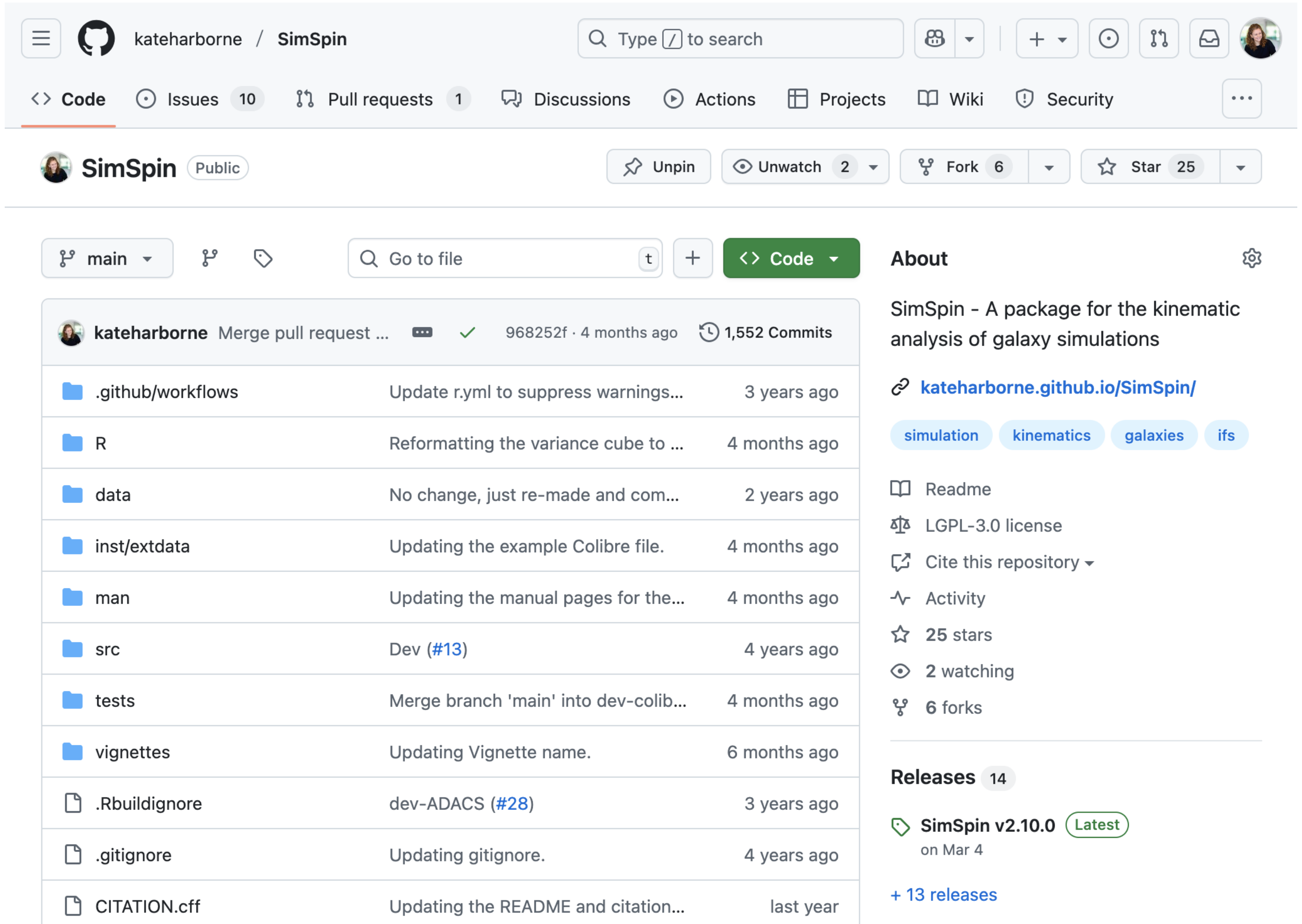
Version management with git for collaborative coding

Dr Kate Harborne
ICC, Durham University
katherine.e.harborne@durham.ac.uk

Outline

- My background
- Why use git for collaborative coding?
- Some quick definitions
- Get some practice within project teams…

My background



kateharborne / SimSpin

SimSpin Public

kateharborne Merge pull request ... 968252f · 4 months ago 1,552 Commits

.github/workflows	Update r.yml to suppress warnings...	3 years ago
R	Reformatting the variance cube to ...	4 months ago
data	No change, just re-made and com...	2 years ago
inst/extdata	Updating the example Colibre file.	4 months ago
man	Updating the manual pages for the...	4 months ago
src	Dev (#13)	4 years ago
tests	Merge branch 'main' into dev-colib...	4 months ago
vignettes	Updating Vignette name.	6 months ago
.Rbuildignore	dev-ADACS (#28)	3 years ago
.gitignore	Updating gitignore.	4 years ago
CITATION.cff	Updating the README and citation...	last year

About

SimSpin - A package for the kinematic analysis of galaxy simulations

kateharborne.github.io/SimSpin/

simulation kinematics galaxies ifs

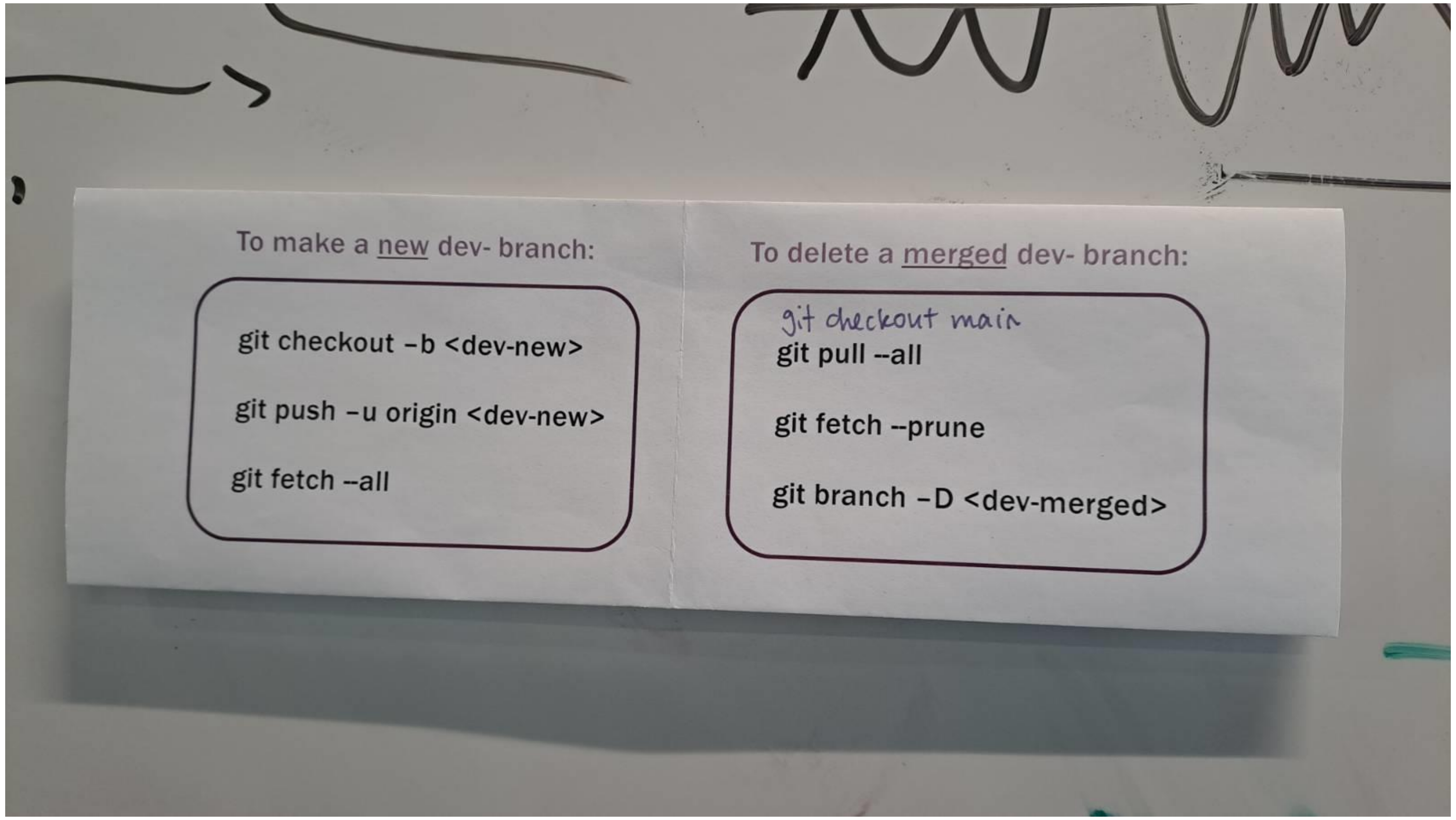
Readme
LGPL-3.0 license
Cite this repository
Activity
25 stars
2 watching
6 forks

