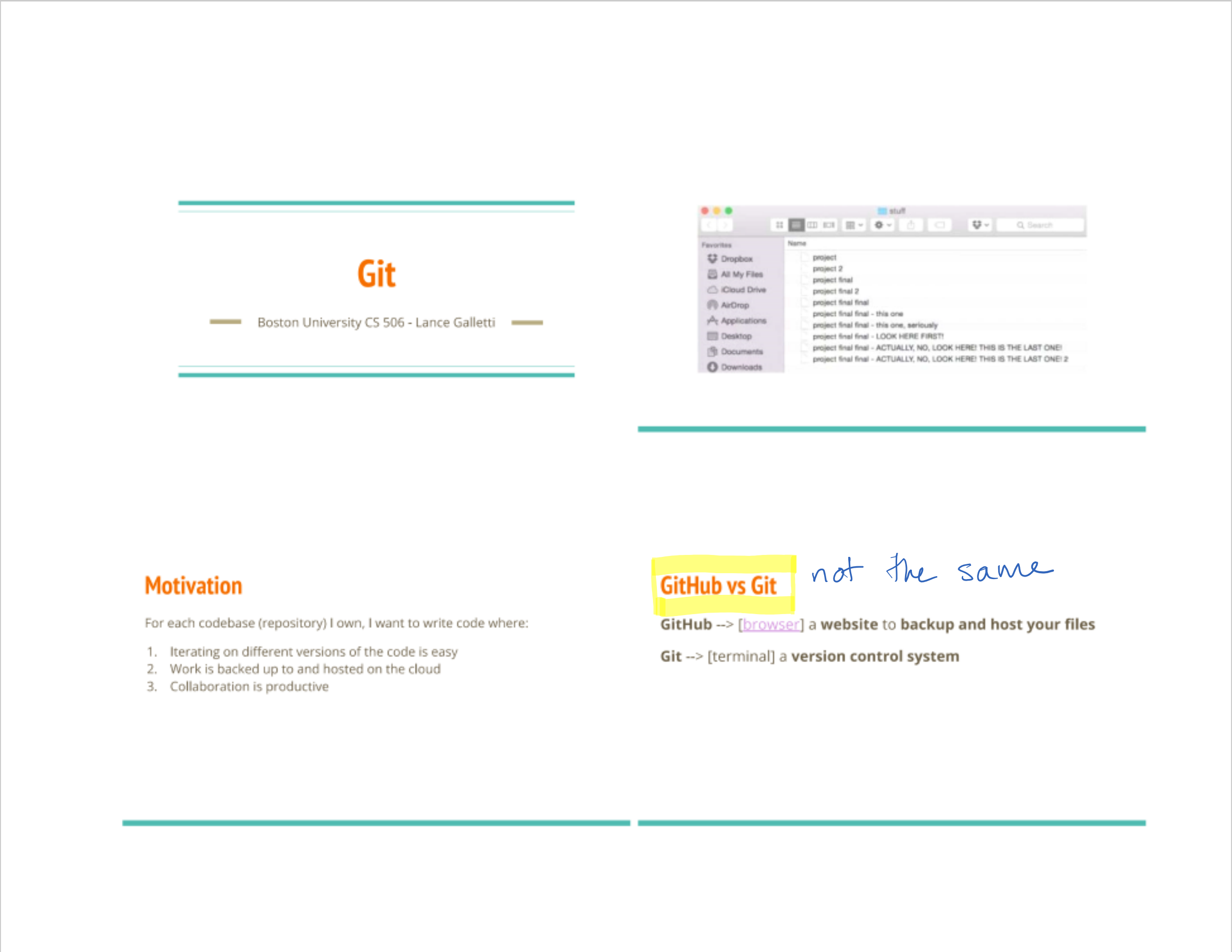
Printout

Wednesday, January 25, 2023 2:29 PM



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:

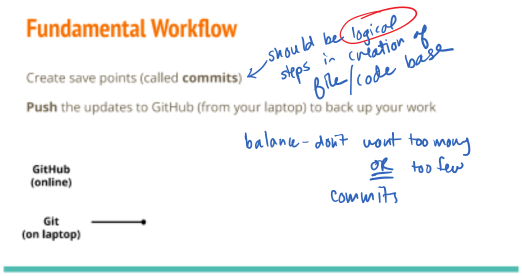
1. Iterating on different versions of the code is easy
2. Work is backed up to and hosted on the cloud
3. Collaboration is productive

GitHub vs Git

not the same

GitHub --> [browser] a **website** to **backup and host your files**

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

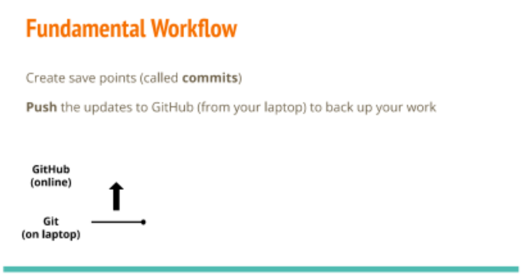


Fundamental Workflow

Create save points (called **commits**) ← should be logical steps in creation of file/code base

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

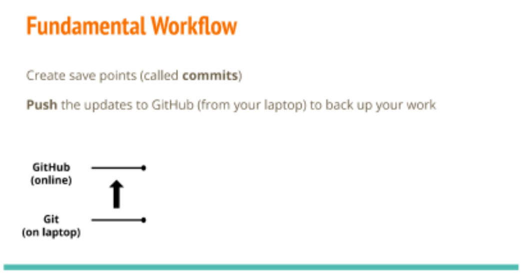
balance - don't want too many <u>OR</u> too few commits



Fundamental Workflow

Create save points (called **commits**)

