Ed-DaSH Workshop

Flic Anderson

18/03/2021

HOW TO (RE-)USE THIS MATERIAL

This is a `.html` presentation created in `R Markdown` with `ioslides`.

(It's been written in a .Rmd file, and we generated .html slides by 'knitting' it in Rstudio.)

You can check out the code used to make these slides at the Workshop repo on Github (and the workshop materials it's based on), and adapt it for your own presentations if you like - I've got a MIT Licence on the repo, which means:

"Basically, you can do whatever you want as long as you include the original copyright and license notice in any copy of the software/source."

Source: tl;drLegal

Things I wish I'd known sooner about collaborating using Git & GitHub

Outline:

- Intro to Git, GitHub
- The Git Workflow
- Issue Tickets
- Branching
- Pull Requests (PRs)
- Helpful Materials
- Questions

Intro:

Git

Git is a *version control system*.

- It doesn't copy your files, as much as keep track of all `changes` ever since you told git to `add` the file to its logs
- Best for `text` files; can't see 'into' binaries (e.g. pdfs, pics)
- Saves the changes you tell it to `commit` to its log in project-specific folders referred to as a repository, or `repo`
- Very useful: undo/redo, backups, switching computers, safely 'experimenting'

It can be used via *Command Line* (the terminal!) or via a *Graphical User Interface* (GUI), on pretty much all systems.

GitHub

GitHub is an *online platform* for storing your git repository. There are others (e.g. GitLab, Bitbucket).

- Keep track of all your repos online - `public` or `private`
- You can tell git to `push` its list of your file changes up to GitHub.
- Or `pull` changes down (yours or someone else's!) and `merge` them with your local copies.
- Contribute to others' work via a `pull request`

Git Workflow

(Fork)

(Clone)

(Branch)