Releases 14

SimSpin v2.10.0 Latest
on Mar 4

+ 13 releases

MY BACKGROUND



To make a new dev- branch:

```
git checkout –b <dev-new>

git push –u origin <dev-new>

git fetch --all
```

To delete a merged dev- branch:

```
git checkout main
git pull --all

git fetch --prune

git branch –D <dev-merged>
```

Why use git?

Functionality

- Make backups
- See previous versions
- Mark “releases”
- Test code

Access

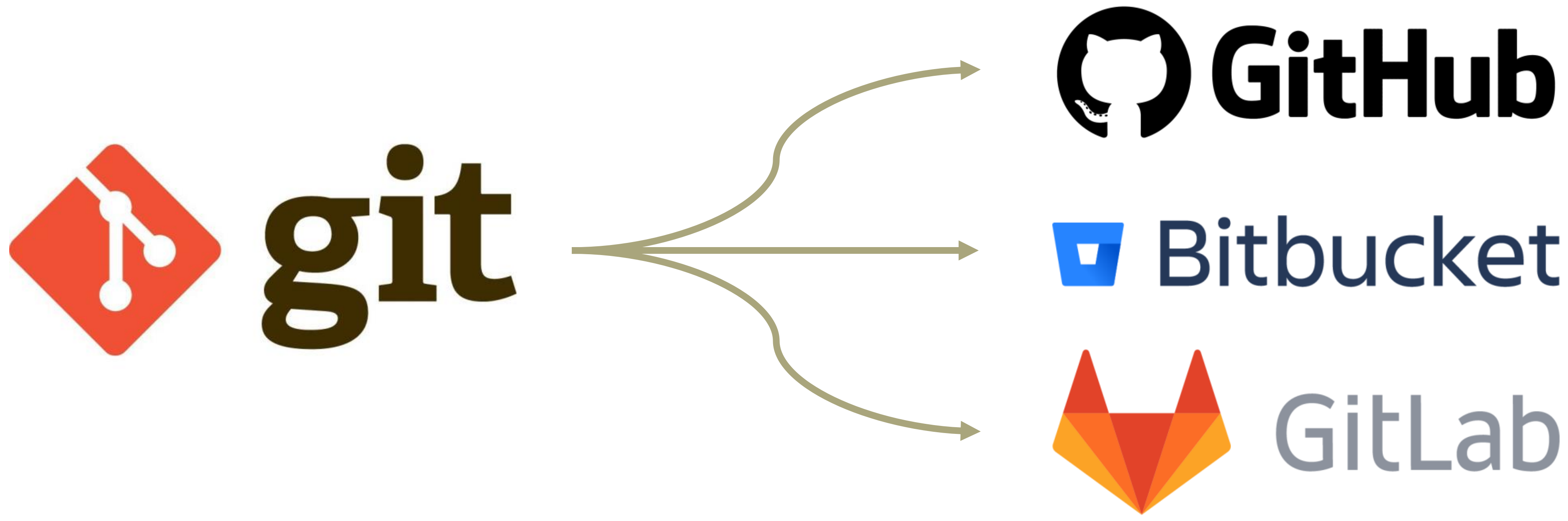
- Access from anywhere
- Synchronise across computers

Collaborate

- Allow others to use your code
- Allow others to changes

Tool

Services



Other tools also exist.
(Mercurial, CVS, subversion, etc.)

Some definitions – To start

https://www.atlassian.com/git

GitHub

Repository ("repo")

ORIGIN

Let’s make a repo for each team project:

- If you don’t already have a GitHub account, go and set one up here.
- If you have not used GitHub in the past, you may need to add an SSH key for authentication and signing of commits: see here to set one up, and here to generate a new key if you need one.
- Only one person needs to initialise the repository in each group.
- Each group member needs to be added as a collaborator on the project to gain push permissions to the repo.

Some definitions – To start

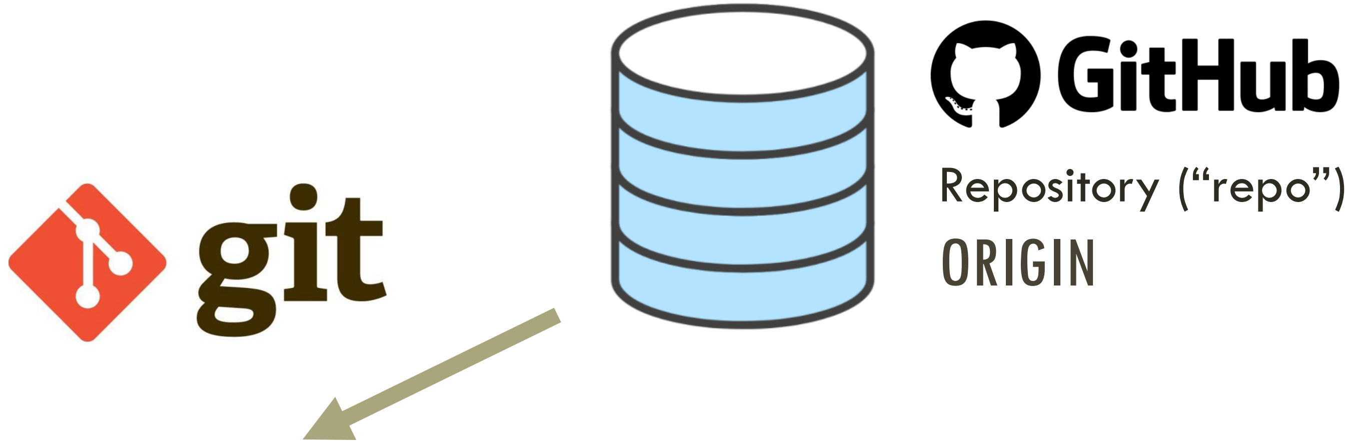
https://www.atlassian.com/git

GitHub

Repository ("repo")

