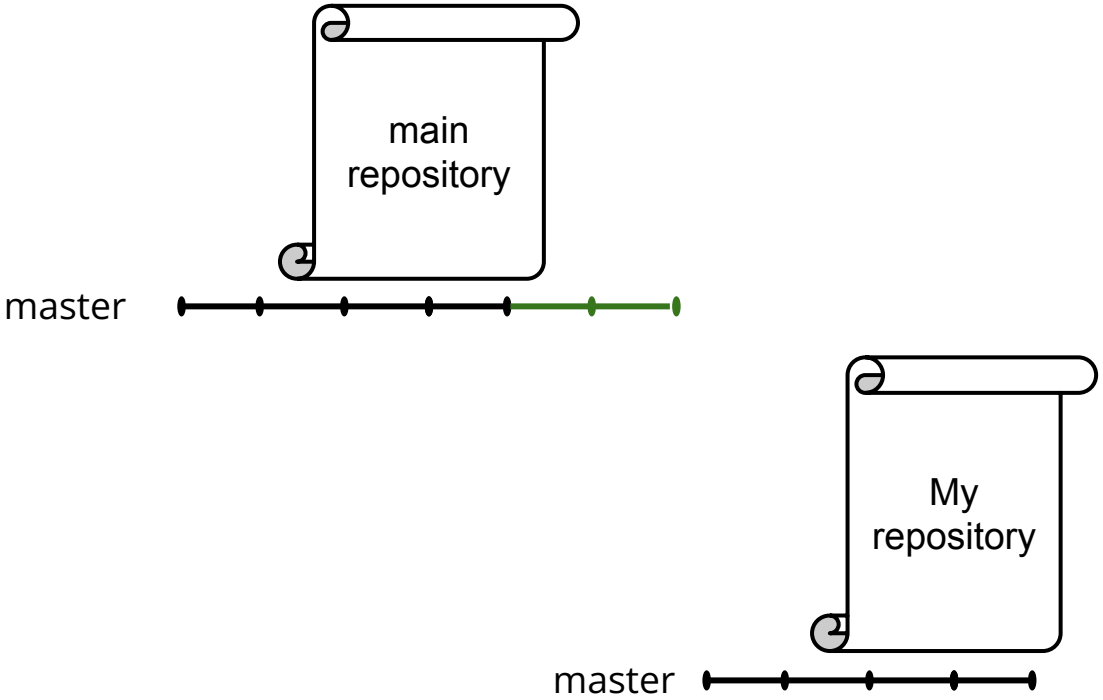
# Collaboration



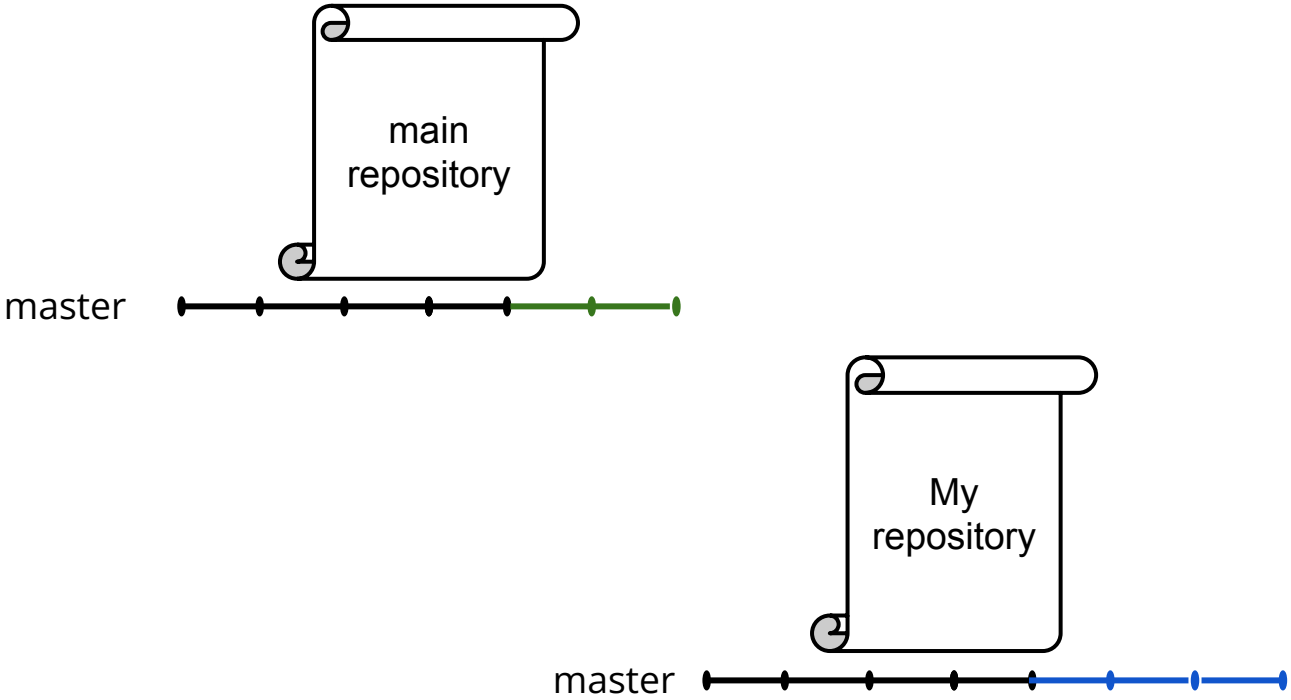
# Collaboration



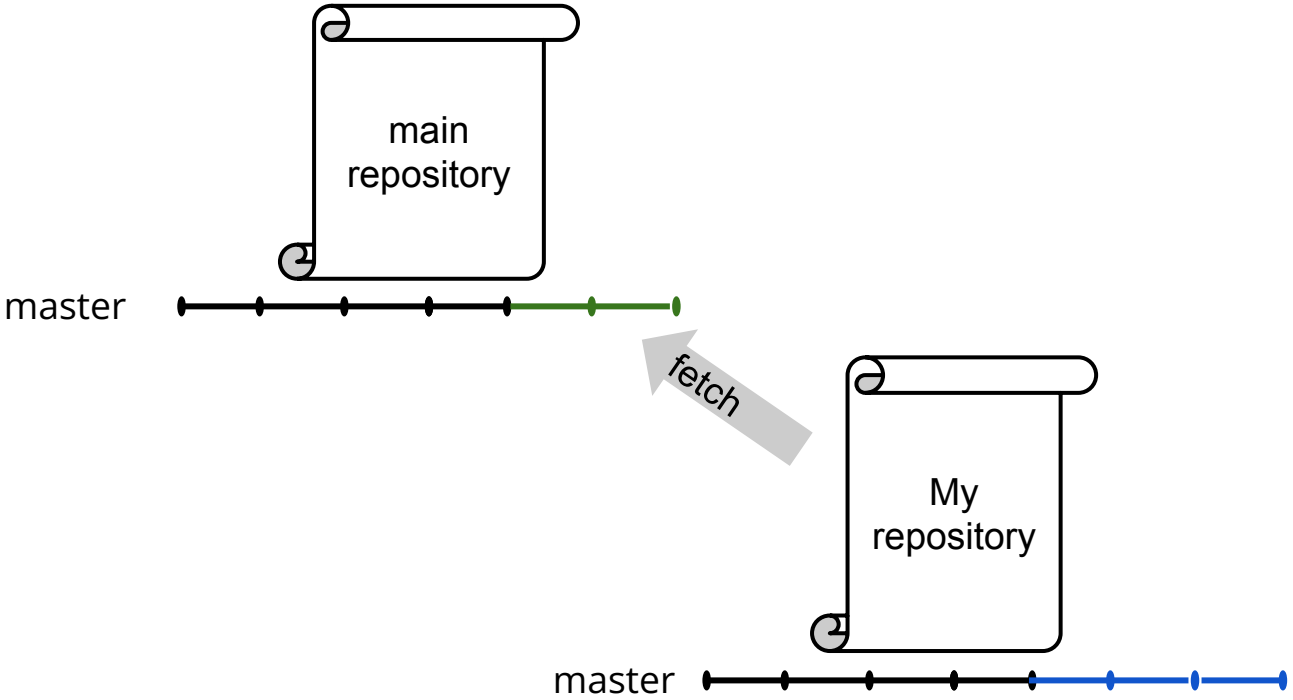