Edit

Add

Commit

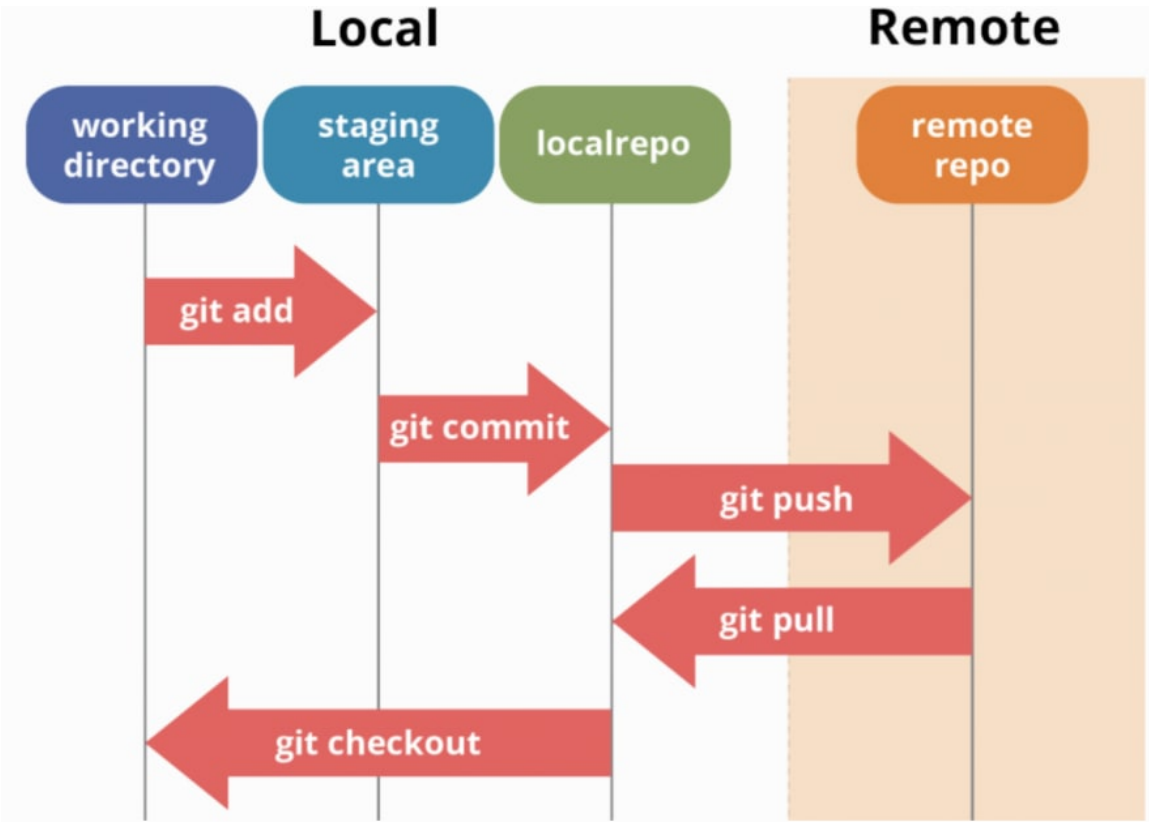
Push

Pull Request

Merge

Git Workflow

Git commands move file-change info between tracking states, locally and remotely.



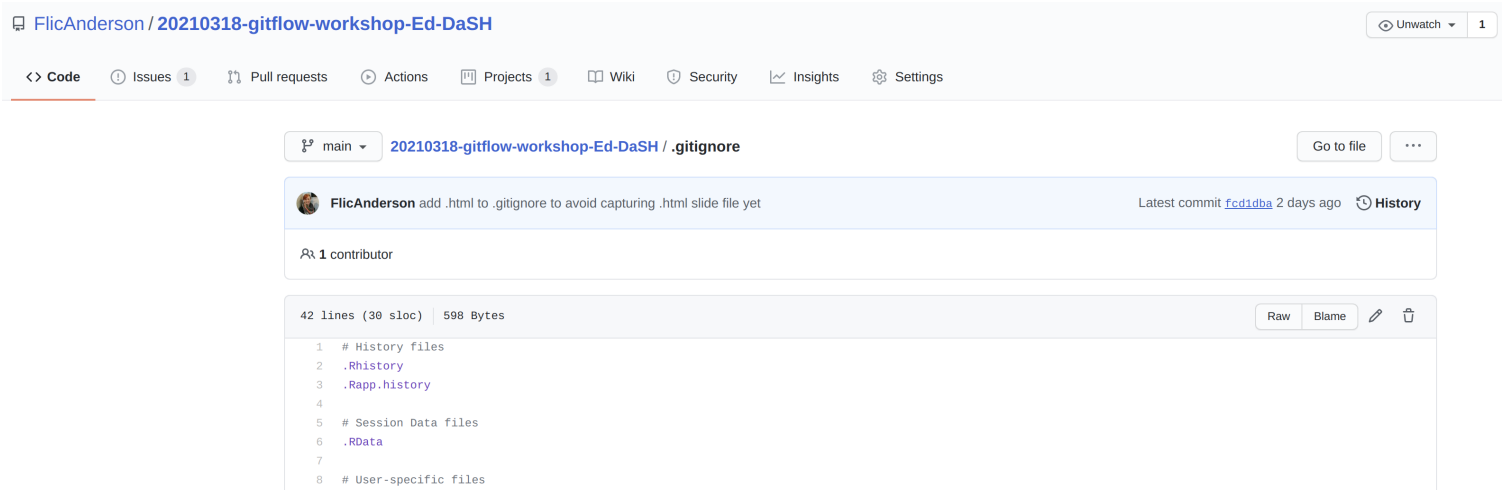
Commits

Commits are the way in which we save a *snapshot of the changes* in each file our git repository tracks