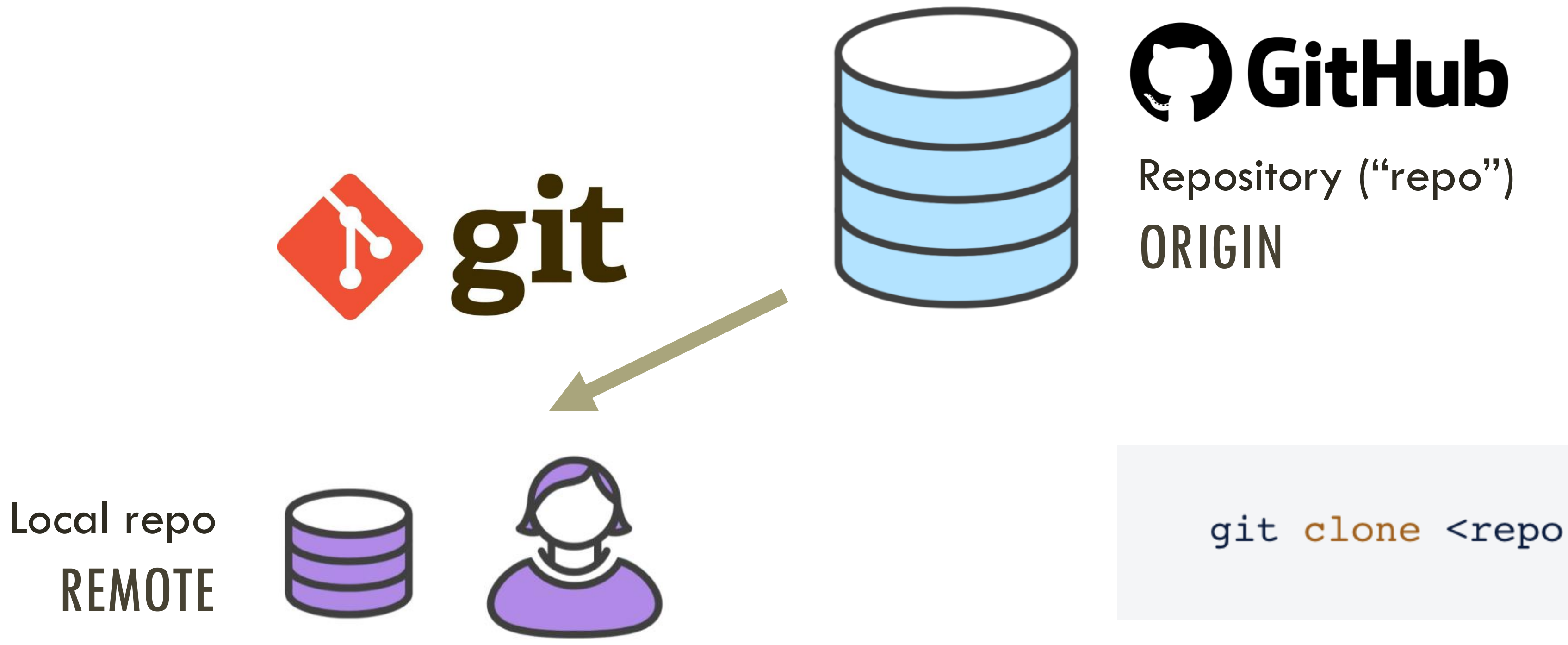
ORIGIN

Some definitions – To start



```
git clone <repo url>
```

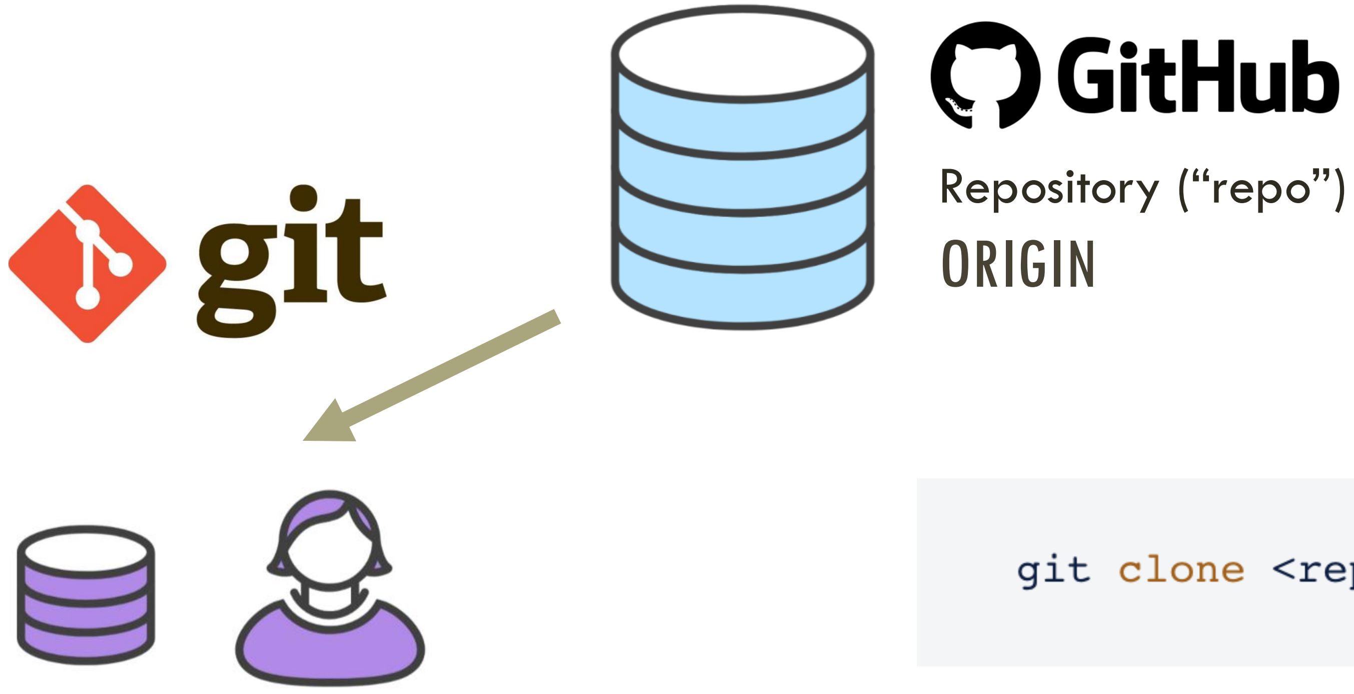
Some definitions – To start



```
git clone <repo url>
```

Some definitions – To start

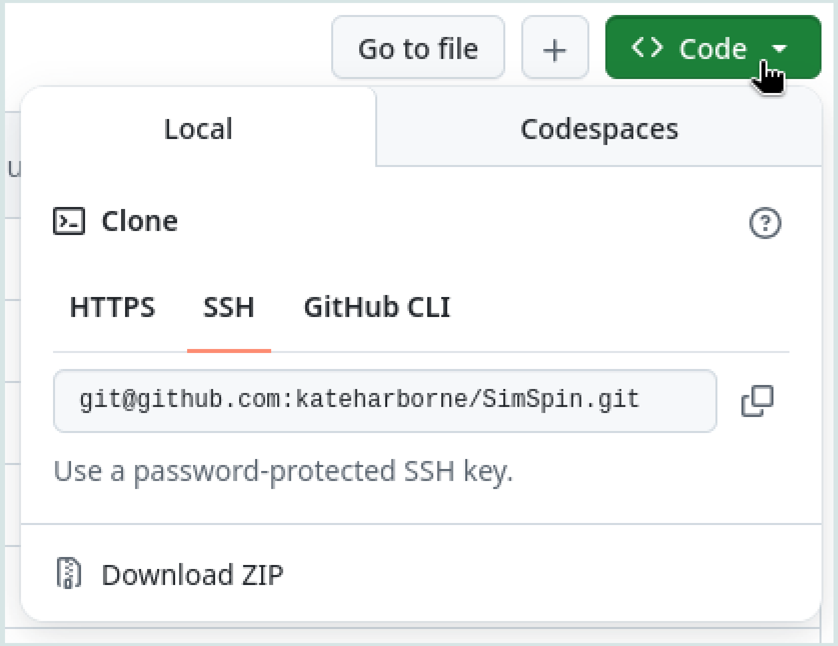
I like to keep all my repos in a particular directory locally (e.g. <home/repos/>)



```
git clone <repo url>
```