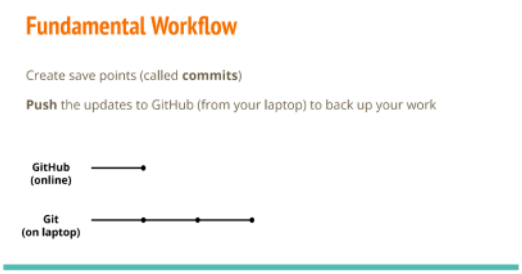
Push the updates to GitHub (from your laptop) to back up your work



Fundamental Workflow

Create save points (called **commits**)

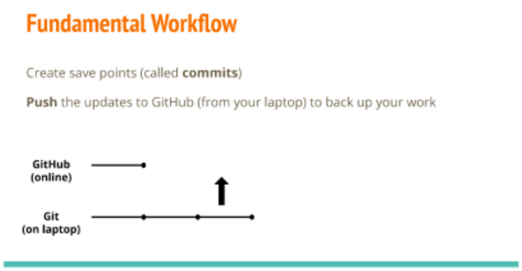
Push the updates to GitHub (from your laptop) to back up your work



Fundamental Workflow

Create save points (called **commits**)

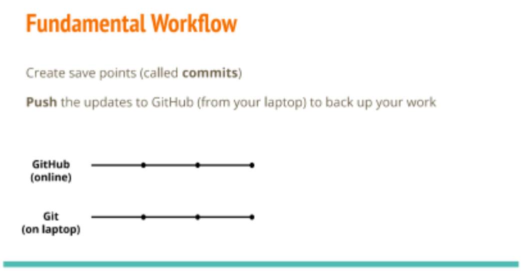
Push the updates to GitHub (from your laptop) to back up your work



Fundamental Workflow

Create save points (called **commits**)

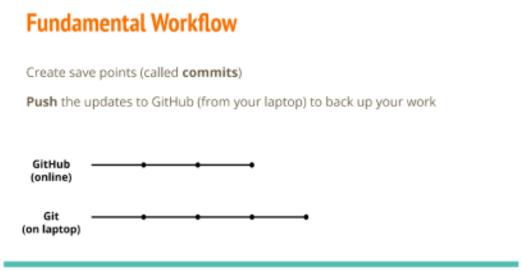
Push the updates to GitHub (from your laptop) to back up your work



Fundamental Workflow

Create save points (called **commits**)

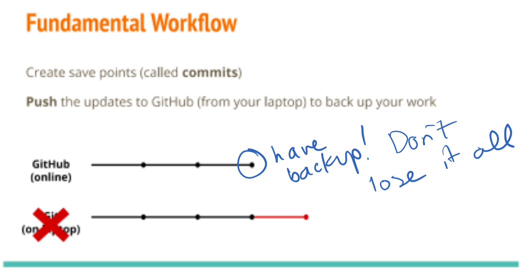
Push the updates to GitHub (from your laptop) to back up your work



Fundamental Workflow

Create save points (called **commits**)

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



Fundamental Workflow

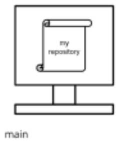
Create save points (called **commits**)

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

have backup! Don't lose it all

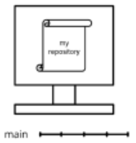
Initialize a repository

```
git init
```



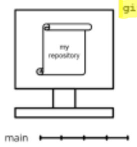
Add and Commit changes

```
git add <files>
git commit -m "some message"
```



Add a remote that points to GitHub

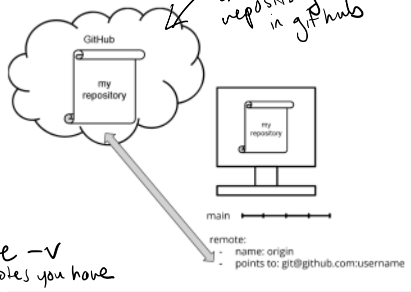
```
git remote add origin <link>
```



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

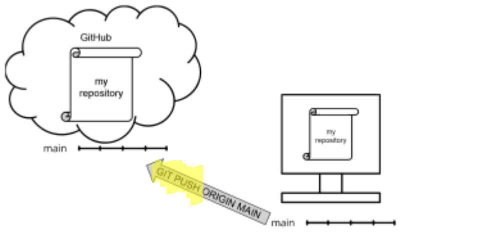
↑ ref remote by name vs. using link each time

git remote -v
see remotes you have



create repository in github

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



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

git log ← see your commits

Demo

Iterating on Different Versions

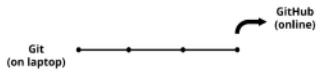
The ease or difficulty of adding a new feature to the code base may depend on the state / version of the codebase.

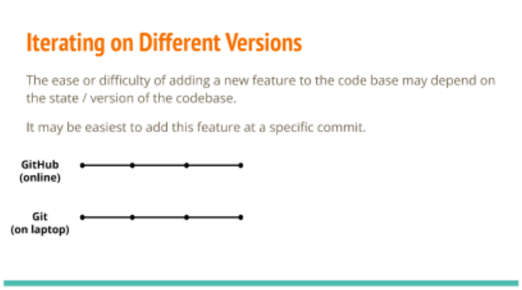
It may be easiest to add this feature at a specific commit.

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.

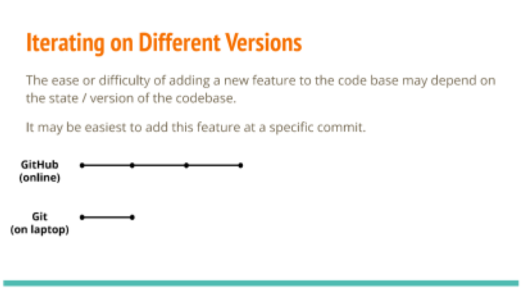




Iterating on Different Versions

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

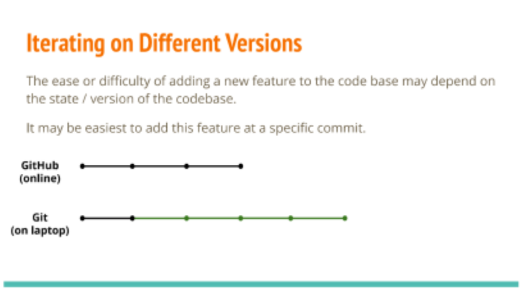
It may be easiest to add this feature at a specific commit.



