GitHub Flow and Merge Conflicts

April 7, 2026

Announcements

- **Final Project Checkpoint 2 due TODAY April 7 at 11:59pm**
  - Push all of the code that you have written so far to GitHub for your first code review. This code should be written in your `final project/` folder and organized using a file structure similar to your labs.
  - *Tip:* If you are completing a real data analysis project, it is recommended to have conducted a preliminary exploratory data analysis and modeling pass through the data. This would allow for more constructive feedback at this checkpoint.
  - **If you want more helpful feedback, leave comments describing what you are trying to do throughout the code and list your planned/future to-dos**

Announcements

- **Final Project Checkpoint 3: code + outline/draft** due **Friday, April 17 11:59pm**
  - Address issues from Project Checkpoint 2 **using GitHub Flow** (branching/merging)
  - Report structure can vary depending on your final project
  - Ex. For a real data analysis, "typical" section headers include:
    - **Introduction:** introduce and motivate the domain problem that you are trying to study
    - **Data collection and cleaning:** describe the available data and all data preprocessing/cleaning steps that you took (including justification)
    - **Methods:** detail your model development pipeline and justify
    - **Results:** present and discuss findings in the domain context
    - **Discussion:** summarize your findings and discuss how your findings fit into the broader problem context
- Undergraduates can submit to **USCLAP** (intermediate category) or **USRESP**
  - Prize $$ involved

Plan for Today

1 Merge conflicts

+ What are merge conflicts?
+ Why do they occur?
+ How to resolve merge conflicts?

2 Branching + GitFlow

+ Issues
+ Branches
+ Pull requests

Review: Basic Git/GitHub Pipeline

Merge Conflicts

What changes when we collaborate with others

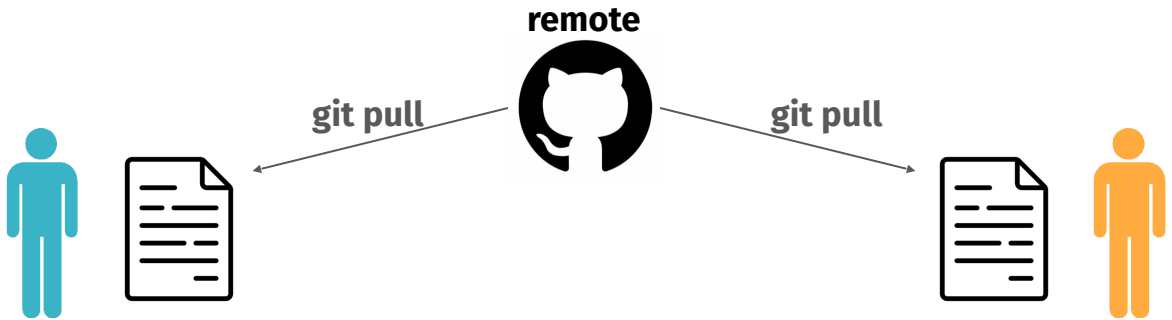
Greater possibility of **merge conflicts** if everyone works on the same branch

- Merge conflicts happen when git does not know how to combine changes from two different commits

What changes when we collaborate with others

Greater possibility of **merge conflicts** if everyone works on the same branch

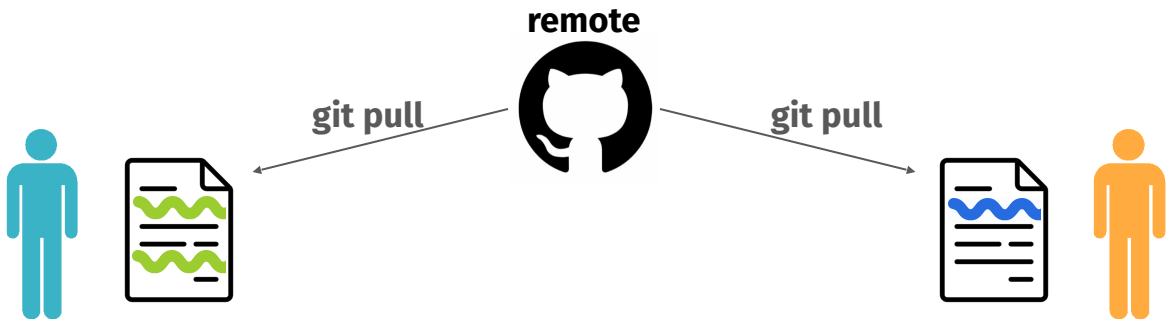
- Merge conflicts happen when git does not know how to combine changes from two different commits



