Resolving Merge Conflicts

(https://github.com/learn-co-curriculum/git-github-resolving-merge-conflicts) (https://github.com/learn-co-curriculum/git-github-resolving-merge-conflicts/issues/new)

Learning Goals

- Resolve merge conflicts.
- Delete branches using `git delete`.

Introduction

So far we've only looked at cases where there are no conflicts between the two branches we're trying to merge. However, there will be times when the two branches contain conflicting changes, i.e., changes to the same file or files that are different from each other. In this lesson we'll learn how to resolve these situations.

Resolving Merge Conflicts

We'll work through the process of resolving merge conflicts using our Python resources project.

> Note: for purposes of illustration, we will intentionally create a merge conflict, then go through the process of resolving it. Normally, of course, we would try to avoid creating conflicts. But they will arise from time to time, so it's good to know how to handle them when they do.

Make sure you're on the `main` branch, then create a new branch, `add-videos`, and switch to it. Add the following to the bottom of `python-resources.md`:

```
## Videos

- [Python Talks Playlist](https://youtube.com/playlist?list=PLiS5HT0ret8jvmP3GGmom3hY3jiFBq-8w)
```

Add and commit the changes, then switch back to the main branch.

On `main`, we'll start by adding some third-level headers into our Books section: Beginner, Intermediate, and Advanced. It should look like this:

```
# Python Study Resources

## Books

### Beginner

- [Composing Programs (Chapter 1-2)](https://composingprograms.com/) [Free]
  - This is a great book for learning compsci concepts in addition to Python
- [Automate the Boring Stuff with Python](https://automatetheboringstuff.com/) [Free]
  - Has short, easy projects that involve different modules
- [Python Crash Course , 2nd Ed](https://nostarch.com/pythoncrashcourse2e)
  - Beginner friendly intro to fundamental concepts

### Intermediate


### Advanced
```

Once you've added the headers, add and commit your changes.

Next, we'll add a book to the intermediate section and one to the advanced section:

```
...

### Intermediate

- [Serious Python](https://serious-python.com/)
  - Does a great job of explaining some of the more difficult topics not covered
    in depth in the beginner books
```

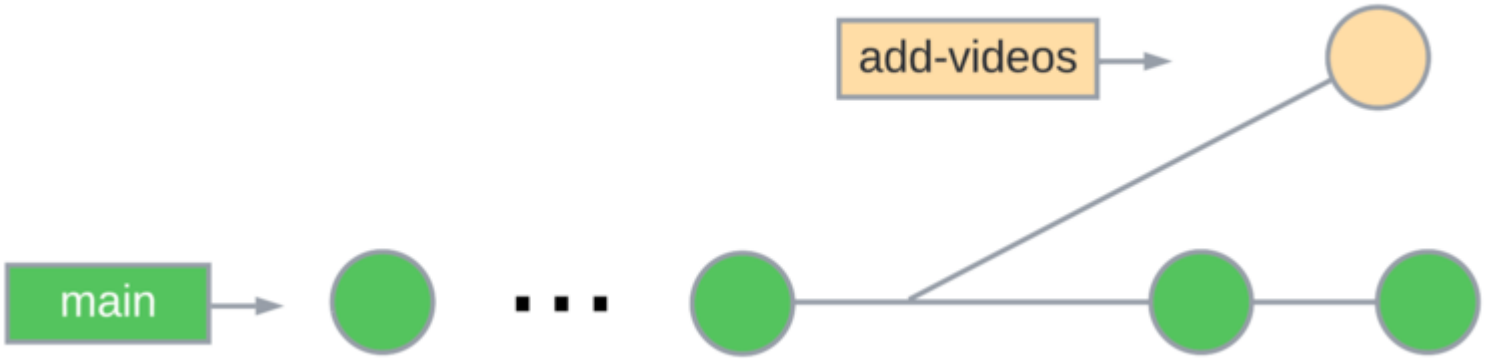
```
### Advanced

- [Fluent Python, 2nd Edition](https://www.oreilly.com/library/view/fluent-python-2nd/9781492056348/)
  - The best reference book for Python
```

Add and commit these changes as well.

