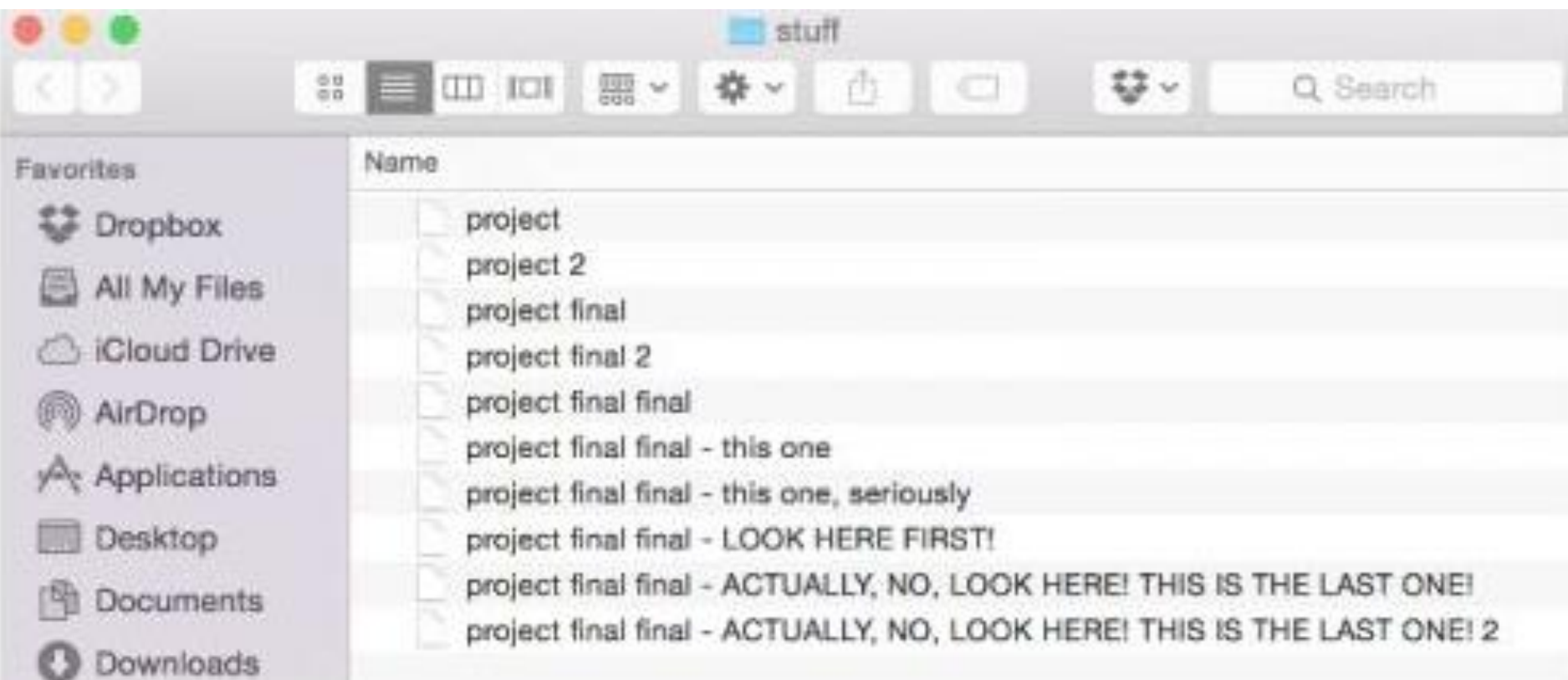


---

# Git

— Boston University CS 506 - Lance Galletti —

---



# Git

A tool to help us manage the **timeline(s)** of a project (also called repository).

Formally called a **Version Control System** or **Source Control Management**

# Fundamental Workflow

As we change the project over time


Create save points (called **commits**) that track the timeline of the project's evolution

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

# Fundamental Workflow

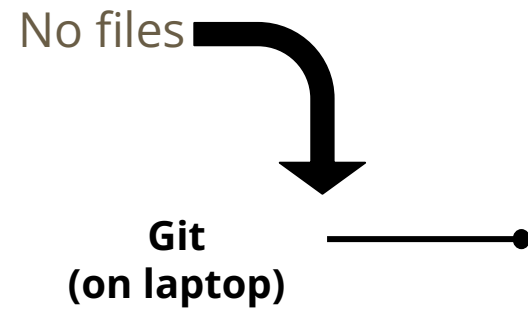
Create save points (called **commits**)

**Git**  
(on laptop)



# Fundamental Workflow

Create save points (called **commits**)



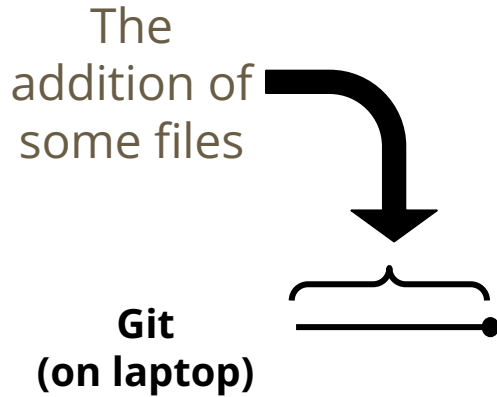
# Fundamental Workflow

Create save points (called **commits**)



# Fundamental Workflow

Create save points (called **commits**)





# Fundamental Workflow

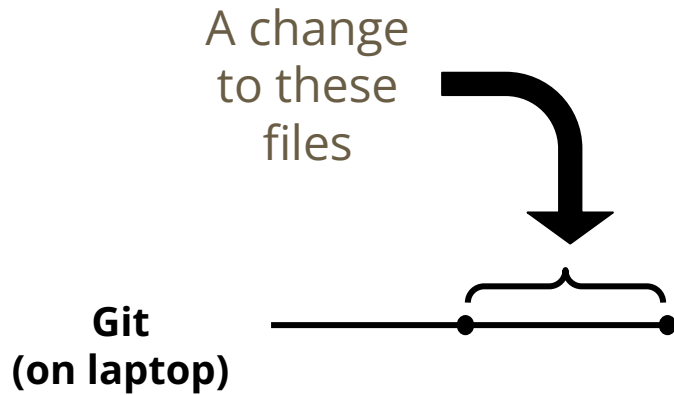
Create save points (called **commits**)

**Git**  
(on laptop)



# Fundamental Workflow

Create save points (called **commits**)



# Fundamental Workflow

Creates a timeline of your code / files

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



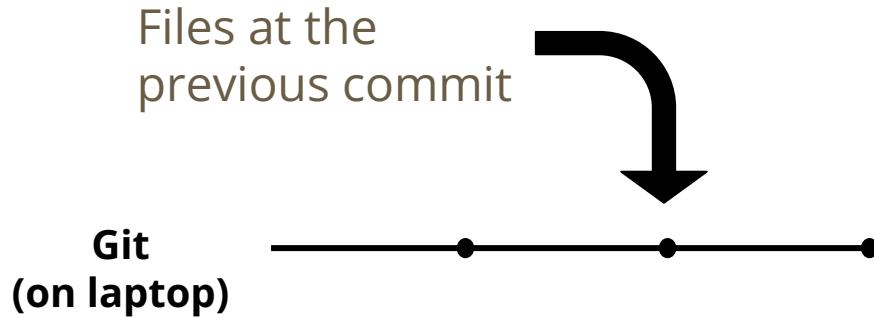
# Fundamental Workflow

Creates a timeline of your code / files



# Fundamental Workflow

Creates a timeline of your code / files



**Demo**

# GitHub vs Git

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

**GitHub** --> [[browser](#)] a **website** to **backup and host** the **timeline(s)** of your project


# Fundamental Workflow

Create save points (called **commits**)

**Push** the updates to GitHub (from your laptop) to back up your work

**GitHub**  
(online)

**Git**  
(on laptop)

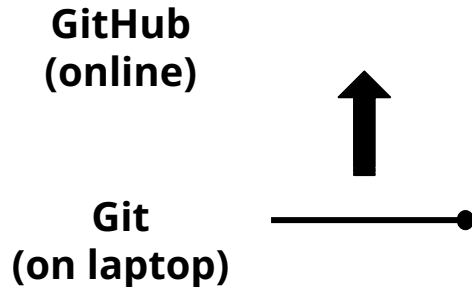




# Fundamental Workflow

Create save points (called **commits**)

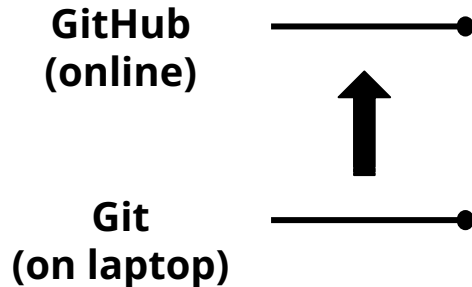
**Push** the updates to GitHub (from your laptop) to back up your work



# Fundamental Workflow

Create save points (called **commits**)

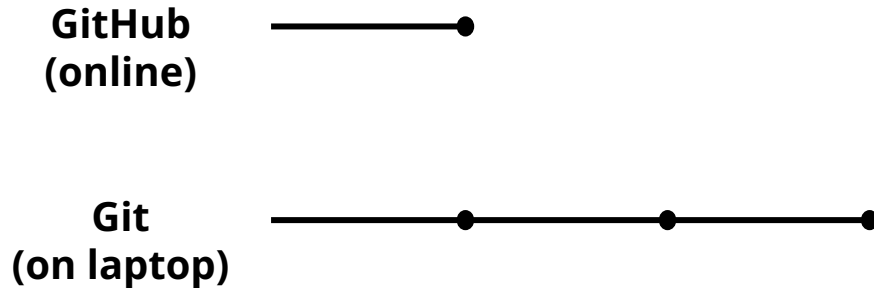
**Push** the updates to GitHub (from your laptop) to back up your work



# Fundamental Workflow

Create save points (called **commits**)

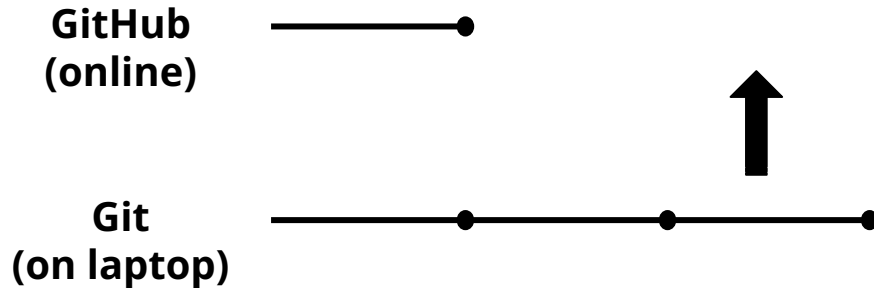
**Push** the updates to GitHub (from your laptop) to back up your work



# Fundamental Workflow

Create save points (called **commits**)

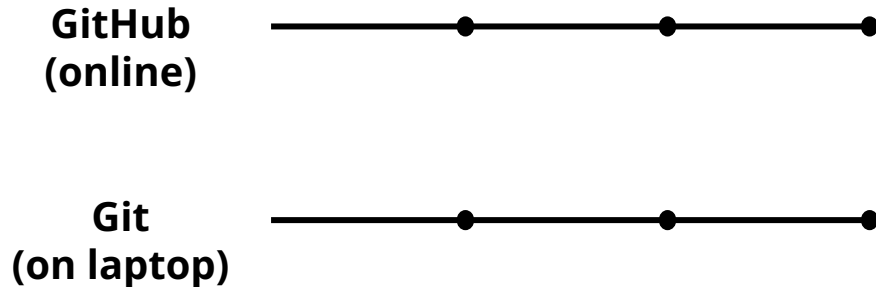
**Push** the updates to GitHub (from your laptop) to back up your work



# Fundamental Workflow

Create save points (called **commits**)

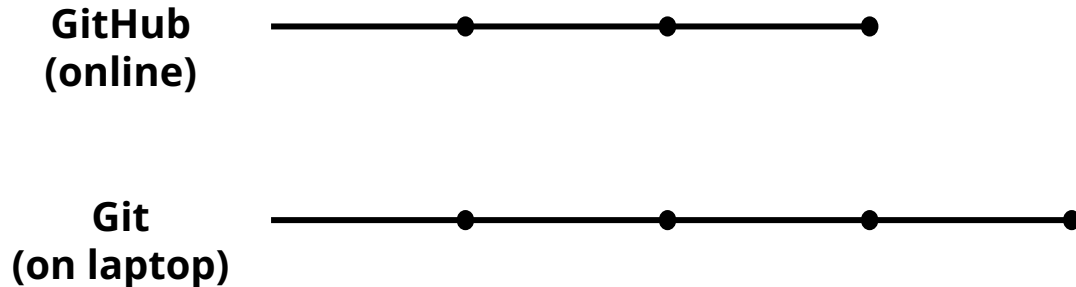
**Push** the updates to GitHub (from your laptop) to back up your work



# Fundamental Workflow

Create save points (called **commits**)

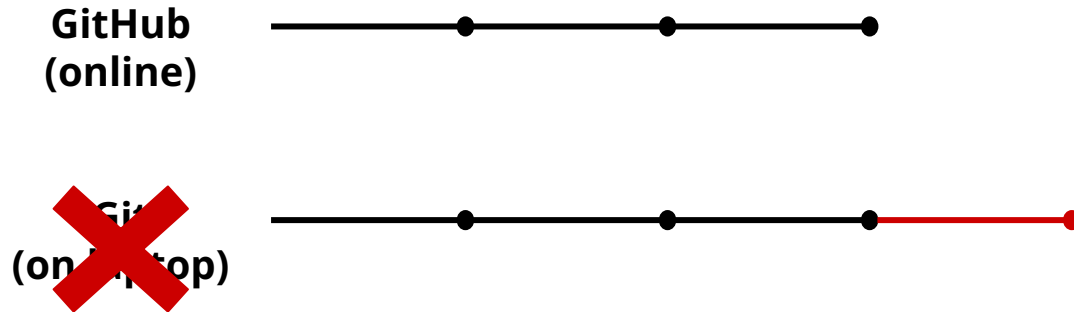
**Push** the updates to GitHub (from your laptop) to back up your work



# Fundamental Workflow

Create save points (called **commits**)

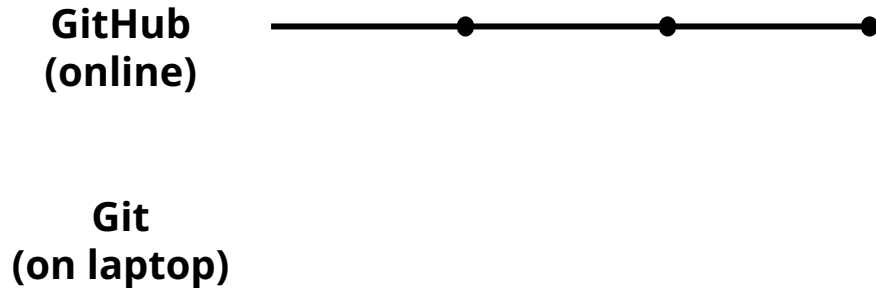
**Push** the updates to GitHub (from your laptop) to back up your work



# Fundamental Workflow

Create save points (called **commits**)

**Push** the updates to GitHub (from your laptop) to back up your work

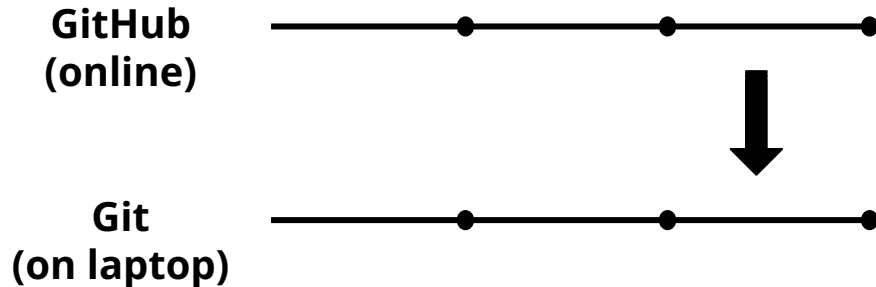




# Fundamental Workflow

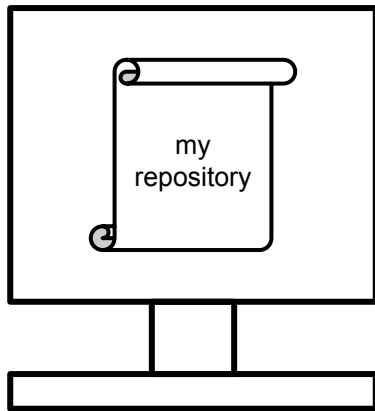
Create save points (called **commits**)

**Push** the updates to GitHub (from your laptop) to back up your work



# Initialize a repository

```
git init
```

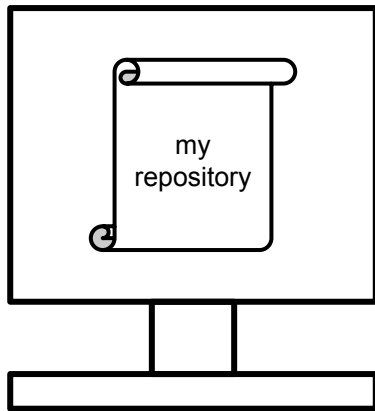


main

# Add and Commit changes

```
git add <files>
```

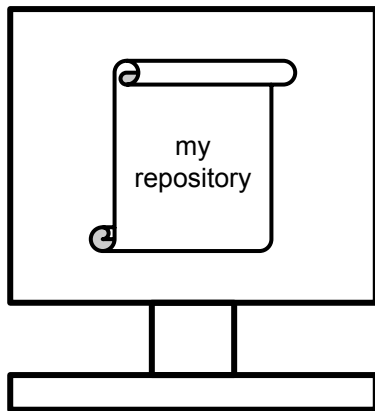
```
git commit -m "some message"
```



main —●—●—●—●—●—

# Add a remote that points to GitHub

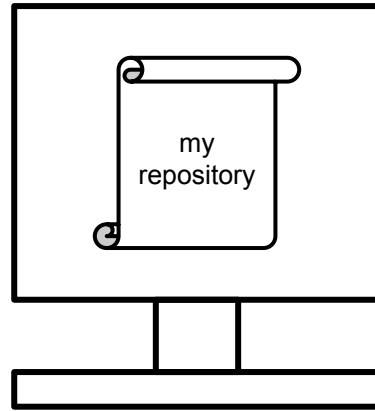
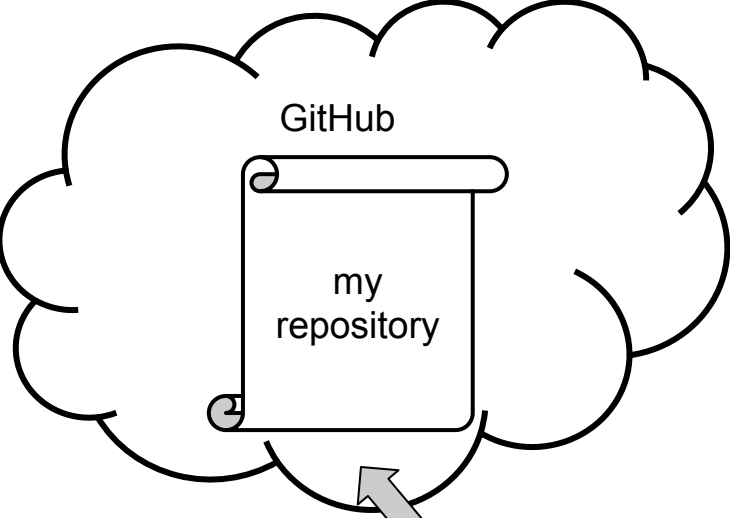
```
git remote add origin <link>
```



main —●—●—●—●—●

remote:

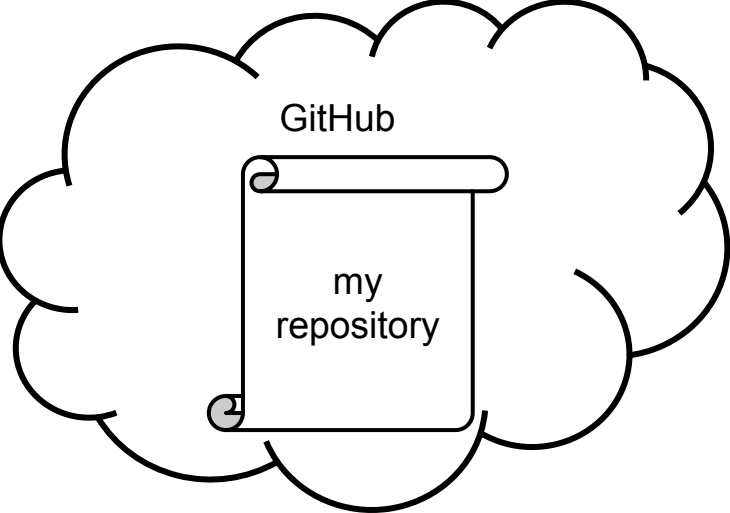
- name: origin
- points to: git@github.com:username




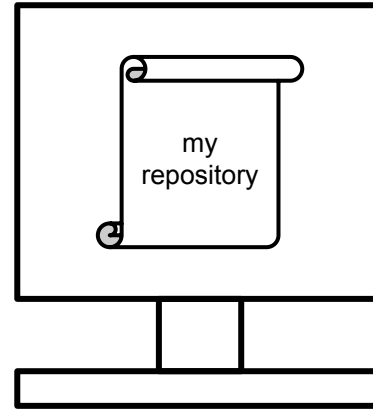
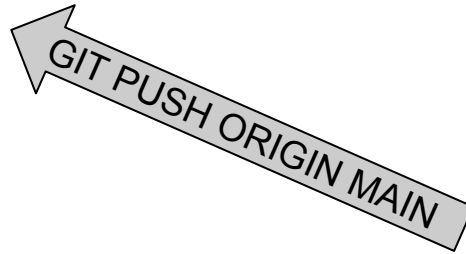
main —●—●—●—●—●—

remote:


- name: origin
- points to: git@github.com:username



main

A horizontal line with five dots representing a sequence of commits.

main

A horizontal line with five dots representing a sequence of commits.

remote:

- name: origin
- points to: git@github.com:username

# Demo + Exercise

# Motivation

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

1. Iterating on (+ keeping track of) different versions of the code is easy
2. Work is backed up to and hosted on the cloud
3. Collaboration is productive



# Iterating on Different Versions

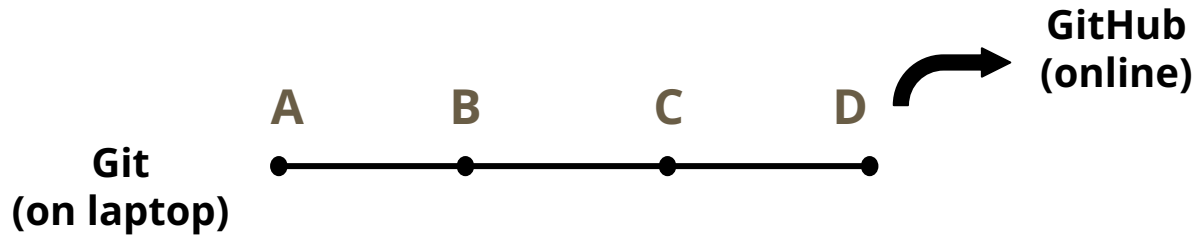
The ease or difficulty of adding a new feature to the code base may depend on the state / version of the codebase.