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.

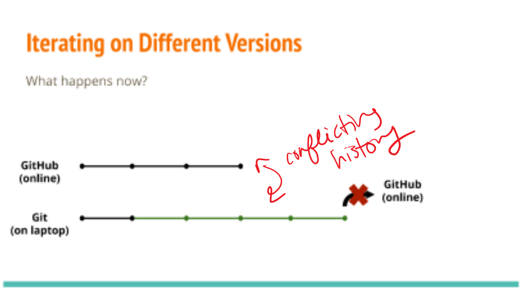
It may be easiest to add this feature at a specific commit.



Iterating on Different Versions

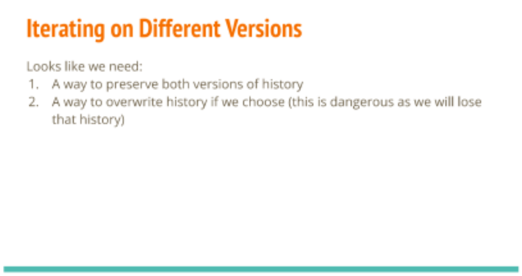
The ease or difficulty of adding a new feature to the code base may depend on the state / version of the codebase.

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



Iterating on Different Versions

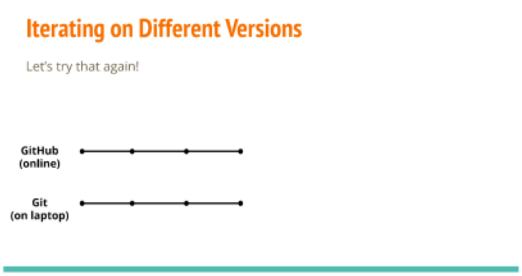
What happens now?



Iterating on Different Versions

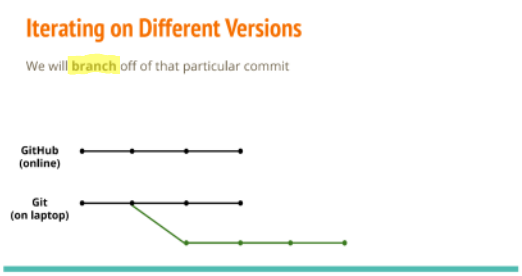
Looks like we need:

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)



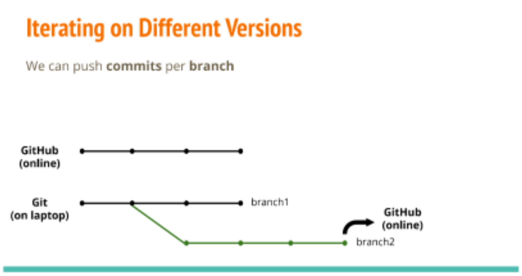
Iterating on Different Versions

Let's try that again!



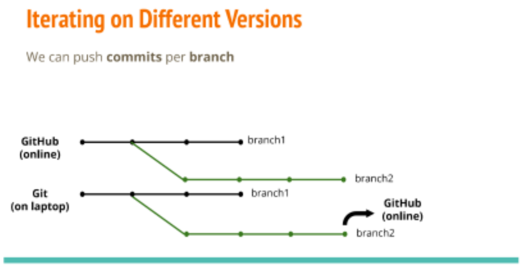
Iterating on Different Versions

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



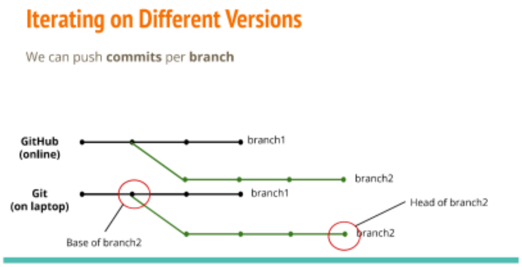
Iterating on Different Versions

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



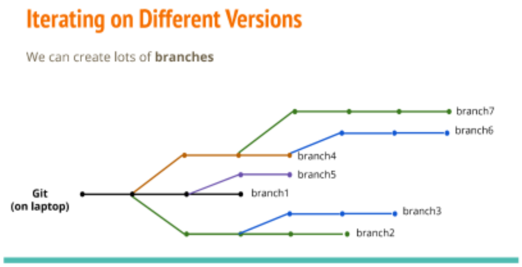
Iterating on Different Versions

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



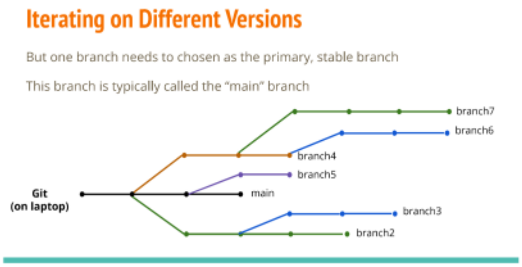
Iterating on Different Versions

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



Iterating on Different Versions

We can create lots of **branches**



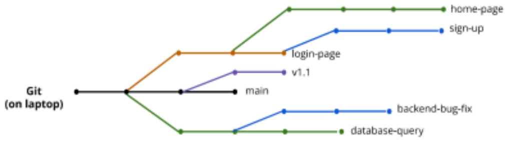
Iterating on Different Versions

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

This branch is typically called the "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

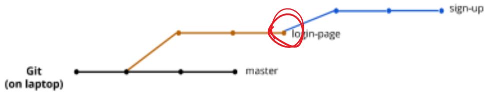


Iterating on Different Versions

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

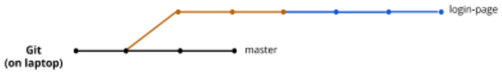
Iterating on Different Versions

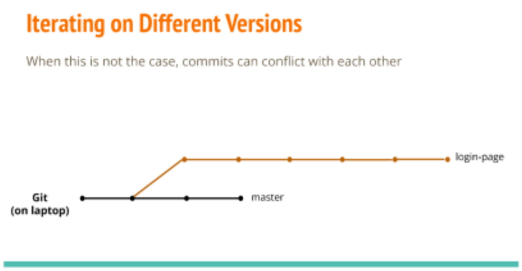
At some point we will want to clean up certain branches by **merging** them with the master / main branch or with each other.