What changes when we collaborate with others

Greater possibility of **merge conflicts** if everyone works on the same branch

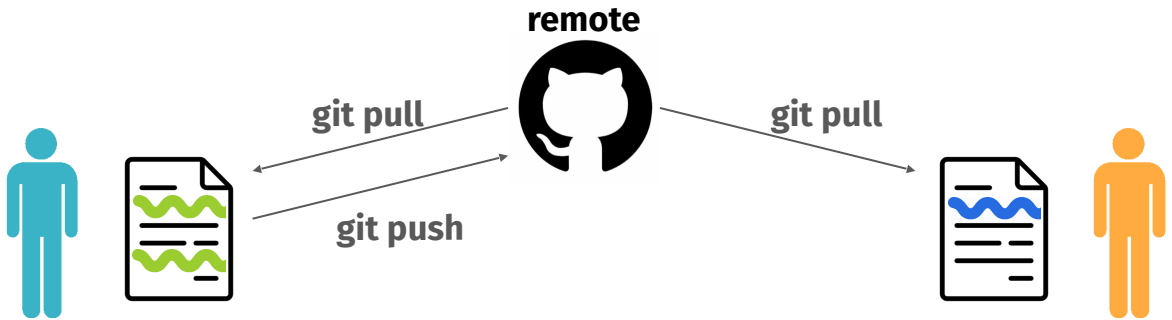
- Merge conflicts happen when git does not know how to combine changes from two different commits



What changes when we collaborate with others

Greater possibility of **merge conflicts** if everyone works on the same branch

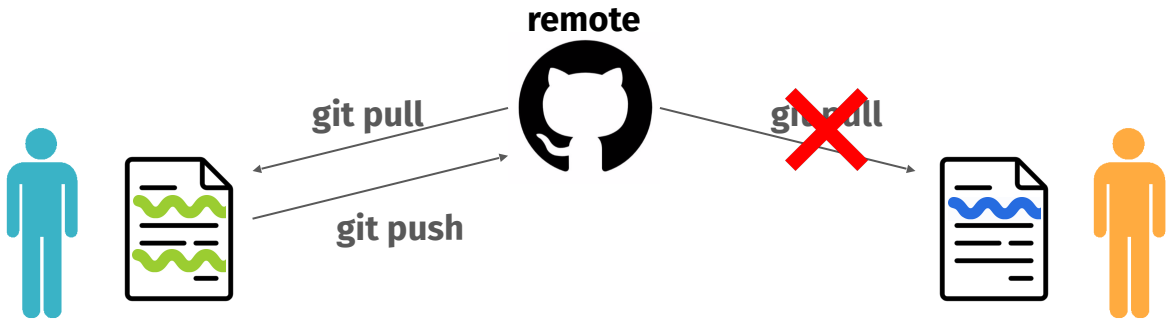
- Merge conflicts happen when git does not know how to combine changes from two different commits