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.

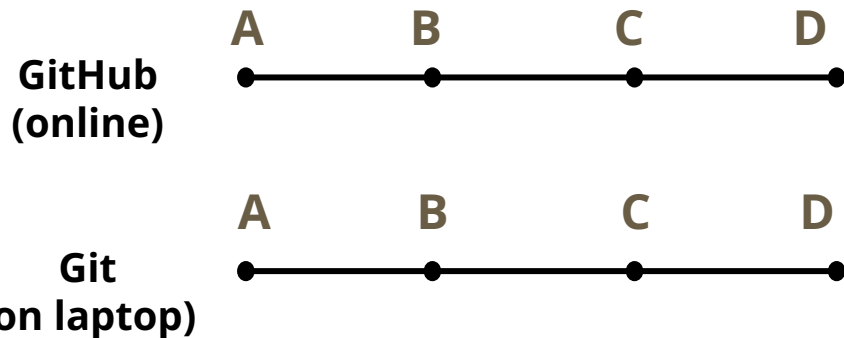
It may be easiest to add this feature at a specific commit.



# 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.

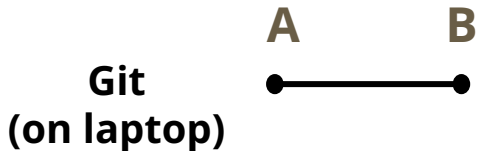
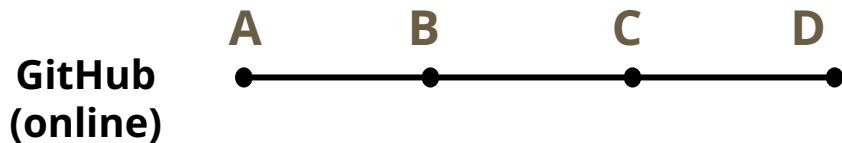
It may be easiest to add this feature at a specific commit.



# 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.

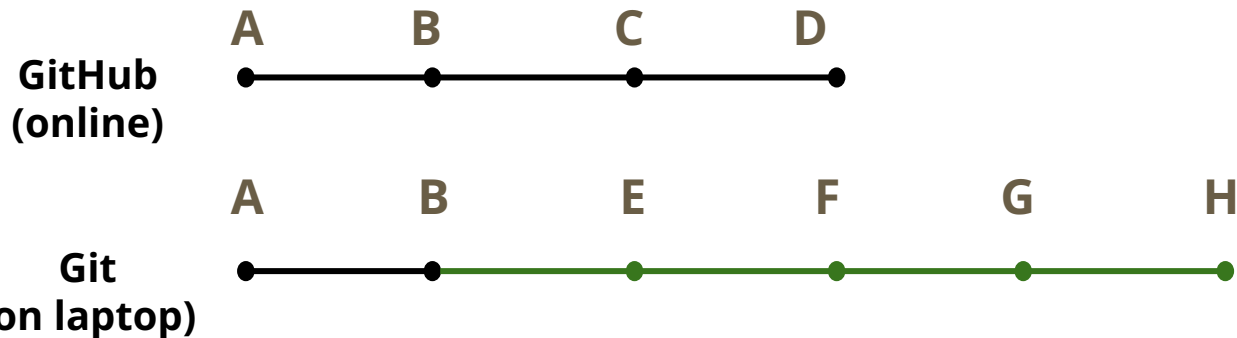
It may be easiest to add this feature at a specific commit.



# 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.

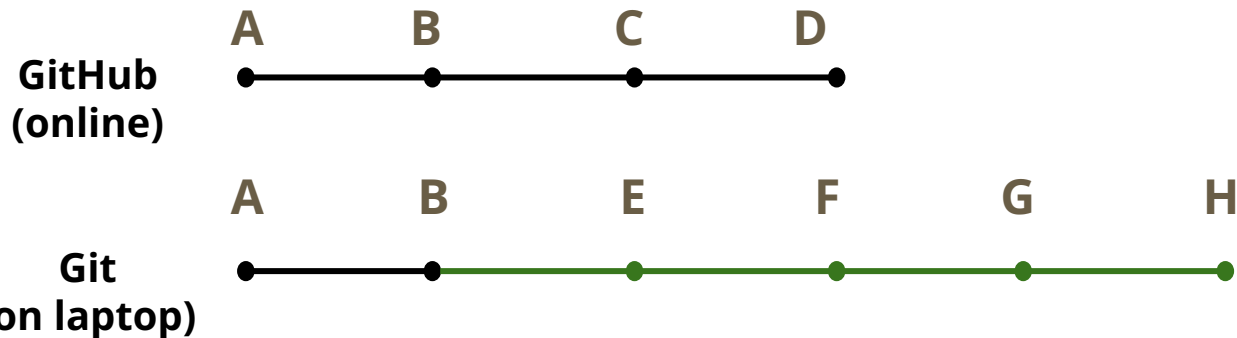
It may be easiest to add this feature at a specific commit.



# Iterating on Different Versions

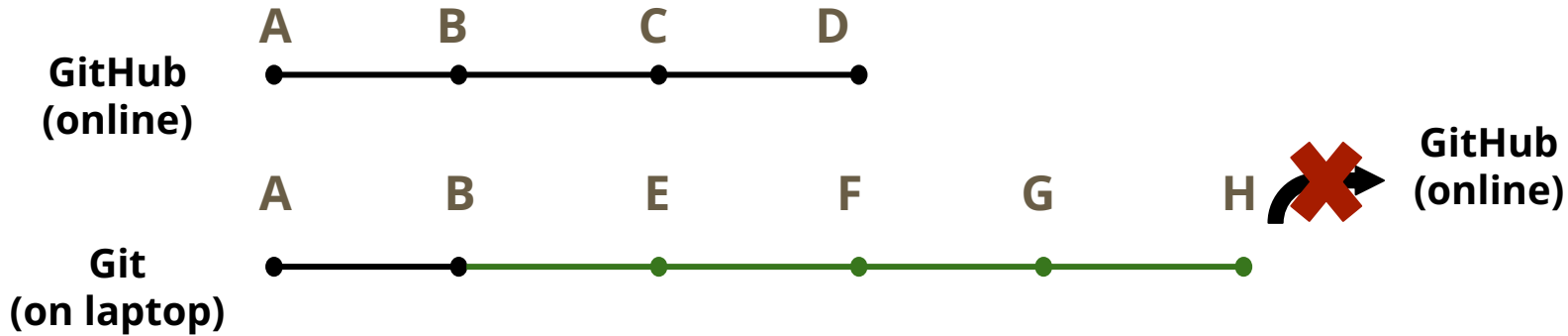
The ease or difficulty of adding a new feature to the code base may depend on the state / version of the codebase.

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



# Iterating on Different Versions

What happens now?



# Iterating on Different Versions

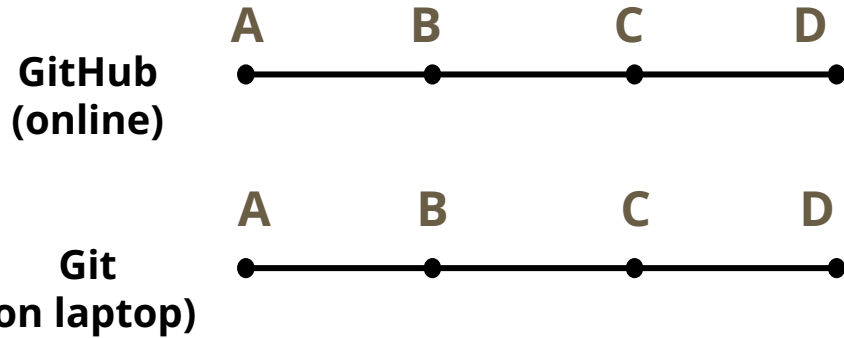
Looks like we need:

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)



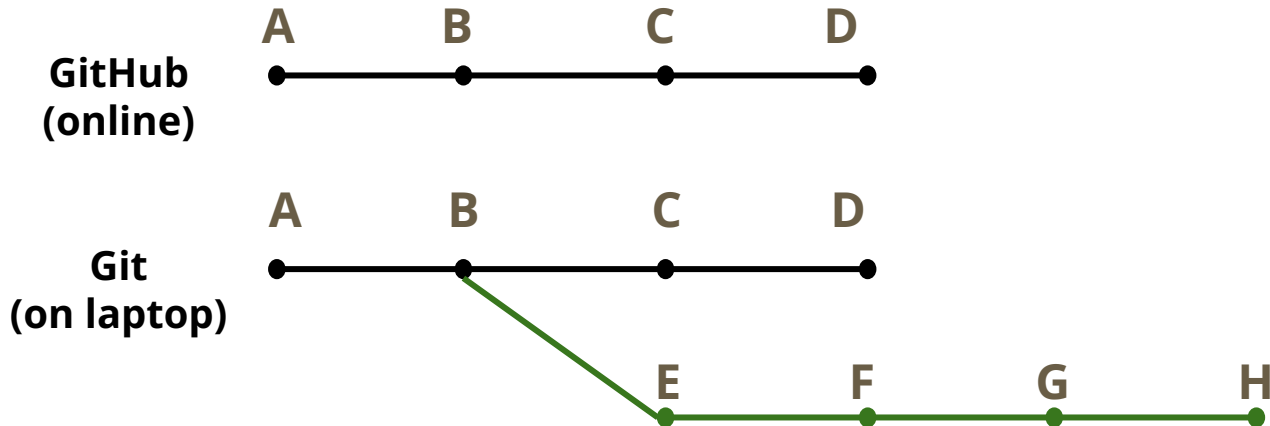
# Iterating on Different Versions

Let's try that again!



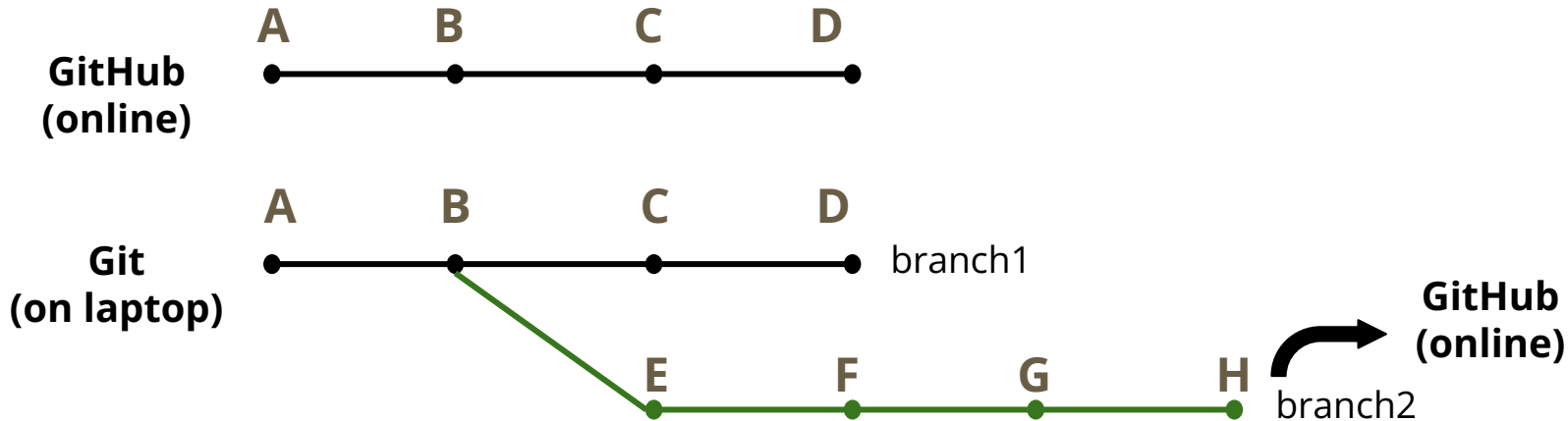
# Iterating on Different Versions

We will **branch** off of that particular commit **to create a new timeline**



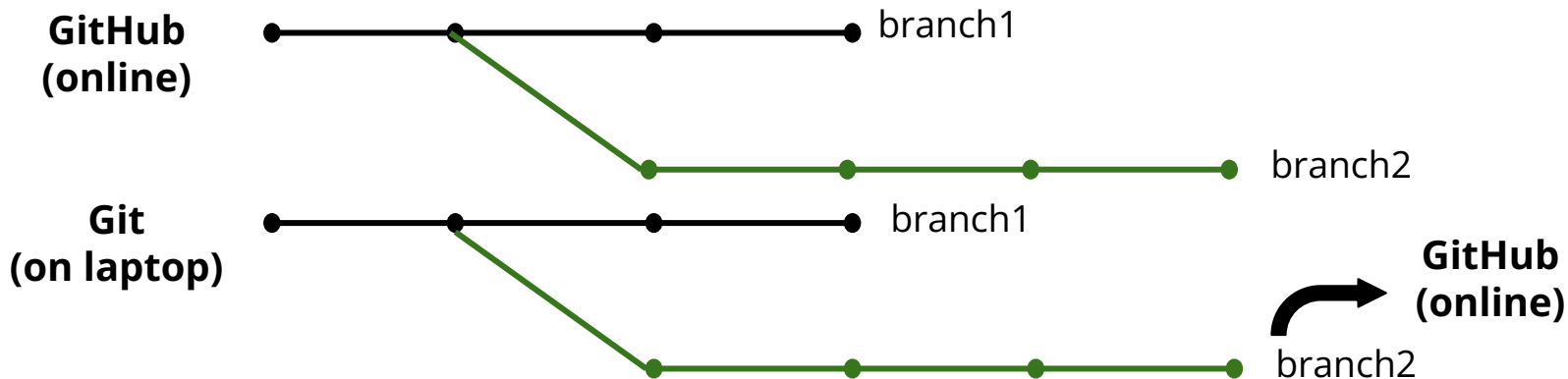
# Iterating on Different Versions

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



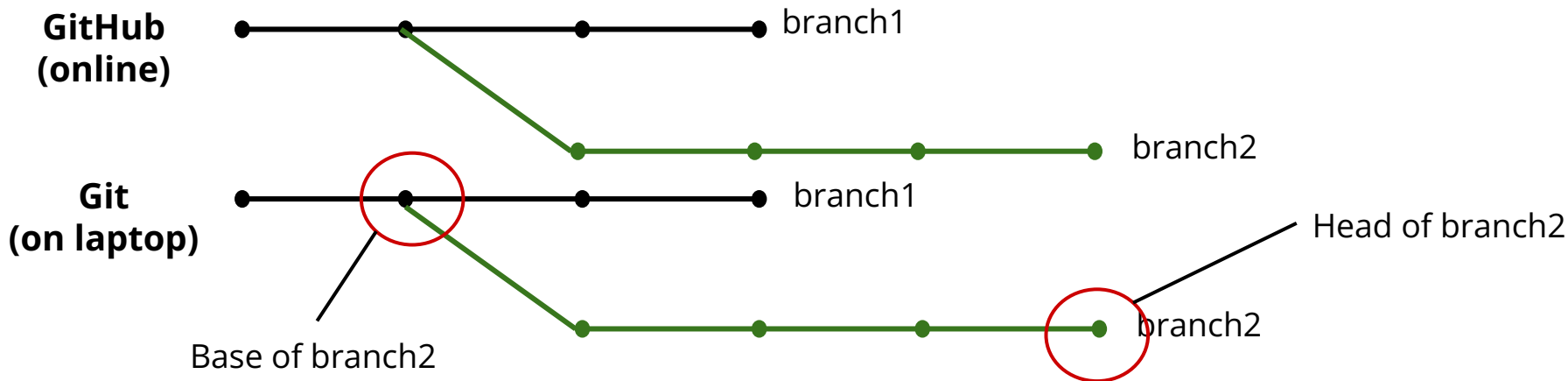
# Iterating on Different Versions

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



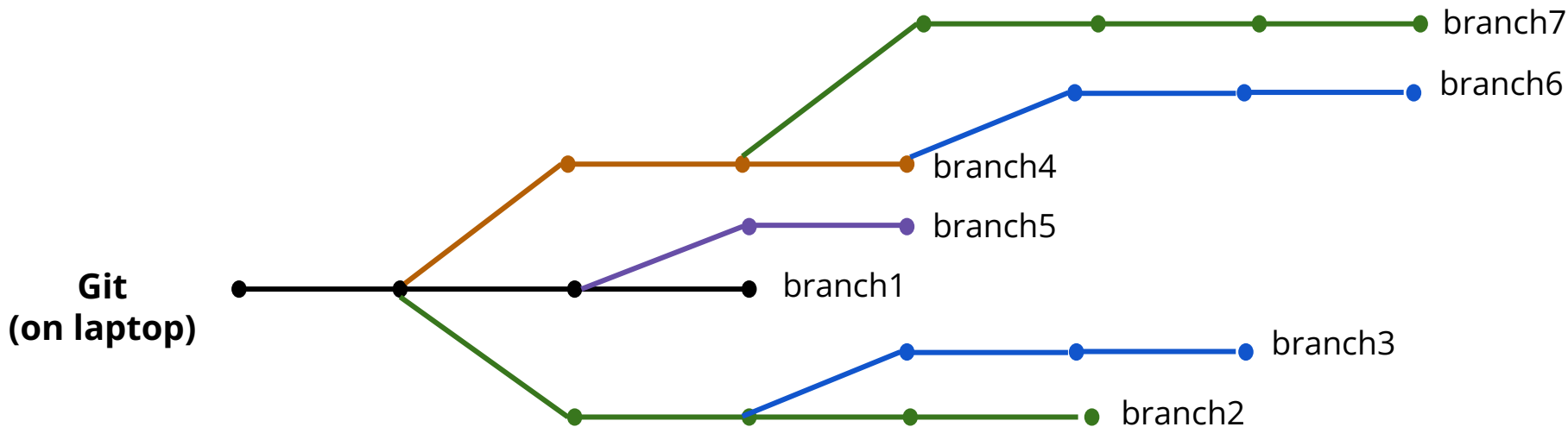
# Iterating on Different Versions

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



# Iterating on Different Versions

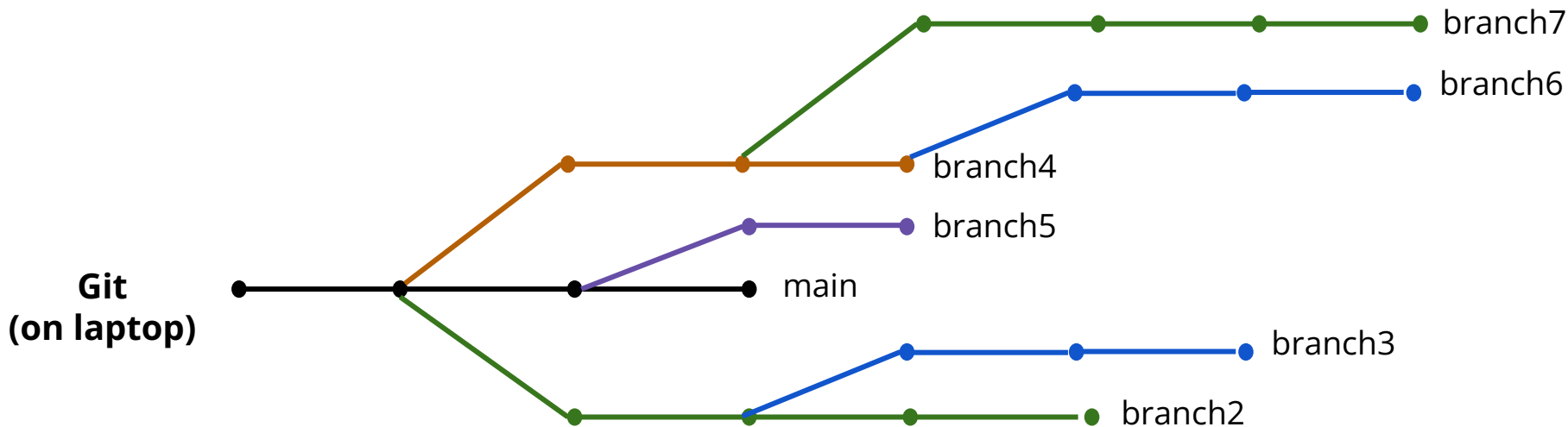
We can create lots of **branches**



# Iterating on Different Versions

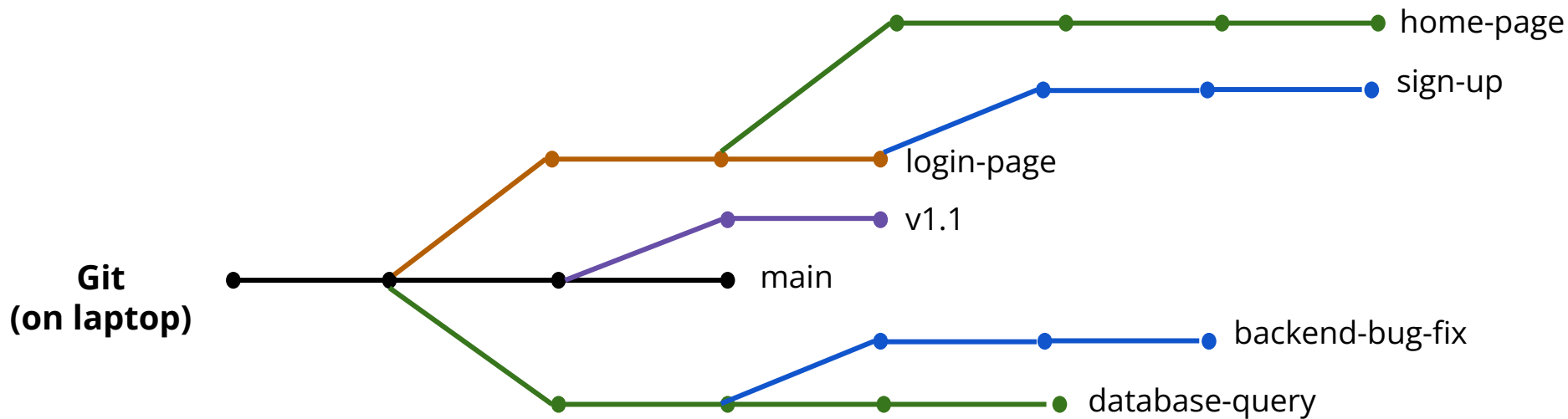
But one branch needs to be chosen as the primary, stable branch

