Welcome

Good afternoon and thank you for attending today's workshop. We will get started shortly.

To follow along with the demo portion of the workshop, you will need a GitHub account and to have Git installed and configured on your own computer; we will not have time to cover this today. These topics are covered in the Basic Git workshop (https://mark-durbin-ucf.github.io/git-novice/).

Please note: today's workshop <u>will not be recorded</u>, but this presentation will be emailed to all registrants tomorrow before noon. Please feel free to email me (mark.durbin@ucf.edu) with any questions you may have.

Version Control with Git – Intermediate

Mark Durbin - Cloud Computing Engineer III
Office of Research Information Technology

September 14, 2023

Requirements

You need to have Git installed on your own computer to follow along with the demo portion of this workshop. You will also need a GitHub account, and have configured an SSH key on your computer. These items are covered in the basic Git workshop: https://mark-durbin-ucf.github.io/git-novice/.

Goal of this workshop

The goal of today's workshop is to prepare you to work collaboratively on coding projects in academic and professional environments, specifically with Git and GitHub.

A note about Git workflows

Git and GitHub workflows vary between teams, individuals, companies, and projects, so the workflow and techniques presented in this workshop should not be considered the "correct" way to use Git or GitHub, but rather examples of different actions that can be taken to maintain a history of changes to your project. Today we will be using the Gitflow workflow, one of many documented workflows.

Before working on a project with others, decide with your collaborators on a general methodology for using Git and GitHub in order to streamline your workflow. If you are joining a project that has already been established, review contributing guidelines, documentation, and speak with your collaborators before committing to minimize conflicts.

Topics we will cover today

- Branching

- Pull requests

- Merging

Have questions about another Git or GitHub topic? If we have time at the end, I'll try to answer your questions.

Note: the instructions for today's workshop include directions to use the terminal text editor called Vim. You do not need to use Vim. Feel free to use any text editor installed on your machine to edit your files.

Branching

Keeping changes separate until they are ready to be combined

Branching – What is a branch?

"Branches allow you to develop features, fix bugs, or safely experiment with new ideas in a contained area of your repository."

Source:
https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches

Branching – When should I use a branch?

You should use a branch whenever you want to make changes to the stable source code of your project. Branches make comparing the latest stable version of your code to your altered code much easier.

Note: You can certainly do all your work on the <u>main</u> branch, but as your code becomes more complex, and you are working on more than one change at a time, with multiple people, you may find that combining multiple code sources is difficult and error-prone.

Branching – Gitflow branching strategy

Feature branches are used to track a specific proposed change to the production files for a project. These branches can be deleted once they are merged into the develop branch.

The development branch (develop) is used to track the latest state of changes that are being proposed to a project. In this case, feature branches are merged into develop as they are completed and tested. This branch is persistent and does not get deleted.

Release branches are created to mark the features that will be made available in the next stable release. These branches can be deleted once they are merged into the production branch.

The production branch (main) of a web application is used to track stable, or production-ready code. Once the code in the release branch is ready to be released, it will be merged into the main branch. This branch is persistent and does not get deleted.
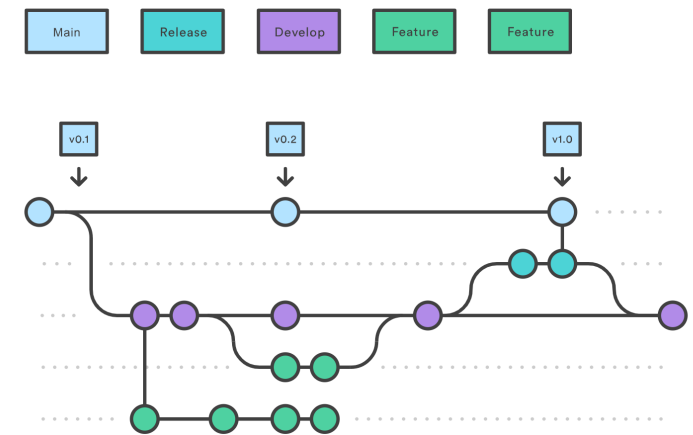