Iterating on Different Versions

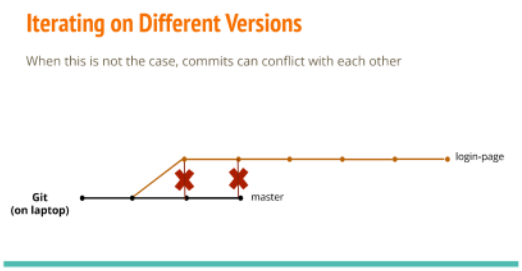
Merging is trivial if the **base** of one branch is the **head** of the other - the changes are "simply" appended.





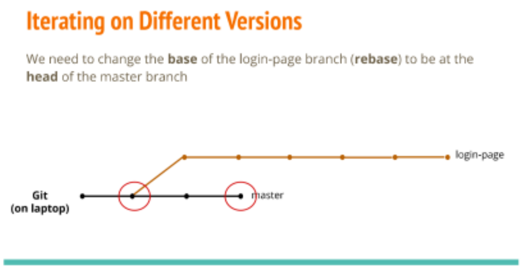
Iterating on Different Versions

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



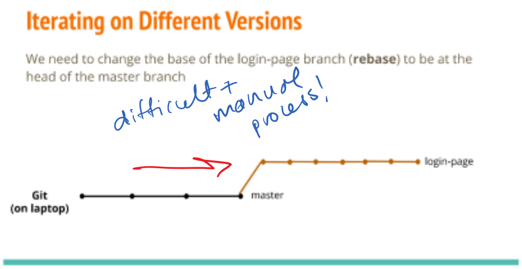
Iterating on Different Versions

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



Iterating on Different Versions

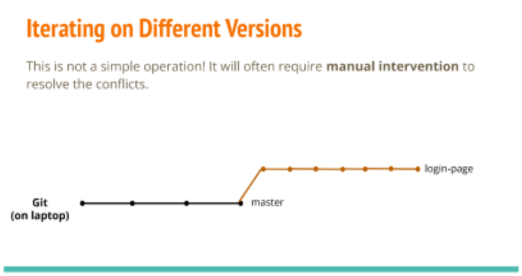
We need to change the **base** of the login-page branch (**rebase**) to be at the **head** of the master branch



Iterating on Different Versions

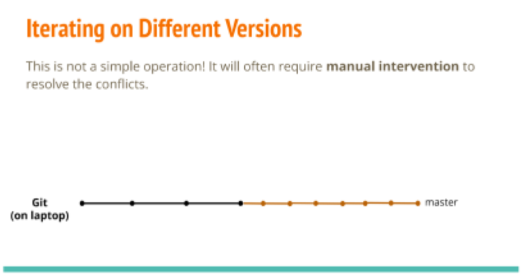
We need to change the base of the login-page branch (**rebase**) to be at the head of the master branch

difficult + manual process!



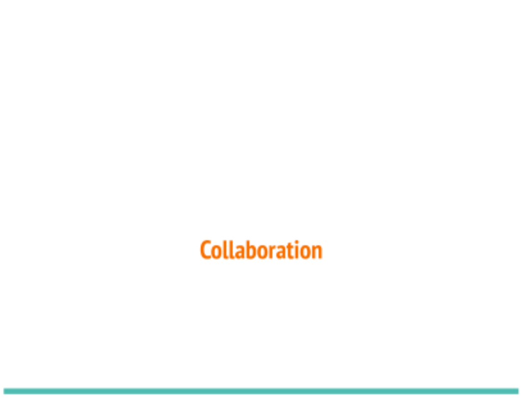
Iterating on Different Versions

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

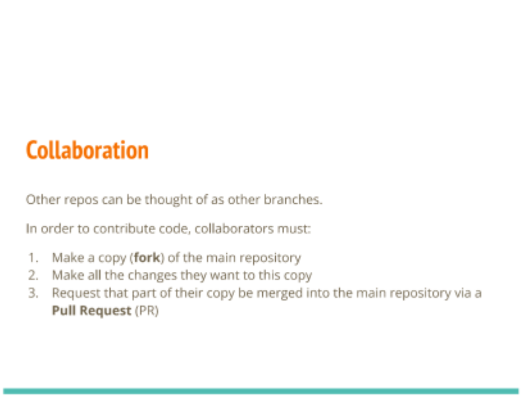


Iterating on Different Versions

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



Collaboration

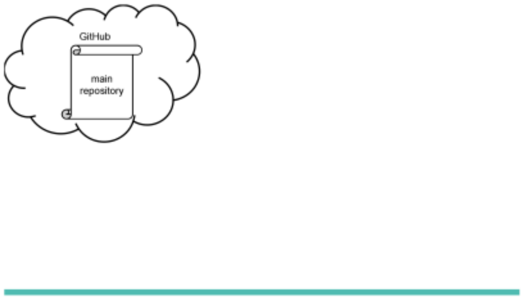


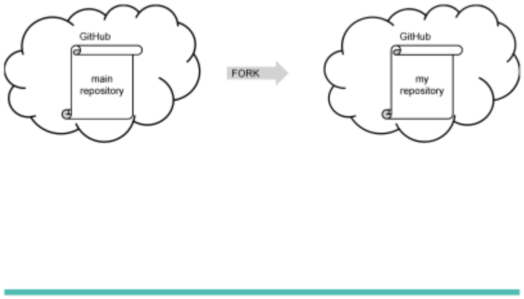
Collaboration

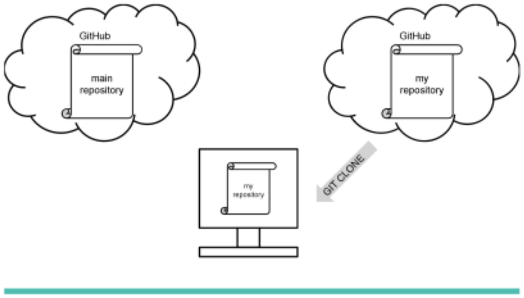
Other repos can be thought of as other branches.