This branch is typically called the “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



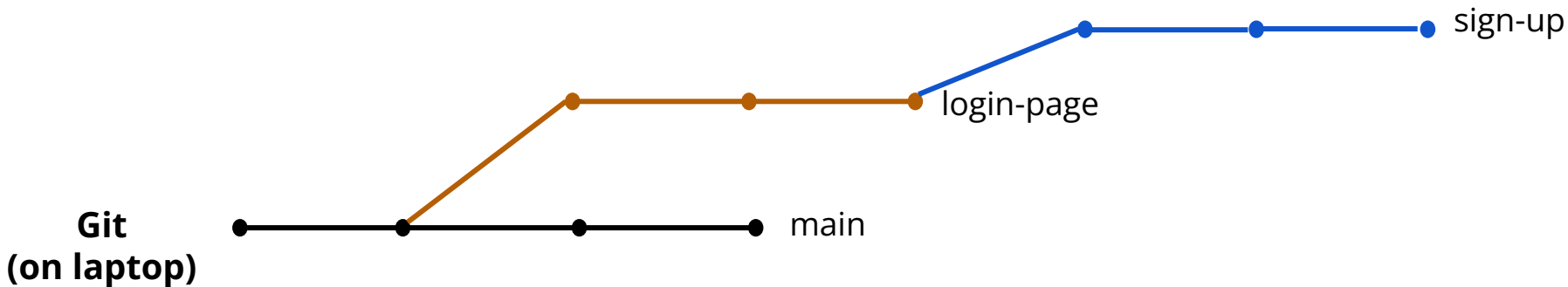


# Iterating on Different Versions

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

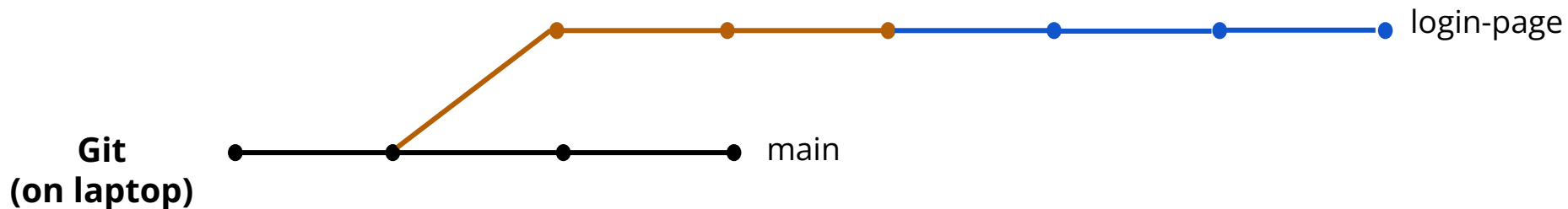
# Iterating on Different Versions

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



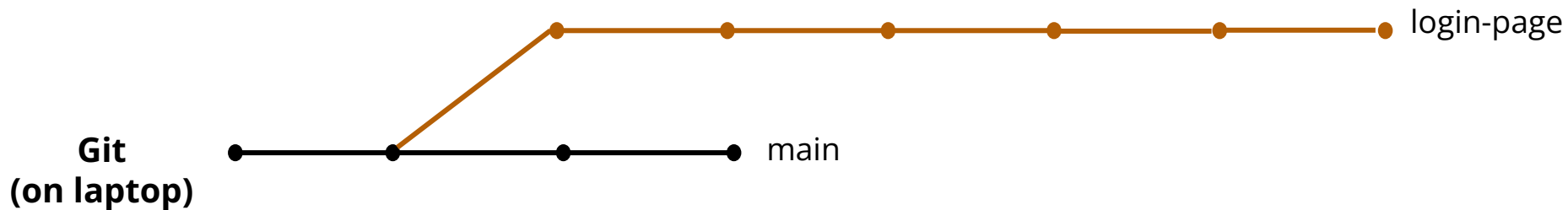
# Iterating on Different Versions

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



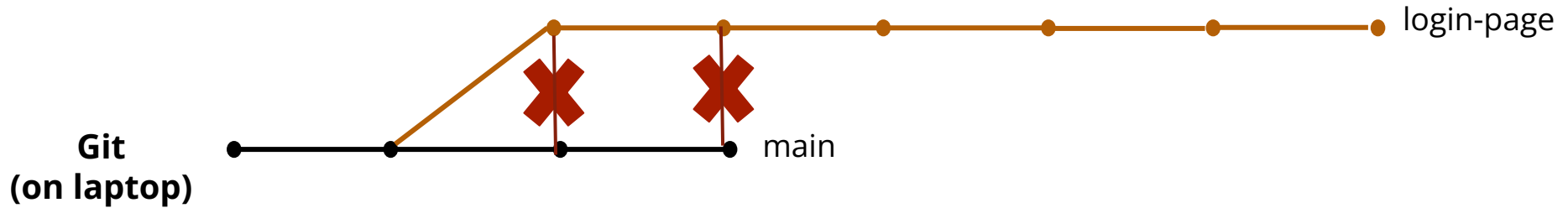
# Iterating on Different Versions

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



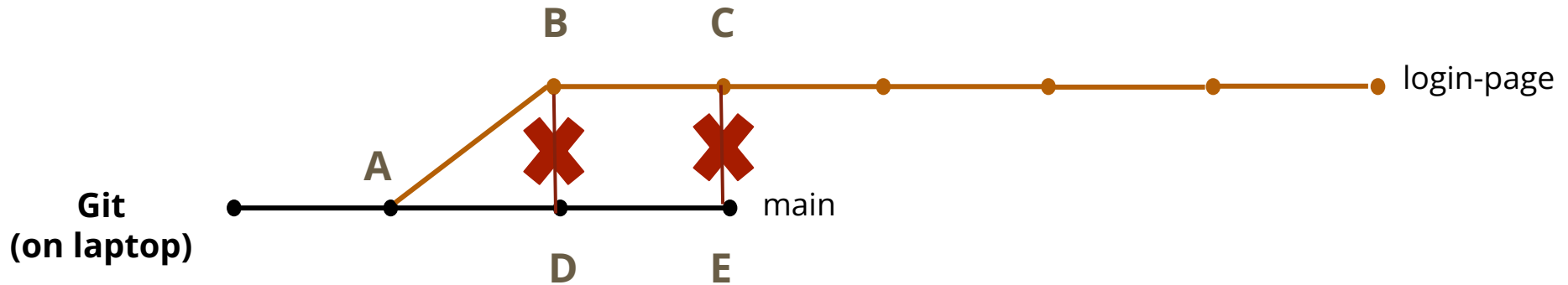
# Iterating on Different Versions

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



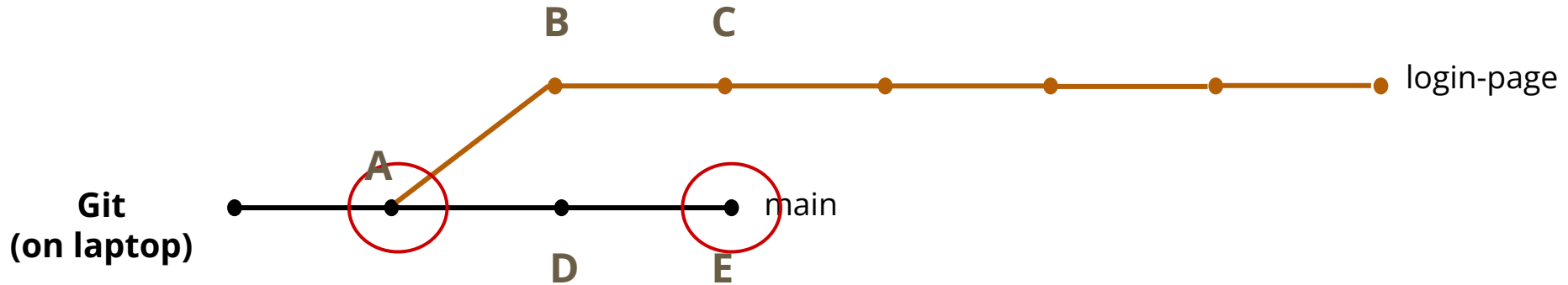
## Iterating on Different Versions

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

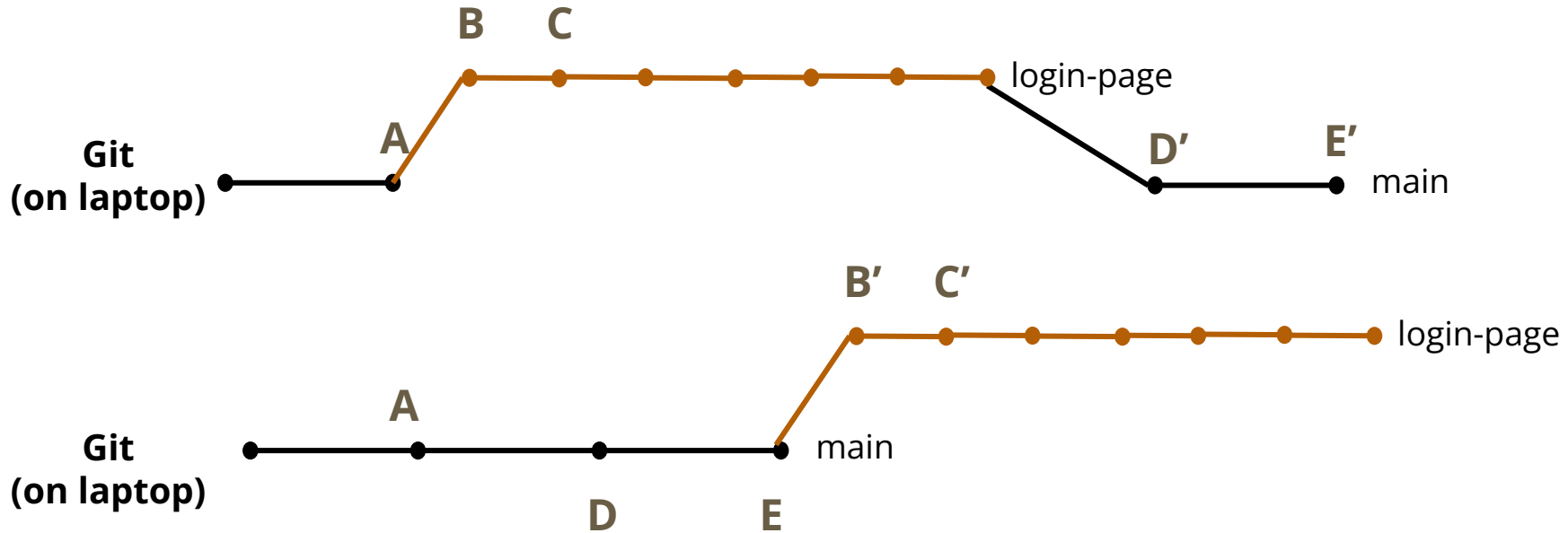


# Iterating on Different Versions

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



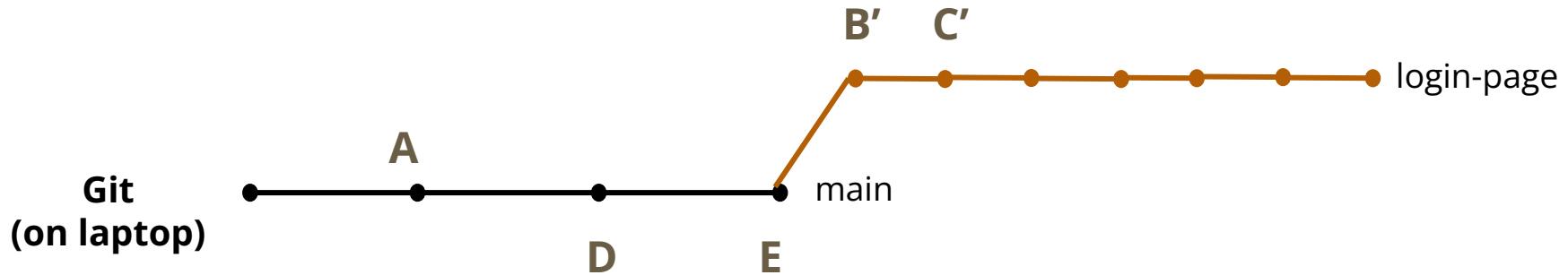
# Two options for Rebasing





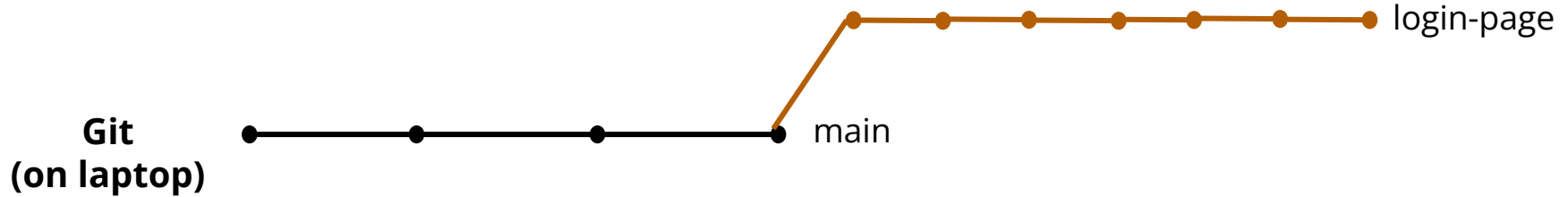
# Iterating on Different Versions

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



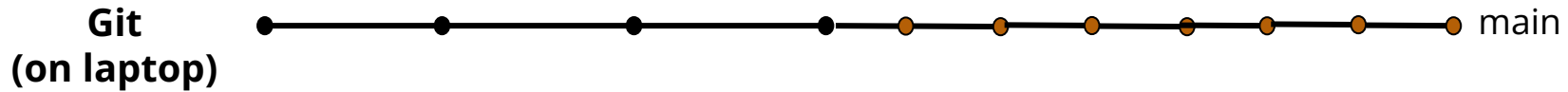
# Iterating on Different Versions

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.



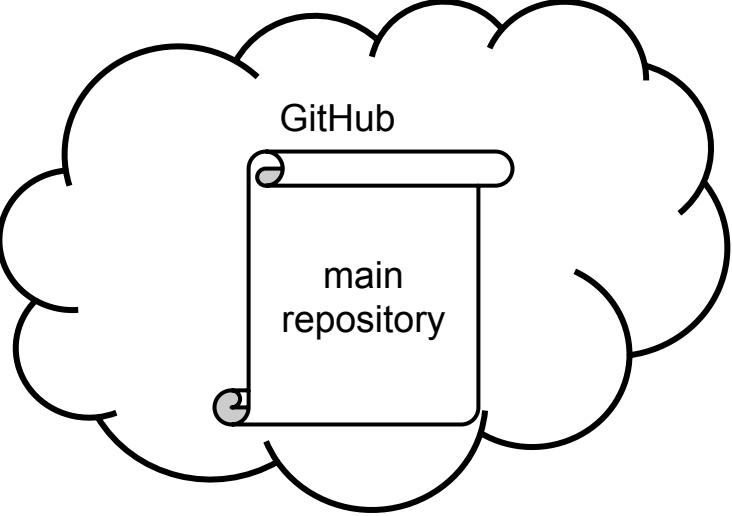
**Collaboration (follow along)**

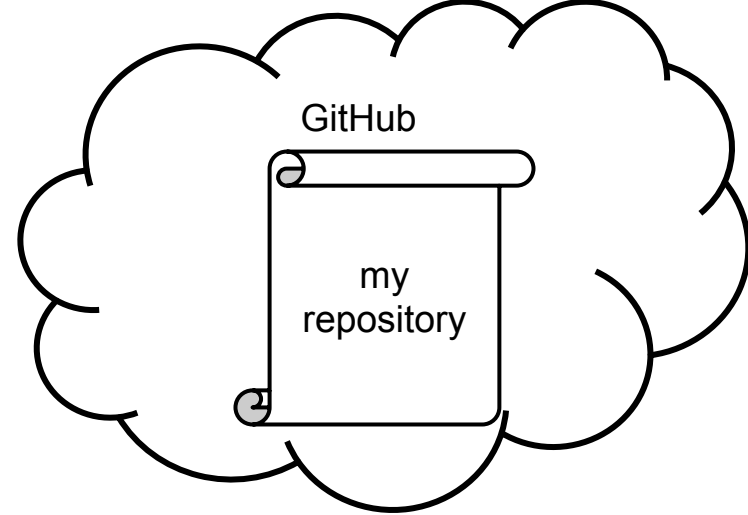
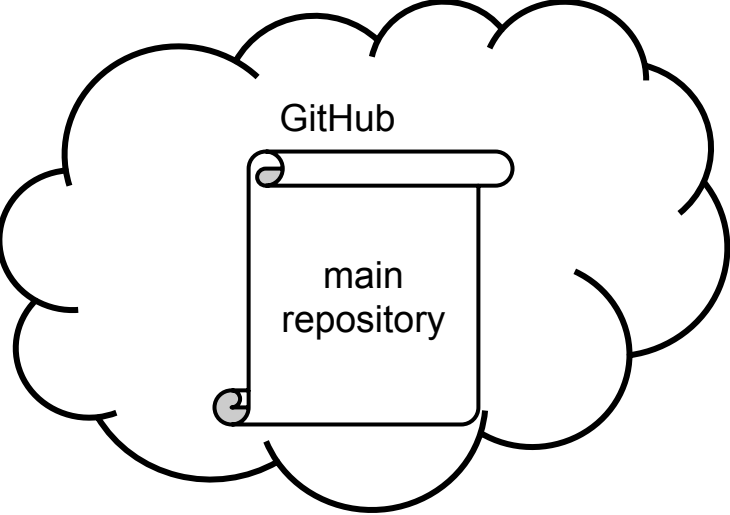
# Collaboration

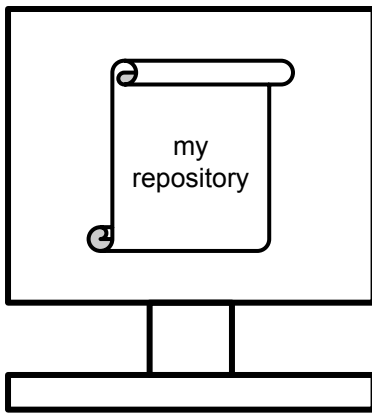
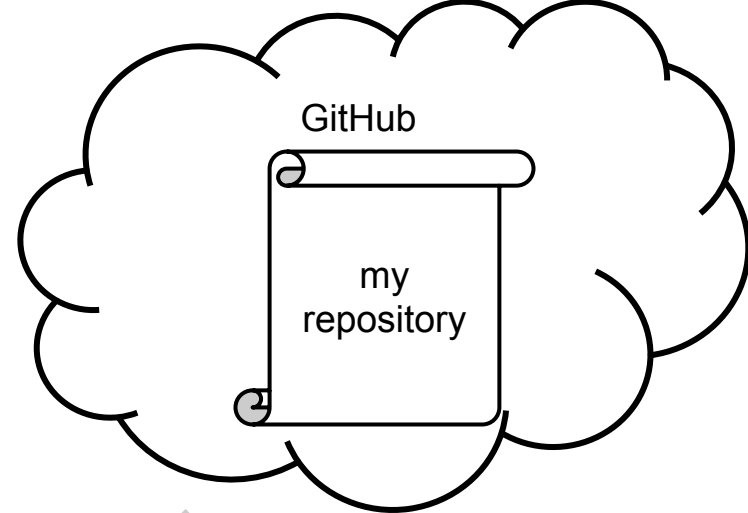
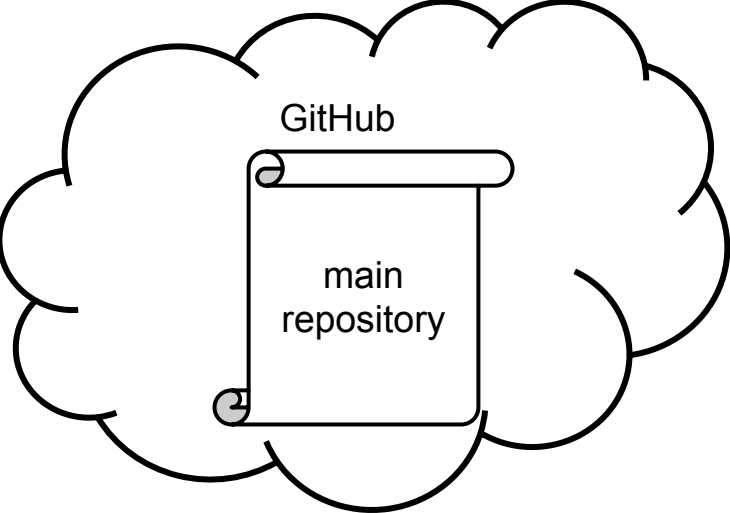
Other repos can be thought of as other branches.