# Collaboration



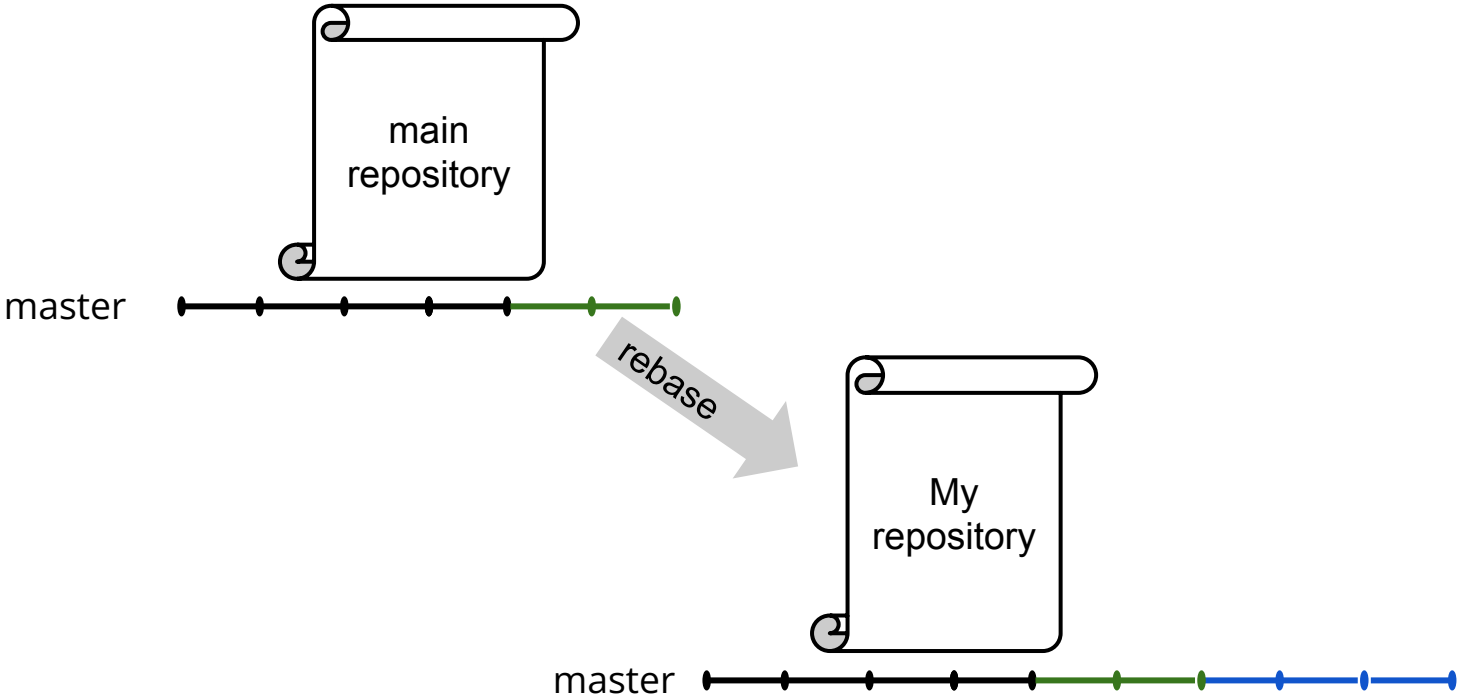
# Collaboration



# Collaboration



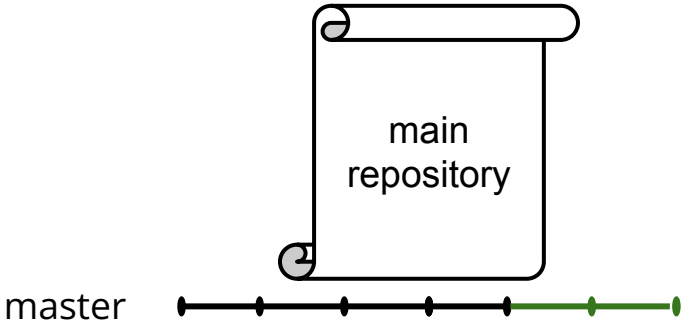