What changes when we collaborate with others

Greater possibility of **merge conflicts** if everyone works on the same branch

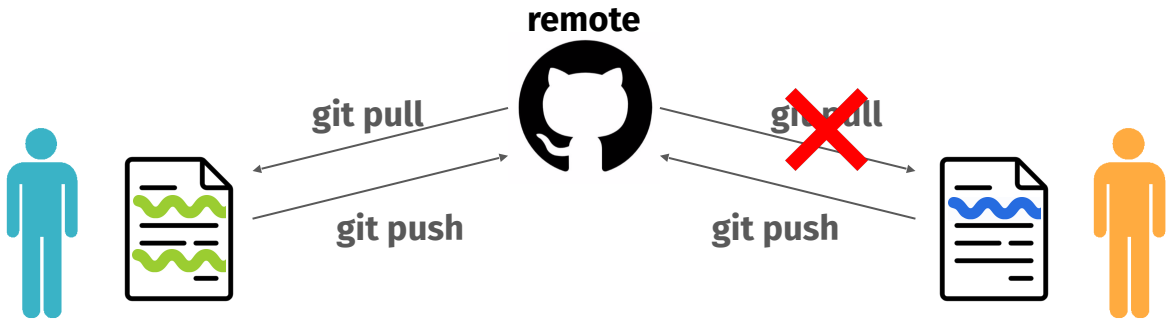
- Merge conflicts happen when git does not know how to combine changes from two different commits



What changes when we collaborate with others

Greater possibility of **merge conflicts** if everyone works on the same branch

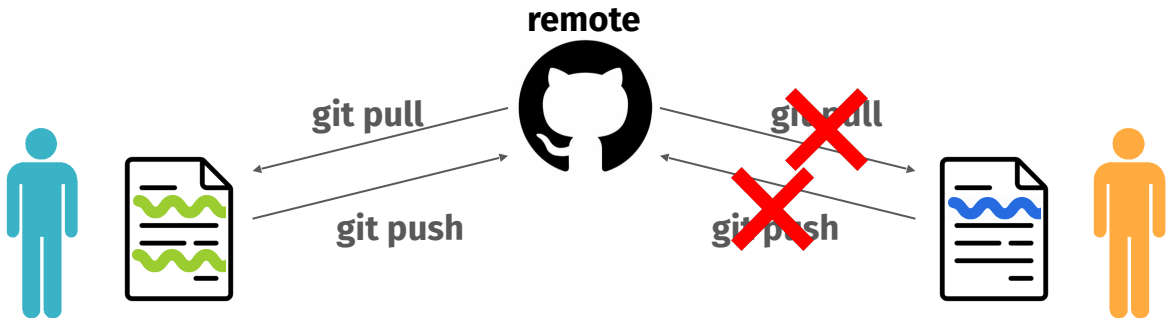
- Merge conflicts happen when git does not know how to combine changes from two different commits



What changes when we collaborate with others

Greater possibility of **merge conflicts** if everyone works on the same branch

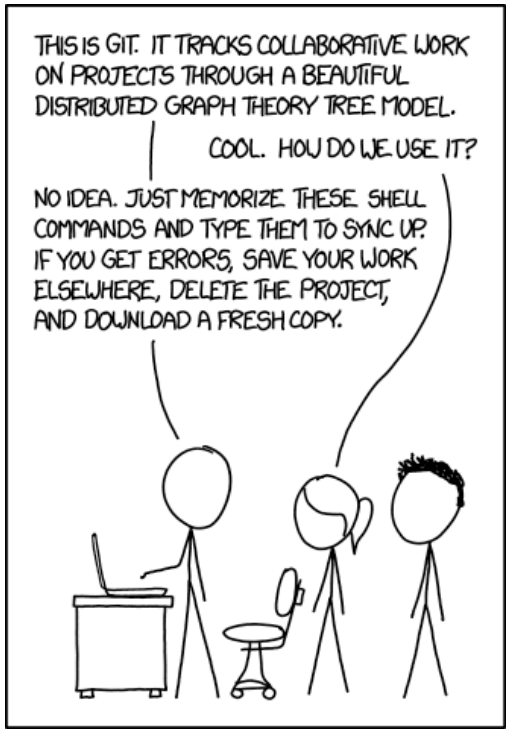
- Merge conflicts happen when git does not know how to combine changes from two different commits



How to fix merge conflicts

Option 1:

(not recommended)



How to fix merge conflicts

Option 2: resolving merge conflicts **manually**

1. Pull from remote: `git pull`
    - If there is a merge conflict, this will fail
2. Stash current changes for later: `git stash`
3. Re-try pulling from remote: `git pull`
4. Try to "pop" your stashes on top of the latest pull: `git stash pop`
5. If there are conflicts, resolve them manually