Next, we'll merge in the `add-videos` branch. Recall that we need to be on the branch that we are merging the changes into, in our case, `main`. Since we are already there, we're ready to run the command to merge in `add-videos`:

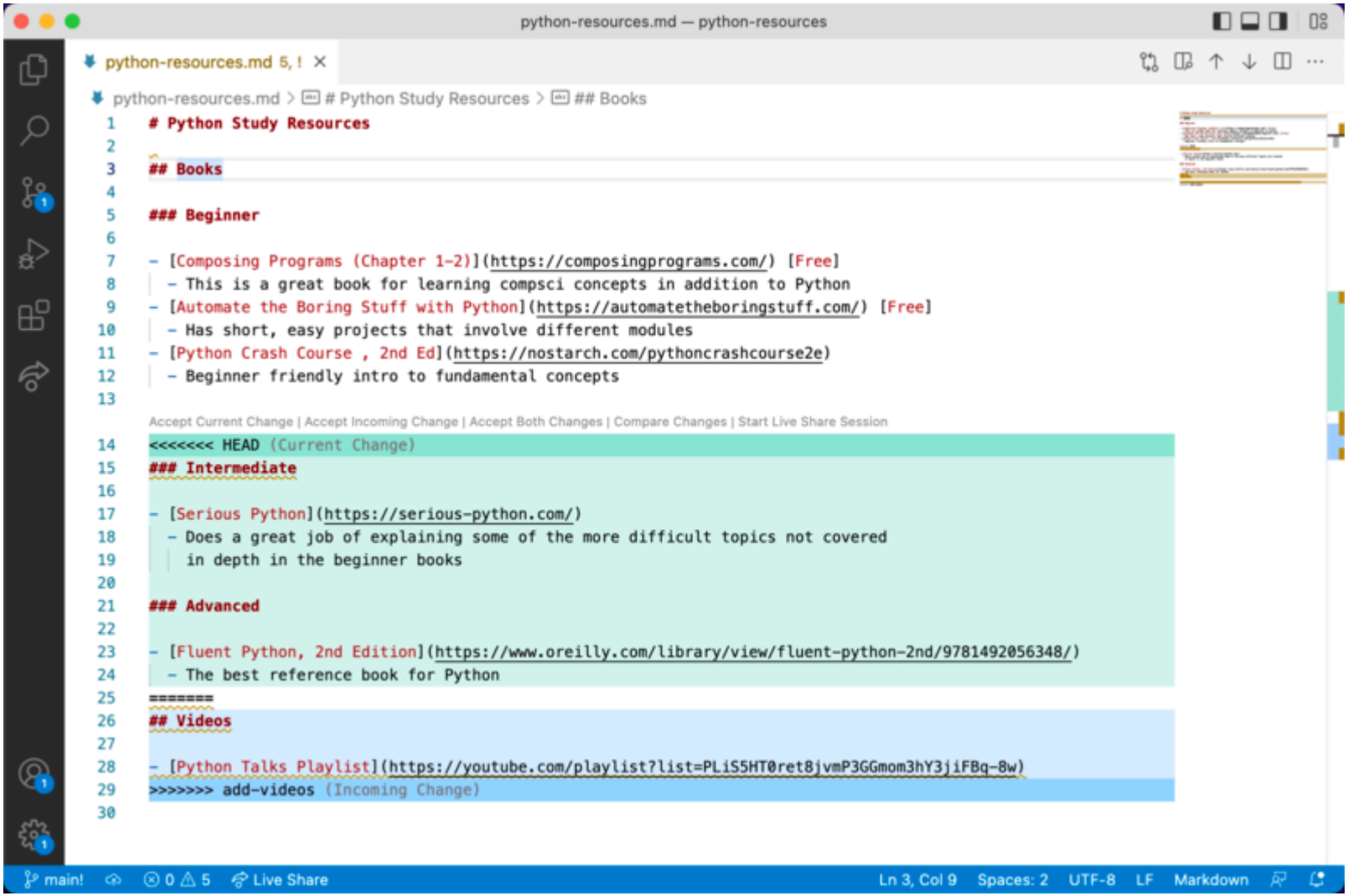
```
$ git merge add-videos
Auto-merging python-resources.md
CONFLICT (content): Merge conflict in python-resources.md
Automatic merge failed; fix conflicts and then commit the result.
```

Git is letting us know there is a *merge conflict*. Our commit history looks like this:



The last commit shared by both branches is the one right before the `add-videos` branch was created. Since that time, changes have been made to `python-resources.md` on both branches. Git has no way of knowing which changes should be used in the merged version of the file, so we will need to resolve the conflicts by hand.

Git and VS Code give us some help in fixing the conflicts. If you take a look at `python-resources.md` in VS Code, you should see something like this:



The color highlighting and strings of characters can be confusing at first, so we'll walk through them.