Clone a local copy of your group's repo

- Decide on a location to store your repository files.
- Open a terminal.
- Change to the directory where you wish to store the repo.
- Use the git clone command as given on GitHub

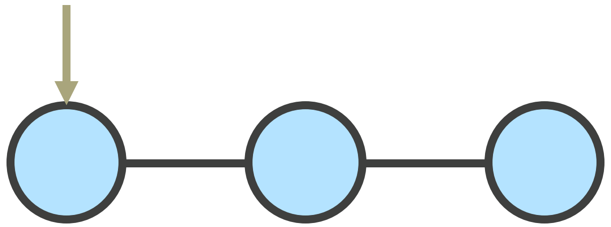


```
git@github.com:kateharborne/SimSpin.git
```

Use a password-protected SSH key.

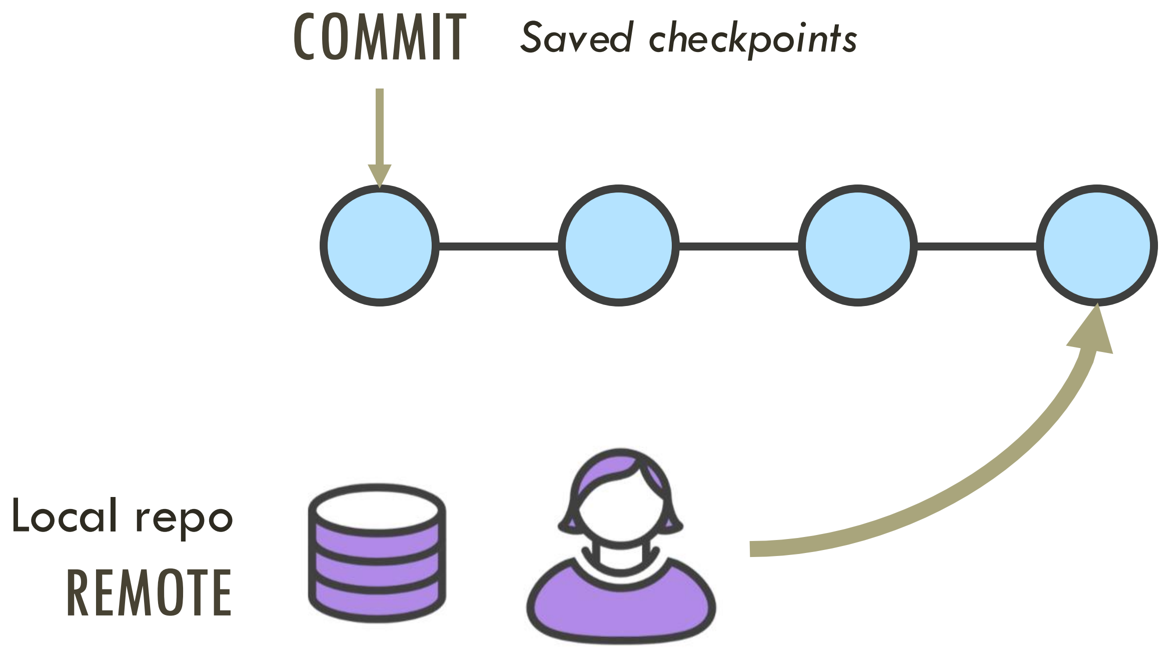
Some definitions – To “SAVE” a change

COMMIT *Saved checkpoints*

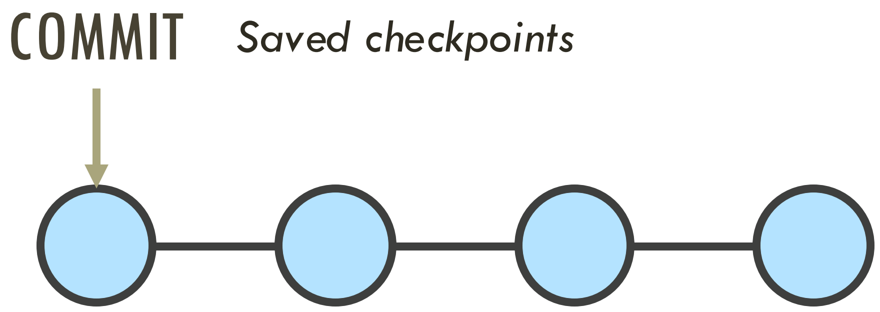


Local repo
REMOTE

Some definitions – To "SAVE" a change



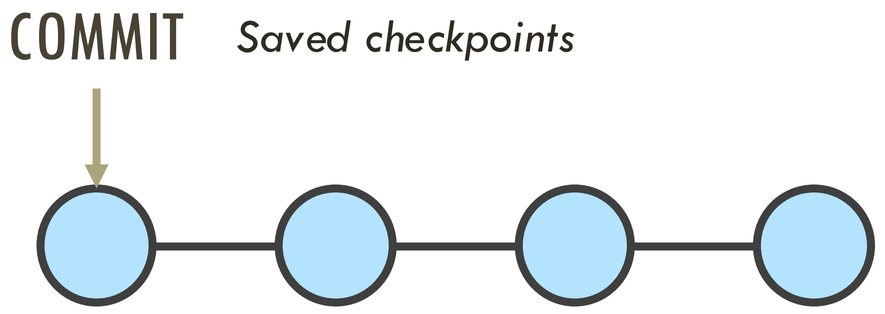
Some definitions – To "SAVE" a change



Local repo
REMOTE

1)
```
git add hello.py
```

Some definitions – To “SAVE” a change



Local repo

REMOTE

1)

```
git add hello.py
```

2)

```
git commit
```

+ add a commit message in the following pop-up

OR

```
git commit -m "commit message"
```

Some tips for meaningful commit messages

Commit messages should be short summaries of what you've changed. It's very tempting to make all commit messages `git commit -m "Bug fix"`

But future you will not be happy when trying to find the point in history you made a particular change. The aim is to be able to revisit these messages and pinpoint when you adjusted something in your code.