Resolving merge conflicts manually

Before resolving merge conflicts, conflicts are shown using <<<<<< and ======

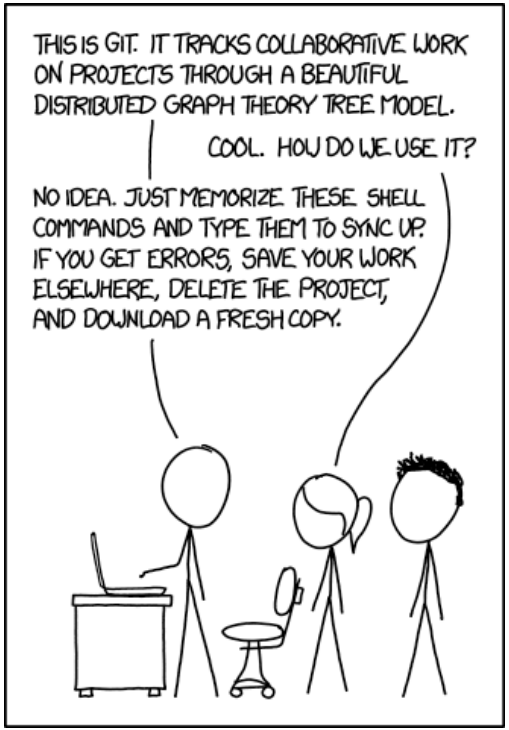
```
1  # dsip
2
3  This is an example repository for Data Science in Practice: Tools and
   Applications.
4
5  <<<<<<< Updated upstream
6  This change was made remotely.
7  =======
8  This change was made locally.
9  >>>>>>> Stashed changes
```

To resolve conflicts, open file and manually add/delete lines to your liking, e.g., if I wanted to keep both remote and local changes:

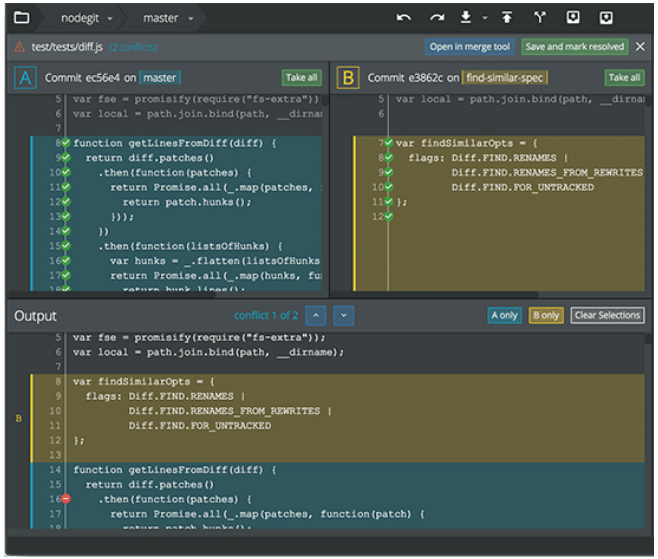
```
1  # dsip
2
3  This is an example repository for Data Science in Practice: Tools and
   Applications.
4
5  This change was made remotely.
6
7  This change was made locally.
```

How to fix merge conflicts

Option 1:

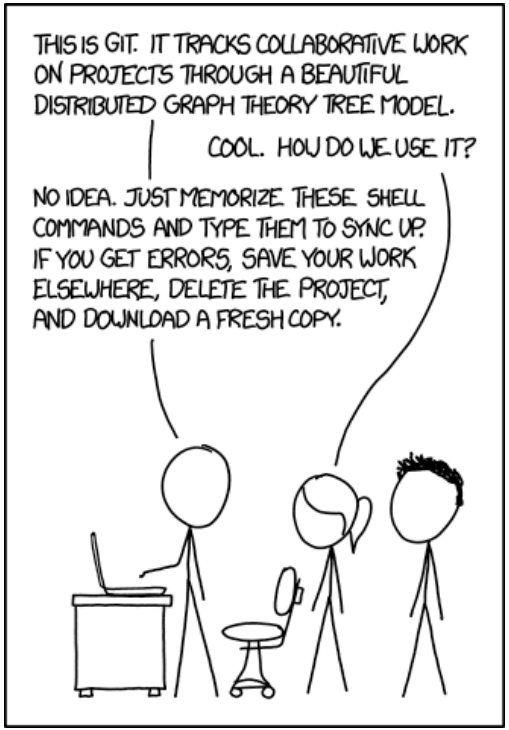


Option 2: Fix merge conflicts manually

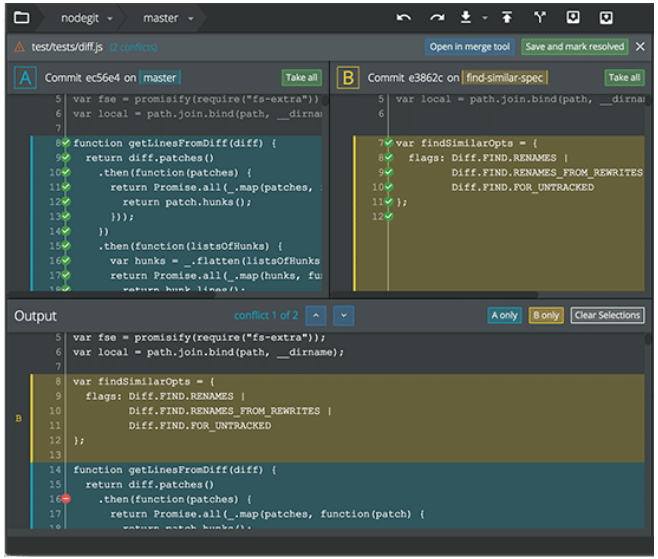


How to fix merge conflicts

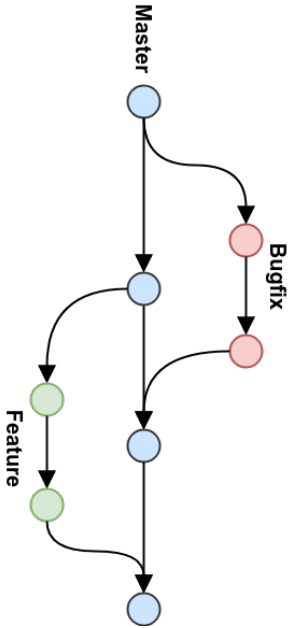
Option 1:



Option 2: Fix merge conflicts manually



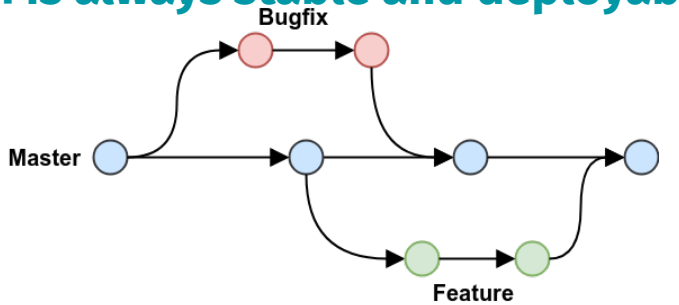
Option 3: Use **branching** for preventative care



Branching + GitHub Flow

Why Branching?

- Preventative measure to **avoid merge conflicts** after every pull/push
  - Typically only need to deal with possible merge conflicts when merging branches
- For **organization**
  - Different features or bug fixes (or people) get their own branch
  - Important to be organized in case these changes introduce bugs and you need to revert your changes (reverting is easier if changes come as one commit or a consecutive chunk of commits)
- So that **main branch is always stable and deployable**



GitHub Flow: branching in a broader workflow

GitHub Flow: branching in a broader workflow

For each change that you want to make:

GitHub Flow: branching in a broader workflow

For each change that you want to make:

1. **Create an issue** on GitHub, detailing the change
    - Issues are not only used to notify others of bugs, but also for keeping track of what everyone is working on (e.g., new features). Think of "issues" like a "to-do" list.