# Collaboration

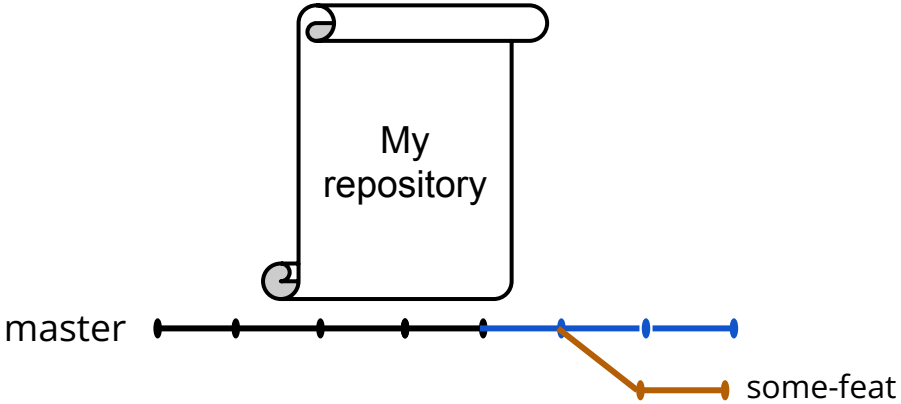


# Collaboration

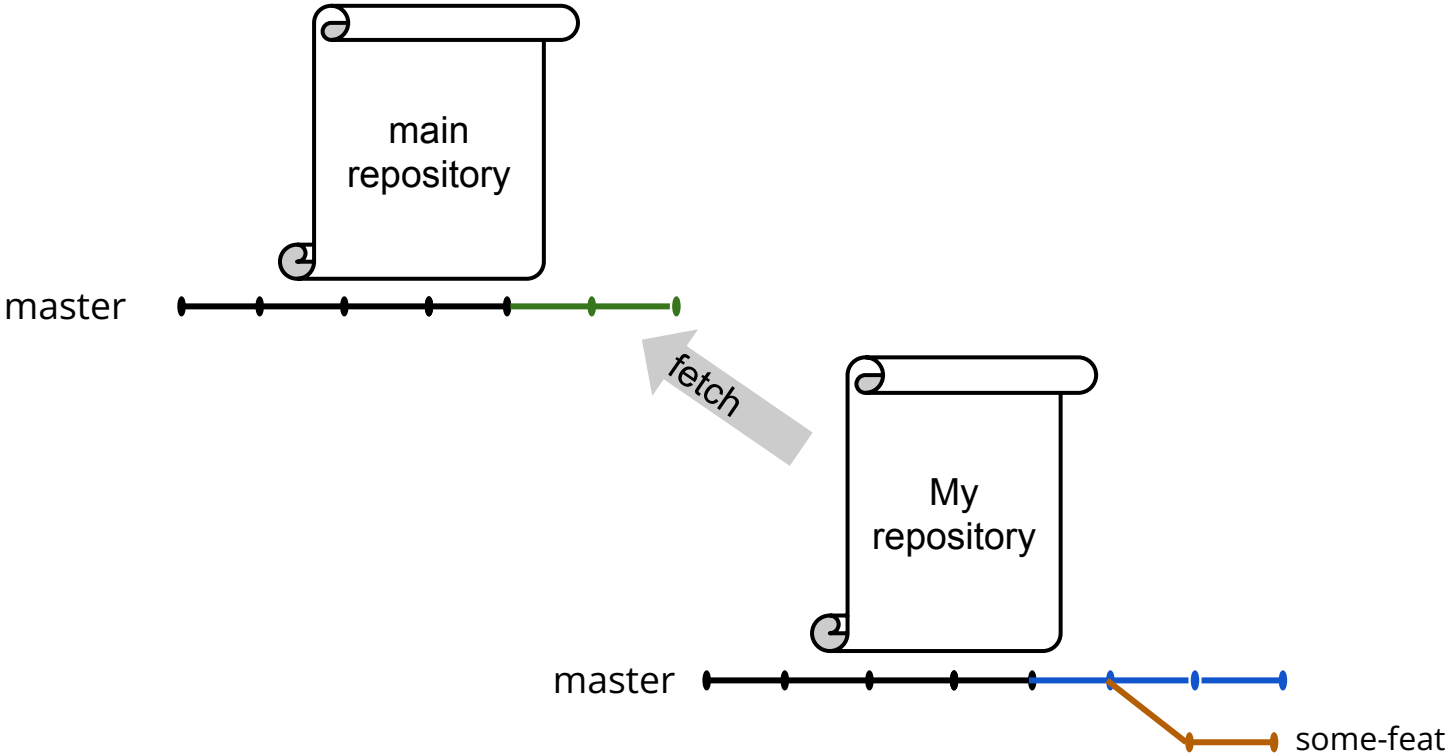
What happens to other branches?