Image source:
https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow
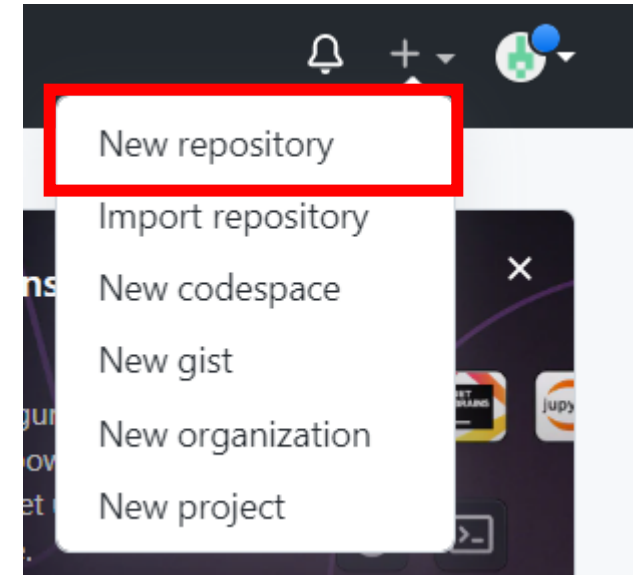
Branching – Create a sample repository

Sign into GitHub

In the upper right corner of the page, click the + symbol and then New repository

Enter "intermediate-git-workshop" for the repository name

Check the Add a README file checkbox
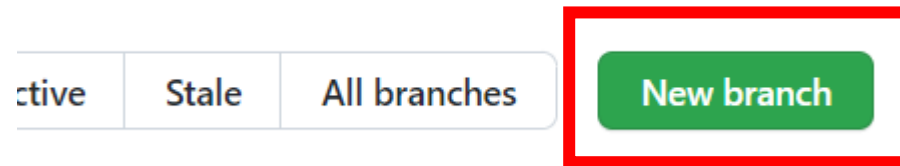
Click Create repository

Branching – Create a development branch

Go to the page for your repository called "intermediate-git-workshop"

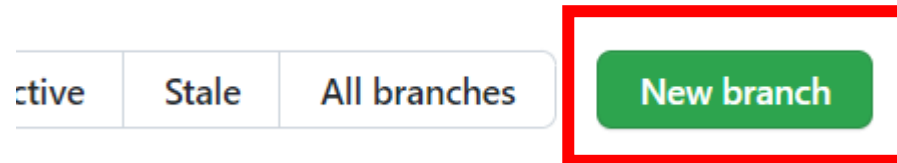Click the branches detail

Click the New branch button

Enter "develop" for the branch name and click Create branch

Branching – Create a feature branch

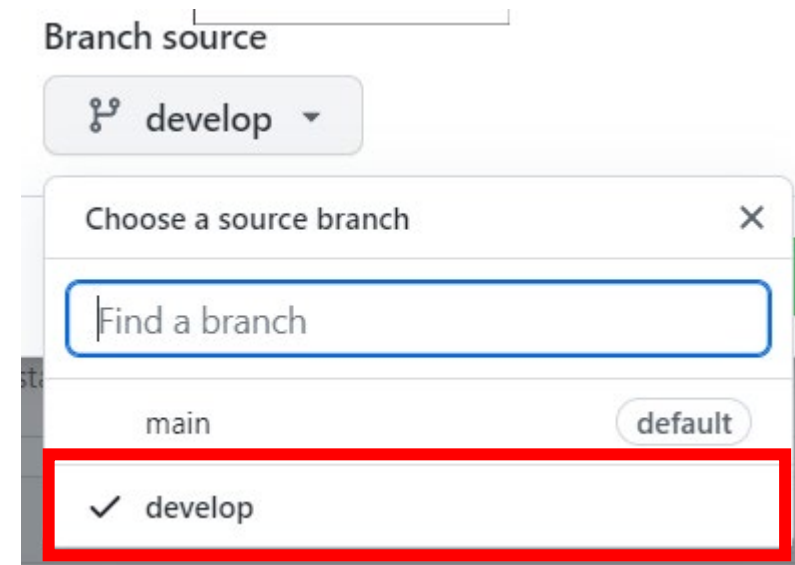Refresh the page (develop won't show up as a branch source if you don't)