Think of it like a video-game 'save': save at the end of each level

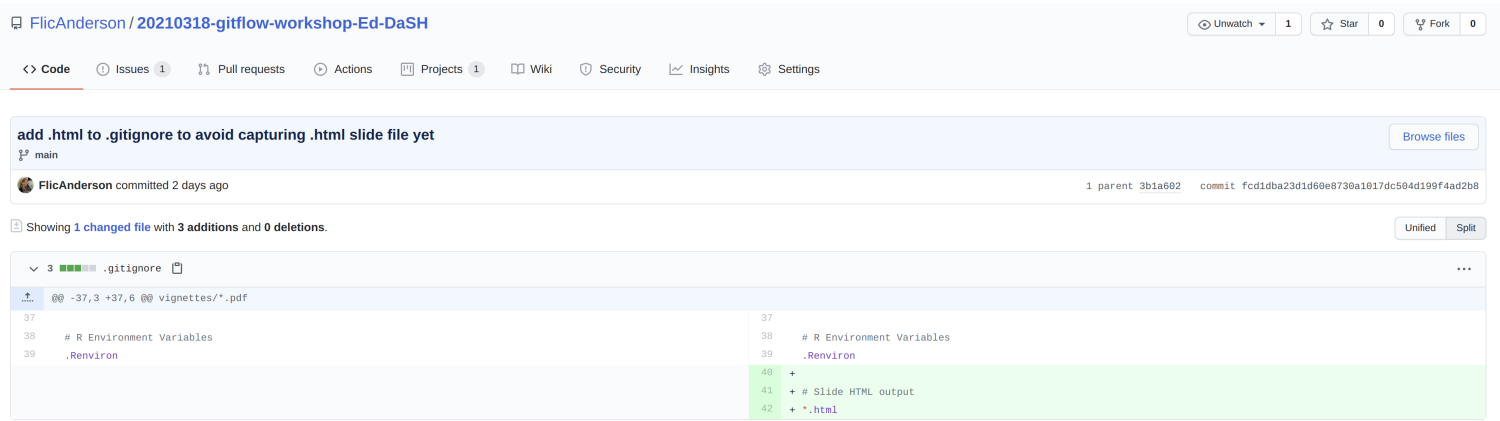
Don't commit everything - *no big files!* - add things to `.gitignore` file

Each commit gets a 'hash' which lets you refer to specific commits (e.g. `fcd1dba`)



Commits

View side-by-side changes included in commits on GitHub:



How To Write A Good Commit Message

```
git commit -m "..."
```

Be concise & consistent!

- WHAT did you do (e.g. fix, rework, add, update)
- WHERE (e.g. filename, function name, section of paper, variable)

```
git commit -m "add resources section to 'README.md'"
```

Issue Tickets

Issue Tickets

- GitHub tool: create ‘issues’ within a repo, where you can add notes.
- markdown format - can include code snippets, formatting, links, emojis!
- @-tag in other GitHub users (“@FlicAnderson, did you get this 404 error before?”)
- assign labels to issues like “help wanted”, “bug”, “question”.
- link issues with pull requests, or keep them separate.
- once the issue is solved, you can mark it as closed!

Issue Tickets

Anatomy of an Issue Ticket: *Example Issue #1*

No issues for this repo yet - need one for demo-ing #1

Edit | New issue

Open **FlicAnderson** opened this issue now · 0 comments

FlicAnderson commented now — Owner

As this is a new repository for my slides, there are **no issues currently in this repository**. No bugs, no nothing! 🐛 🚫

This is unhelpful as I need an issue to demonstrate with!

I (**@FlicAnderson**), have decided to create an issue ticket to resolve this issue!

FlicAnderson added the good first issue label now

FlicAnderson self-assigned this now

Write | Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Close issue | Comment

Remember, contributions to this repository should follow our GitHub Community Guidelines.

Assignees
FlicAnderson

Labels
good first issue

Projects
None yet

Milestone
No milestone

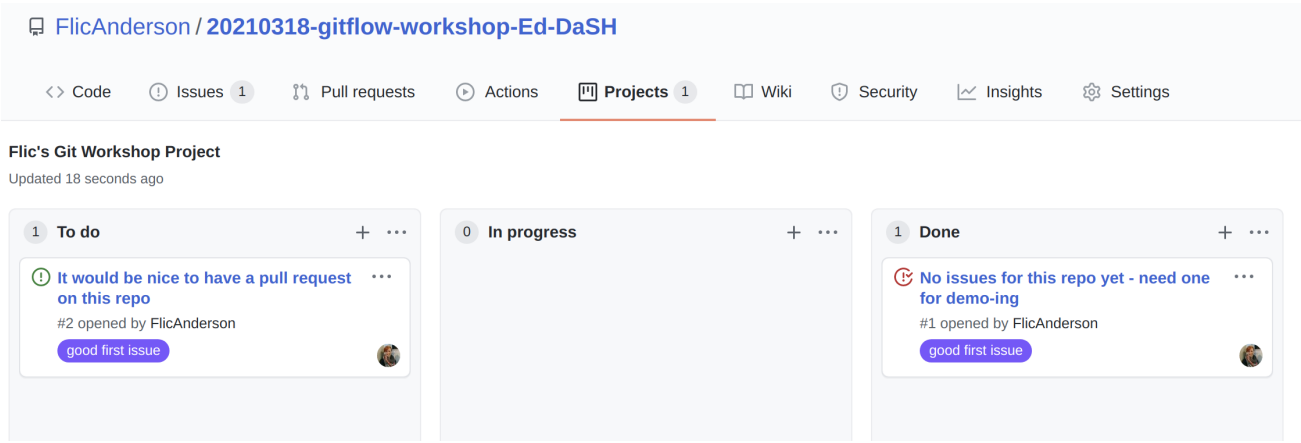
Linked pull requests
Successfully merging a pull request may close this issue.
None yet