# Collaboration

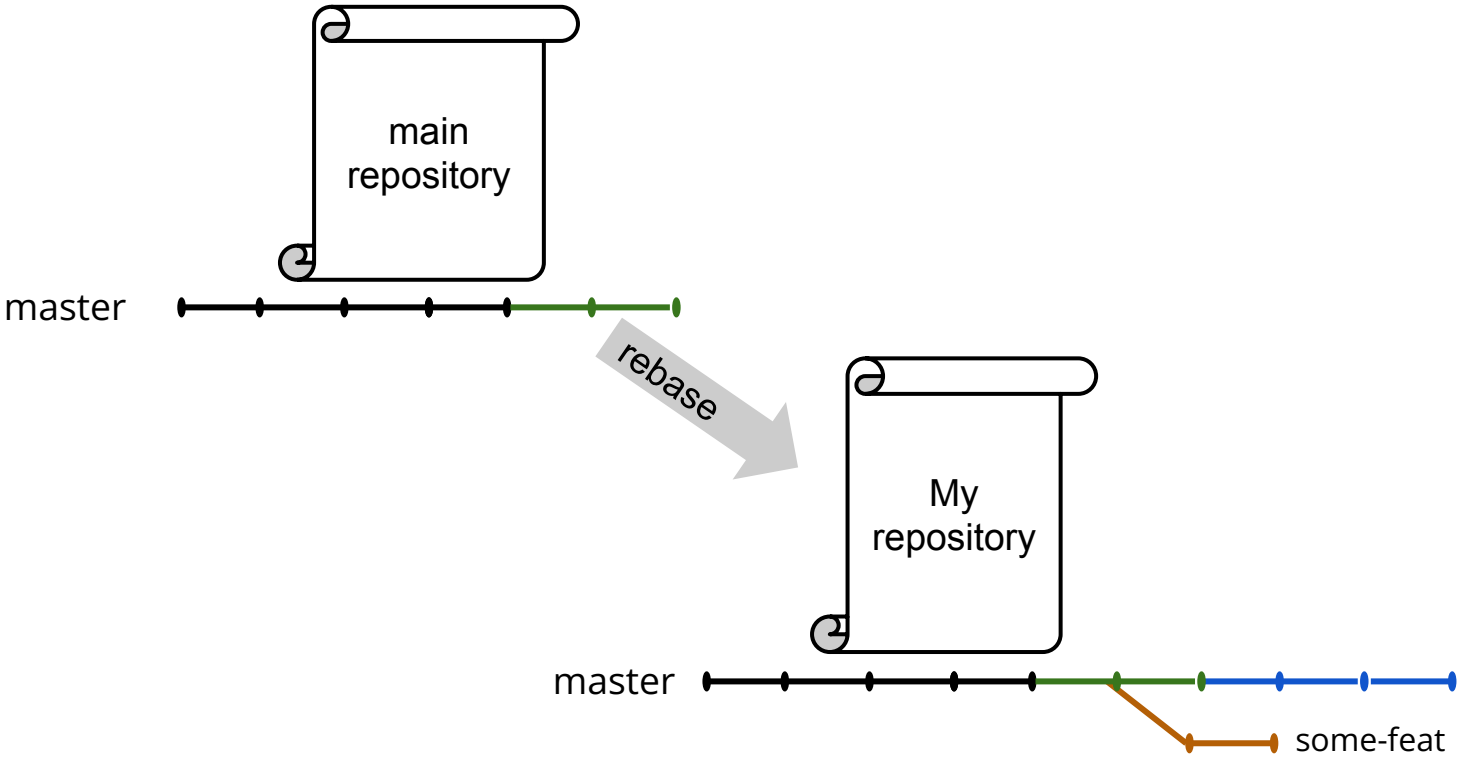


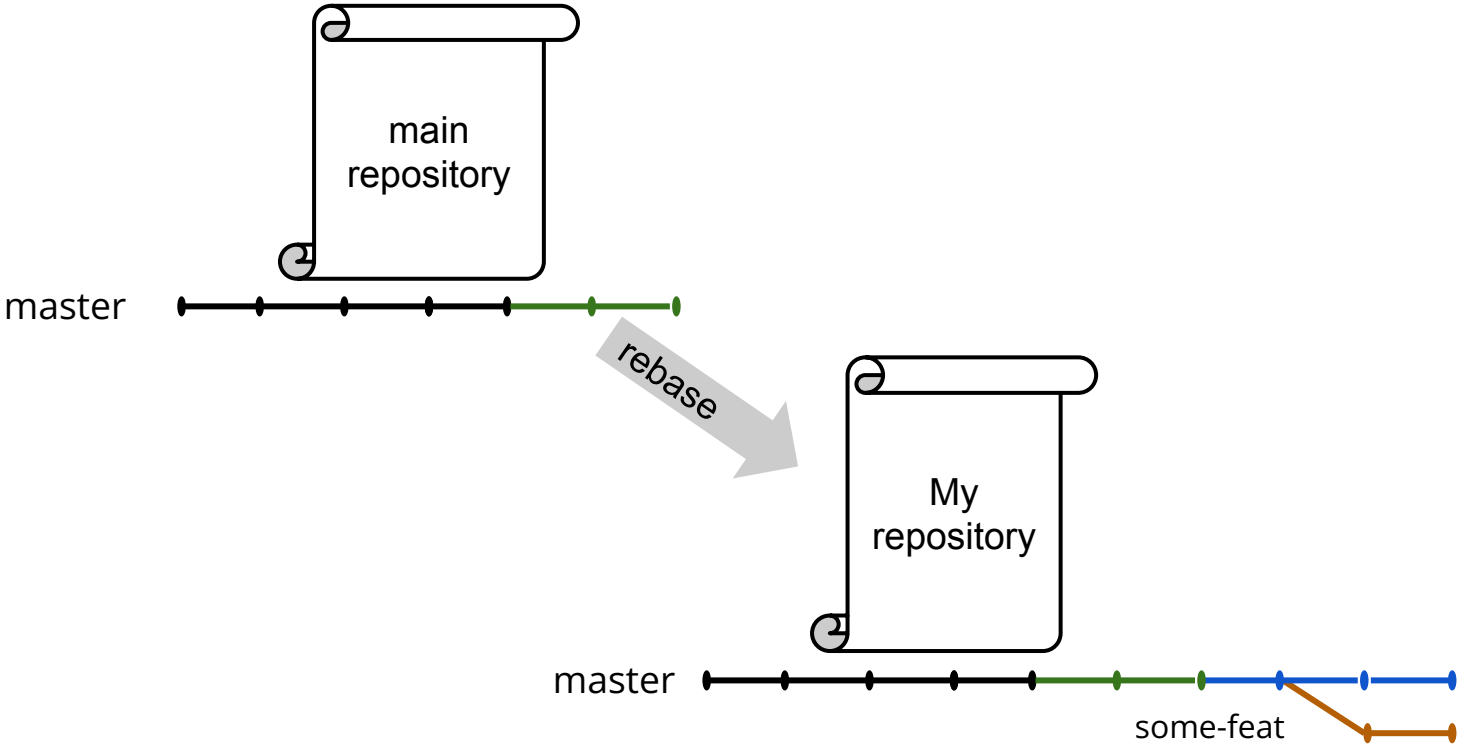
# Collaboration



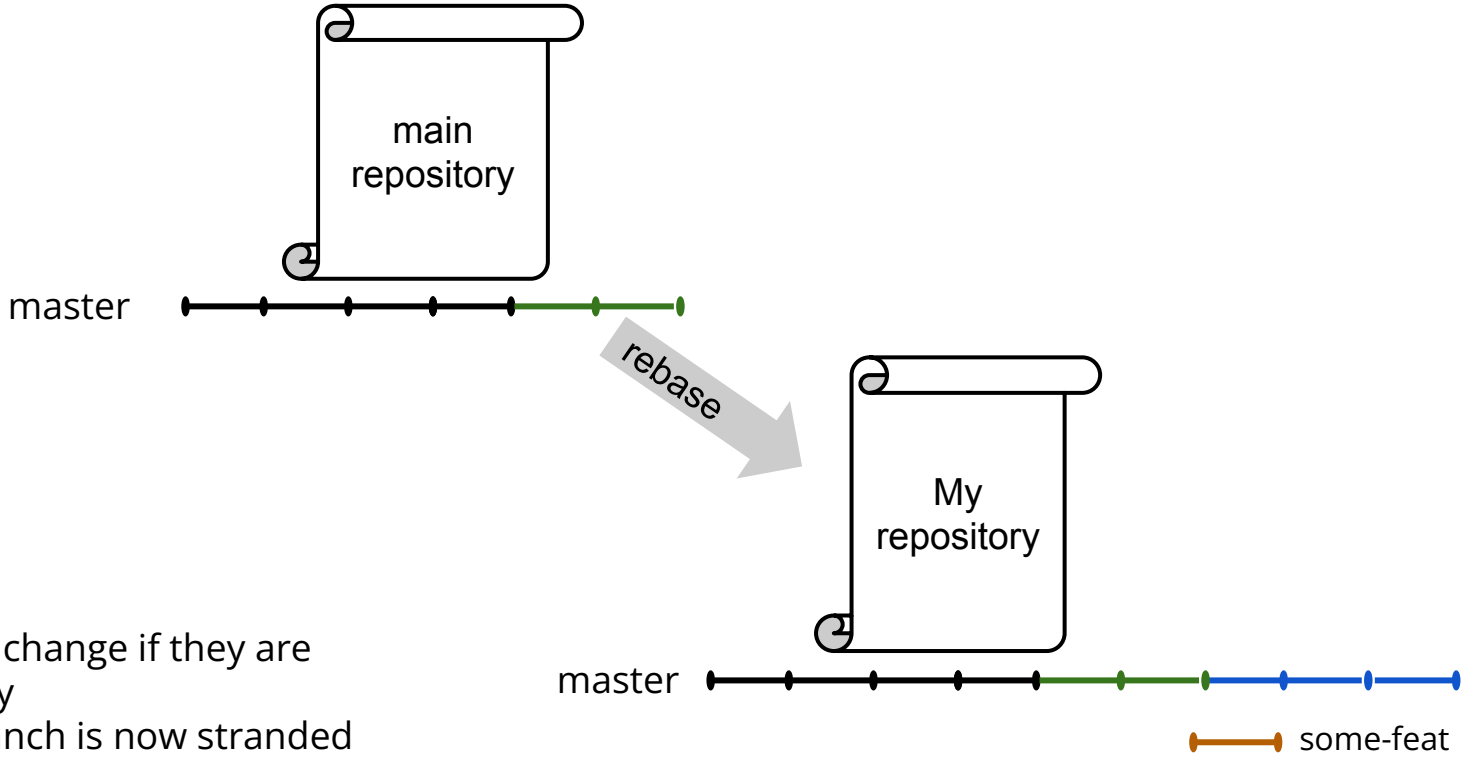
# Collaboration



# Collaboration



# Collaboration



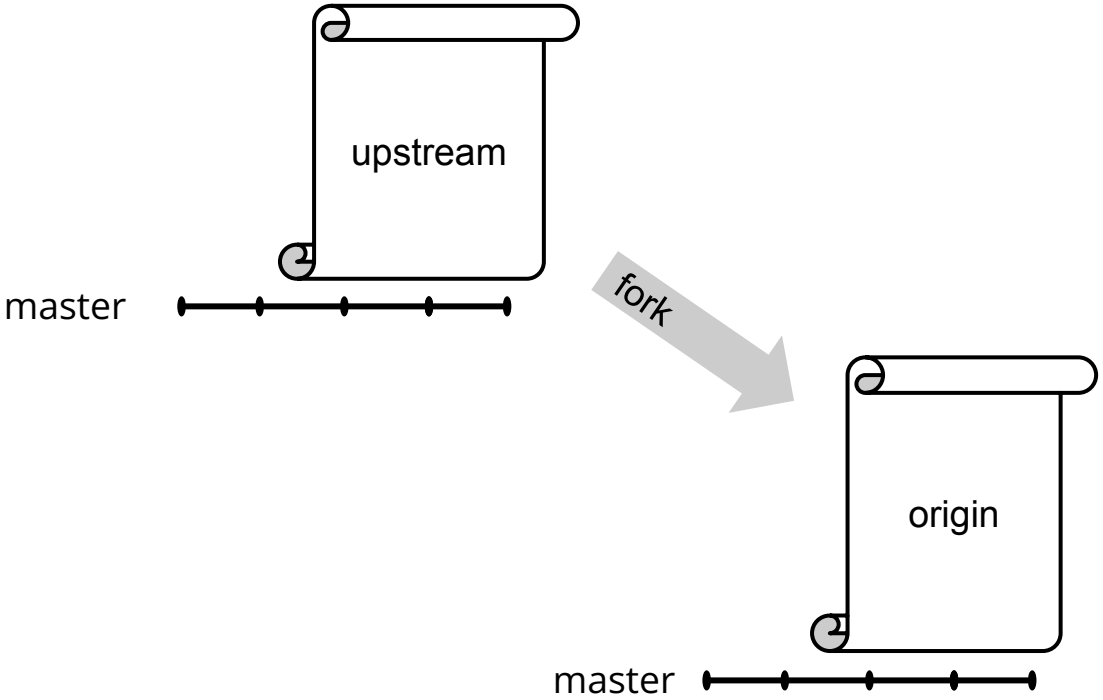

commit IDs (shas) change if they are ordered differently
The some-feat branch is now stranded