Click the New branch button

Enter "feature" for the branch name

Change branch source to develop

Click Create branch

Branching – Branch protection

Persistent branches like main and develop should have branch protection rules to prevent accidental deletion.

Note: As of September 2023, protected branch rules are not enforced on private repositories until you move to a GitHub team or Enterprise organization account.

Branching – Add branch protection rules

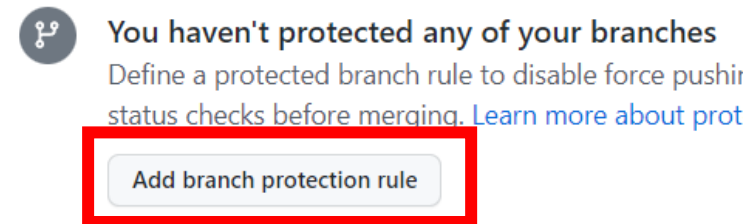On your repository page, click <u>Settings</u>

Click <u>Branches</u>
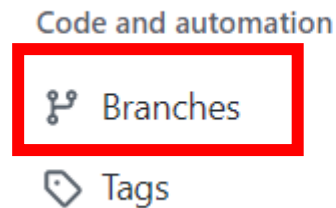
Click add a branch protection rule

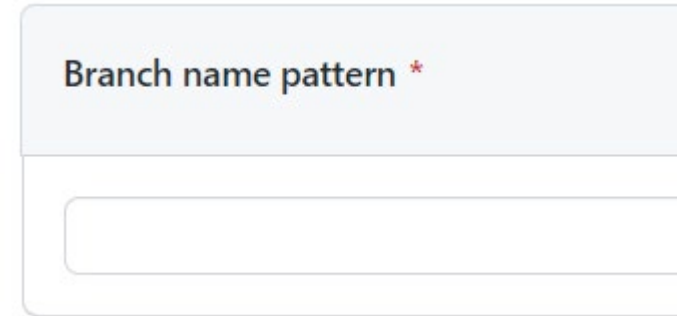# Branching – Protect main and develop

Type "main" in the Branch name pattern input

Click <u>Create</u>

Add another rule for the develop branch

Branch name pattern *

Pull Requests

Requesting updates to the project code

Pull requests – What is a pull request?

"Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch."

Source:

https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests

Pull requests – When should I submit a PR?