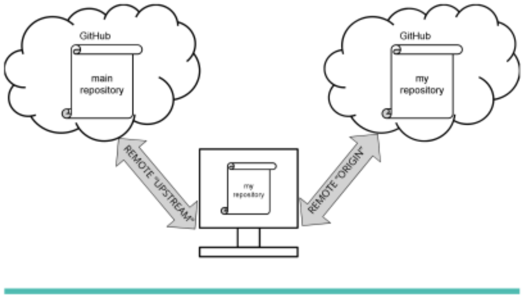
In order to contribute code, collaborators must:

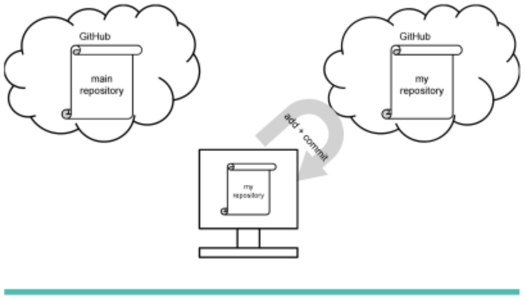
1. Make a copy (**fork**) of the main repository
2. Make all the changes they want to this copy
3. Request that part of their copy be merged into the main repository via a **Pull Request** (PR)

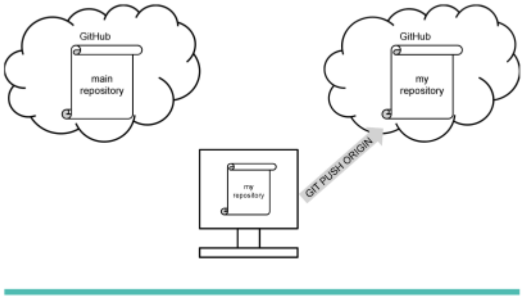


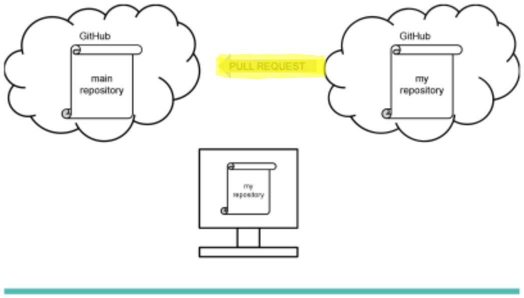












git fetch upstream
↳ see what is happening in upstream repo

git rebase upstream main
↳ merges branches - move commits onto new base commit

DEMO

git push origin main -f → override to force update

git checkout -b → create new branch

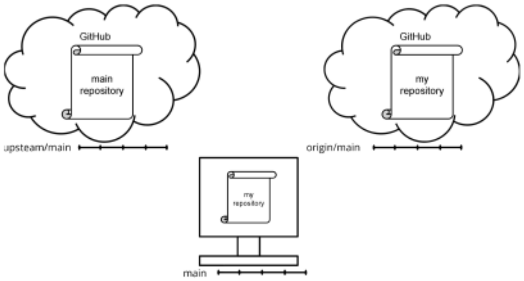
git checkout -b → create new branch

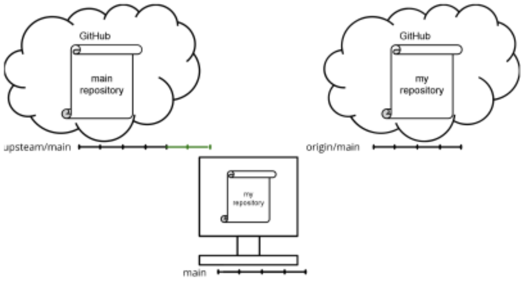
use to move btwn branches

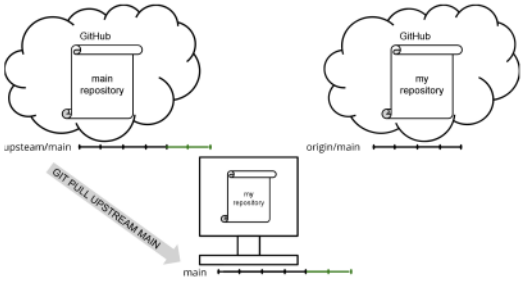
Job search tips

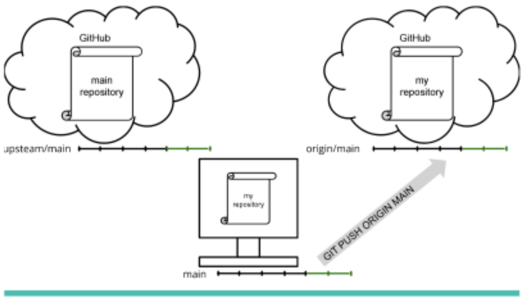
- what do you believe in? What will you be uncompromising about?
- be prepared for the basic interview q's
- don't change yourself for a company
- interviews are about fit more than technical competency
- when you are in an interview…you are also interviewing them!
- mentorship is important when you're starting out
- no one knows everything - be transparent and be willing to learn