# Collaboration

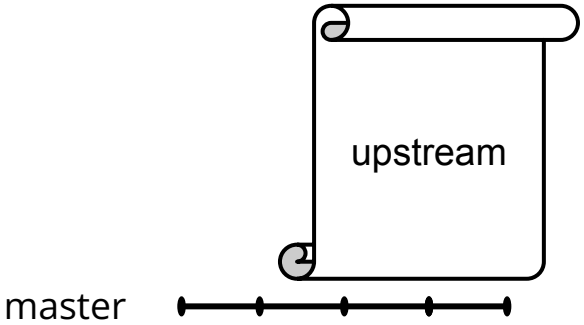
If you **never commit anything to your master branch**, keeping your master branch in sync with the main repository's is easy! And keeping all branches attached comes for free!

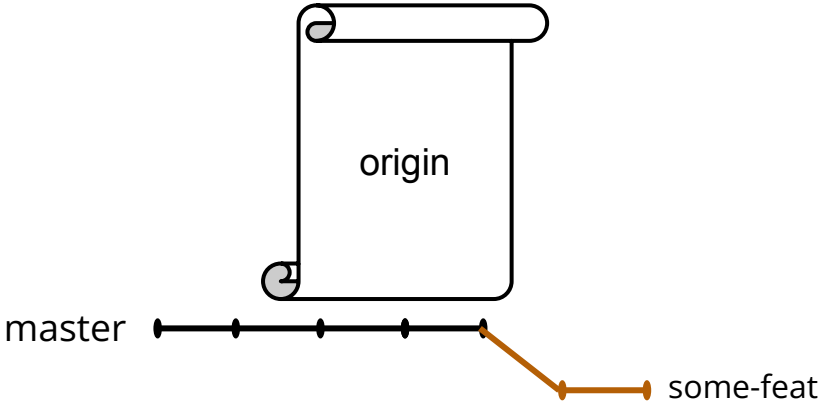
As a rule, always create a new branch when developing - **never commit directly to the master branch**