Working on a project with others, you have created a branch to do your work in and have completed the work. Now, you want to combine your work with others in the development branch before releasing it. You submit a pull request to move your code changes in the feature branch into the development branch.
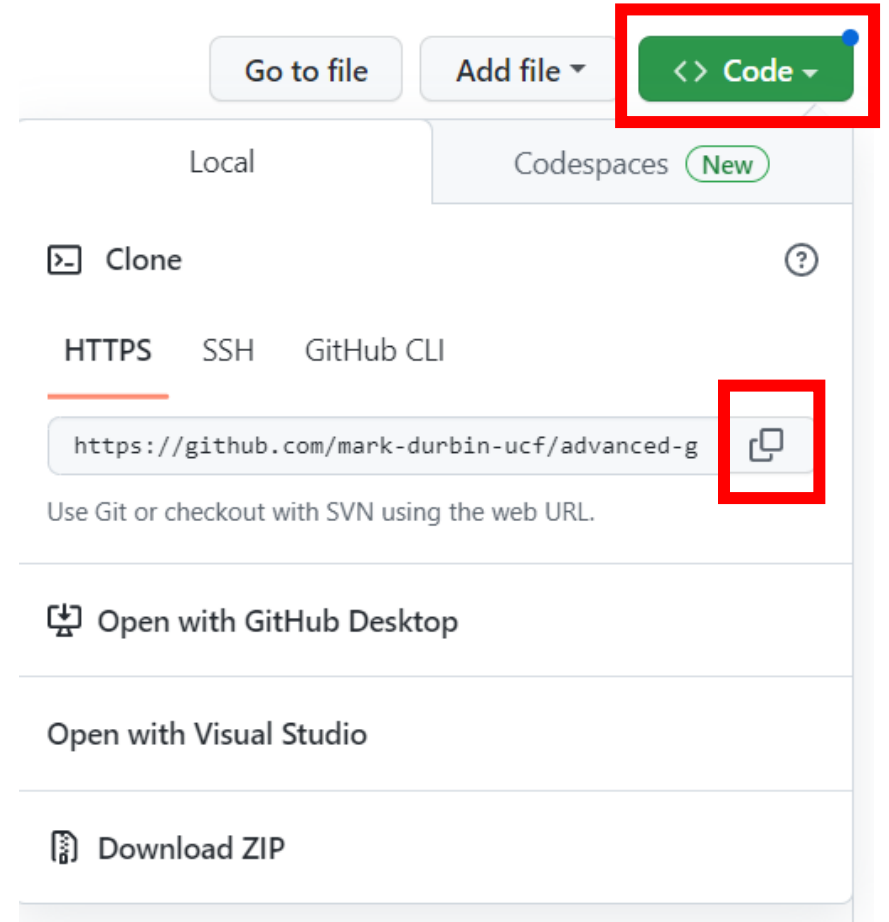
Pull requests – Clone your repository

Go to your repository on GitHub and click Code, then the copy-to-clipboard button to copy the URL for your repository

Open a terminal on your computer

Clone the remote repository in your home folder:

```
cd ~/
git clone YOUR_REPO_URL
```

Pull requests – Create a new file in your repo

Fetch remote branches:

```
git fetch --all
```

Update your local repo with remote changes:

```
git pull --all
```

Check out the branch we created on GitHub

```
git checkout feature
```

Create a new file in your local repository called my-file.txt:

```
touch my-file.txt
```

Pull Requests – Edit my-file.txt in your repo

Open the file in Vim:

`vim my-file.txt`

Press the `i` key to start insert mode, and type the following sentence:

`This file was made in the feature branch`

Press the `esc` key to exit insert mode, type `:wq`, and then press the `enter` key to save the file

Pull Requests – Commit and push changes

Stage the file for commit:

```
git add my-file.txt
```

Commit the file:

```
git commit -m "Create my-file.txt"
```

Push the changes to GitHub:

```
git push
```

Pull requests – Submit the pull request
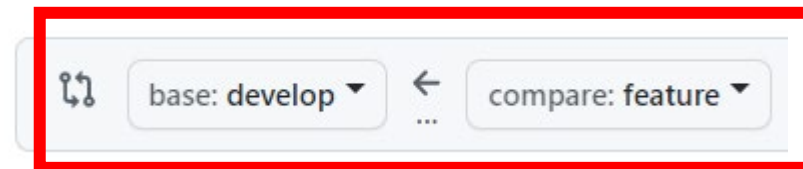
Go to your repository on GitHub

Click Pull Requests

Click New pull request

Select to pull feature into develop

Click Create pull request button

Merging

One of a few ways to combine code from different branches

Merging – What is a merge?

"Merging is Git's way of putting a branched history back together again. The git merge command lets you take the independent lines of development created by git branch and integrate them into a single branch."