Keeping your fork updated





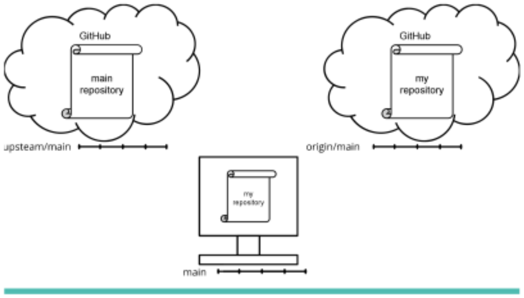


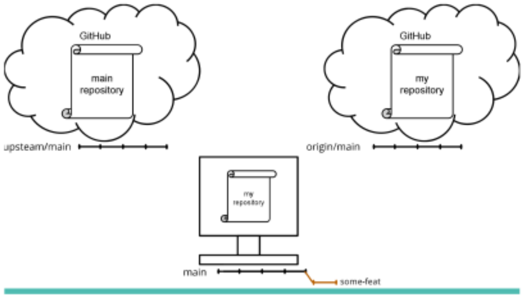


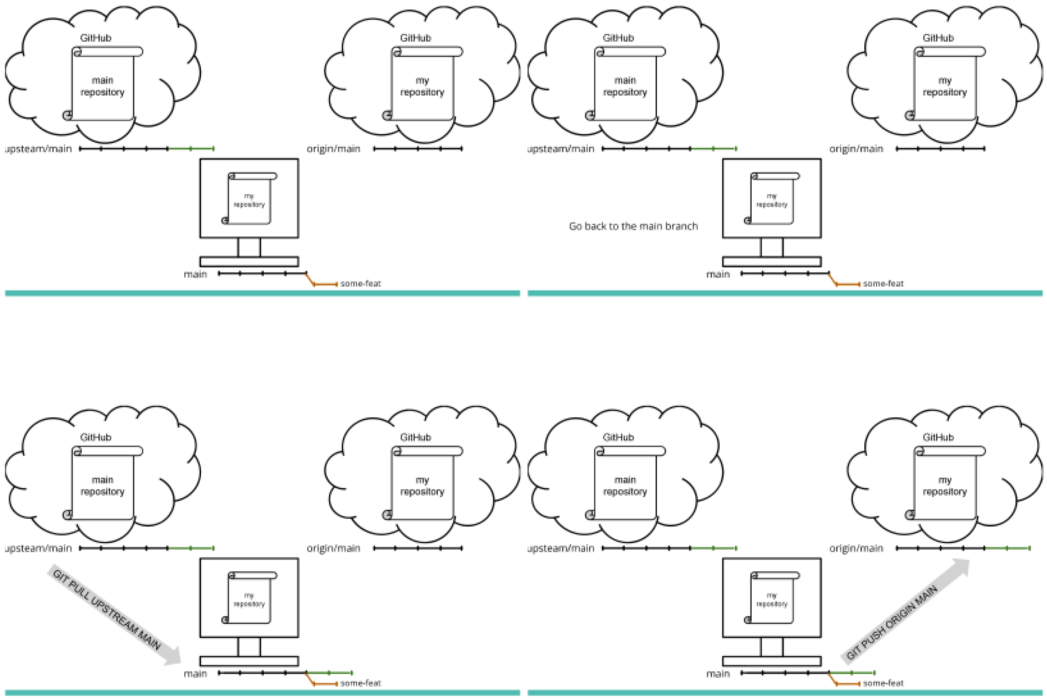
Best Practices

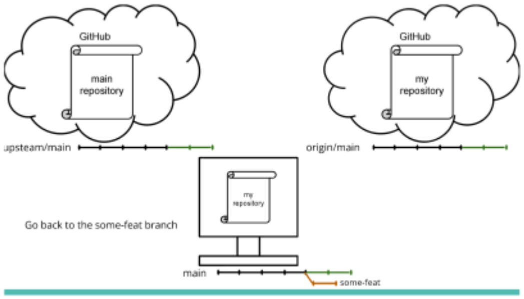
If you **never commit anything to your main branch**, keeping your main branch in sync with the main repository's is easy!