In order to contribute code, collaborators must:

1. Make a copy (**fork**) of 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)



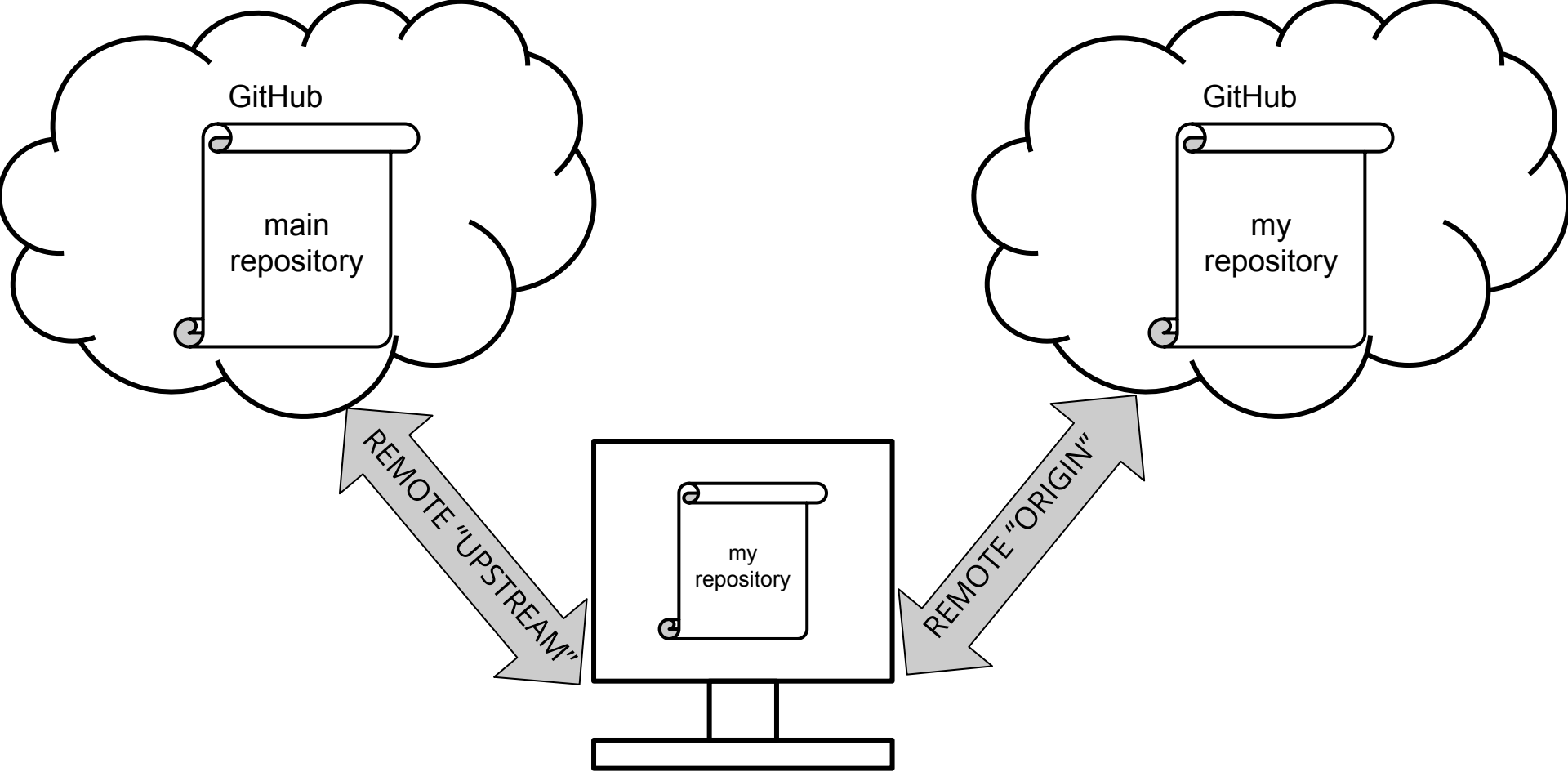


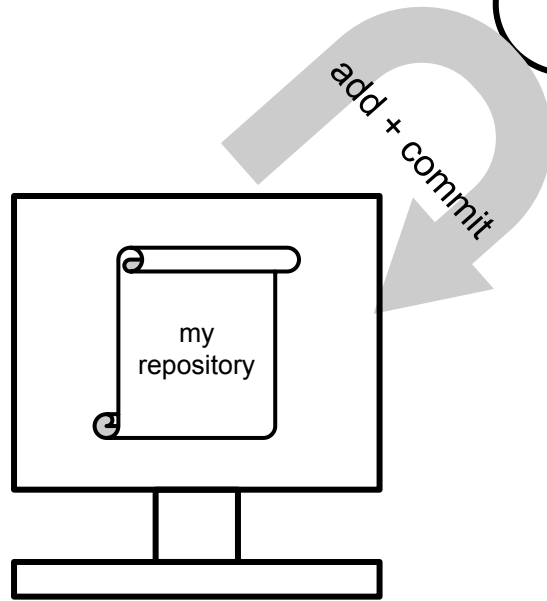
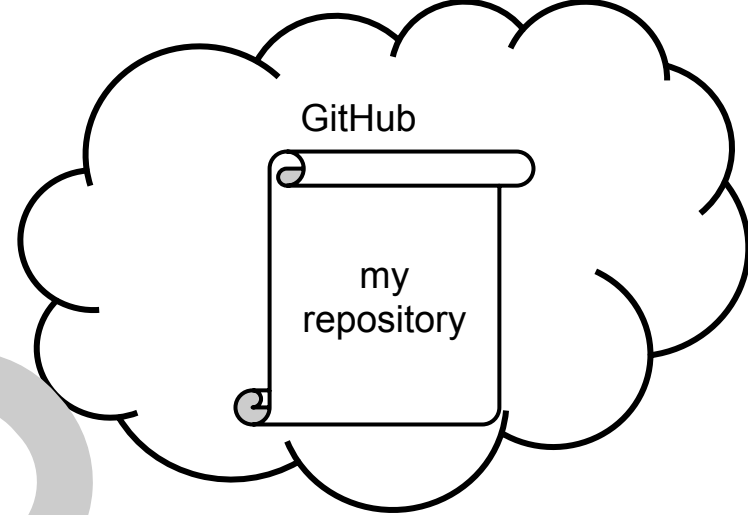
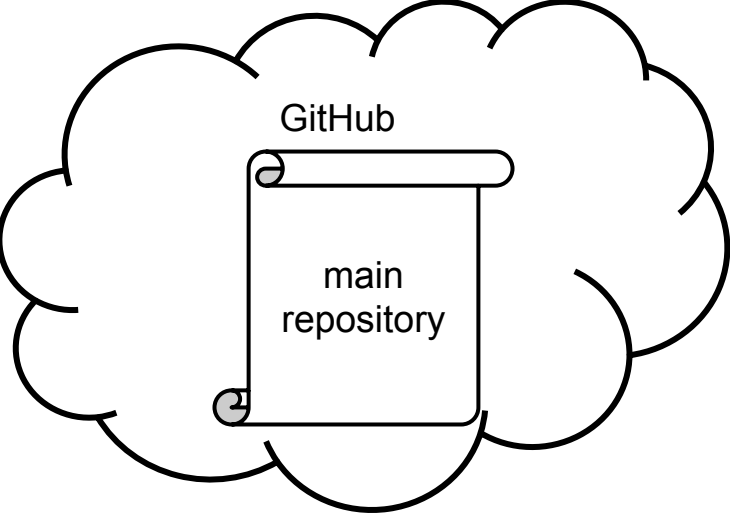


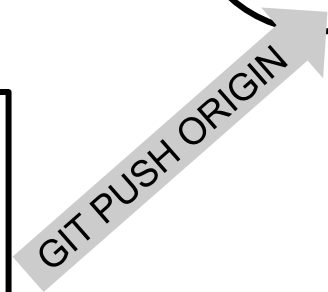
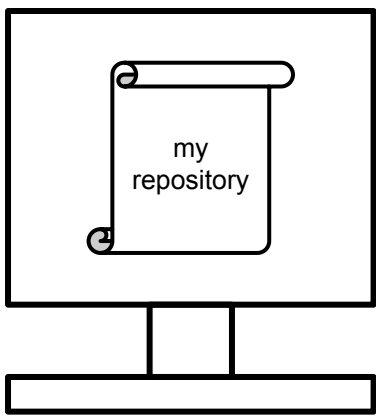
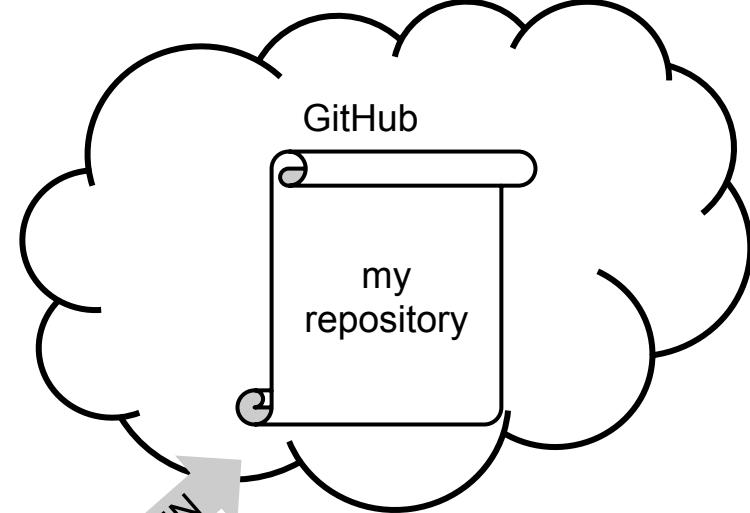
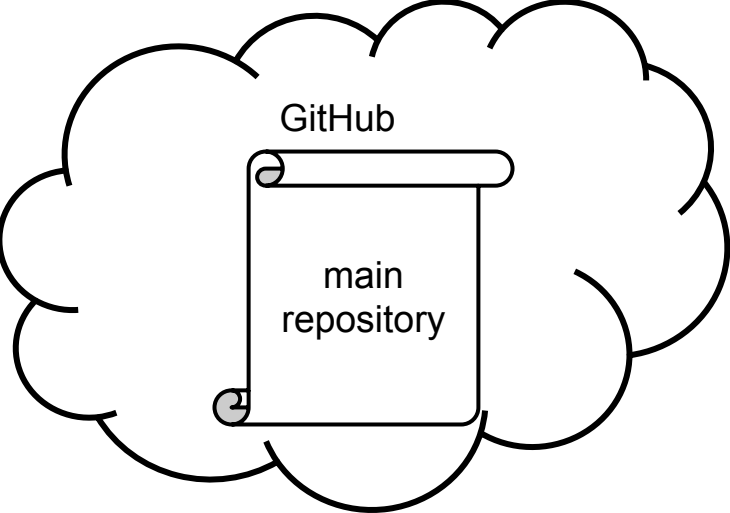
GIT CLONE

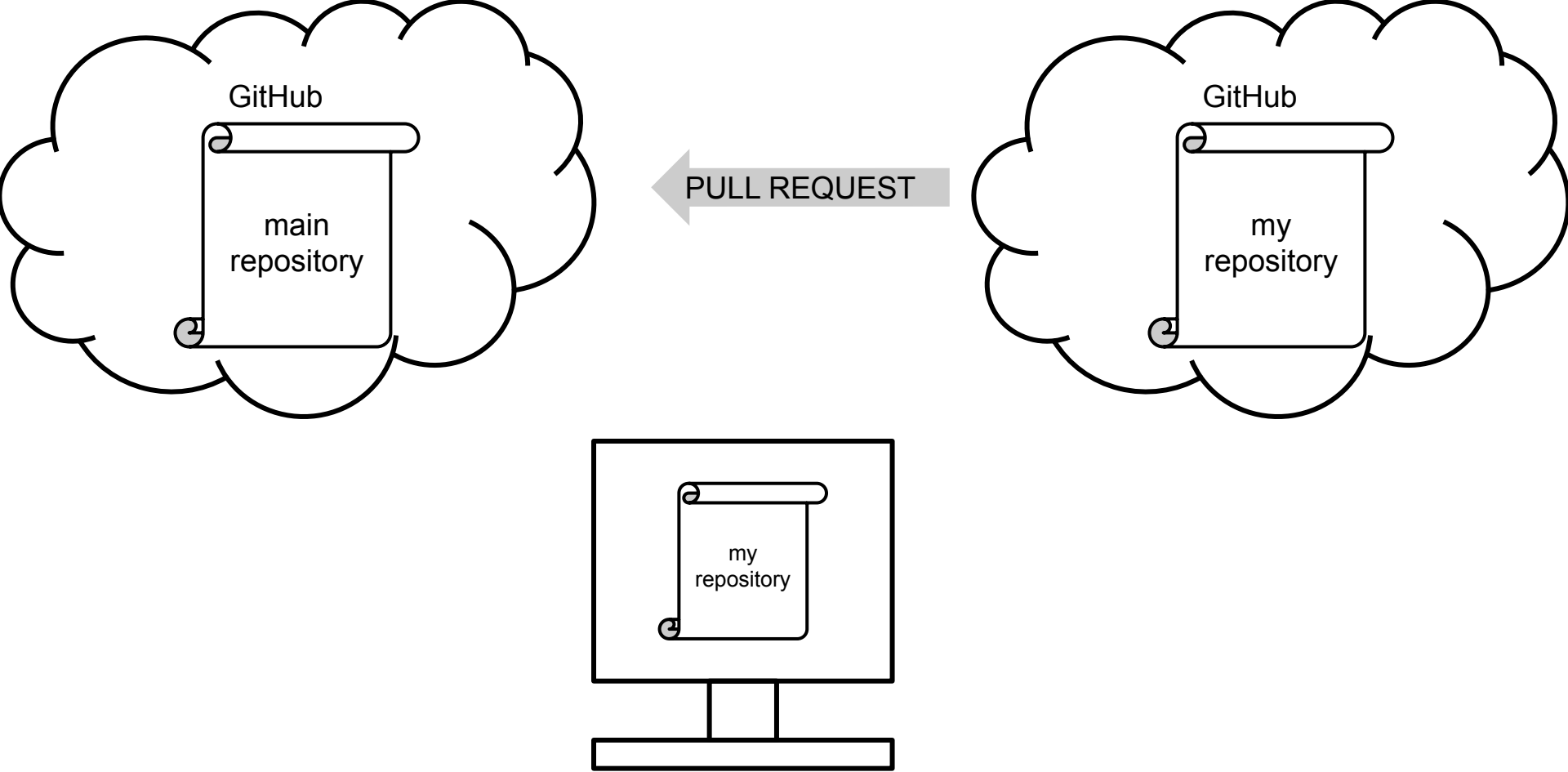
A gray arrow pointing from the "my repository" cloud to the computer monitor, with the text "GIT CLONE" written inside it.



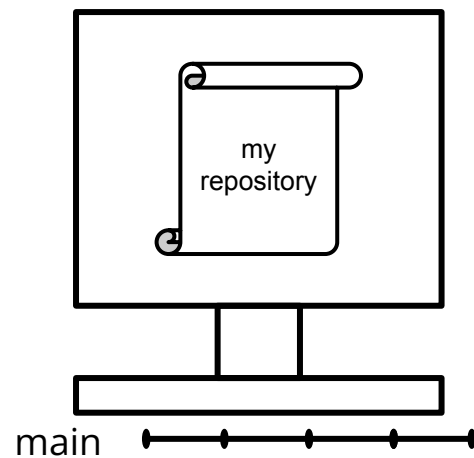
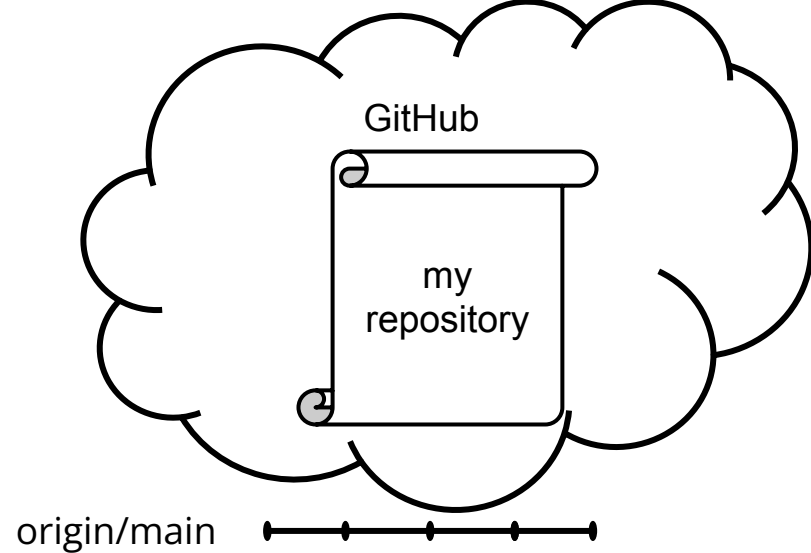
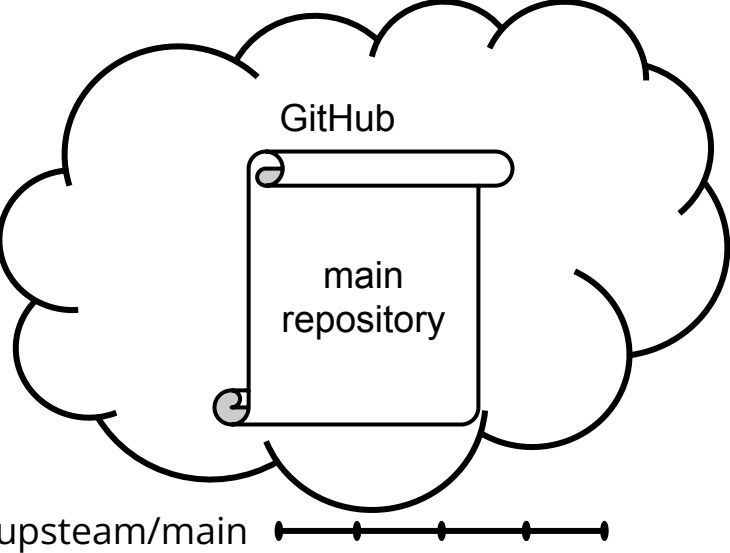