Source:
https://www.atlassian.com/git/tutorials/using-branches/git-merge

# Merging – When should I merge?

A sample merge use case:

Work has been completed and tested in a <u>feature branch</u>, and you need to update the code in the <u>development branch </u>with these changes. You submit a pull request to merge the feature branch into the develop branch.

Merging – Create a merge commit option

Merging with the Create a merge commit option will preserve all the commits from the branch you are merging, allow you to provide a summary of the changes made in those commits in the body of the merge commit, and provide a single commit to revert in case you want to revert the changes from all of the commits.

Create a merge commit on GitHub is equivalent to running `git merge --no-ff`.
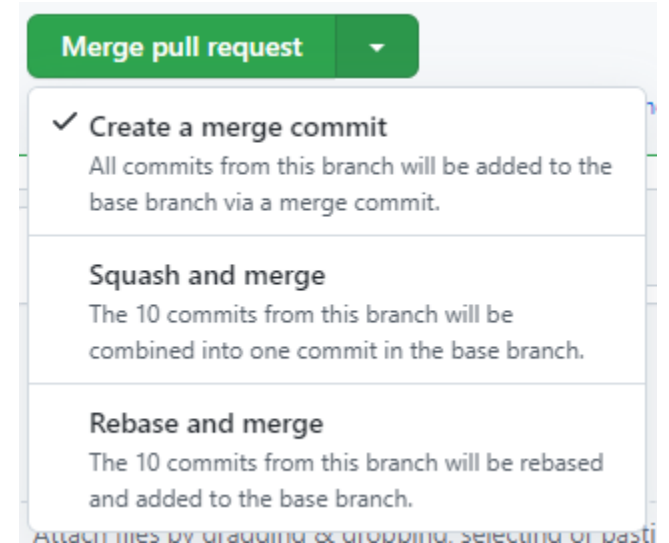
Merging – Squash and merge option

Merging using the Squash and merge command will combine the commits from the branch you are merging into a single, new commit.
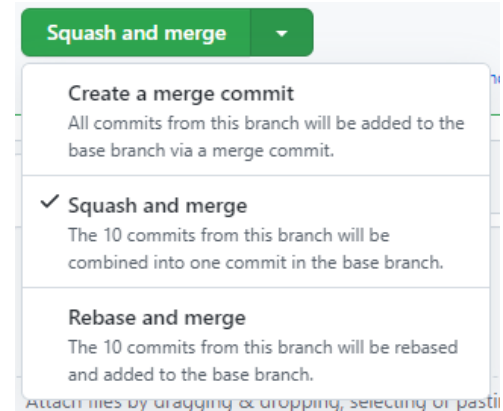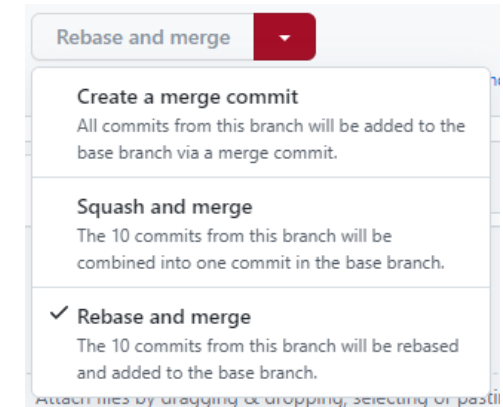
Squashing is possible on the command line by using the `--squash` option with the command `git merge`.

Merging – Rebase and merge option

Merge using the <u>Rebase and merge</u> option will preserve all the commits from the branch you are merging but will assign them new commit IDs.