GitHub Flow: branching in a broader workflow

For each change that you want to make:

1. **Create an issue** on GitHub, detailing the change
    - Issues are not only used to notify others of bugs, but also for keeping track of what everyone is working on (e.g., new features). Think of "issues" like a "to-do" list.
2. **Create a new branch** to fix that issue
    - Best practice is to create a separate branch for each issue but this is not a hard rule; do what makes the most sense for your needs

GitHub Flow: branching in a broader workflow

For each change that you want to make:

1. **Create an issue** on GitHub, detailing the change
    - Issues are not only used to notify others of bugs, but also for keeping track of what everyone is working on (e.g., new features). Think of "issues" like a "to-do" list.
2. **Create a new branch** to fix that issue
    - Best practice is to create a separate branch for each issue but this is not a hard rule; do what makes the most sense for your needs
3. **Make changes** to your code and commit *frequently* to the branch

GitHub Flow: branching in a broader workflow

For each change that you want to make:

1. **Create an issue** on GitHub, detailing the change
    - Issues are not only used to notify others of bugs, but also for keeping track of what everyone is working on (e.g., new features). Think of "issues" like a "to-do" list.
2. **Create a new branch** to fix that issue
    - Best practice is to create a separate branch for each issue but this is not a hard rule; do what makes the most sense for your needs
3. **Make changes** to your code and commit *frequently* to the branch
4. **Create a pull request** (i.e., a request to merge branch into the main branch)

GitHub Flow: branching in a broader workflow

For each change that you want to make:

1. **Create an issue** on GitHub, detailing the change
    - Issues are not only used to notify others of bugs, but also for keeping track of what everyone is working on (e.g., new features). Think of "issues" like a "to-do" list.
2. **Create a new branch** to fix that issue
    - Best practice is to create a separate branch for each issue but this is not a hard rule; do what makes the most sense for your needs
3. **Make changes** to your code and commit *frequently* to the branch
4. **Create a pull request** (i.e., a request to merge branch into the main branch)
5. **Review the pull request** and resolve any merge conflicts

GitHub Flow: branching in a broader workflow

For each change that you want to make:

1. **Create an issue** on GitHub, detailing the change
    - Issues are not only used to notify others of bugs, but also for keeping track of what everyone is working on (e.g., new features). Think of "issues" like a "to-do" list.
2. **Create a new branch** to fix that issue
    - Best practice is to create a separate branch for each issue but this is not a hard rule; do what makes the most sense for your needs
3. **Make changes** to your code and commit *frequently* to the branch
4. **Create a pull request** (i.e., a request to merge branch into the main branch)
5. **Review the pull request** and resolve any merge conflicts
6. **Merge the pull request** into the main branch

GitHub Flow: branching in a broader workflow

For each change that you want to make:

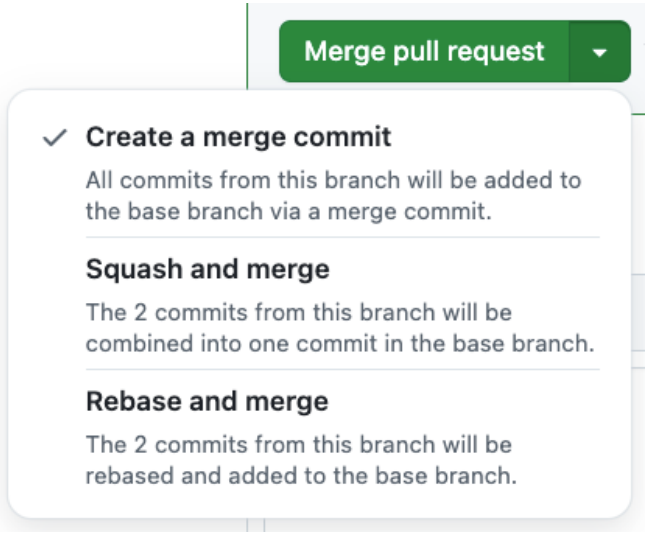
1. **Create an issue** on GitHub, detailing the change
    - Issues are not only used to notify others of bugs, but also for keeping track of what everyone is working on (e.g., new features). Think of "issues" like a "to-do" list.
2. **Create a new branch** to fix that issue
    - Best practice is to create a separate branch for each issue but this is not a hard rule; do what makes the most sense for your needs
3. **Make changes** to your code and commit *frequently* to the branch
4. **Create a pull request** (i.e., a request to merge branch into the main branch)
5. **Review the pull request** and resolve any merge conflicts
6. **Merge the pull request** into the main branch
7. **Delete the branch**