Notifications Customize
Unsubscribe
You're receiving notifications because you're watching this repository.

1 participant

Issue Tickets

Managing Your Issue Tickets with A Project Kanban Board:



How To Write A Good Issue Ticket

Give a clear and concise description of the problem or new feature to help others get up to speed quickly

- how you found the issue & what you've done already to try fix/investigate it/ideas
- tag in colleagues who might offer insights, or answer any questions
- remember to update the issue regularly as progress happens!
- link to other issues in that repo using the hashtag & issue number e.g. #42
- add the most useful labels / create your own
- check with people before randomly assigning them issues! :)

#DYUTIT? - “Did You Update The Issue Ticket?”

Why is #DYUTIT? a helpful project motto?

AVOID:
* “I didn’t know you’d already fixed that” / “I already spent hours fixing that”
* “I can’t remember what was I trying to do here??”
* “I didn’t think this was a priority for anyone else”

GAIN:
* “oh, hey, I have an idea for this fix”
* “this reminds me of something I think might be related”

Branching

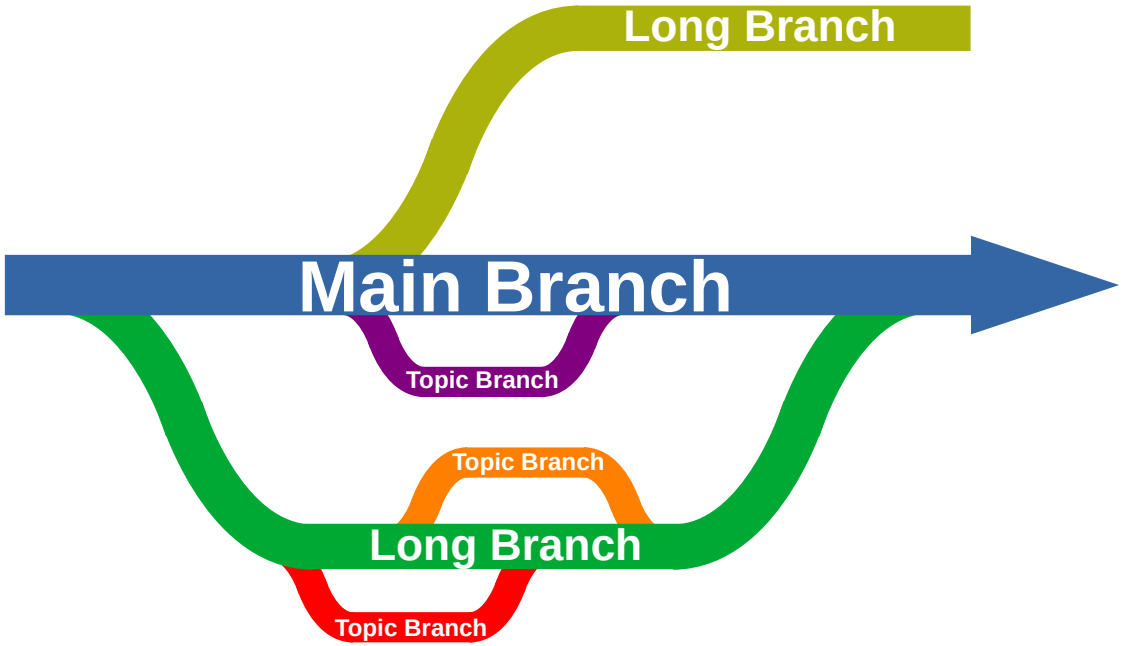
Branching helps bring: good coding practice and good project management.