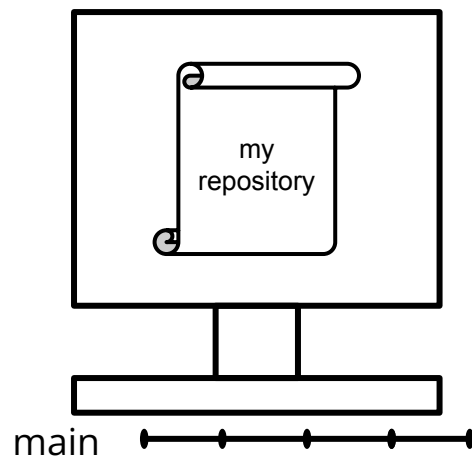
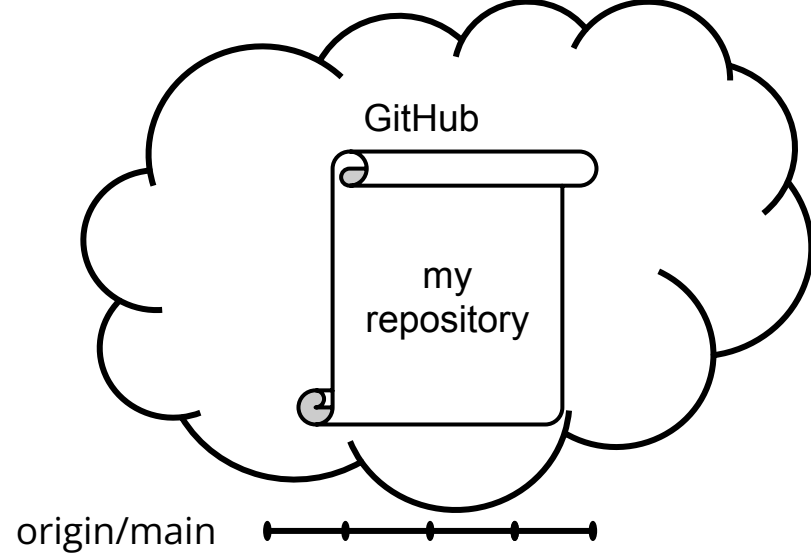
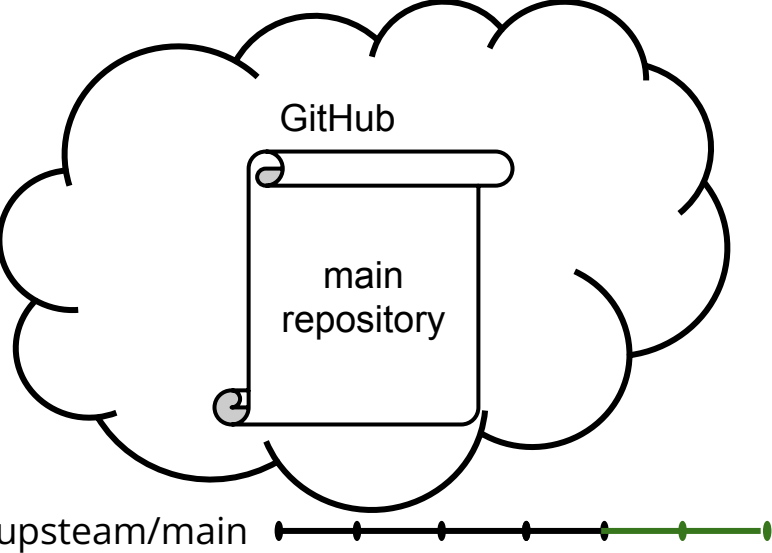


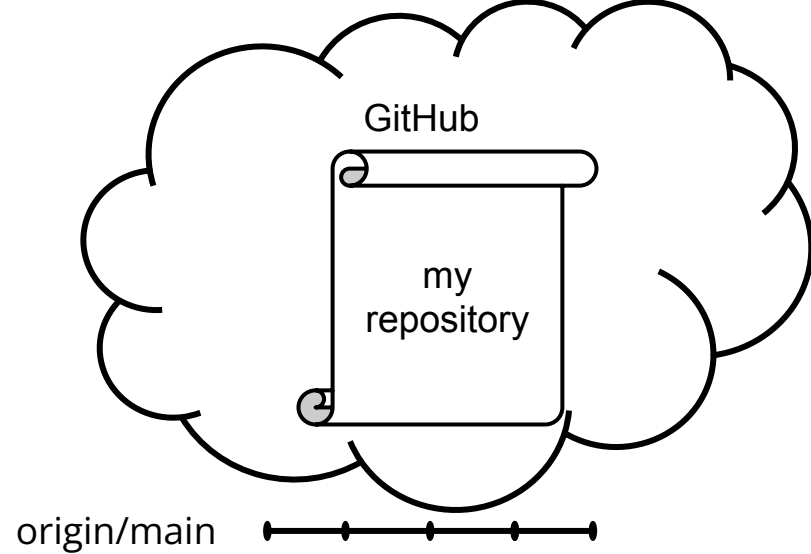
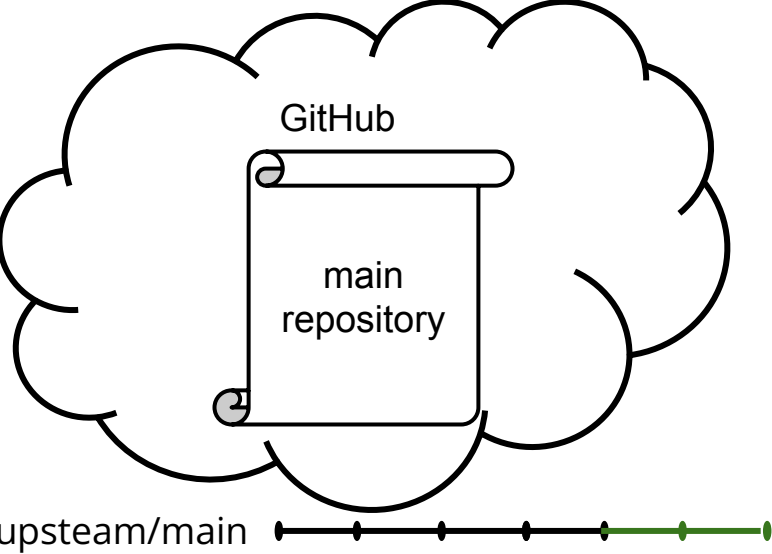




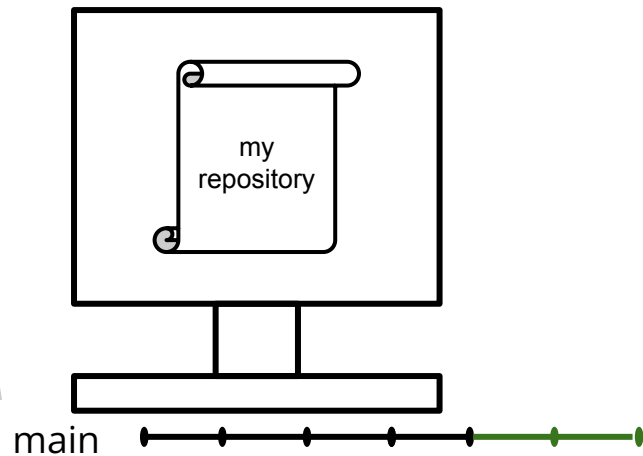
# Keeping your fork updated



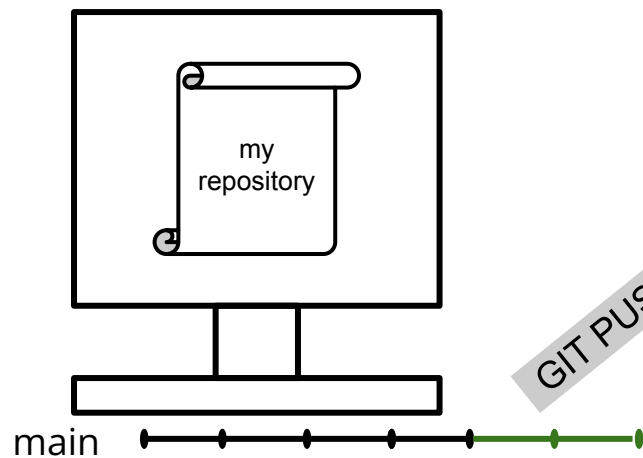
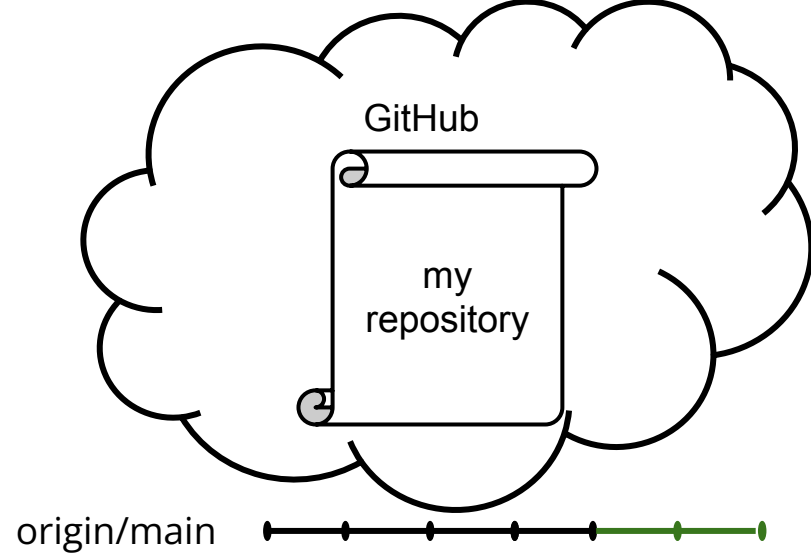
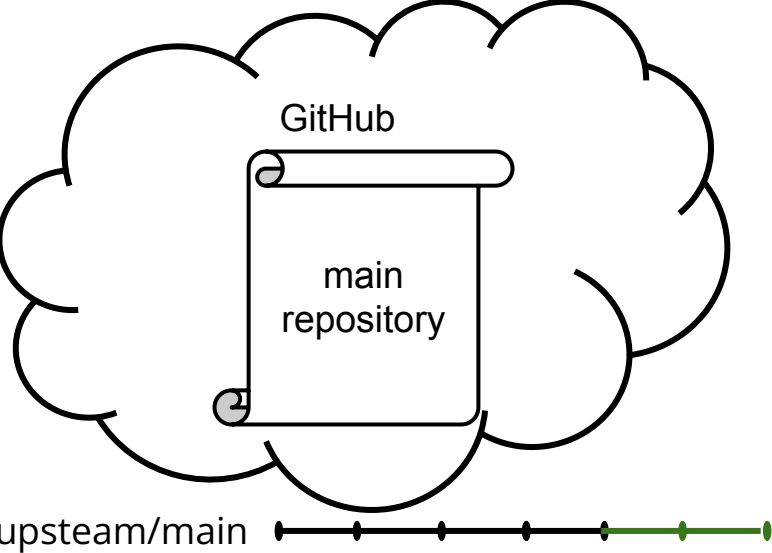




GIT PULL UPSTREAM MAIN





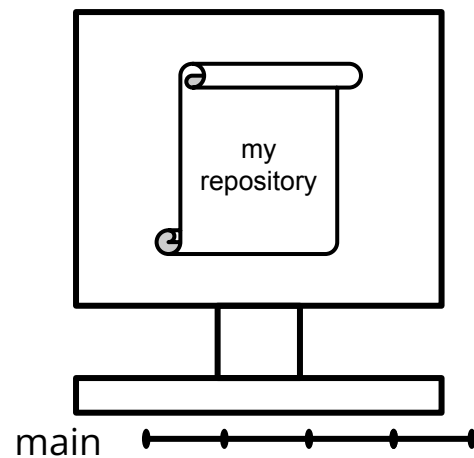
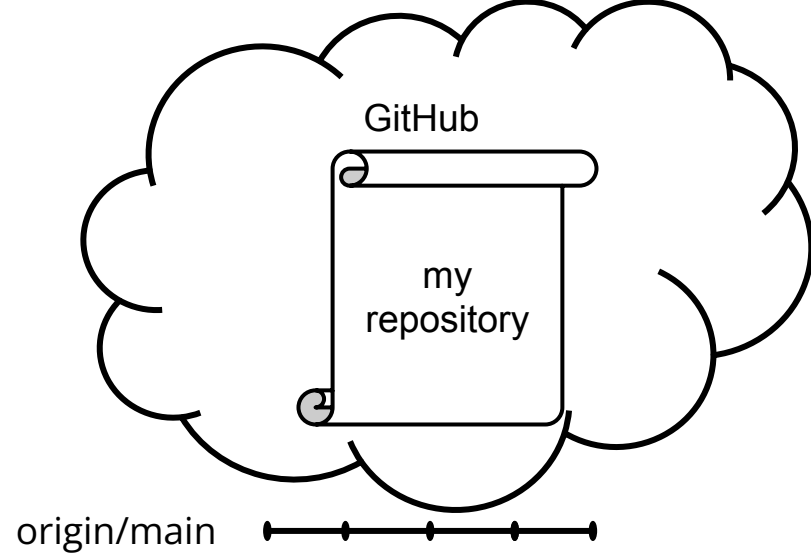
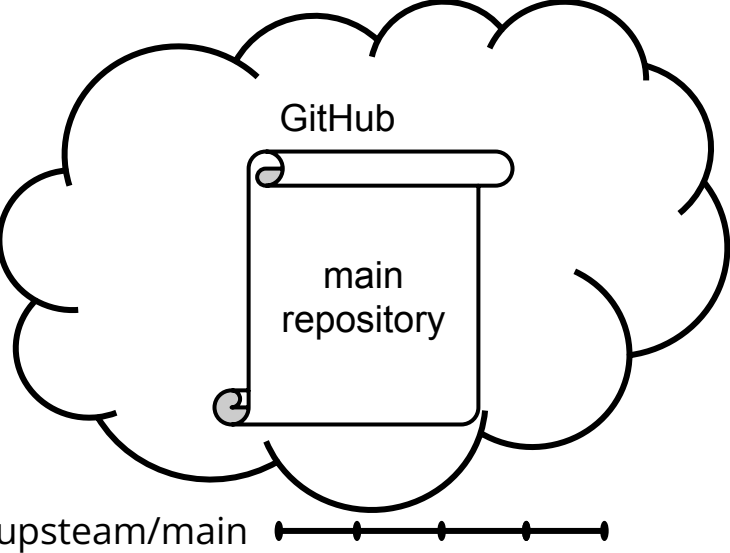


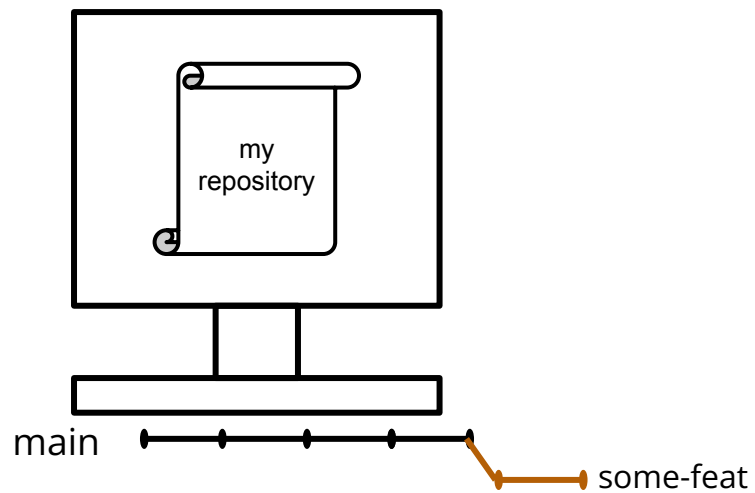
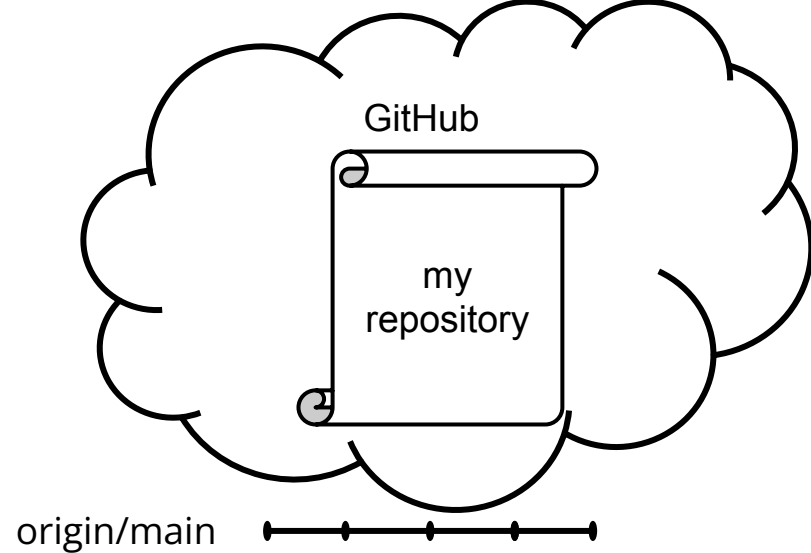
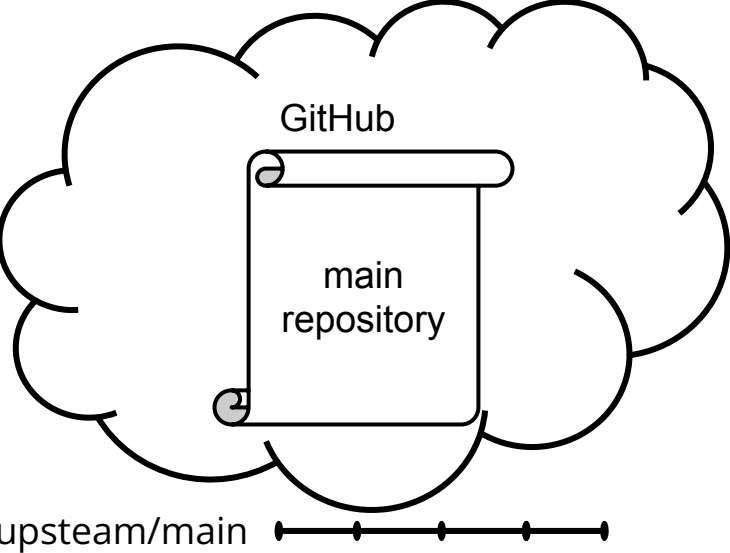
GIT PUSH ORIGIN MAIN

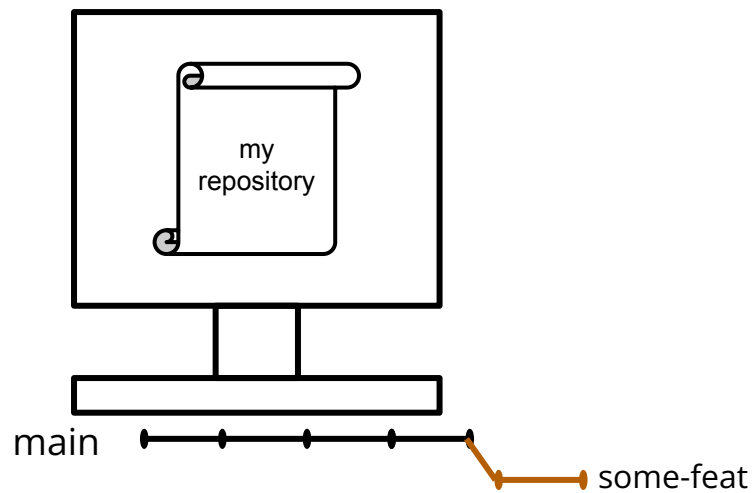
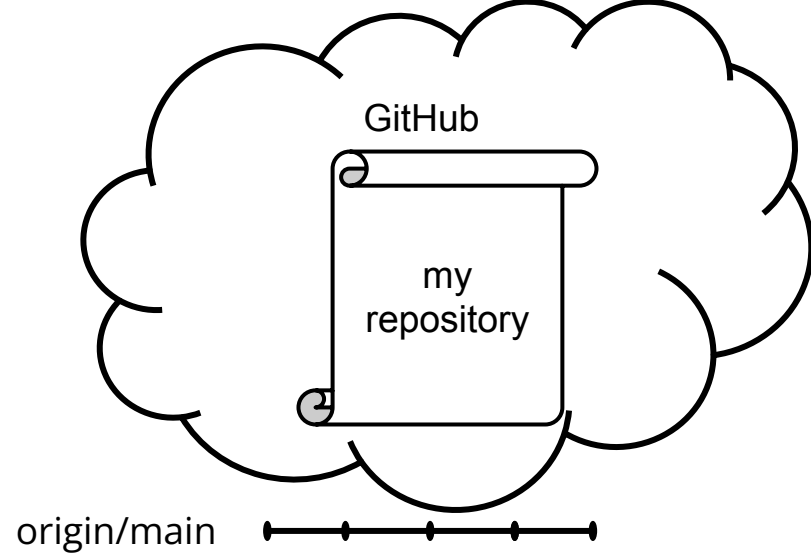
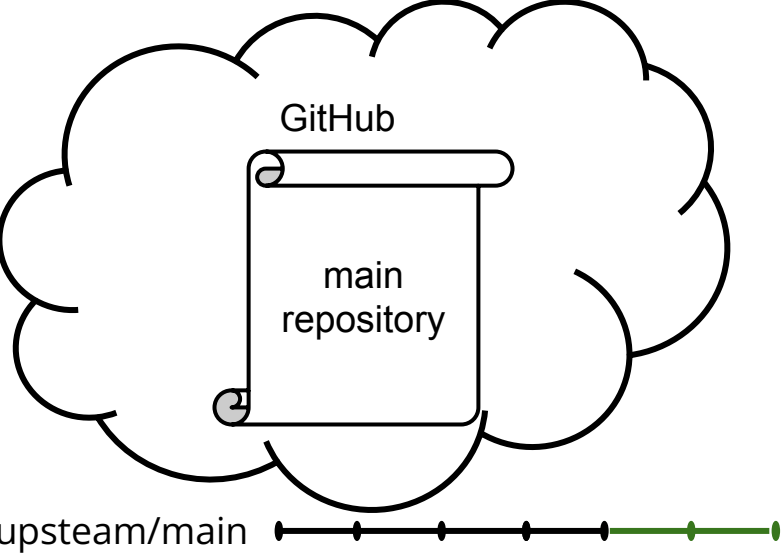
# Best Practices

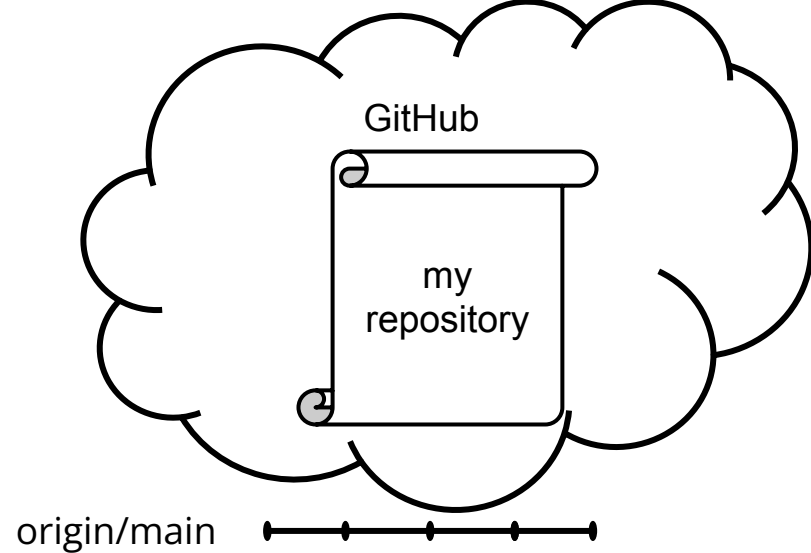
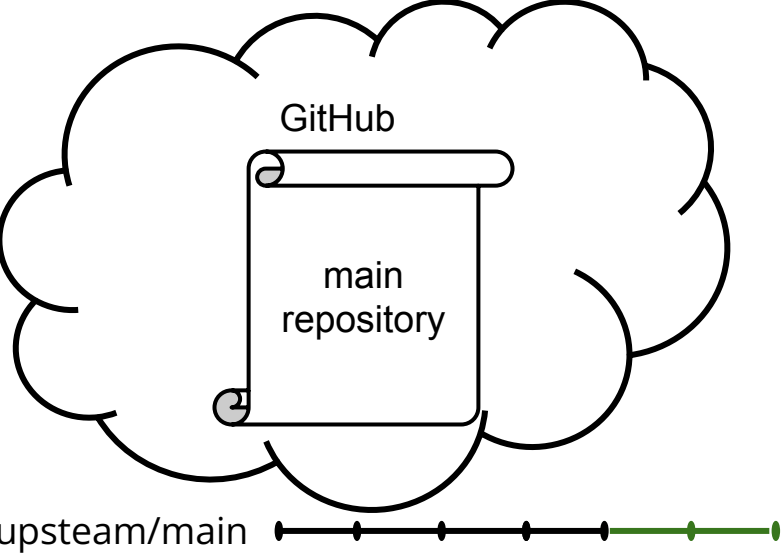
If you **never commit anything to your main branch**, keeping your main branch in sync with the main repository's is easy!