It's good to try and have a convention for how you intend to format your commits, e.g. see here.

```
<type>[optional scope]: <description>

[optional body]

[optional footer(s)]
```

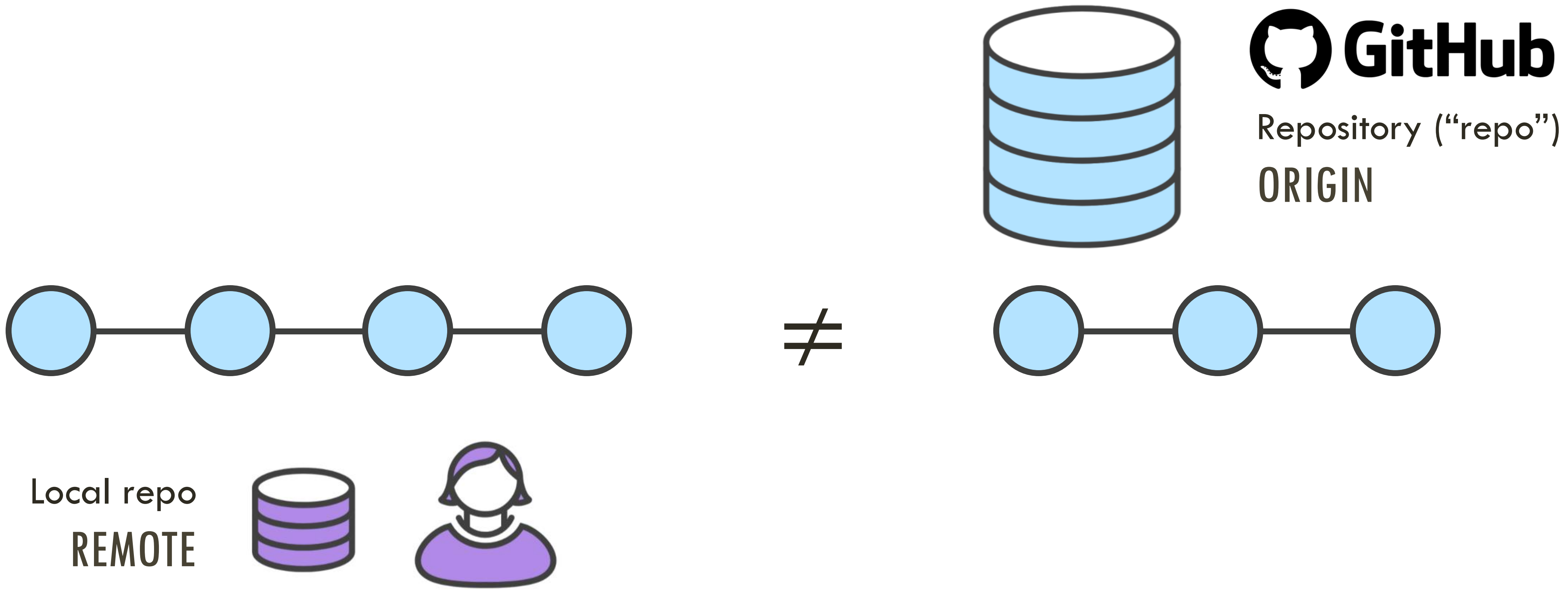
- <type> = `fix`, `feat`, `docs`, `style`, `test`, etc.
- <description> = descriptive title of the commit (but all needs to be < 100 characters).
- Body can include a more detailed breakdown.
- Footers should include any *issues* that the commit addresses.

Make a change to some files in the repo:

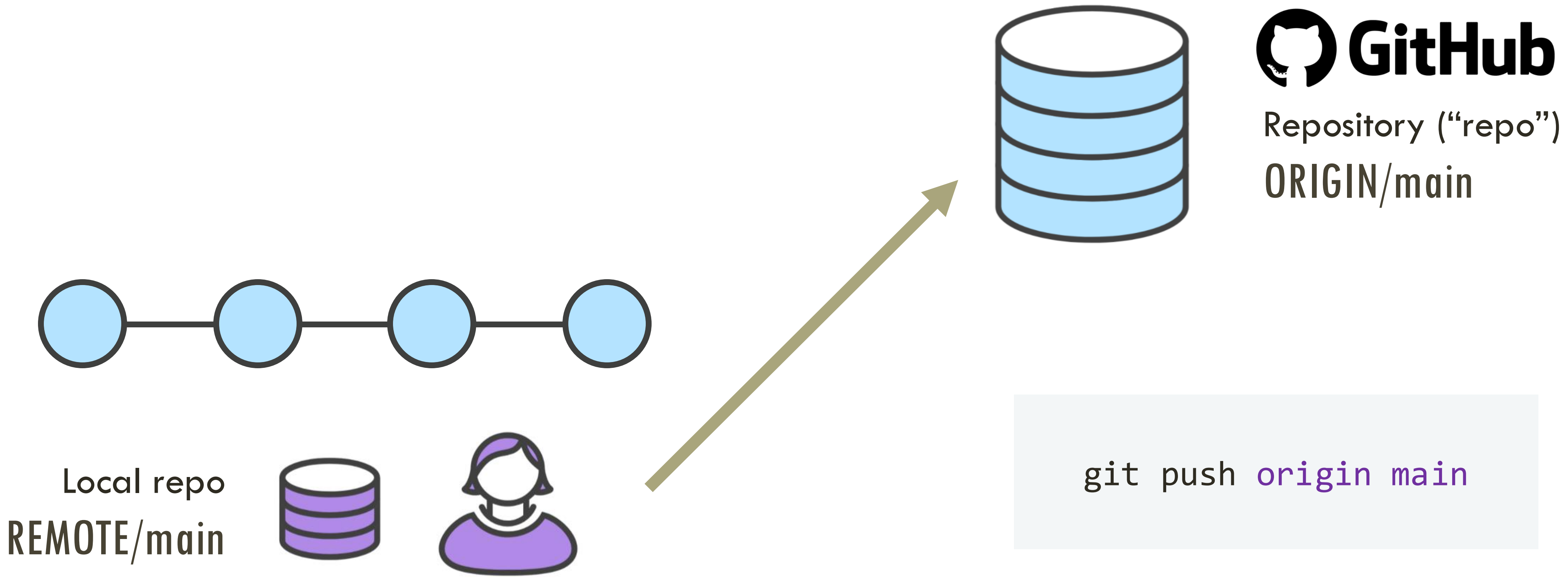
Nominate someone to:

1. Add some text to the `README.md` describing the project.
2. Set up a `NEWS.md` file for recording repo updates.
3. Generate a `pyproject.toml` with the basic headings and version number.
4. Build a `.readthedocs.yaml` file for initialisation of documentation.
5. Add a `CODEOFCONDUCT.md` file as following the PSF/Astropy example.
6. Add some contribution guidelines in a `CONTRIBUTING.md` file with a basic outline.

Any remaining group members? Try also editing the README.md…


GitHub
Repository ("repo")
ORIGIN
≠
Local repo
REMOTE

Some definitions – To "SAVE" a change



```
git push origin main
```

But this all assumes you are the only contributor to the code and that you have no external users who may be affected by your update.

It’s easy to imagine that when other people are also collaborating on a repository, a change may have been made to that `main` branch before you manage to `push` your change.