Rebasing is possible on the command line by using the `git rebase` command.
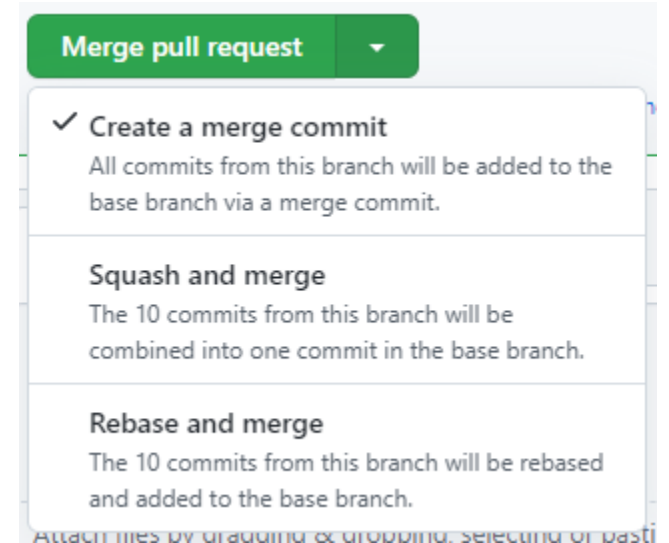
Merging – Merge feature branch

Navigate to the pull requests for your repo and select the pull request you made earlier

Select Create a merge commit in the dropdown

Click Merge pull request

Click Confirm merge

Merging – Dealing with conflicts

Merge conflicts happen when two changes to the same line of code have occurred between two branches that are being merged.

We are going to intentionally create a <u>merge conflict</u> using two new branches, feature-2 and feature-3, and then resolve the merge conflict.

Merging – Create a feature-2 branch

In your terminal, check out the <u>develop</u> branch:

```
git checkout develop
```

Pull the latest changes from GitHub:

```
git pull
```

Create a new branch called feature-2:

```
git checkout -b feature-2
```

Merging – Create my-file.txt in feature-2

Open my-file.txt in Vim:

`vim my-file.txt`

Press the `i` key on your keyboard to enter insert mode in Vim and use the arrow keys to bring your cursor to the end of the line. Then, press the `enter` key to create a new line.

Type the following sentence:

`This file was edited in the feature-2 branch`

Press the `esc` key to exit insert mode, type `:wq` (including the colon) and then press the `enter` key to save the file.

Merging – Commit and push to GitHub

Stage the file for commit:

```
git add my-file.txt
```

Commit the file:

```
git commit –m "Edited my-file.txt in feature-2"
```

Push the changes to GitHub:

```
git push –u origin feature-2
```

Merging – Create a feature-3 branch

In your terminal, check out the <u>develop</u> branch:

```
git checkout develop
```

Pull the latest changes from GitHub:

```
git pull
```

Create a new branch called feature-3:

```
git checkout -b feature-3
```

Merging – Create my-file.txt in feature-3

Open my-file.txt in Vim:

```
vim my-file.txt
```

Press the `i` key on your keyboard to enter insert mode in Vim and use the arrow keys to bring your cursor to the end of the line. Then, press the `enter` key to create a new line.

Type the following sentence:

```
This file was edited in the feature-3 branch
```

Press the `esc` key to exit insert mode, type `:wq`, and then press the `enter` key to save the file

Merging – Commit and push to GitHub

Stage the file for commit:

```
git add my-file.txt
```

Commit the file:

```
git commit –m "Edited my-file.txt in feature-3"
```

Push the changes to GitHub:

```
git push –u origin feature-3
```

Merging – Create a feature-4 branch

In your terminal, check out the <u>develop</u> branch:

```
git checkout develop
```

Pull the latest changes from GitHub:

```
git pull
```

Create a new branch called feature-4 and push to GitHub:

```
git checkout -b feature-4
```

Merging – Edit my-file.txt in feature-4 branch

Open my-file.txt in Vim:

`vim my-file.txt`

Press the `i` key on your keyboard to enter insert mode in Vim and use the arrow keys to bring your cursor to the end of the line. Then, press the `enter` key to create a new line.