Separates projects into larger objectives and smaller, manageable tasks.

It also enables you to prototype changes in isolation, minimising risk, and provides checkpoints to quickly return to when things go wrong.

Working on separate branches lets you include fixes and changes others are working on, without discarding your own changes.

Branching


Long Branch
Main Branch
Topic Branch
Topic Branch
Long Branch
Topic Branch

Branching

Commits vs branches

```
# create new branch called "reviewer-changes" & move to that branch
git checkout -b reviewer-changes

# add an edited file to git's version control 'staging' area
git add Article_Manuscript.Rmd

# 'commit' the changes made to this file with a descriptive message
git commit -m "update abstract of manuscript to include study size"
```

Branching

Local vs remote

GitHub is an amazing collaborative tool, but significantly complicates code management.

Branches and commits will appear on the remote repo that are not on your local repo (and vice versa)

Remember to keep pulling down others' work, and keep your local repos up to date

Branching

Where am I?

```
# show local branches:
git branch
```

```
* master
```

```
# show local branches AND the remote branches (on GitHub)
git branch -a
```

```
* master
  remotes/upstream/master
  remotes/upstream/staging
```

Pull Requests (PRs)

Pull Requests are a key feature of working collaboratively with version control.

“Here’s some work I’ve done, please include it with the rest!”

Can link to issue tickets (close both auto-magically!)

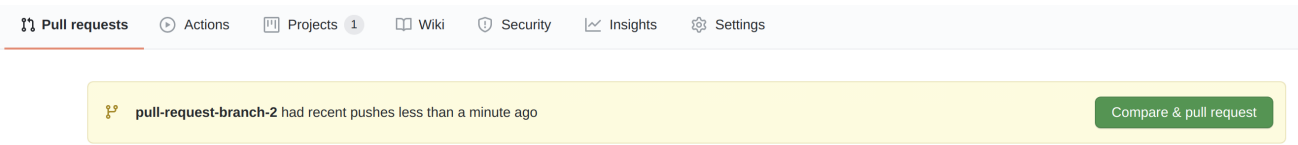
Good opportunity for review

Allow members of the project community to help contribute to the codebase.

Pull Requests (PRs)

When you've made changes, and used `git add` and `git commit` to record them, then used `git push` to push them up to the remote GitHub repository, then you're ready to create a Pull Request.