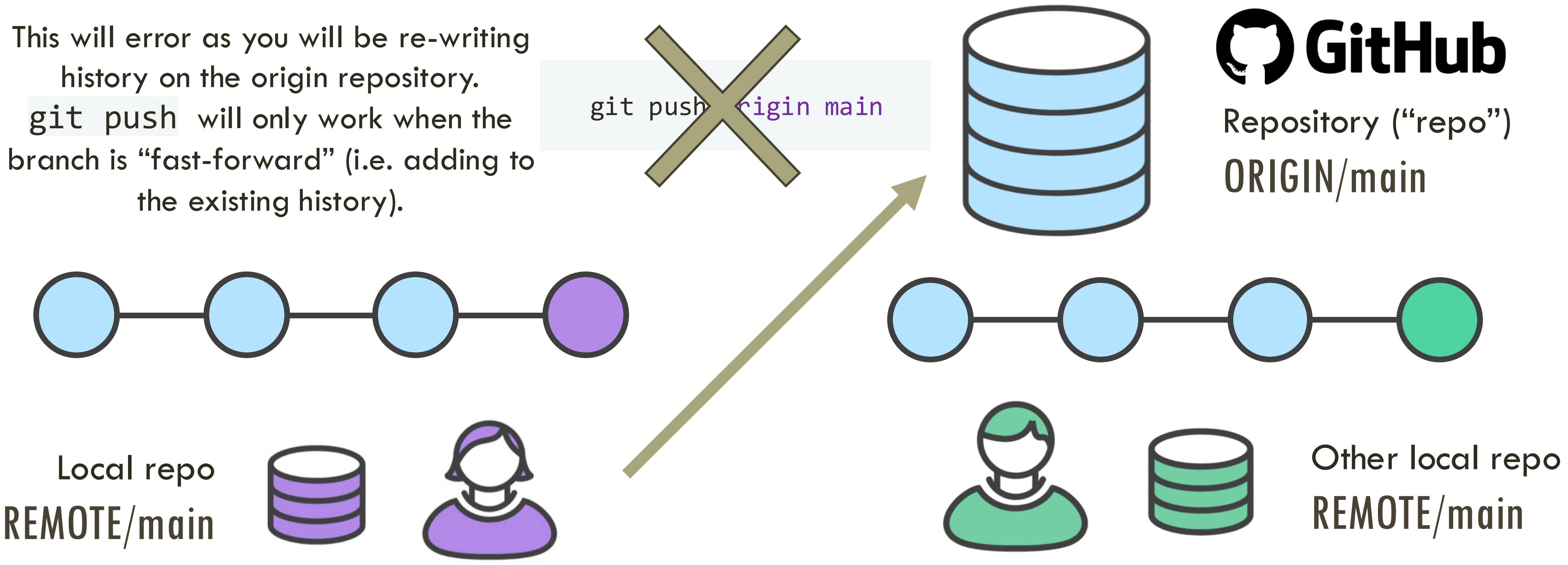
Or that your `push` may work for you - but may break other user’s code.

So collaborative coding on public repositories with git requires a few extra procedures.

Some definitions – To "SAVE" a change

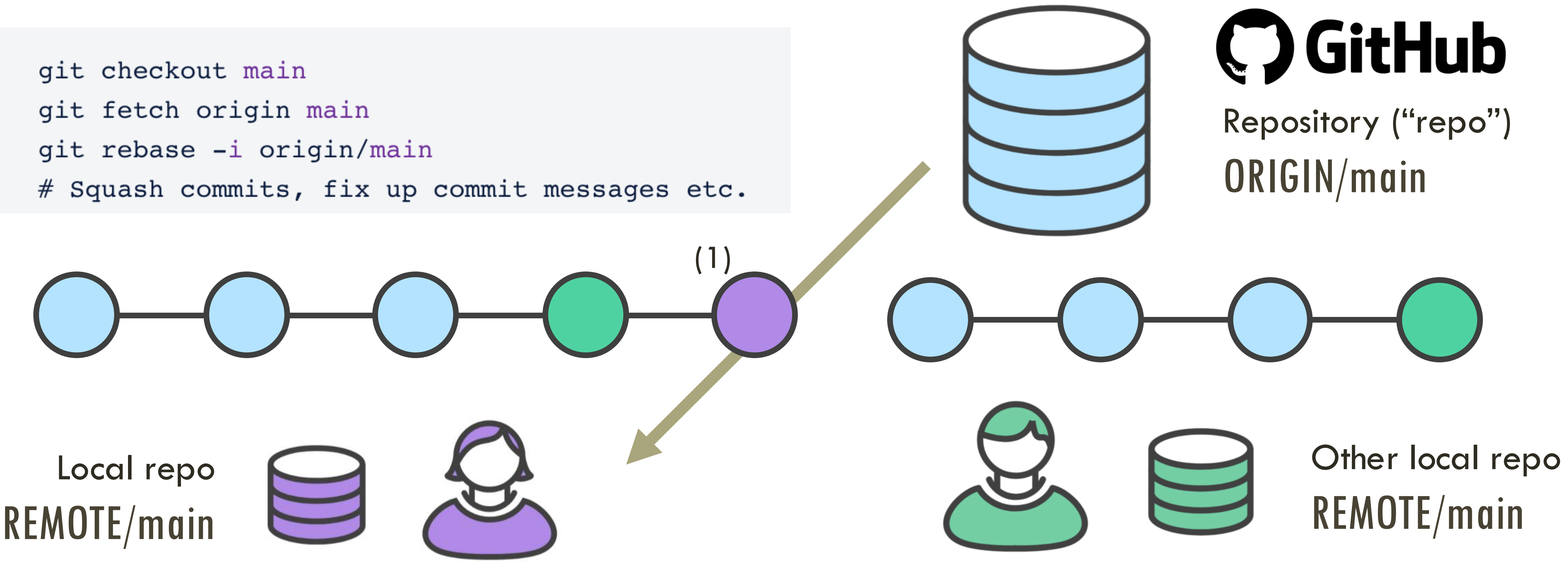
This will error as you will be re-writing history on the origin repository. `git push` will only work when the branch is "fast-forward" (i.e. adding to the existing history).

~~`git push origin main`~~



Some definitions – To "SAVE" a change

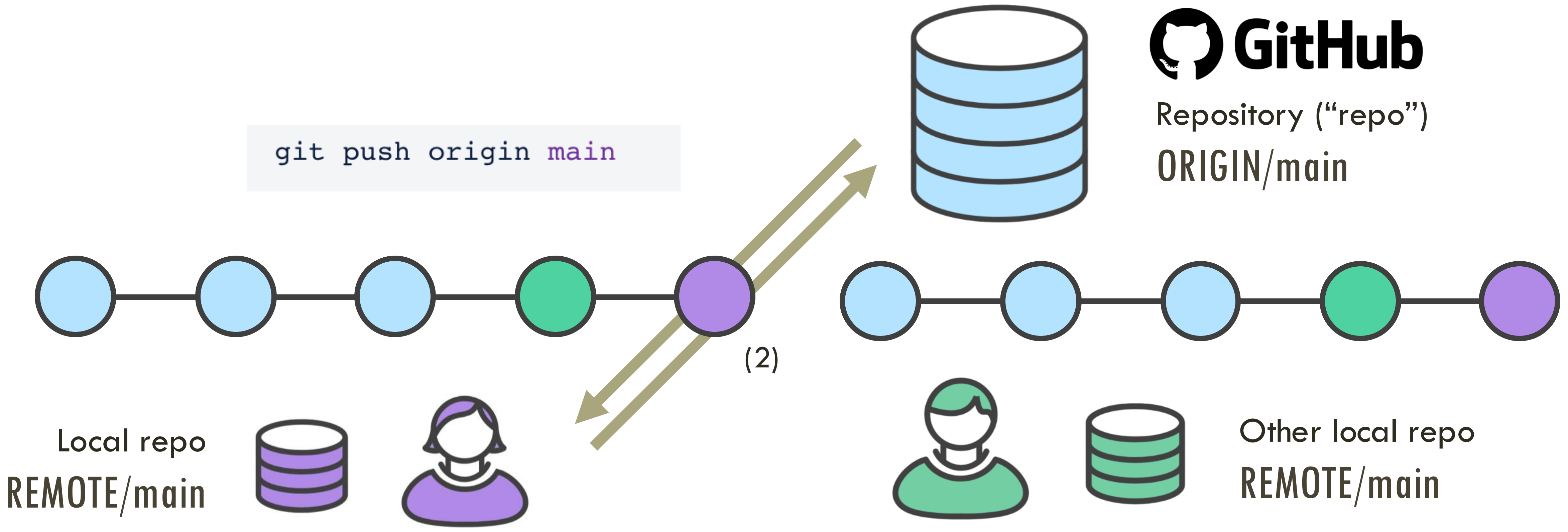
```
git checkout main
git fetch origin main
git rebase -i origin/main
# Squash commits, fix up commit messages etc.
```



Note, it is considered bad practice to re-write public history with rebase commands. For more details, see here.