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

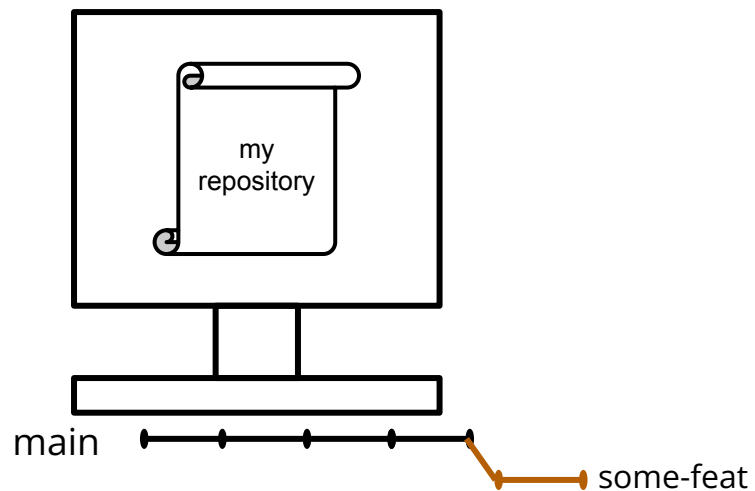


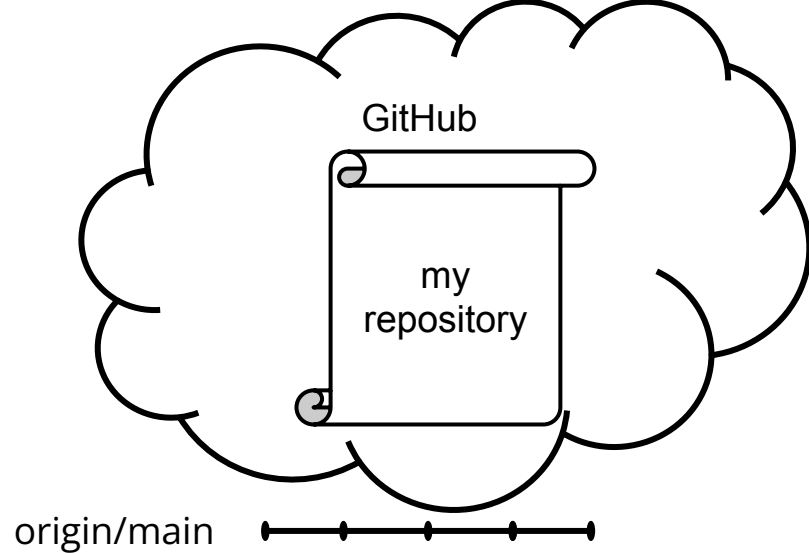
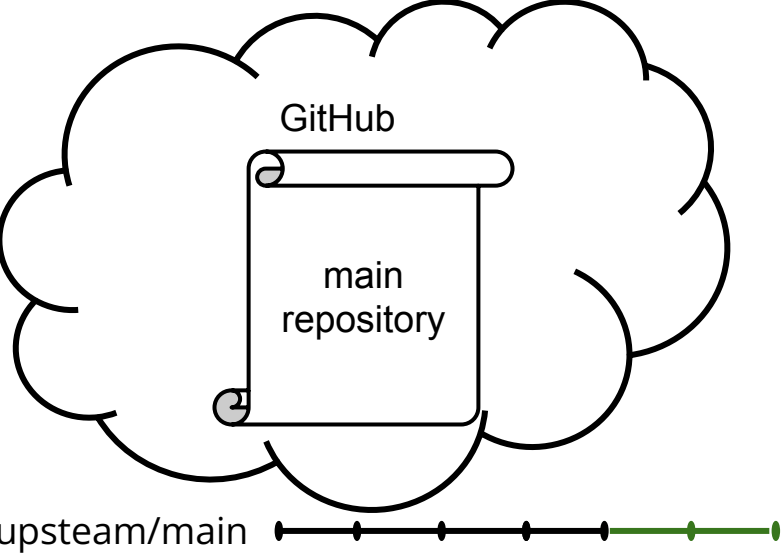




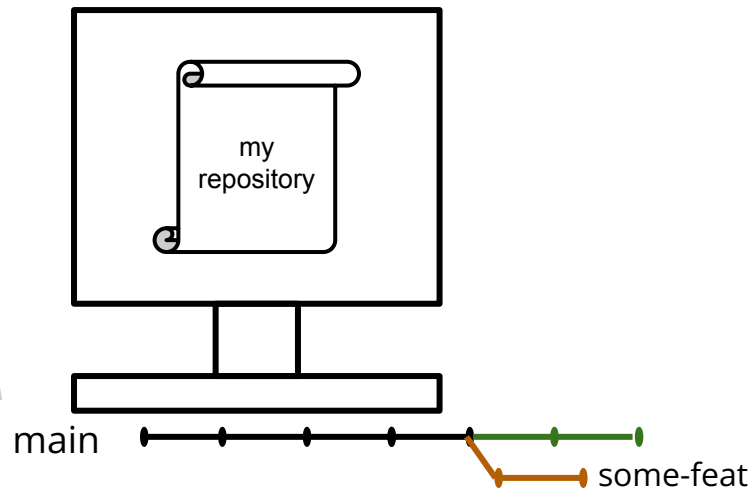


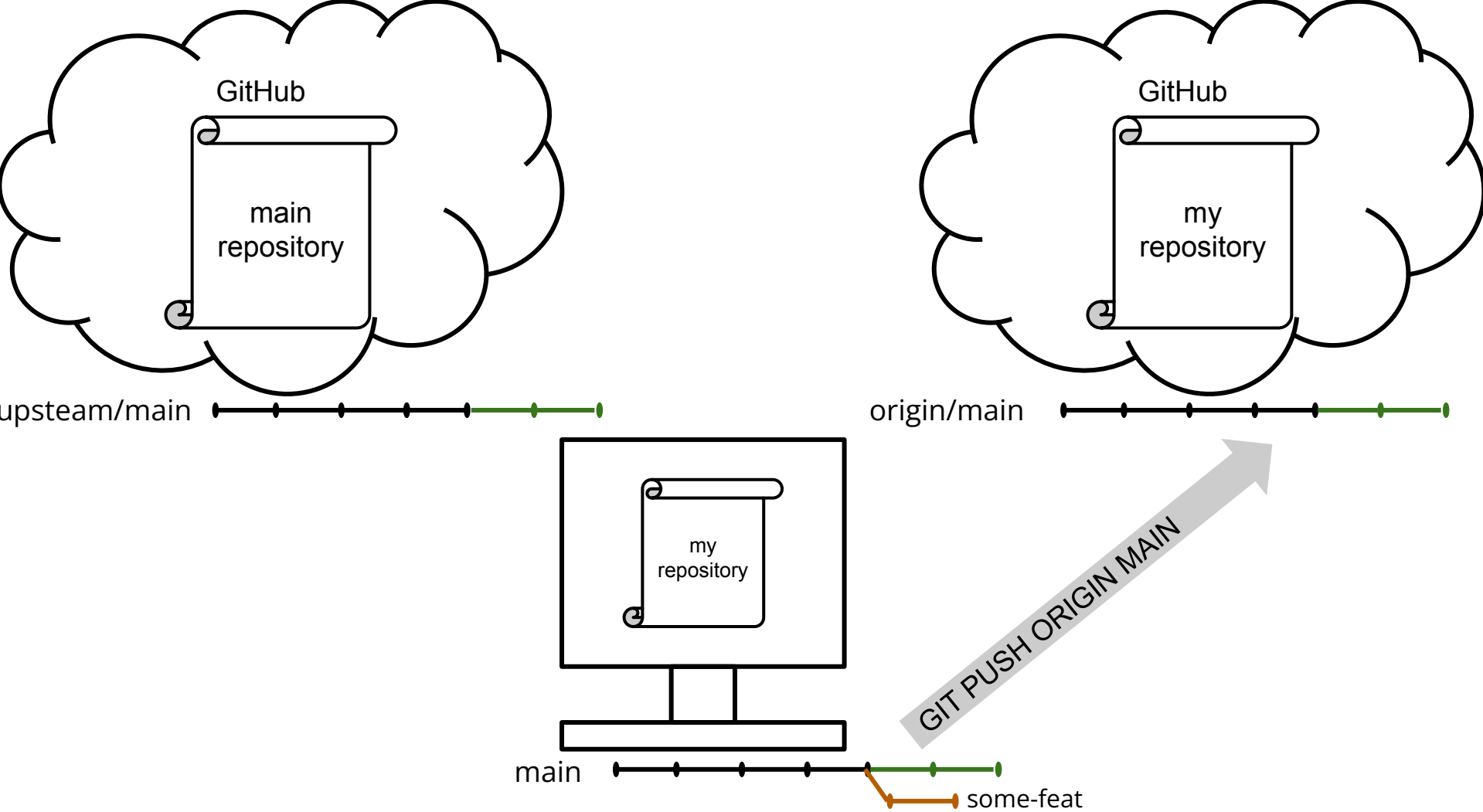
Go back to the main branch



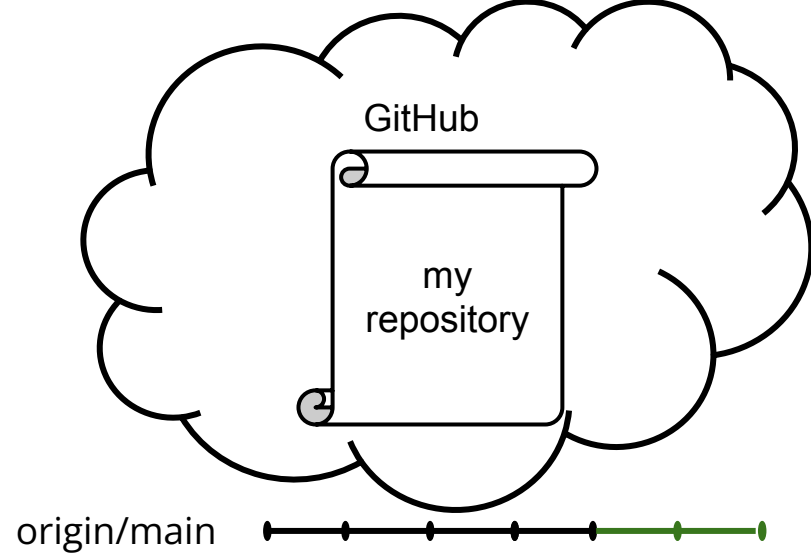
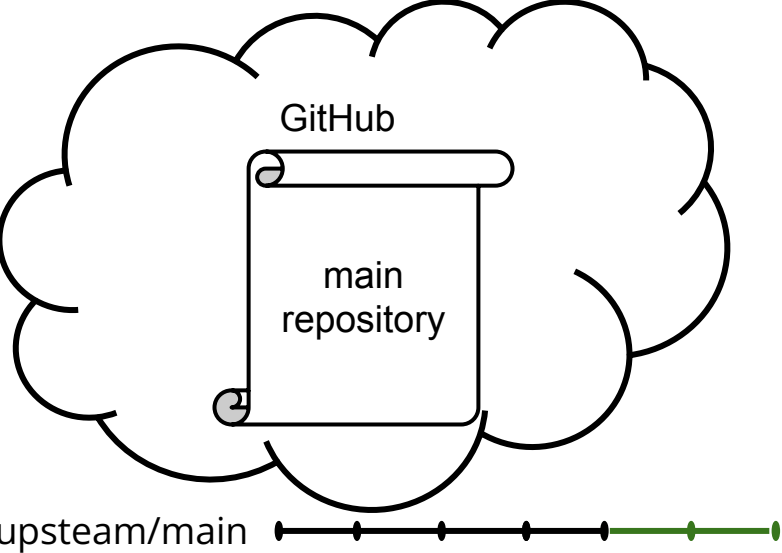


GIT PULL UPSTREAM MAIN

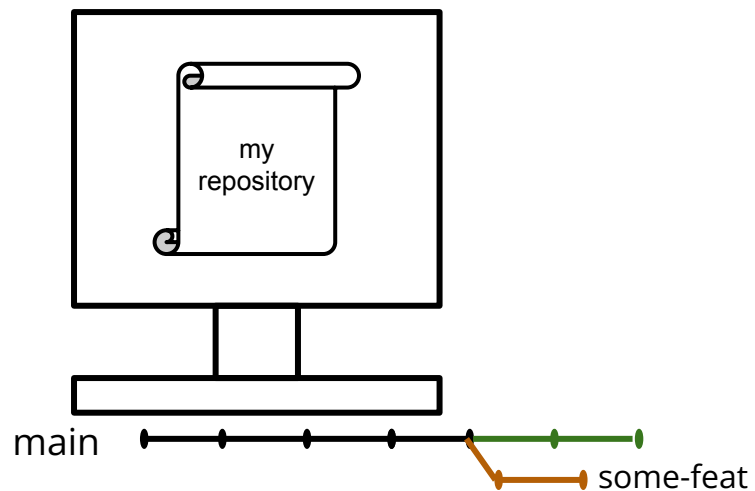


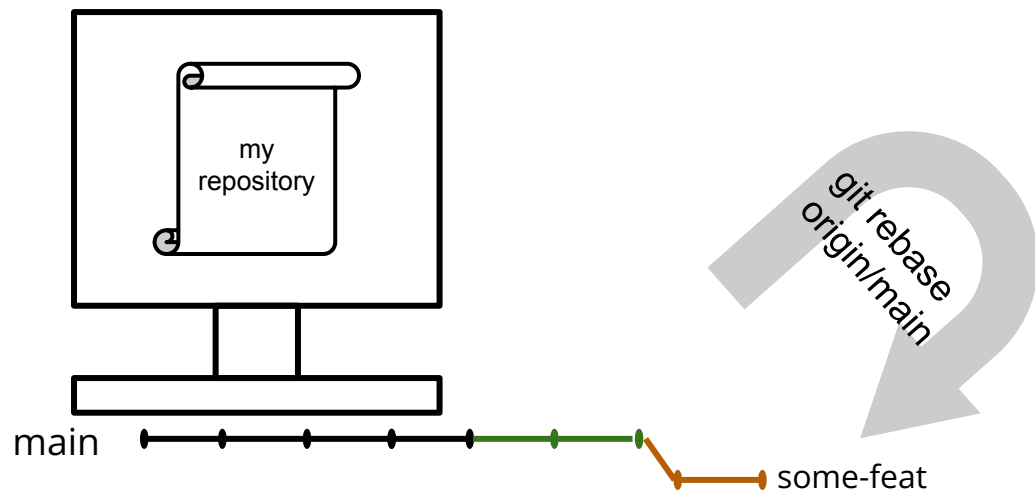
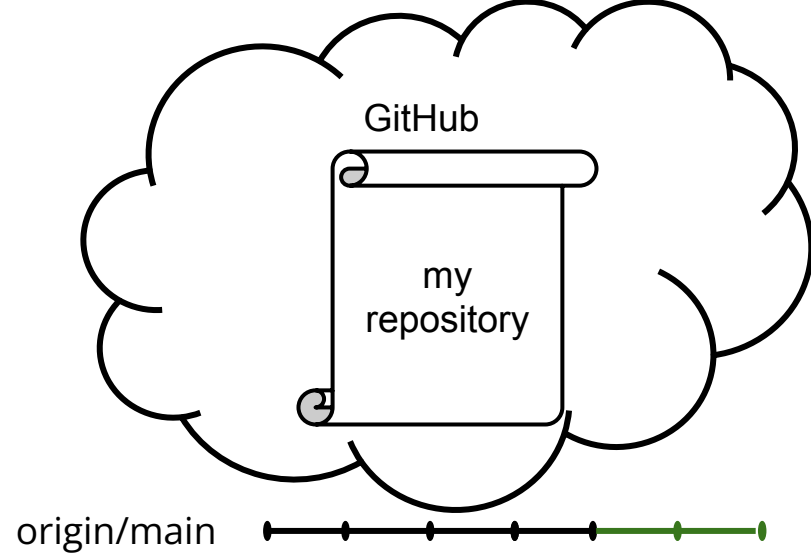
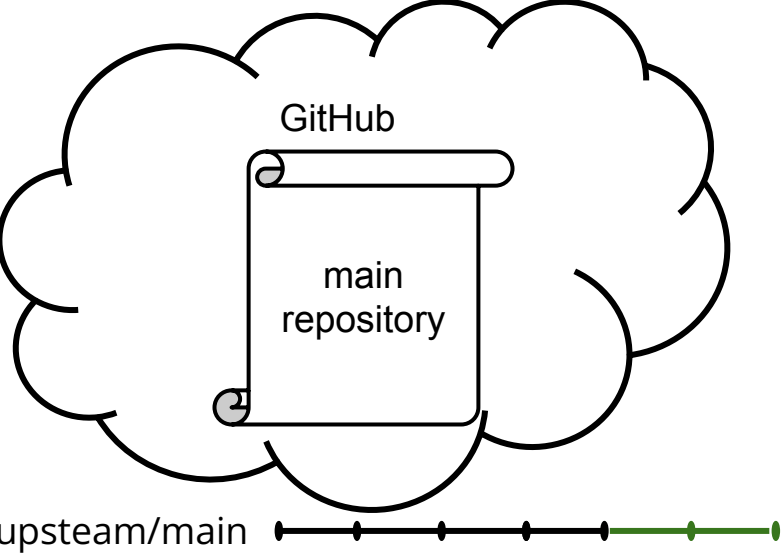