The character strings are what Git uses to help us see the conflicting information and know where each version comes from. The `<<<<<<< HEAD` string marks the beginning of the conflicts and indicates that the changes in the section that follows come from `HEAD` (in our case, the `main`

branch). The `>>>>>>> add-videos` marks the end of the conflicts, and indicates that the changes in the section immediately preceding that marker come from `add-videos`. The `=======` separates the two.

The highlighting, which is created by VS Code, does the same thing, but makes it a little easier to see. The upper highlighted section (in green) shows the "Current Change", i.e., the change from the branch we're currently on. The lower highlighted section (in blue) shows the "Incoming Change", the change in the branch we're trying to merge in.

Resolving the conflicts is a simple matter of deciding which part or parts of the conflicting section we want to keep, then editing the file accordingly. We have complete control over how we resolve the changes: we can choose to keep the Current changes, the Incoming changes, or some combination of the two. We can also choose to keep **all** the changes, which is what we want to do in this case.

To keep all the changes, we just need to remove the lines that contain the Git markers, lines 14, 25, and 29 in the screenshot above. Be sure there's a blank line before the "Videos" subheading. When you remove the markers, the highlighting will disappear.

Alternatively, you can make use of the helpers VS Code provides. If you look just above the opening marker, you'll see a series of clickable commands, including "Accept Both Changes". Clicking that option will have the same effect as removing the marker lines by hand. (Don't forget to add a blank line before the "Videos" subheading.)

Once you've resolved the conflict and saved the file, run `git status`:

```
On branch main
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   python-resources.md

no changes added to commit (use "git add" and/or "git commit -a")
```

Git is telling us that the merge is still in progress, and that we need to resolve the conflicts then commit the changes. We've fixed the conflict, so the last step is to make our commit. This commit will be the merge commit, so we can provide a commit message to indicate that:

```
$ git commit -am 'merge add-videos into main'
[main d550f94] merge add-videos into main
```

Recall that the combination of the `a` and `m` flags allows us to add and commit our changes in a single command.

That's it — we've merged in our feature branch! If you run `git status` again, you'll see the last commit we made (the merge commit) at the end of the commit history. You'll also see the commit we made on the `add-videos` branch earlier in the history.

This example was fairly straightforward, but in some cases, you may have multiple files to correct, or multiple conflicts within individual files (or both). However, the process is the same: work through each section of conflicts identified, determine which changes you want to incorporate, and then edit the file accordingly, either by hand or using the commands provided by VS Code.

Deleting Branches

Once we've finished working on a feature branch and merged it into the main branch, everything we did is now captured in `main`, so we can safely delete the feature branch. You might also delete a branch if the work you were doing on it has been discontinued (assuming you're sure you won't need it later).

We can't delete a branch that we are currently on, so the first step is to switch to a different branch, if necessary. From there, we run the `branch` command with the `d` flag, followed by the name of the branch we want to delete:

```
$ git branch -d <name-of-branch>
```

If the branch has not been fully merged (e.g., if we decide to discontinue work on that branch), we will get an error:

```
$ git branch -d unmerged
error: The branch 'unmerged' is not fully merged.
If you are sure you want to delete it, run 'git branch -D unmerged'.
```

At this point, we can either merge the branch in then run the delete command again, or, if we're sure we don't need the unmerged changes, we can force delete it by using the `D` flag in place of `d`:

```
git branch -D <name-of-branch>
```

Exercise

It's a good idea to practice the process we went through in this lesson, both to help understand how merge conflicts arise and to practice resolving them. The steps are:

1. Create a new branch to add a new resource. This can be whatever you like, but one idea is practice coding sites (e.g., **Codewars (https://www.codewars.com/)**).
2. Switch to the new branch and add the resource at the end of the file.
3. Switch back to `main` and add some **different** new content to the end of the file. Again, this can be whatever you like.
4. Run `git merge <new-branch>`. You should get a merge conflict error.
5. Open the file in VS Code and resolve the conflicts.
6. Add and commit the changes; this will be your merge commit.
7. Verify that the merge is complete: the commit history should contain the commits from both branches as well as the merge commit.
8. When you're done, be sure to delete any branches that are no longer needed.

Conclusion

The best way to handle merge conflicts is to keep them from happening in the first place. But no matter how careful you are, they will inevitably arise, especially when you're working as part of a team. We will talk about habits and processes that will help you avoid conflicts in the next lesson on collaborative workflows, but the information in this lesson will prepare you to deal with them when they do arise.

Resources

- **Git merge conflicts (https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts)**