Some definitions – To "SAVE" a change

```
git push origin main
```

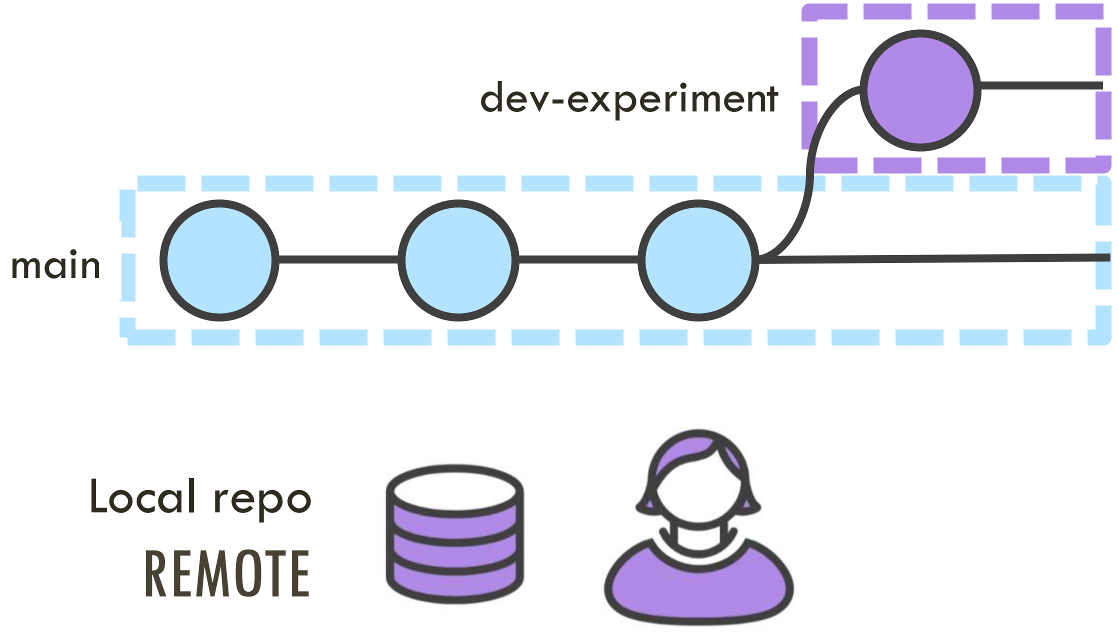


You will have to `pull` before you `push` changes, ensuring your additions are made to the end of the branch of commits. It helps to run a `pull` BEFORE you begin making changes locally.

But this doesn’t prevent you breaking existing code. Currently, we have no checks or procedures in place to avoid this.

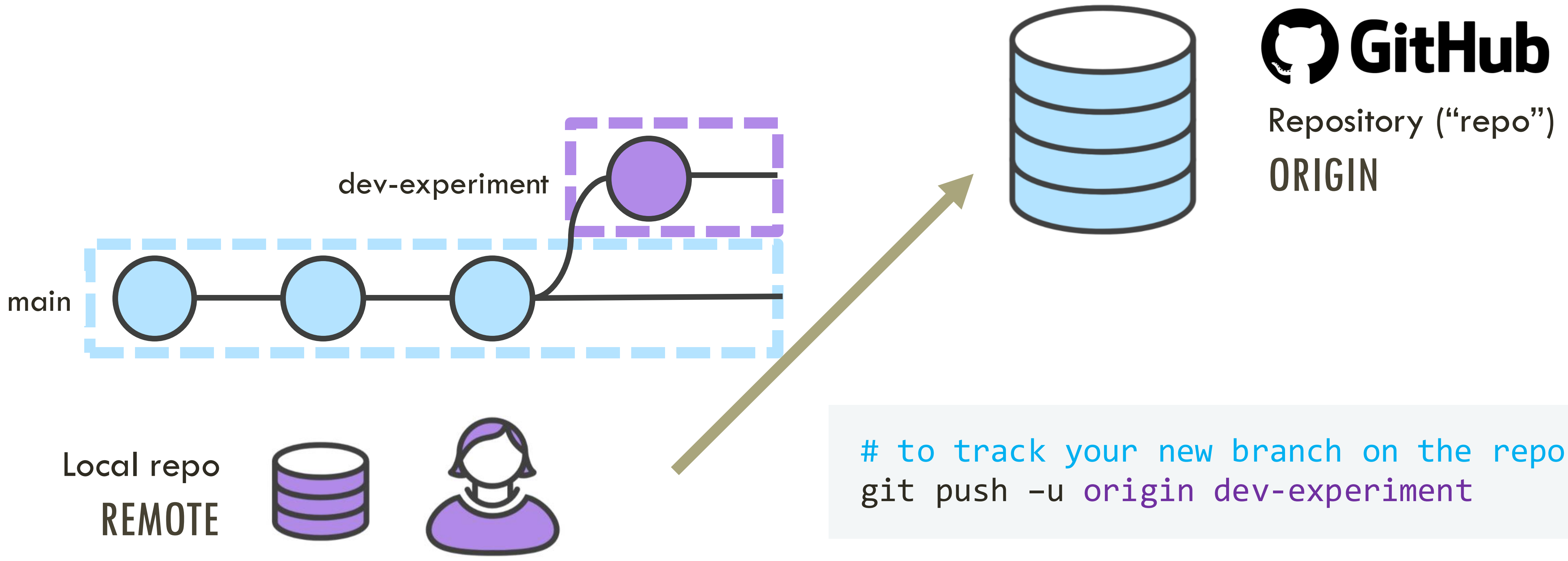
Git has some clever methods in place to help this, but we also need some other tools for performing tests that check our code will work as expected.