Go back to the some-feat branch

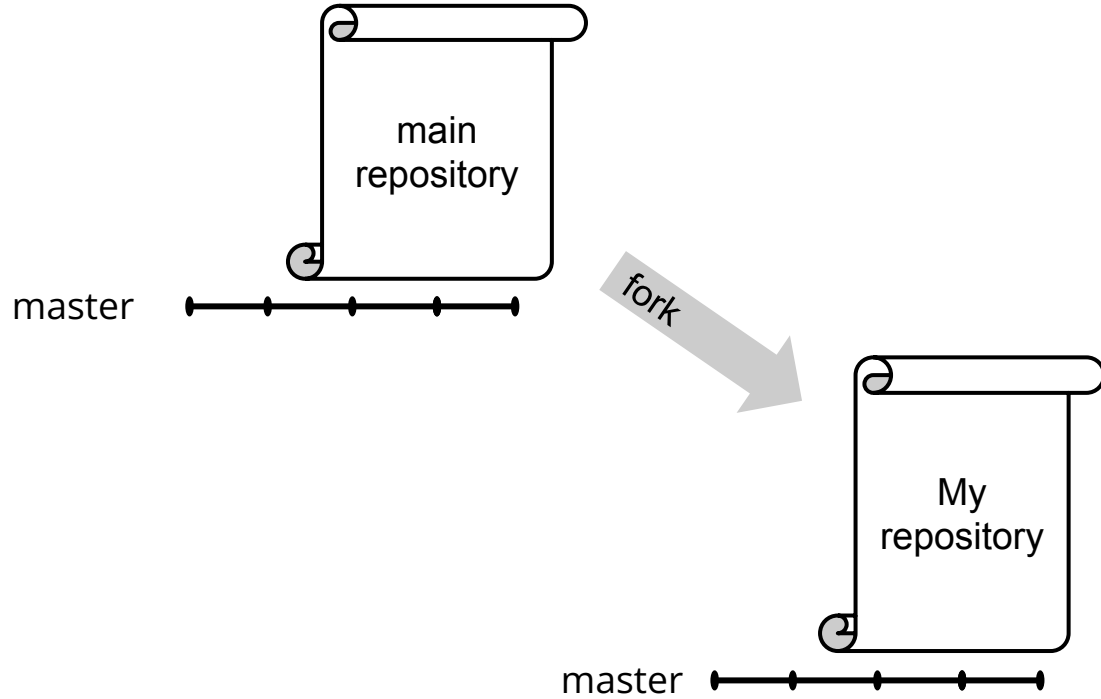


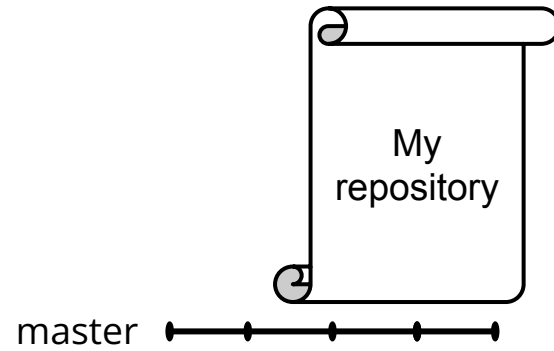
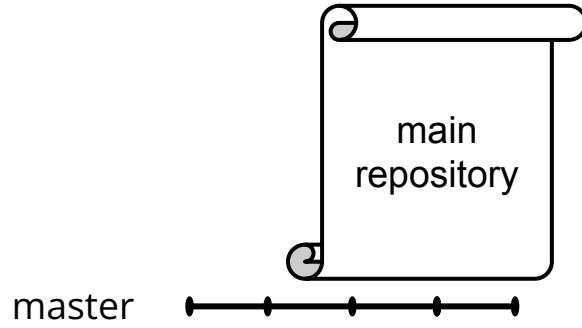


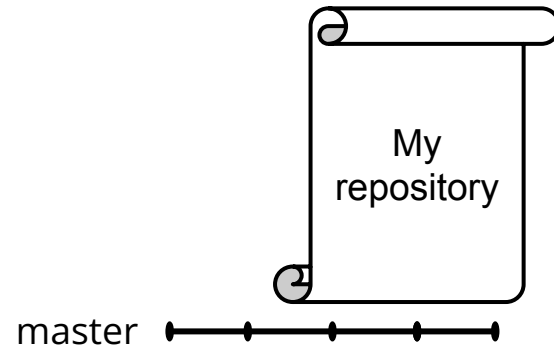
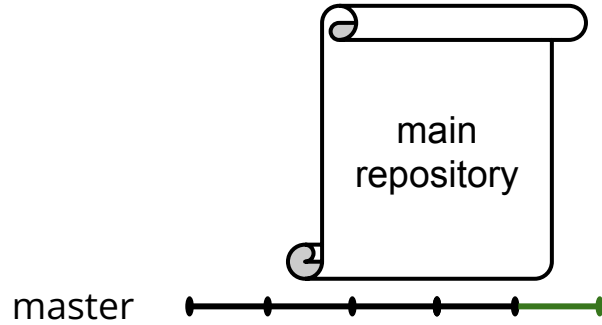
**EXTRA**

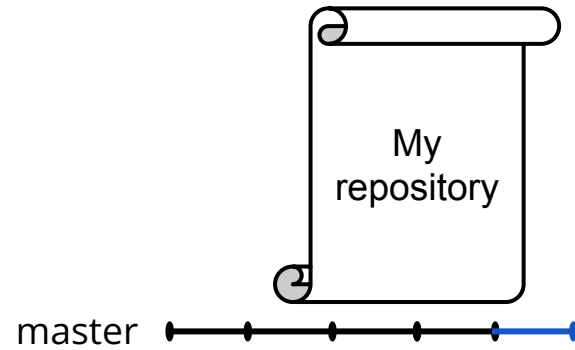
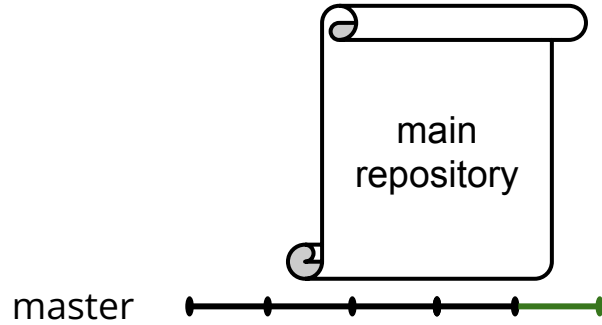
# EXTRA

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

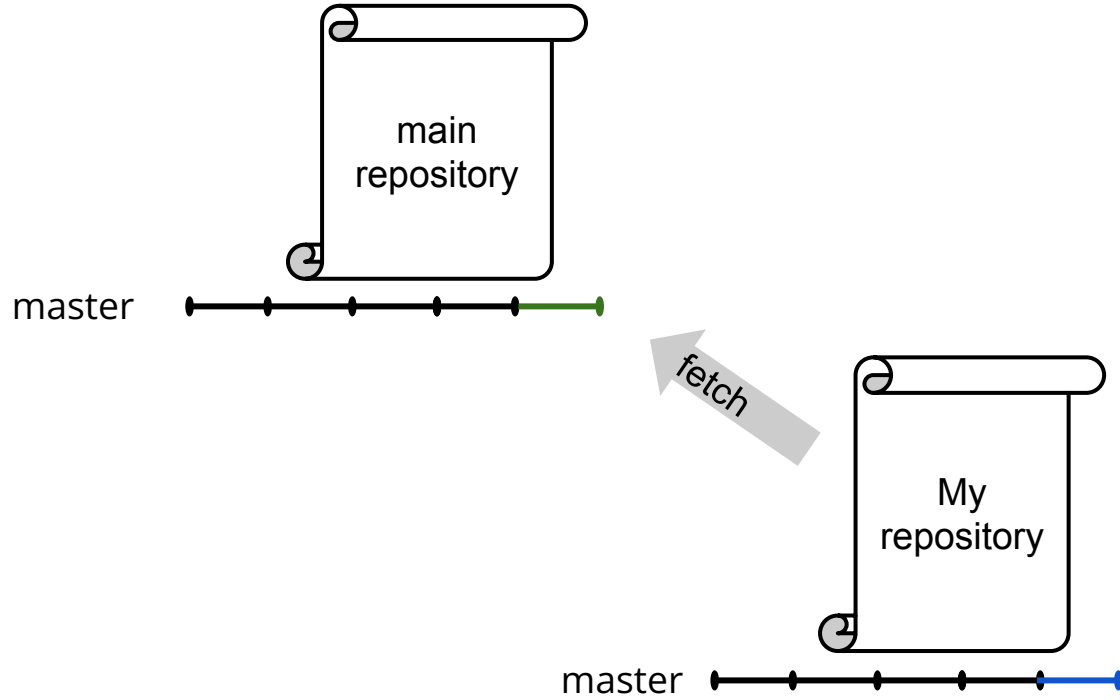


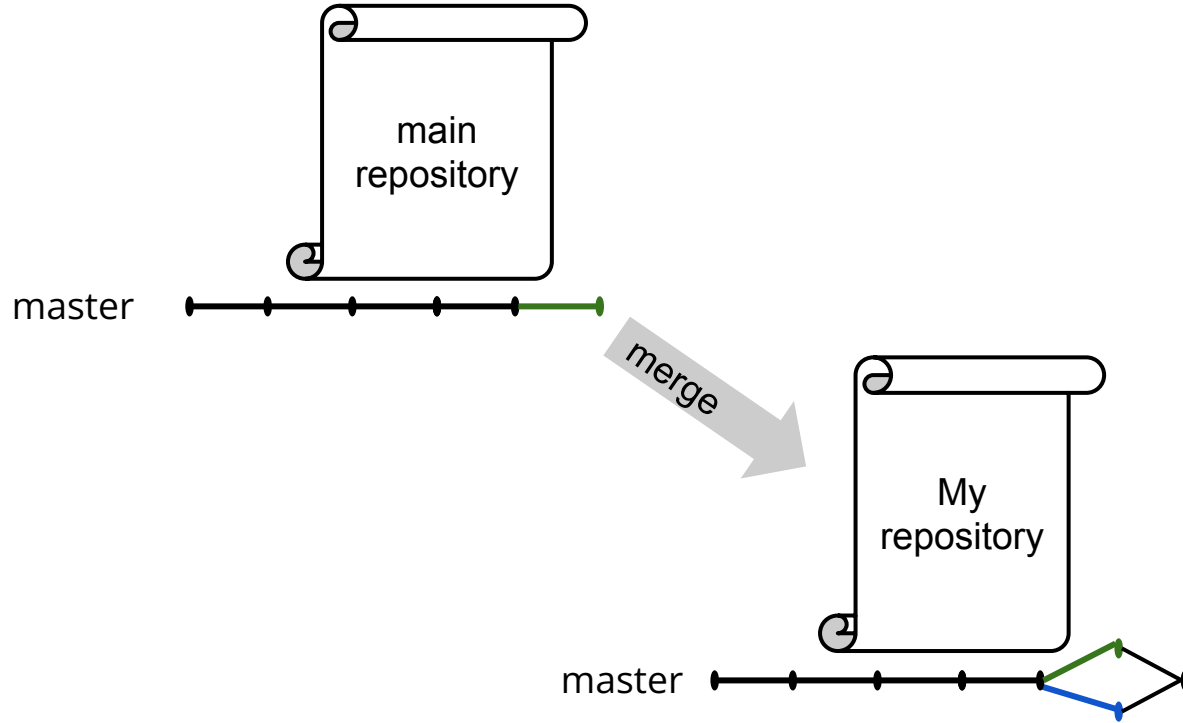


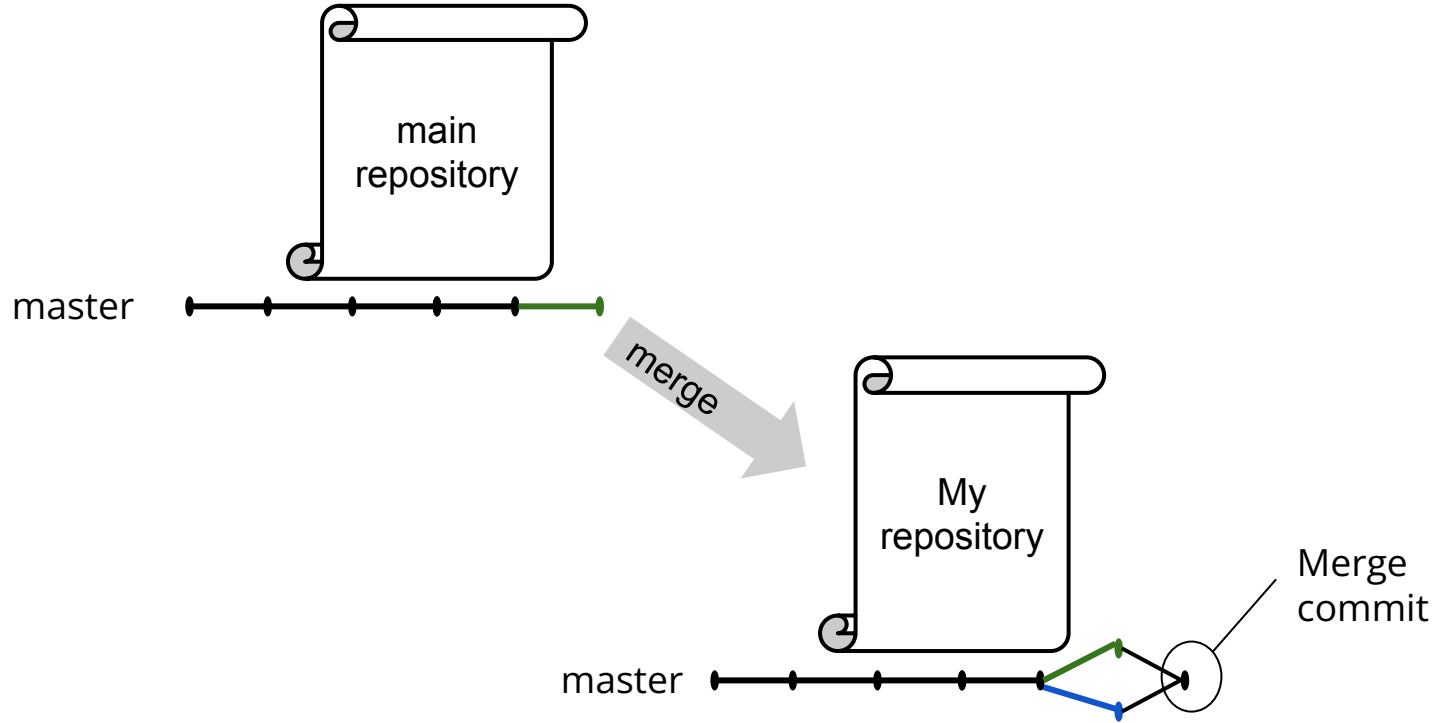






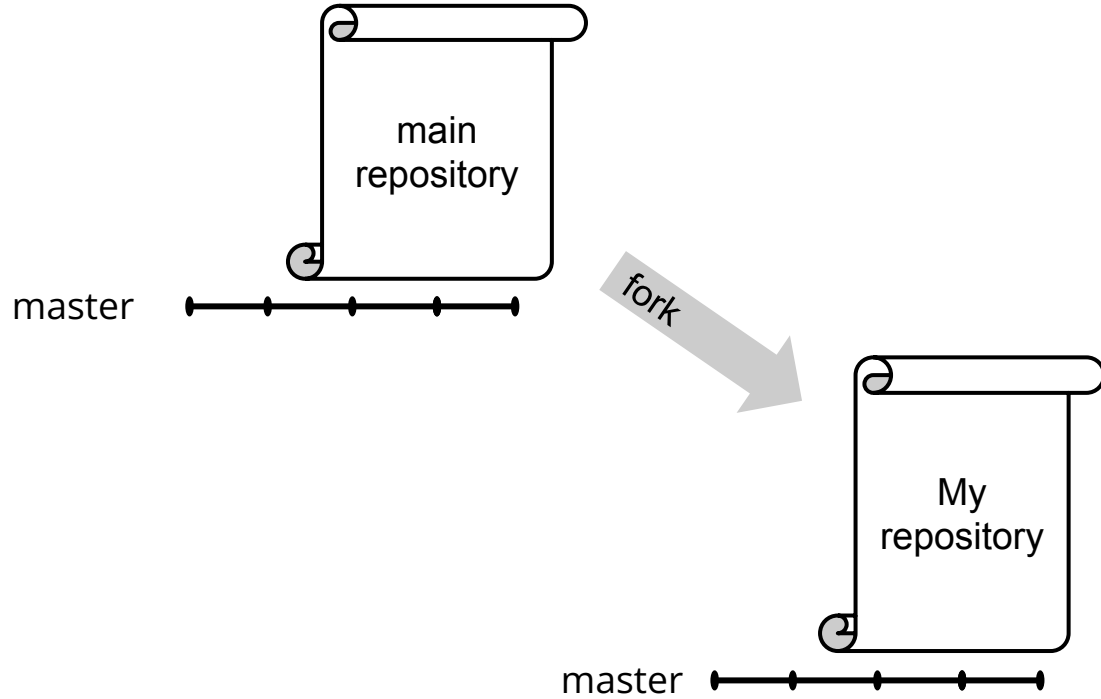


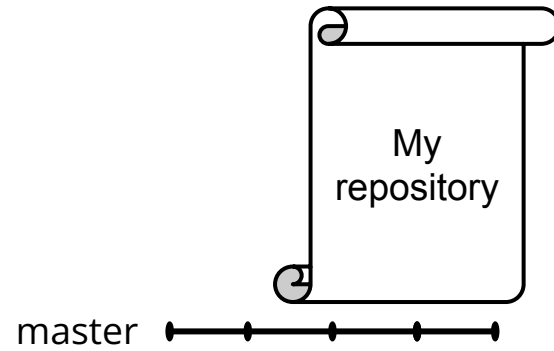
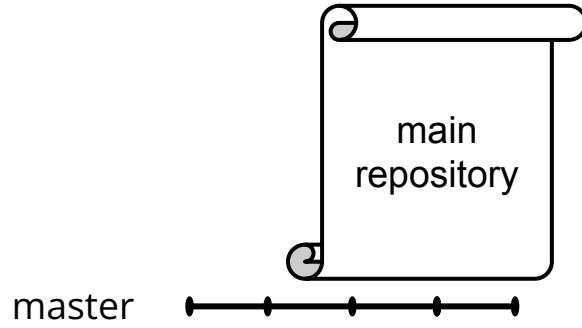


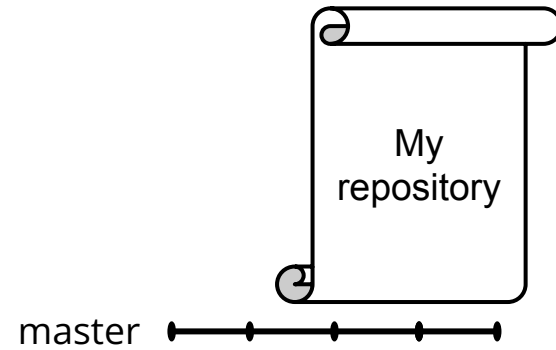
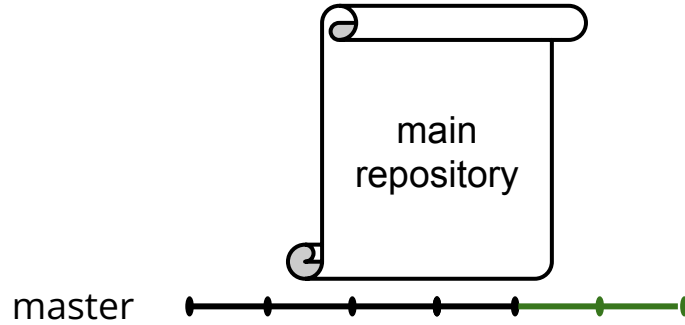


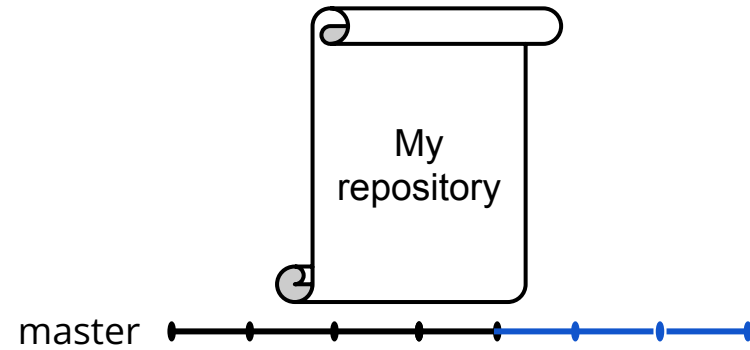
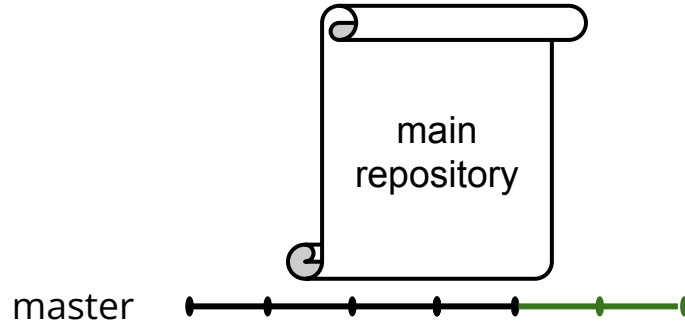
# EXTRA