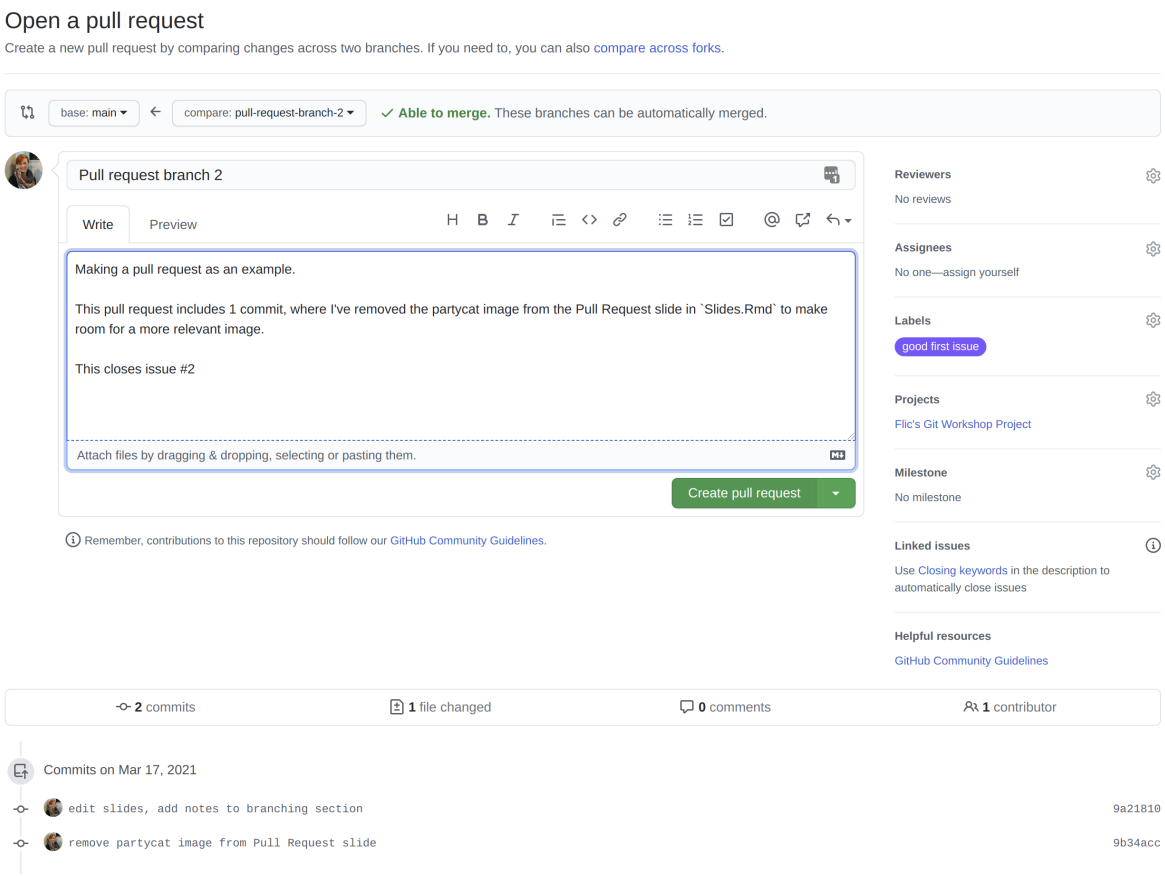
There's often a **"Compare & pull request"** button on the code page for that branch:



Alternatively, find the **"New pull request"** button.

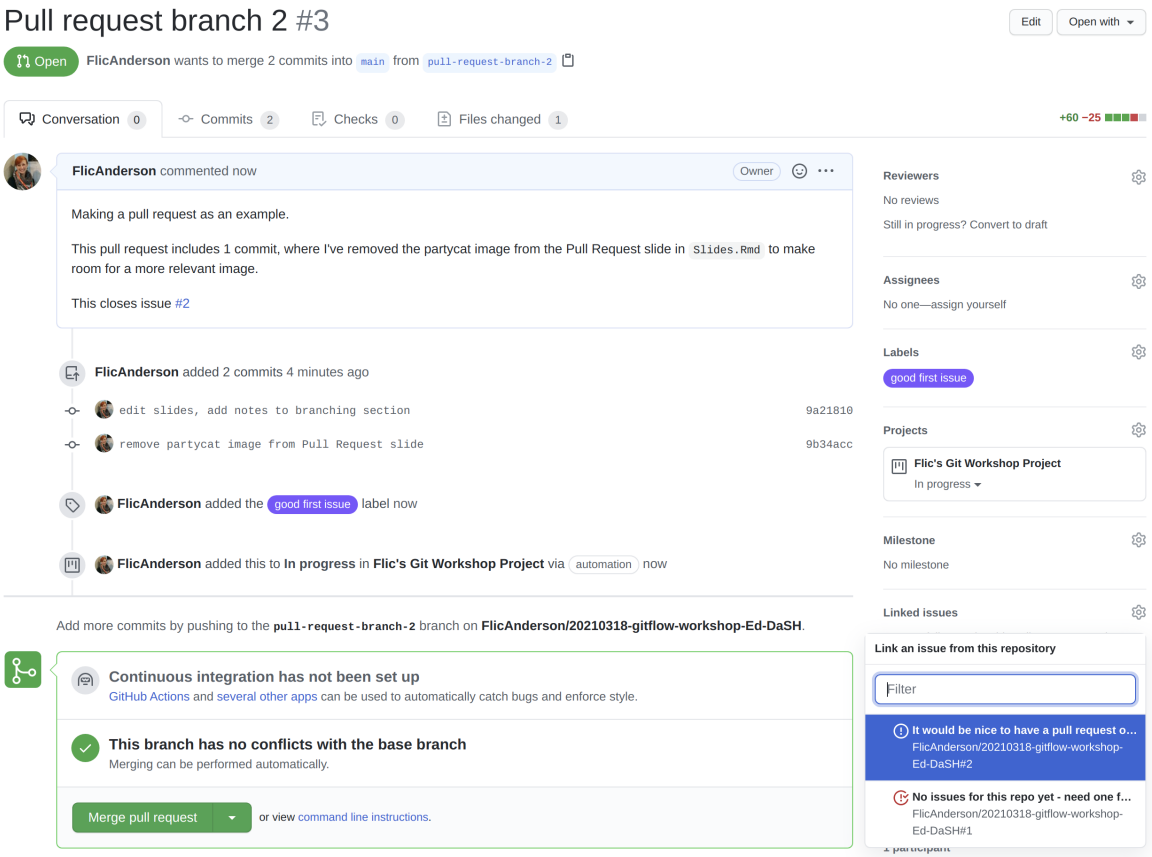
Pull Requests (PRs)

