Introduction

Form pairs of two (required!) and collaborate using your own laptops/machines. Use the git-mania repository referenced below.

Note: The setup instructions below are sometimes intentionally vague. You may have to ask google or your peers questions like "how do I fork a repo?". Remember to always research first prior to asking for clarification/help.

Repository: https://github.com/cicccDerrick/git-mania

One of you will be Developer A, and the other Developer B.

Setup

1. Developer A: fork the repo git-mania
2. Developer A: clone the new fork of the repo
3. Developer A: give Developer B's github profile collaborator access to your fork of the repo (in the settings)
4. Developer B: clone Developer A's fork of the repo

Points to raise and understand at this stage:

- What does it mean to fork a repository?
- Is forking a git thing or a github thing?
- In fact, what is the difference between git and github?
- What does it mean to clone a repository? How is it different to forking?
- Is cloning a git thing or a github thing?
- Why does Developer A have to give Developer B access to their repo? Especially since the repo is already public.
- What is the difference between a public and private repo on github?

AutoMerge Example

1. Developer A: Add content to the HTML.
2. Add a city to the *end* of the Ordered List of cities
3. Developer A: commit
4. Developer A: push the commit
5. Developer B: add content to another part of the HTML
6. Add a city to the *beginning* of the Ordered List of cities
7. Developer B: commit
8. Developer B: attempt to push (it will fail/error out)
9. Why does this fail?
10. Because Git notices that the remote also has a changed version of the HTML file.

11. The developer now needs to pull the remote changes and merge them to the local changes (as a merge commit).
12. Developer B pulls changes
13. Since there were new commits on the remote (origin), Git downloads it. The content change made by Developer A is now downloaded and available to B. Since the change was to a different part of the file, git has no trouble merging it automatically
14. Developer B: push changes
15. Developer A: pull changes

Both developers now have each others changes, without any extra work. This is the case even though they worked on the *same file*, at the *same time*. Git automatically merged the changes. This was only possible because the developers did not modify the same lines of code / parts of the file.

Append Conflict Example

1. Developer A: append content to HTML document
2. Add a city to the end of the Ordered List of cities
3. Developer A: commit
4. Developer A: push

5. Developer B: append content to HTML document
6. **Note:** Do not pull before doing this.
7. Add a city to the *end* of the Ordered List of cities.
8. Developer B: commits
9. Developer B: attempt to push (it will fail/error out)
10. Why does this fail?
11. Because Git notices that the remote also has a changed version of the HTML file.
12. The developer now needs to pull the remote changes and merge them to the local changes (as a merge commit).
13. Developer B: pull and receive a conflict
14. Since the file was updated in the same place, it cannot be auto-merged.
15. Use git status and you'll notice that the file is changed and needs to be committed. But it has a conflict that needs our attention, so we can't just commit it right away.
16. Developer B: resolve conflict
17. Open the conflicting file and notice the `<<<<` and `>>>>>` around the conflicting area. Edit the file so that both additions to the `OL` are kept and the conflict tags (`<<<<<`, `>>>>>` and `=====`) are also removed.
18. Developer B: add and commit the file
19. For conflict resolution commits, you don't need to provide a message, just run `git commit` by itself (with out a `-m` flag).

20. Developer B: push

Both developers now have each others changes. However, one of the developers (the one who attempted to push second) had to resolve the conflict and merge the changed lines of code manually by performing conflict resolution. A manual merge conflict resolution is required when multiple developers edit the same line of code.

Questions:

Question 1: During this linear process, where Dev B changed the document after Dev A had already committed and pushed their change to the remote (github), what extra step(s) could Developer B have taken to completely avoid this merge conflict?

Question 2: Given your answer in Q1, does this mean that merge conflicts can be completely (always) avoided in the real world? Why or why not?