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

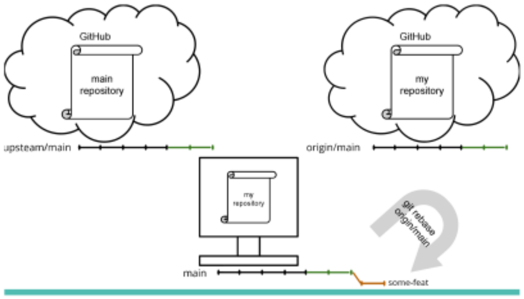





Go back to the main branch



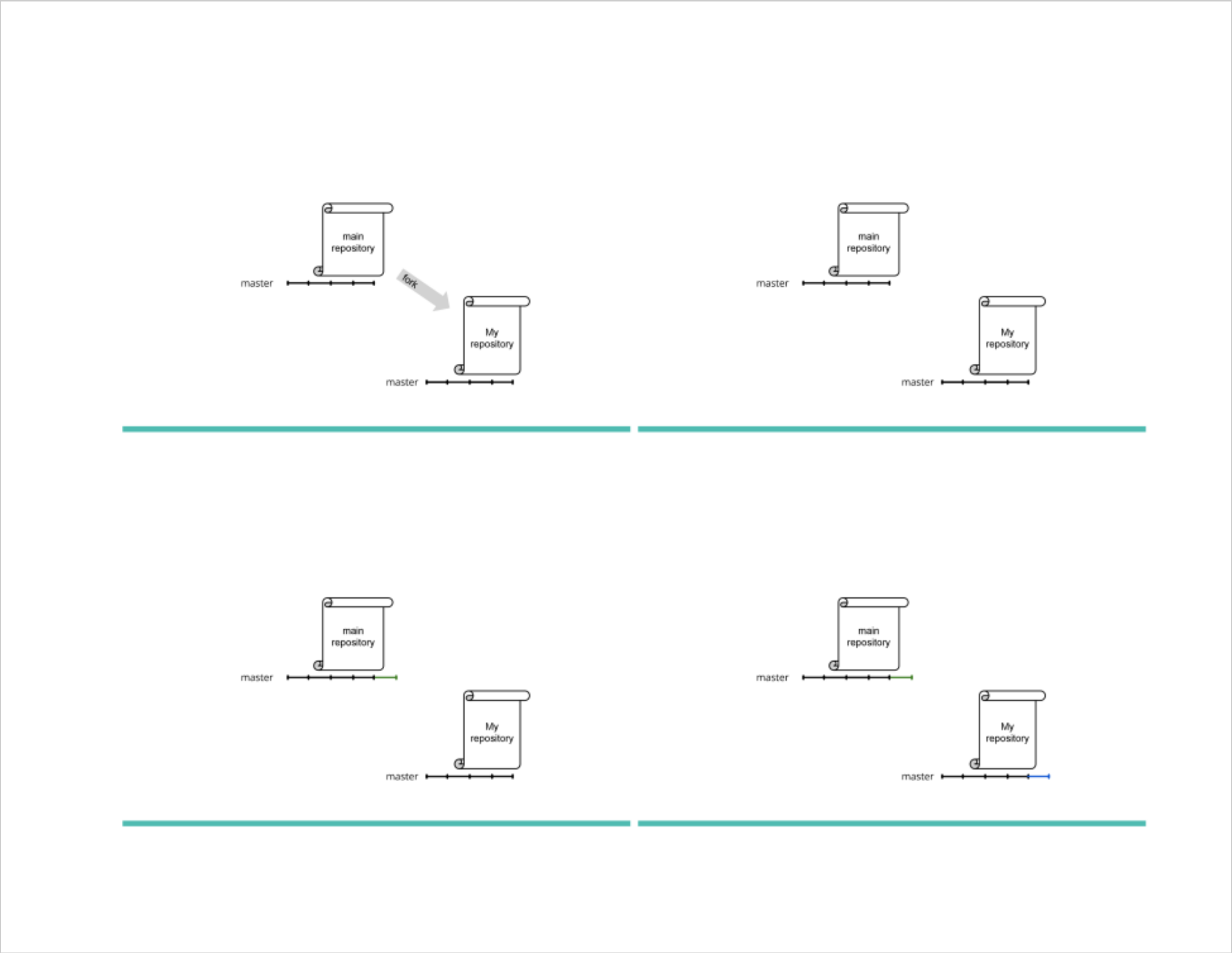
Go back to the some-feat branch

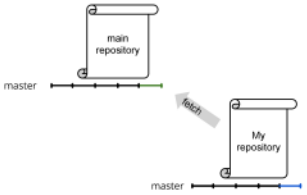


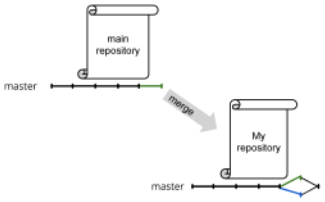
EXTRA

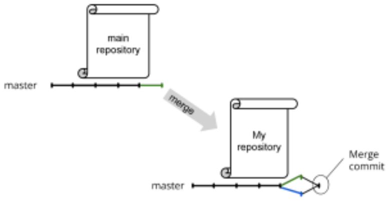
EXTRA

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


main repository
master
fork
My repository
master
main repository
master
My repository
master
main repository
master
My repository
master
main repository
master
My repository
master