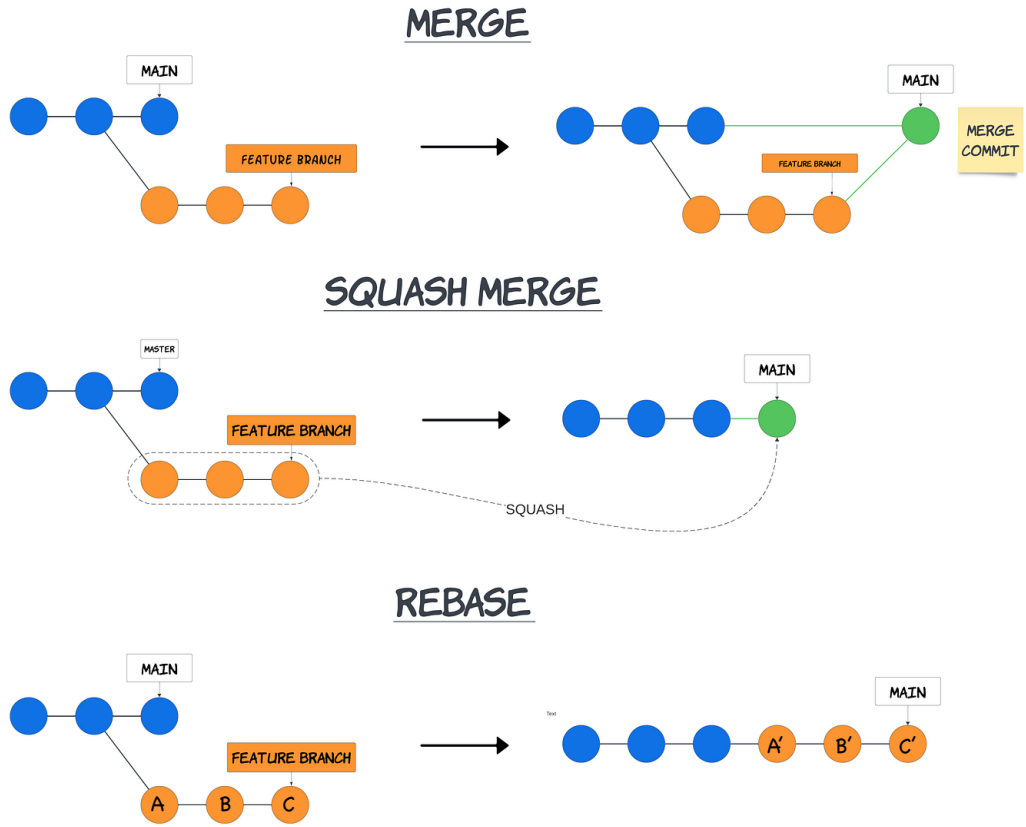
GitHub Flow: branching in a broader workflow

As part of your final project grade, you will need to use GitHub Flow at least once:

1. **Create an issue** (or use one of the issues I've created for you)
2. **Create a new branch** for that issue
3. **Make changes** to your code to resolve that issue
4. **Create a pull request**
5. **Ask the instructor to review your pull request before merging**
6. **Merge the pull request** into the main branch
7. **Delete the branch**

Different Types of Merges





Resolving extremely minor issues

If you have a small issue that can be fixed in a single commit (with no chance of introducing bugs), branching might be an overkill.

An alternative to branching for these minor fixes:

+ In your commit message/description, include "closes #<issue_num>"
+ This will link your commit to the issue and will automatically close the issue (as completed) when you push your changes to the main branch

Recap + Next Time

Recap

+ **To rectify merge conflicts:** use git stash and if that fails, can manually resolve these merge conflicts
+ **GitHub Flow:** use issues, branches, and pull requests to organize your changes/to-dos

Next time

+ LLMs