Fill out the PR info & check the branches!



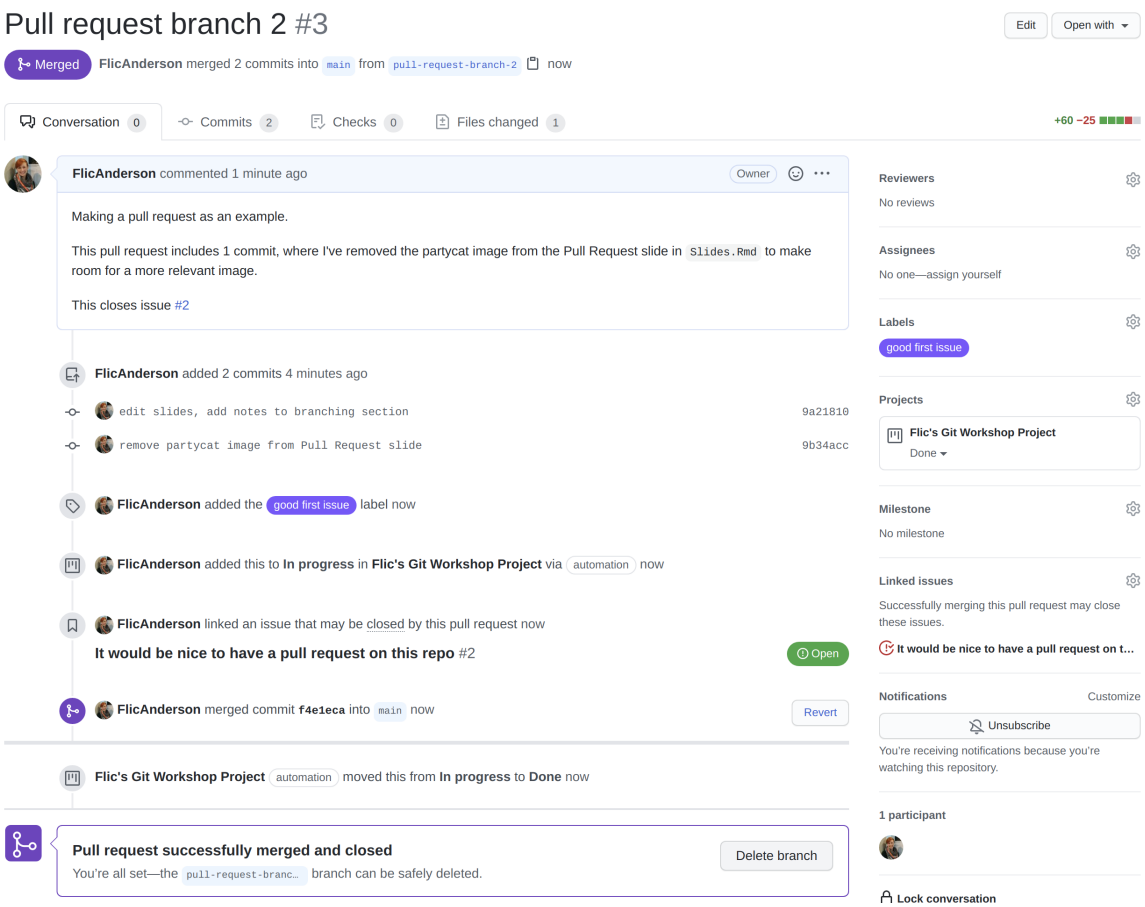
Pull Requests (PRs)

Link related issues to the PR



Pull Requests (PRs)

Press big green **“Merge pull request”** button to complete!