Type the following sentence:

`This file was edited in the feature-4 branch. We're not done yet.`

Press the `esc` key to exit insert mode, type `:wq`, and then press the `enter` key to save the file

Merging – Commit and push to GitHub

Stage the file for commit:

```
git add my-file.txt
```

Commit the file:

```
git commit –m "WIP"
```

Push the changes to GitHub:

```
git push –u origin feature-4
```

Merging – Merge feature-2 into develop

Submit a pull request on GitHub to merge feature-2 into develop, and then open your terminal on your computer to your local repository.

Check out the develop branch, merge the feature-2 branch, and push to the remote repository:

```
git checkout develop
git merge --no-edit --no-ff feature-2
git push
```

On the pull request page on GitHub, you should see that the pull request has been closed, because you pushed develop to GitHub.
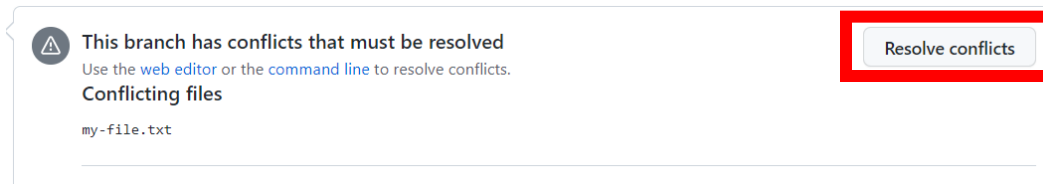
Merging – Merge feature-3 into develop

Start creating a pull request to merge feature-3 into develop.

GitHub will tell you that the two branches can't be automatically merged. This means you will need to resolve the merge conflict.
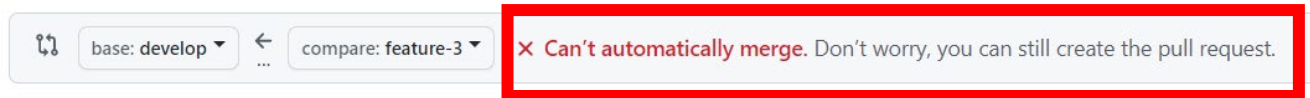
Click <u>Create pull request</u>

After you create the pull request, view it, and click <u>Resolve conflicts</u>

Merging – Resolve conflict

Click <u>Resolve conflicts</u> and open the editor. Remove the contents of the lines that are marked below.
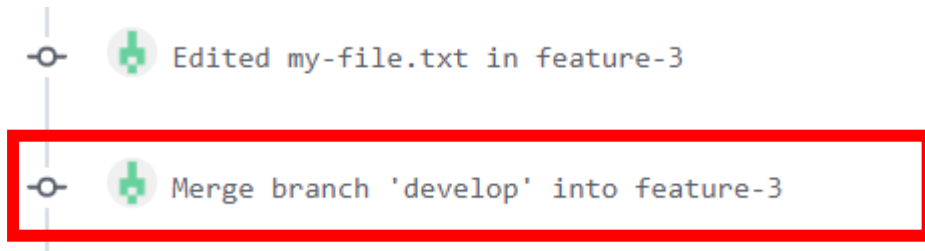
Click <u>Mark as resolved</u>, then click <u>Commit merge</u>

Merging – Merge pull request

After the conflict is resolved, you will see an additional commit in the pull request that indicates that you have merged develop into feature-3; this is where you resolved the merge conflict.

On the pull request page, click <u>Merge pull request</u>

Click <u>Confirm merge</u>

Questions?

If you have any questions, please type them in the chat or unmute yourself.

If you have questions but don't feel like asking them now in front of other people, that's okay too. Email me at mark.durbin@ucf.edu and CC ResearchIT@ucf.edu.

Additional Resources

Git tutorials from Atlassian:
https://www.atlassian.com/git/tutorials

Git courses on LinkedIn Learning
(free to students and employees, log in here:
https://digitallearning.ucf.edu/linkedin-learning/):
https://www.linkedin.com/learning/topics/git?u=57691257

Using Git with VSCode IDE:
https://code.visualstudio.com/docs/sourcecontrol/overview