# Collaboration

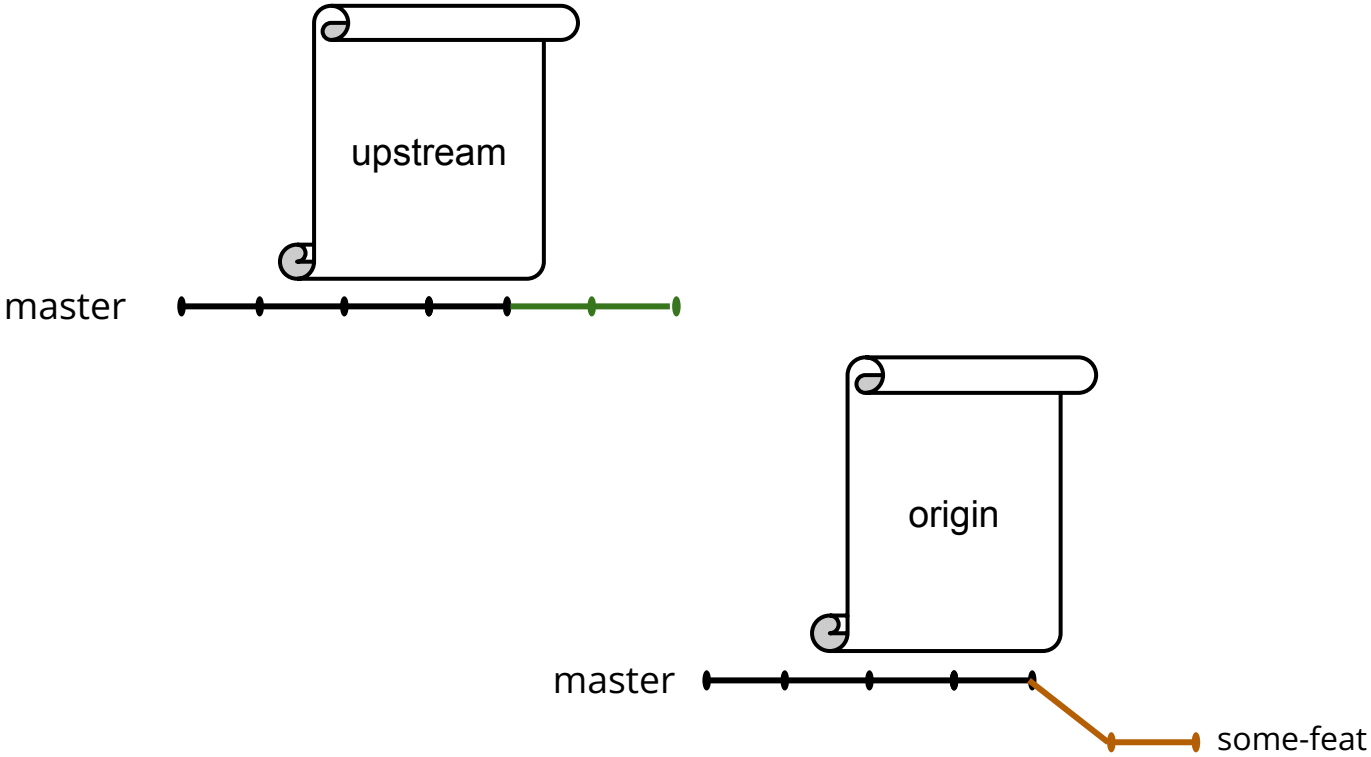


# Collaboration

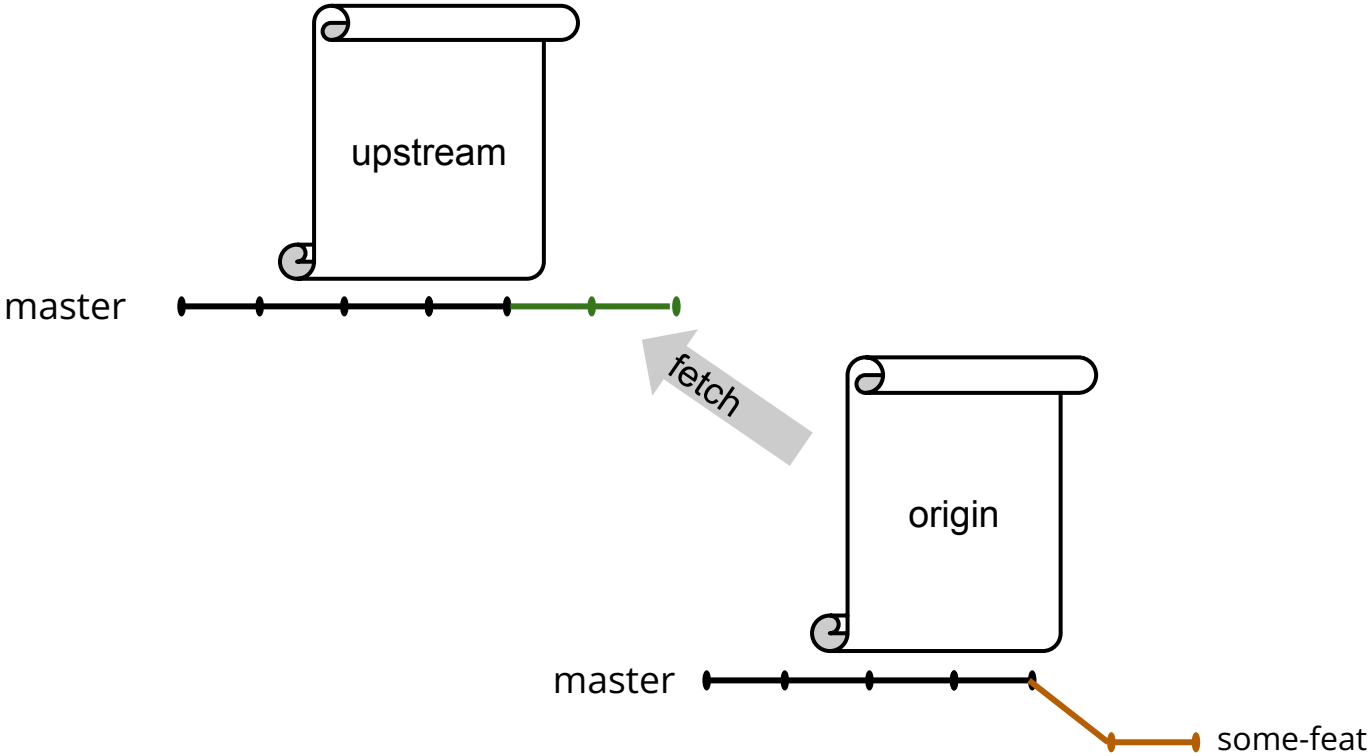