Tips for Success When Collaborating With Git & Github

- Don’t commit unfinished code or merge incomplete branches
- Use `git pull` regularly before starting any new work locally (avoids conflicts later)
- Check which branch you’re on frequently with `git status` or `git branch`
- Keep issue/feature branches short & focused, and delete merged branches often
- Make your messages & notes helpful and informative

Helpful guides

Git cheatsheet

https://education.github.com/git-cheat-sheet-education.pdf

Git documentation

https://git-scm.com/doc

GitHub Guides

https://guides.github.com/

Software Carpentry material

Version Control With Git

Questions?

Troubleshooting

Troubleshooting

Staying Up To Date

Run `git pull` regularly to make sure you've got any newly merged or edited content from the remote branches.

Luckily, git sometimes tells you if there are new commits:

```
fanders6@ceres:~/20210318-gitflow-workshop-Ed-DaSH$ git checkout main
Switched to branch 'main'
Your branch is behind 'origin/main' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)
fanders6@ceres:~/20210318-gitflow-workshop-Ed-DaSH$ git pull
Updating 9a21810..f4e1eca
Fast-forward
 Slides.Rmd | 10 +++++-----
 1 file changed, 5 insertions(+), 5 deletions(-)
```

Troubleshooting

`.gitignore`

Edit this file to include files you don't want git to track.

Consider adding:
* big files
* temporary files
* code-generated output (e.g. contents of `/Results` folder)
* system-specific files (ie Mac/Windows-only files) or system files (e.g. `Thumbs.db`)

```
# in .gitignore:

# Slide HTML output
*.html
```

Troubleshooting

Merge locally to avoid conflicts in PRs

```
# move to local copy of 'main' branch
git checkout main

# get latest changes from GitHub into our local repo
git pull

# move across to our `reviewer-changes` branch where our edits are
git checkout reviewer-changes
```

Troubleshooting

Merge locally to avoid conflicts in PRs

```
# merge up-to-date 'main' into our 'reviewer-changes' branch
git merge main

# manually fix any conflicts by removing merge tags

# push updated 'reviewer-changes' branch to GitHub
git push

# switch to the repo on GitHub in your browser
# & create a Pull Request via the UI
```

Troubleshooting

Merge Conflicts

Fix a merge conflict by manually selecting which lines to keep and which to ignore by editing the relevant files:

```
Auto-merging Article_Manuscript.Rmd
CONFLICT (content): Merge conflict in Article_Manuscript.Rmd
Automatic merge failed; fix conflicts and then commit the result.
```

Troubleshooting

Merge Conflicts

Open file `Article_Manuscript.Rmd`, and scroll to find `>>>>>>>` tags like this:

Article_Manuscript.Rmd

```
Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Various abstract text info of significant interest right here.
<<<<<<< HEAD
This is the abstract content on the main branch. No study size details.
=======
This is an edit on the reviewer-changes branch. Study size was 42.
>>>>>>> reviewer-changes
```

Troubleshooting

Merge Conflicts

Delete the version you don't want to keep (ie in this case from the main branch):

Article_Manuscript.Rmd

```
Abstract

Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Various abstract text info of
significant interest right here.
This is an edit on the reviewer-changes branch.
Study size was 42.

```

Then `git add` and `git commit` the changes - conflict solved!

Troubleshooting

Stashing

```
git stash [push]
```

Reverts the code back to the last commit and saves changes in a temp file.

Perfect for when you realise you're working in the wrong branch.

Swap to right branch and pop the changes back

```
git checkout other_branch
git stash pop
```

Troubleshooting

Reverting and cherrypicking

```
git revert e8cc44d3ce4b01fb2d211bffec0c69cbaa0d80f4
```

Eventually you will accidentally commit a bug into your branch and need to remove it.

```
git cherry-pick e8cc44d3ce4b01fb2d211bffec0c69cbaa0d80f4
```

Some cases may require you to select exactly which commits to merge (rather than a whole branch)