- 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

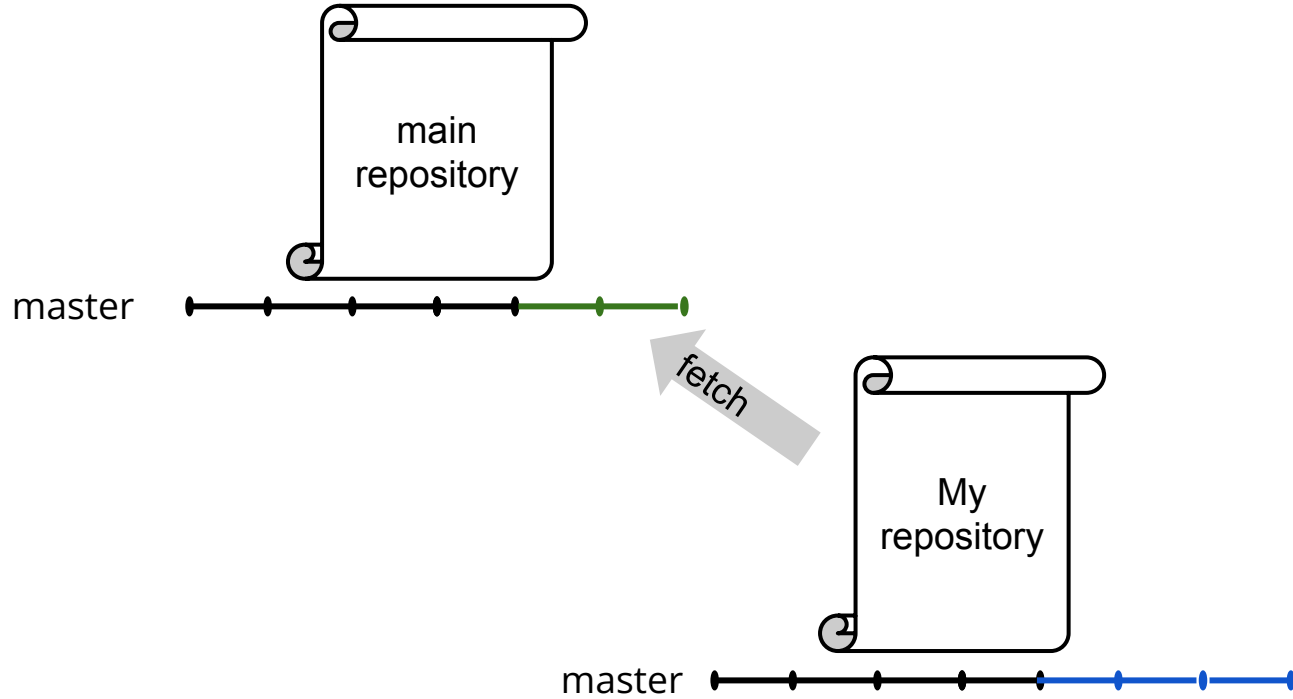


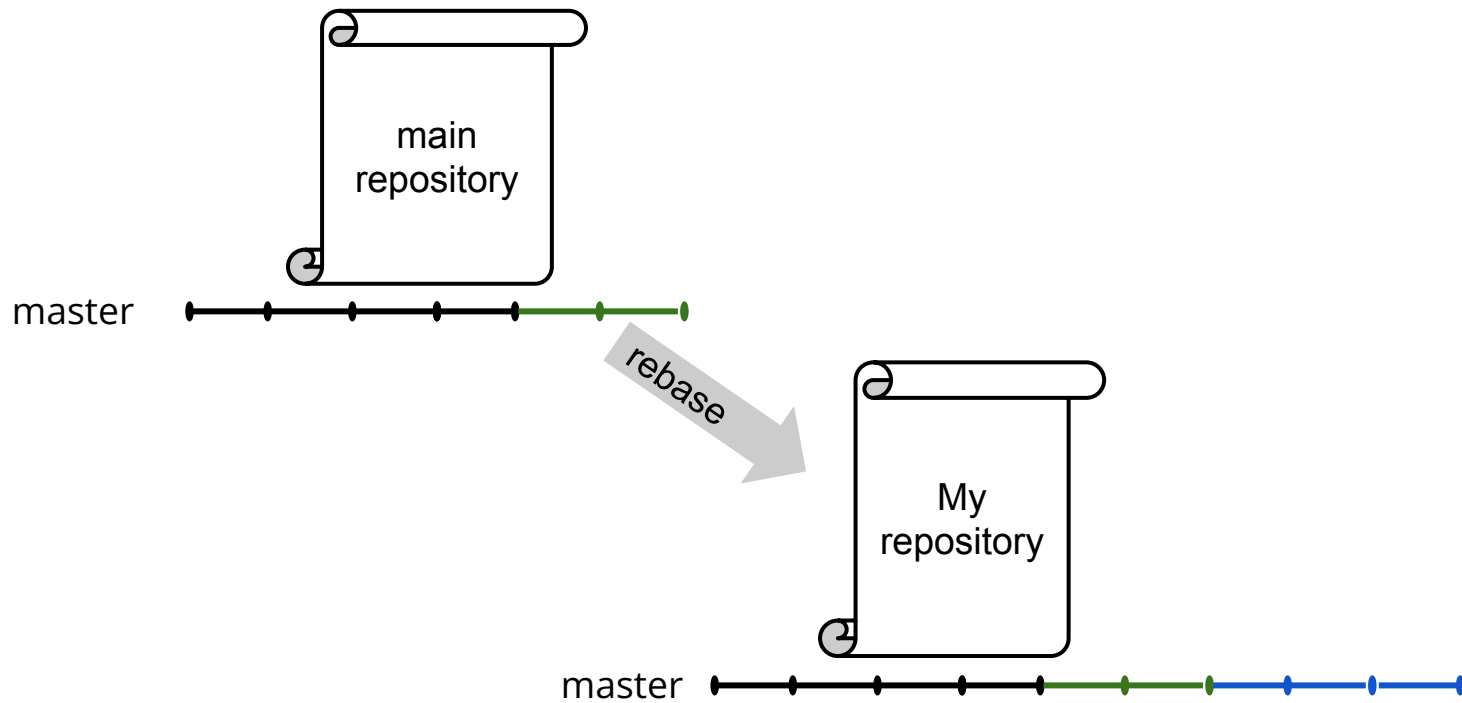






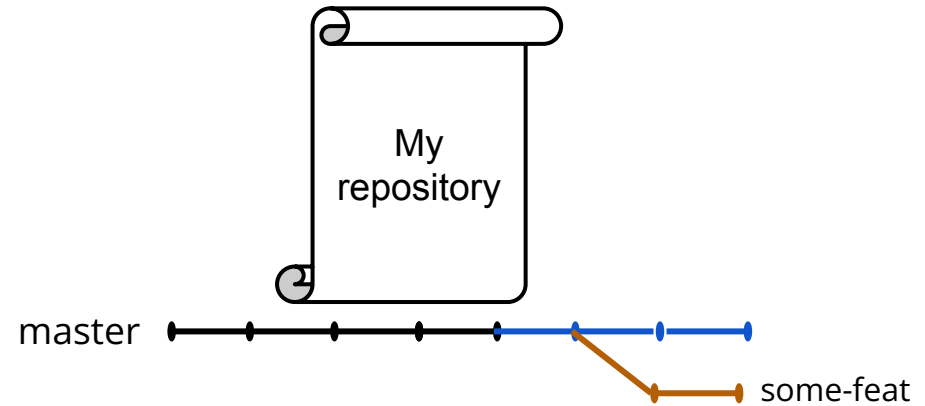
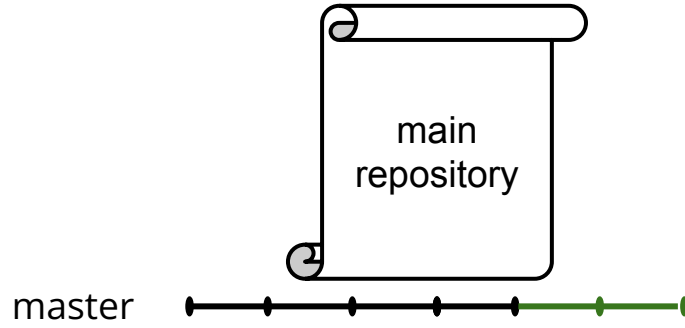


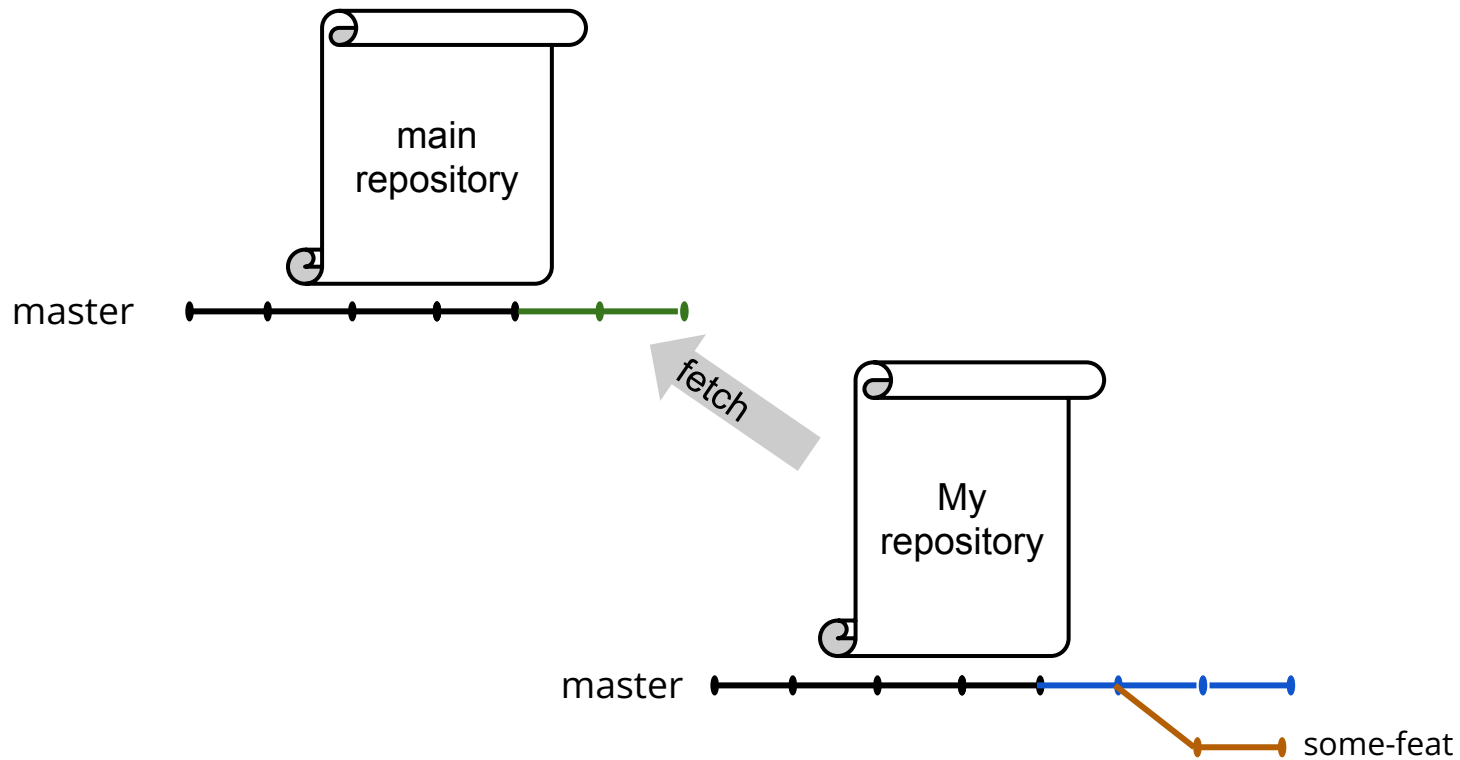


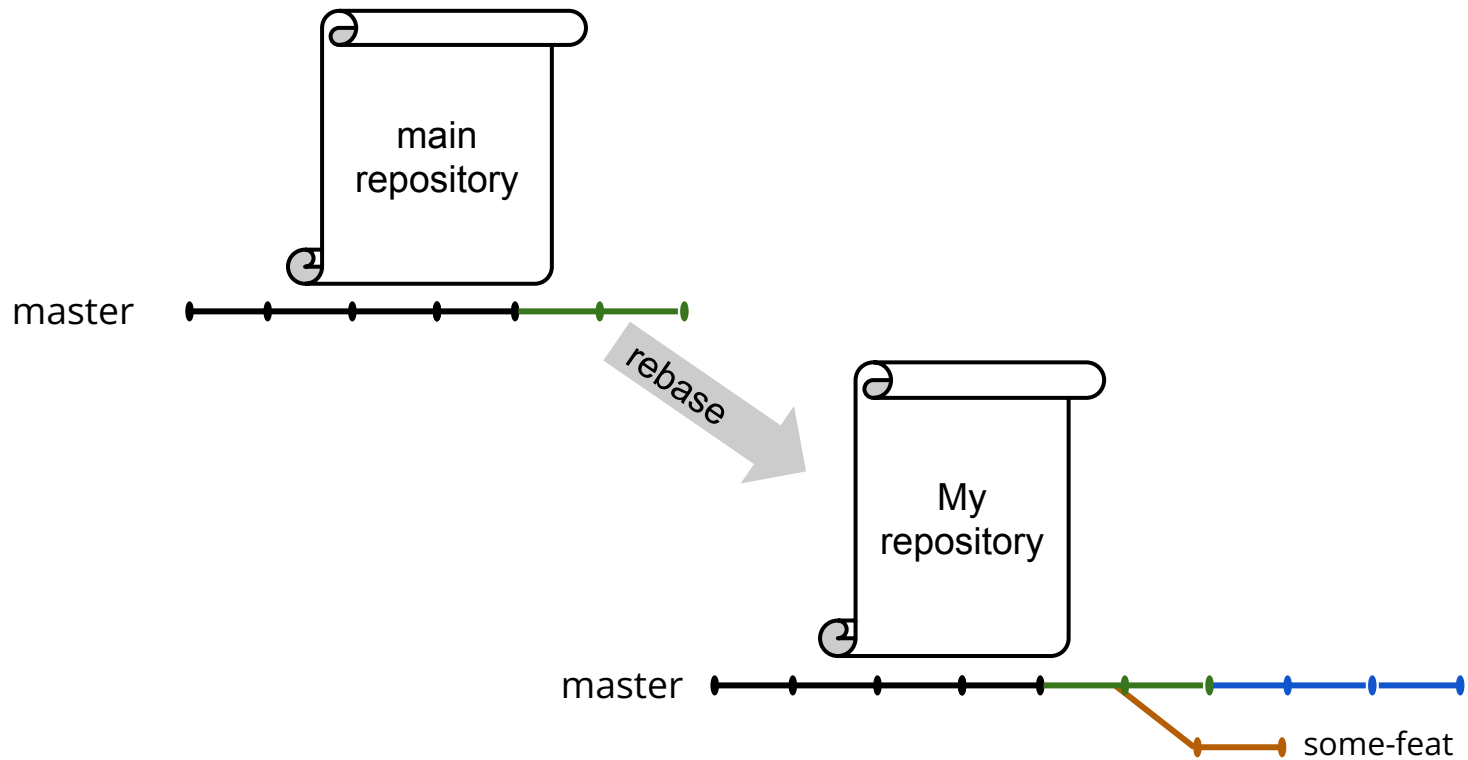


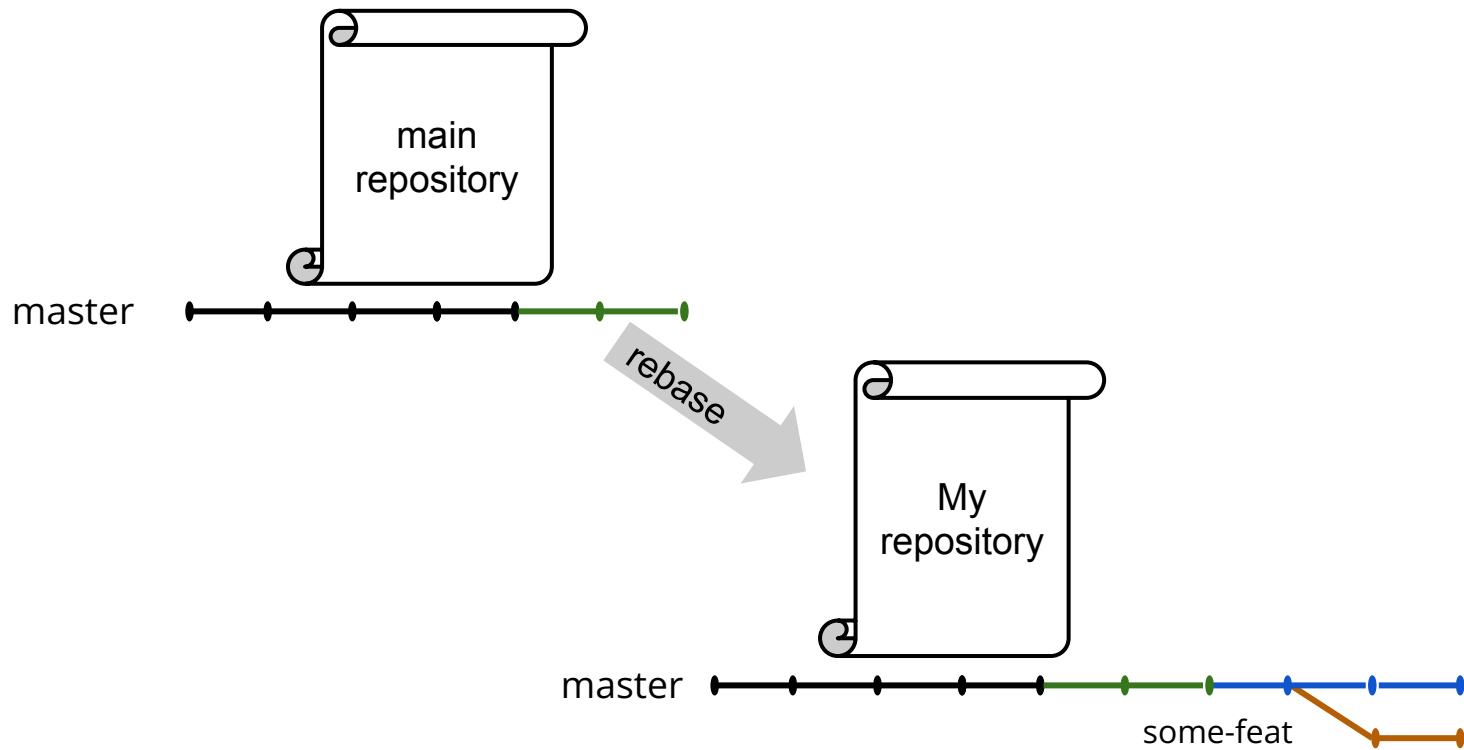
# EXTRA

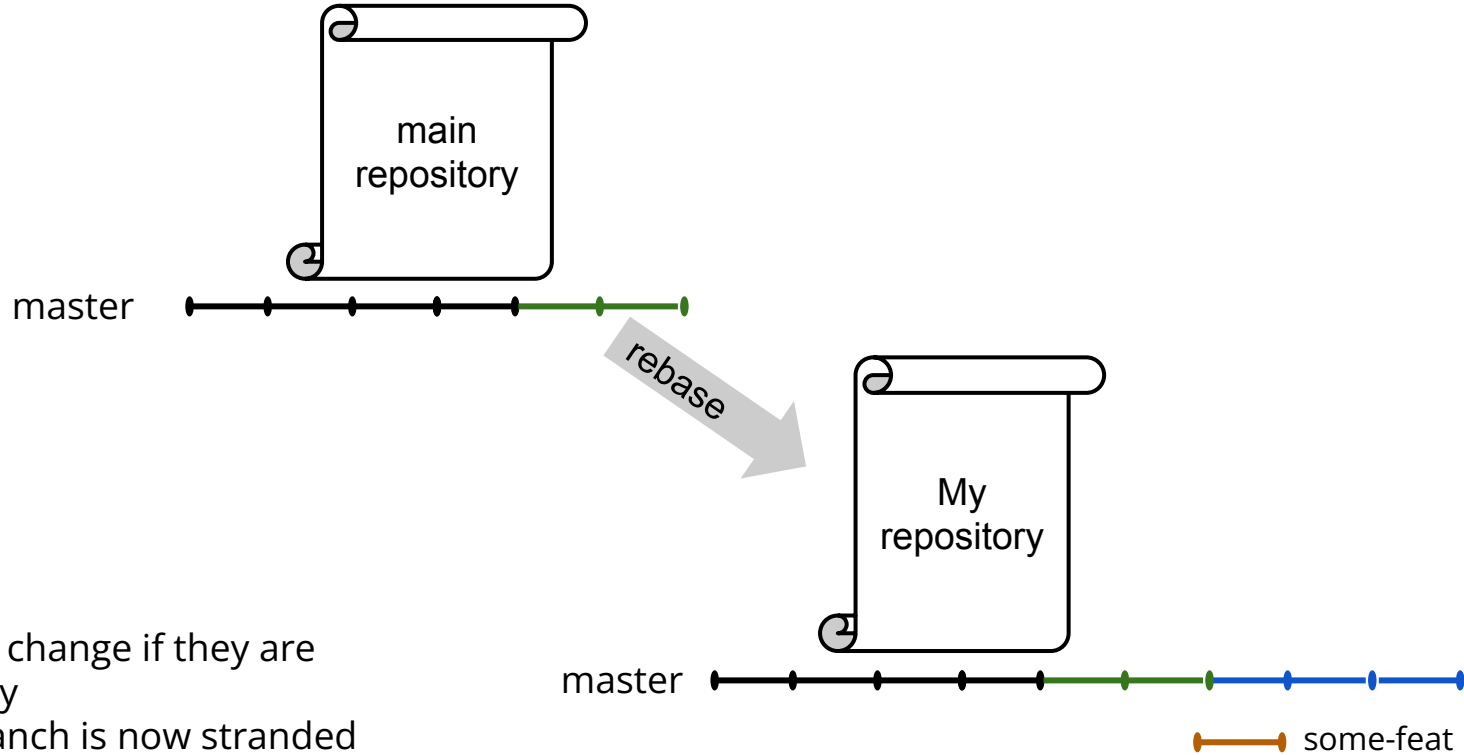
What happens to other branches?











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