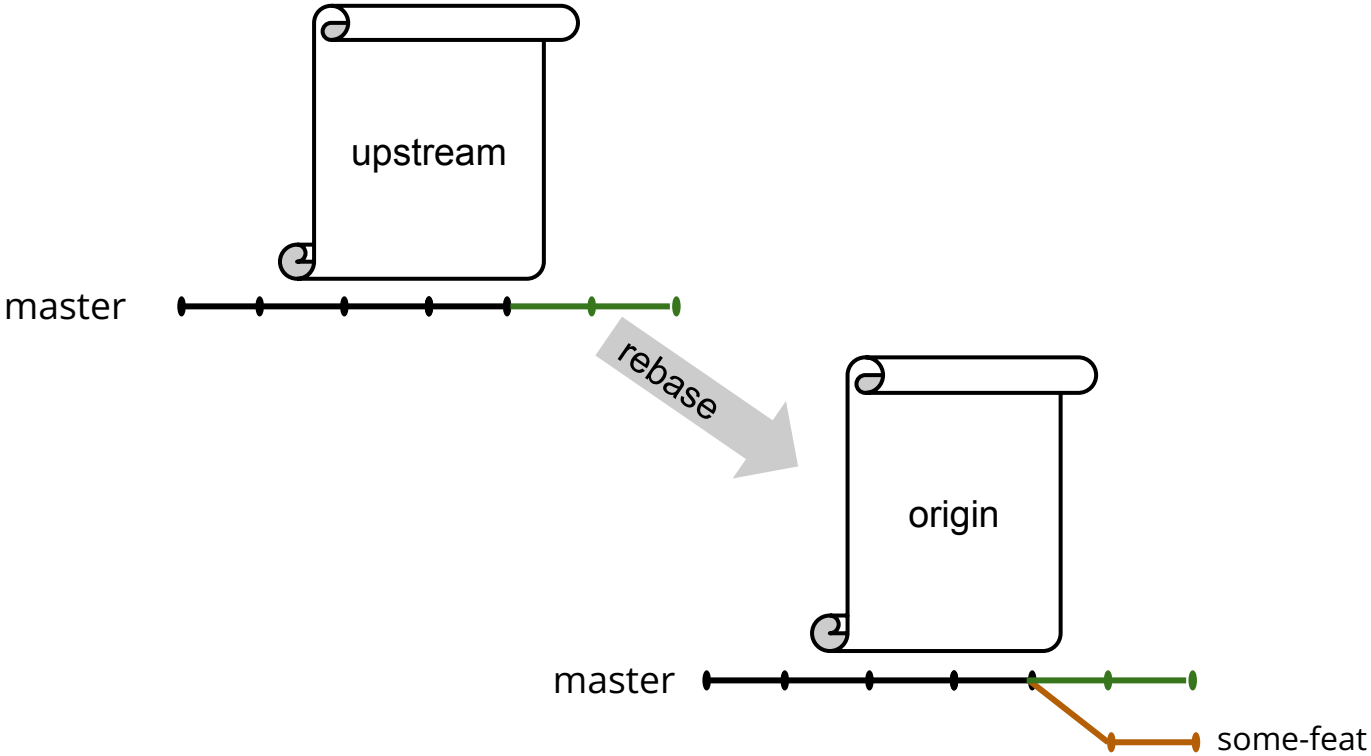
# Collaboration



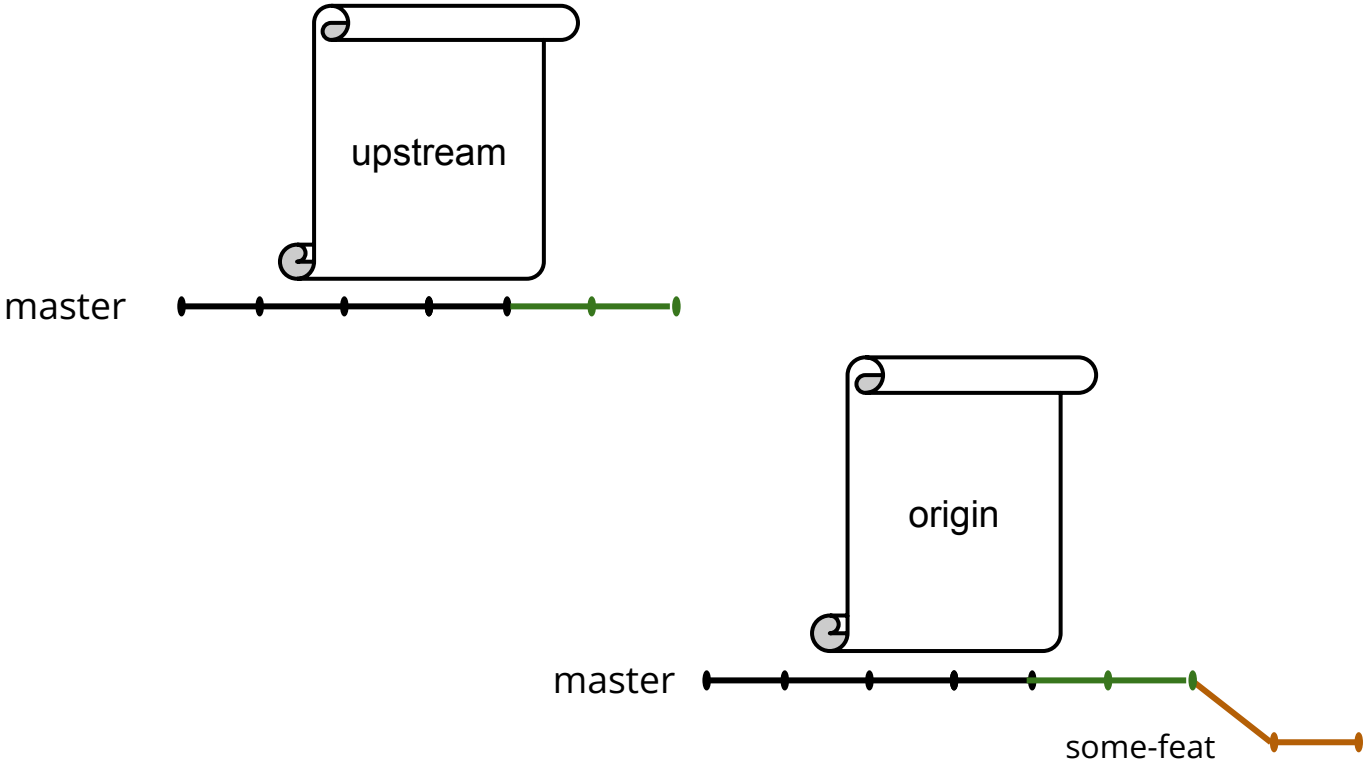
# Collaboration



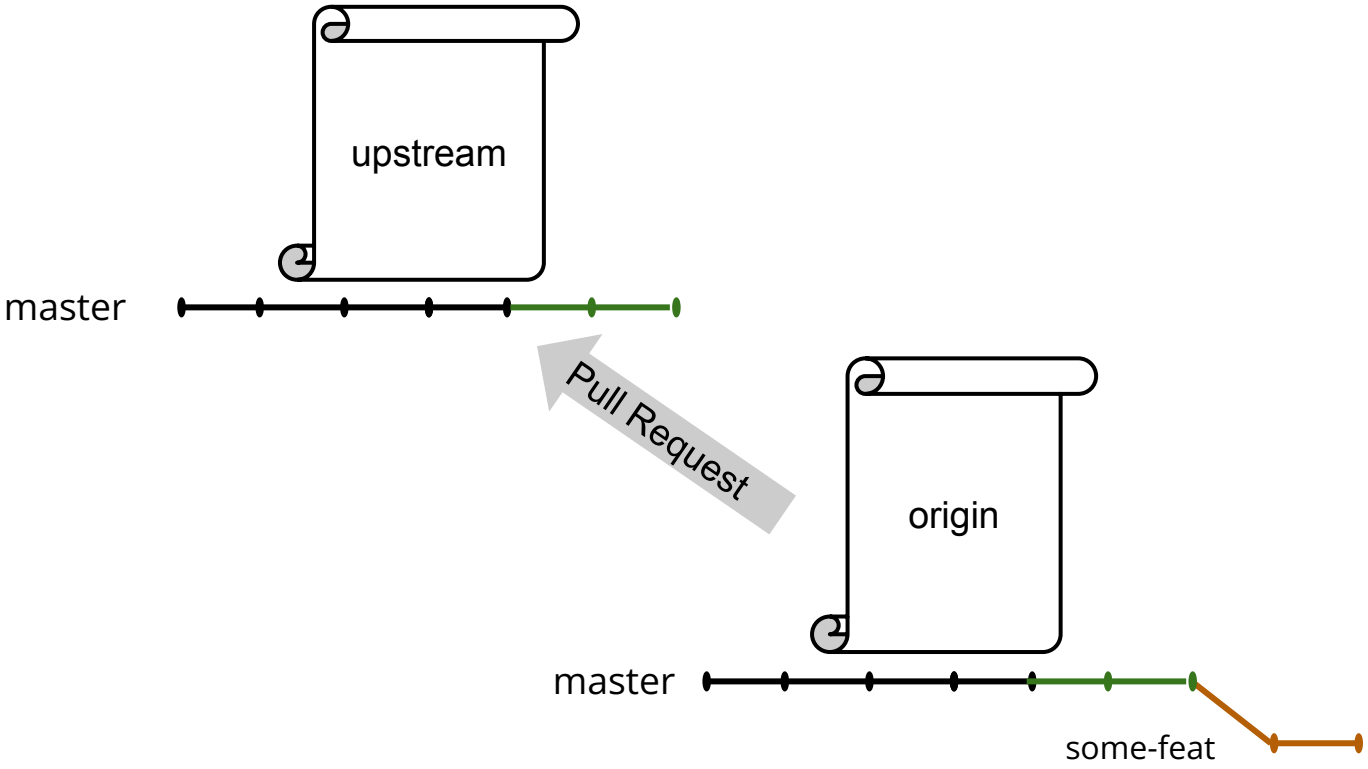