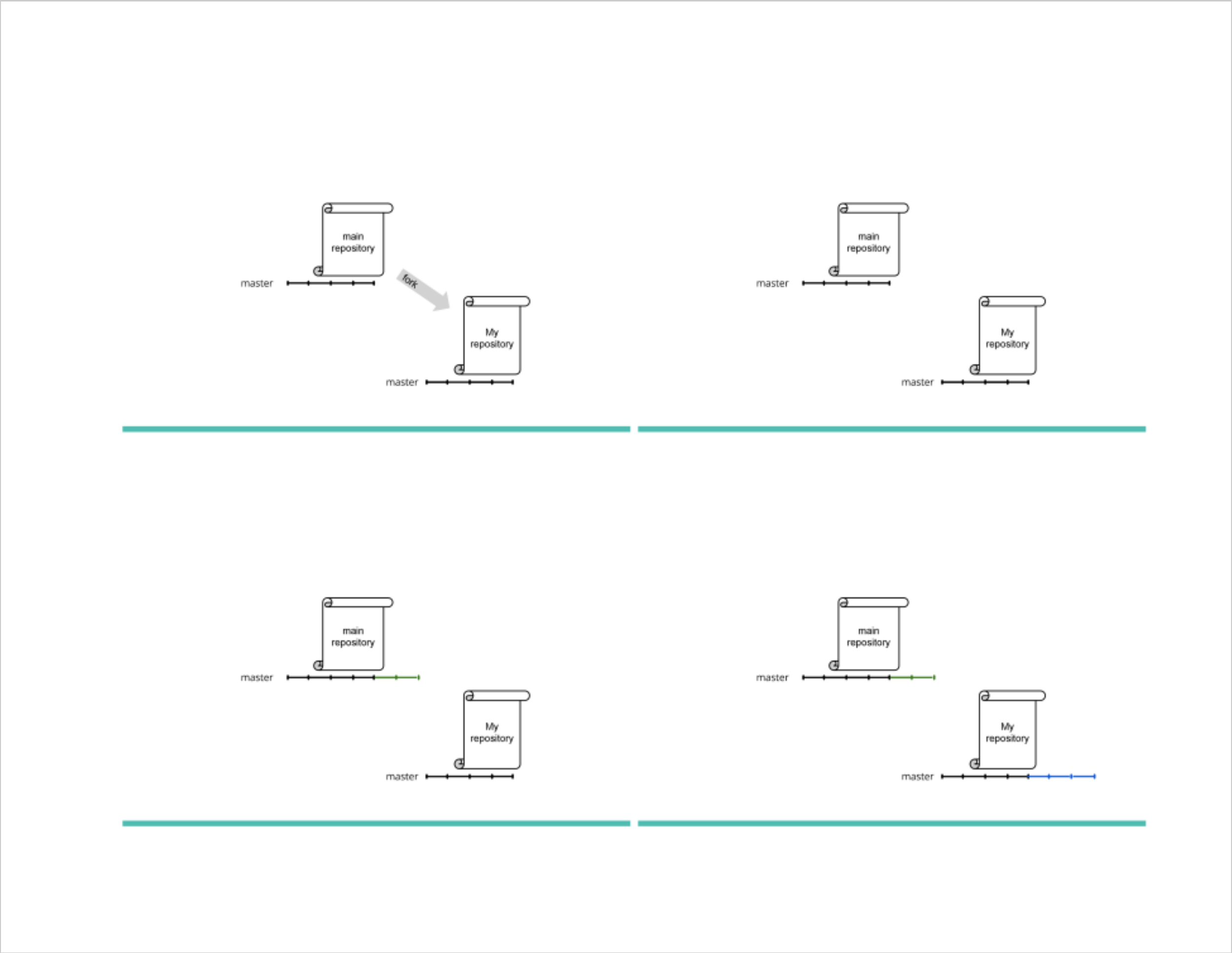


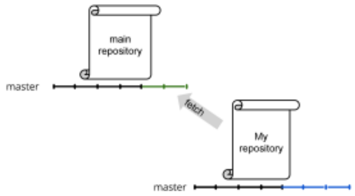


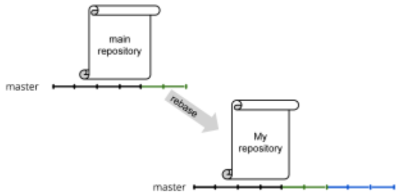


EXTRA

- 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

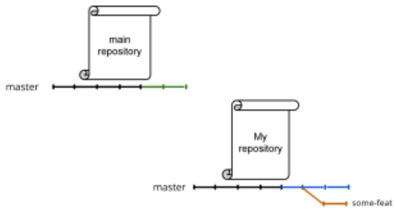

main repository
master
fork
My repository
master
main repository
master
My repository
master
main repository
master
My repository
master
main repository
master
My repository
master

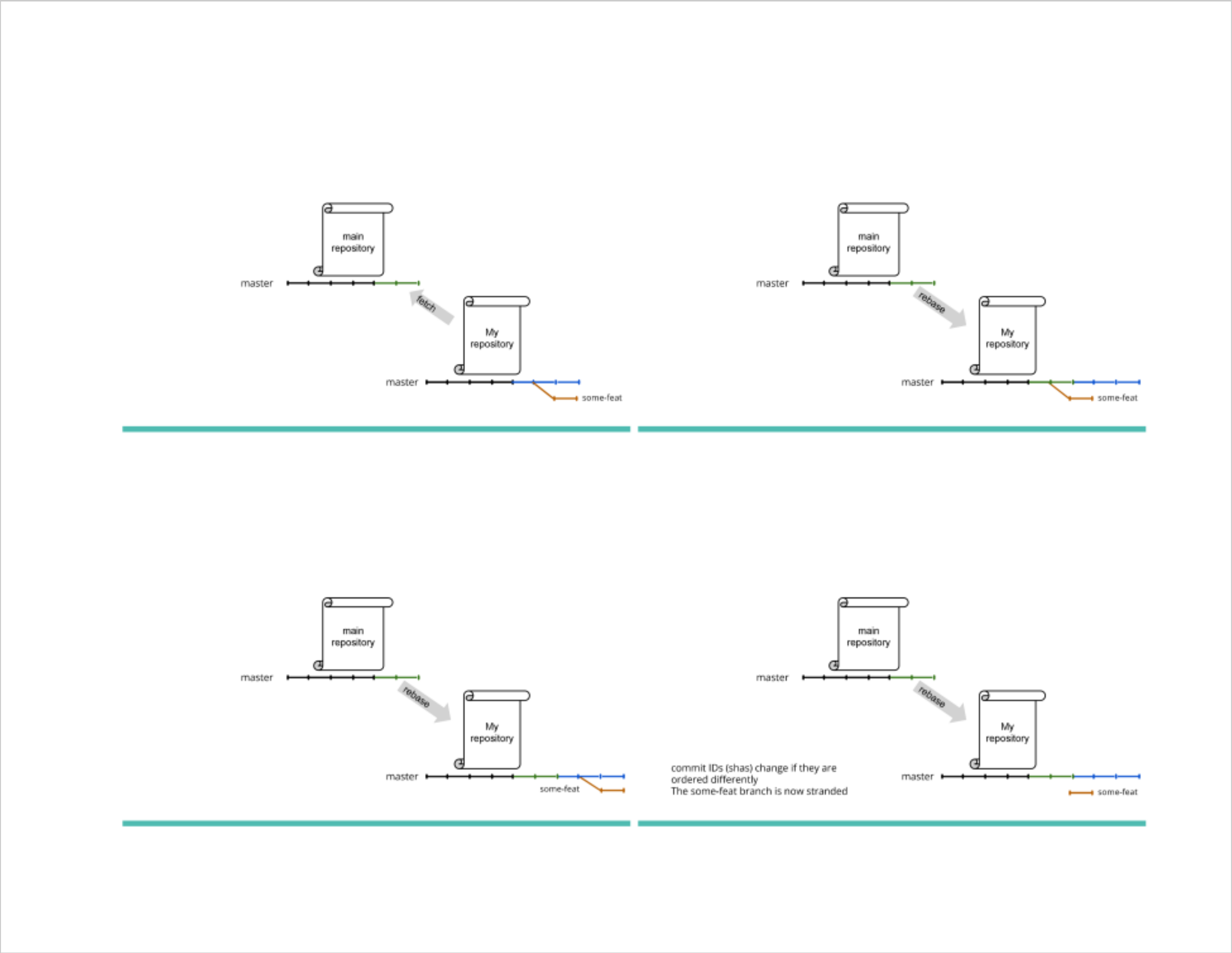




EXTRA

What happens to other branches?





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