# Collaboration



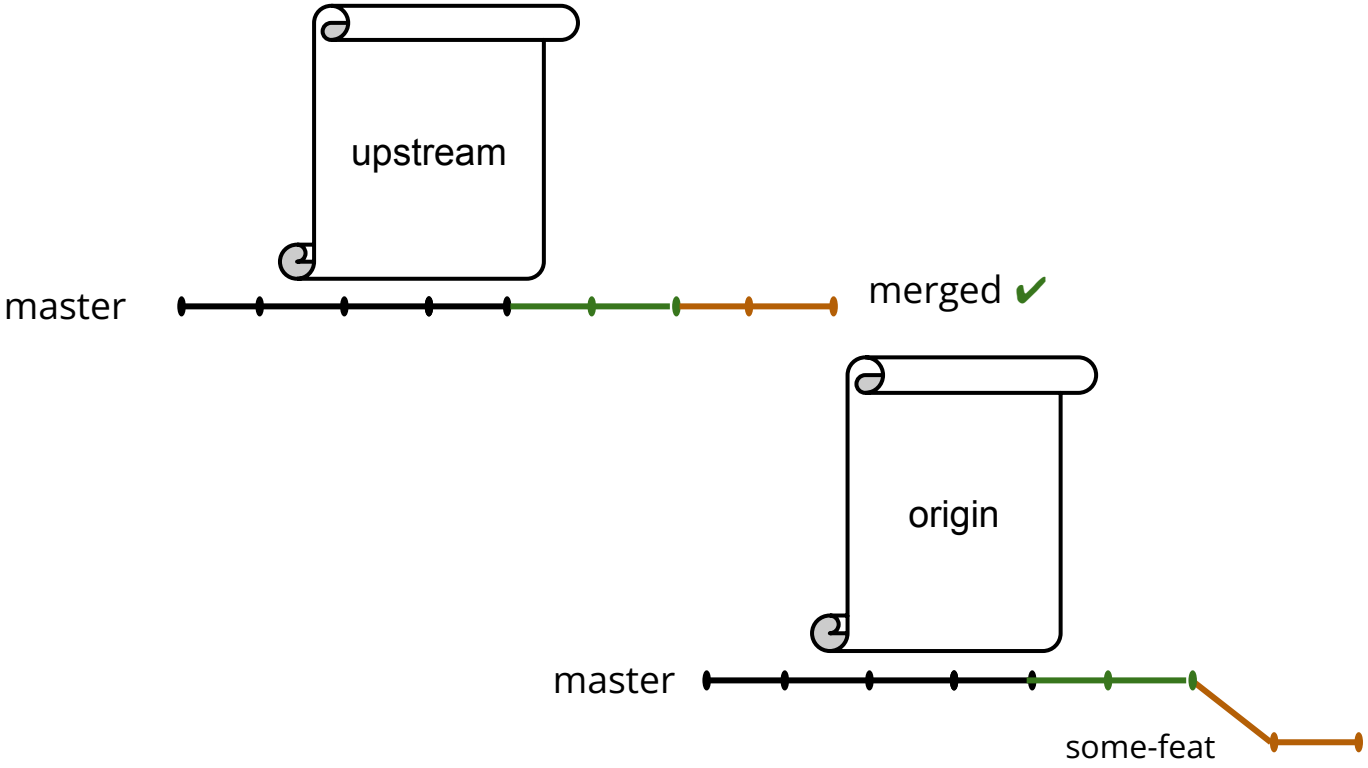
# Collaboration



# Collaboration



# Collaboration



# Demo