Some definitions – Making a "BRANCH"



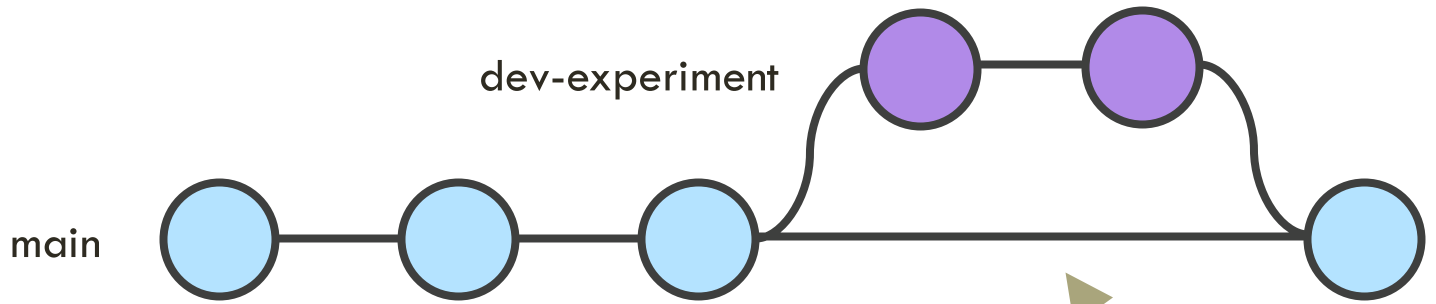
```
# where do you want to branch from?
git checkout main

# make a new branch, but you're still on main
git branch dev-experiment

# now on new branch
git checkout dev-experiment
```

Some definitions – Making a “BRANCH”



```
# to track your new branch on the repo
git push -u origin dev-experiment
```

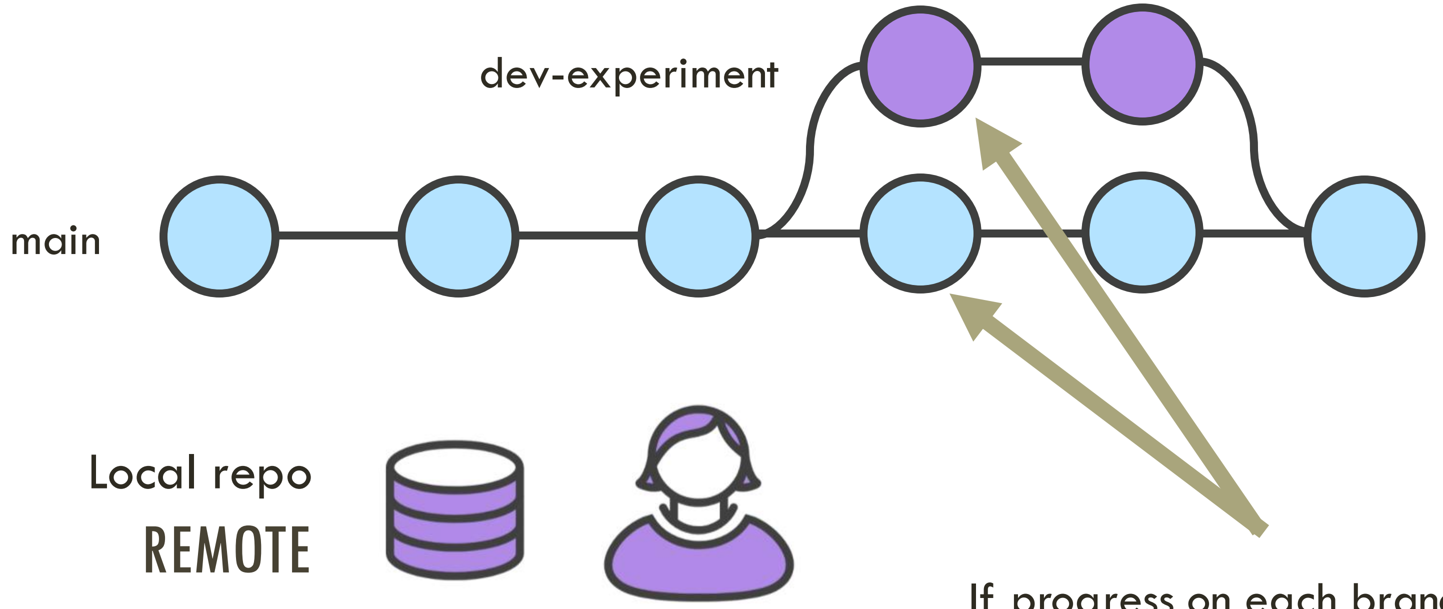
Some definitions – Merging a <u>local</u> "BRANCH"



```
# go back to main
git checkout main

# merge dev-experiment into main
git merge dev-experiment

# delete dev-experiment branch
git branch –d dev-experiment
```

This will work so long as no changes have been made to the main branch in the meantime (i.e. all changes are "fast-forward").

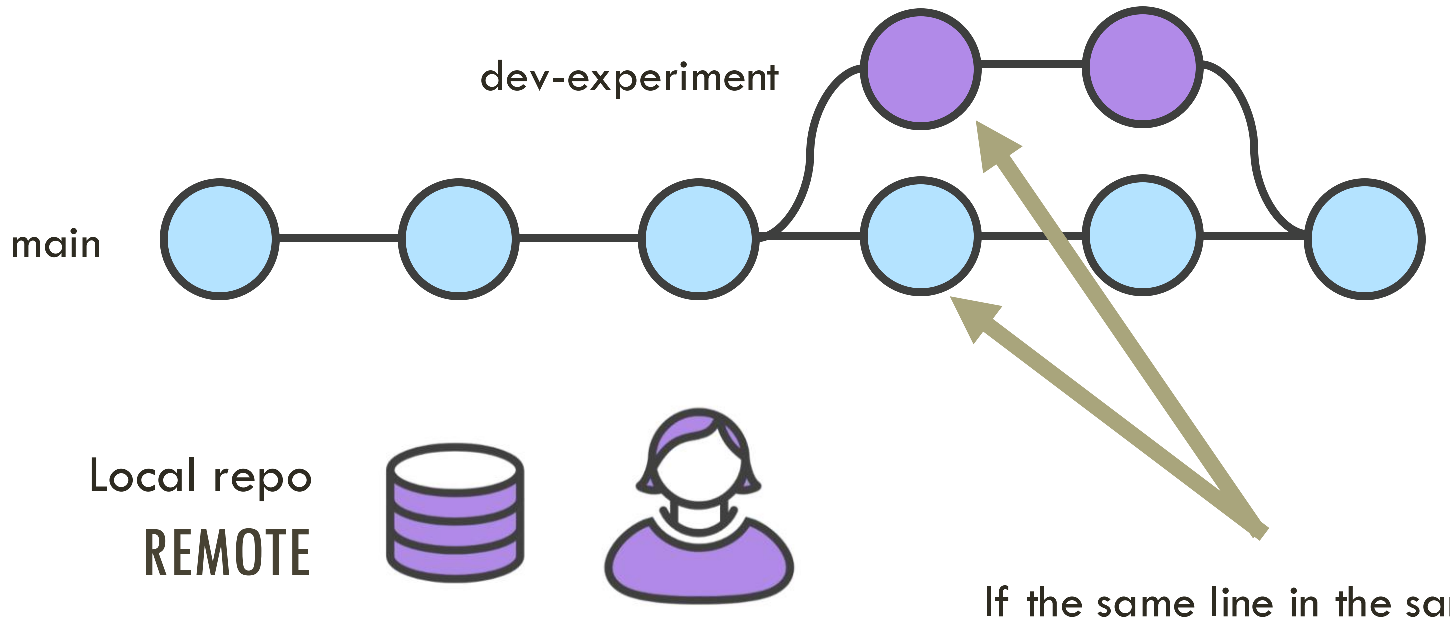
Some definitions – Merging a <u>local</u> "BRANCH"



```
# go back to main
git checkout main

# merge dev-experiment into main
git merge dev-experiment

# delete dev-experiment branch
git branch –d dev-experiment
```

If progress on each branch is made to independent files within the repo, this will also work.

Some definitions – Merging a local "BRANCH"



```
# go back to main
git checkout main

# merge dev-experiment into main
git merge dev-experiment

# resolve conflicts before
#   proceeding
```

If the same line in the same file is changed on each branch, you will have to tell git which version of the file you wish to keep.
For more on resolving merge conflicts, see here.

Version management locally

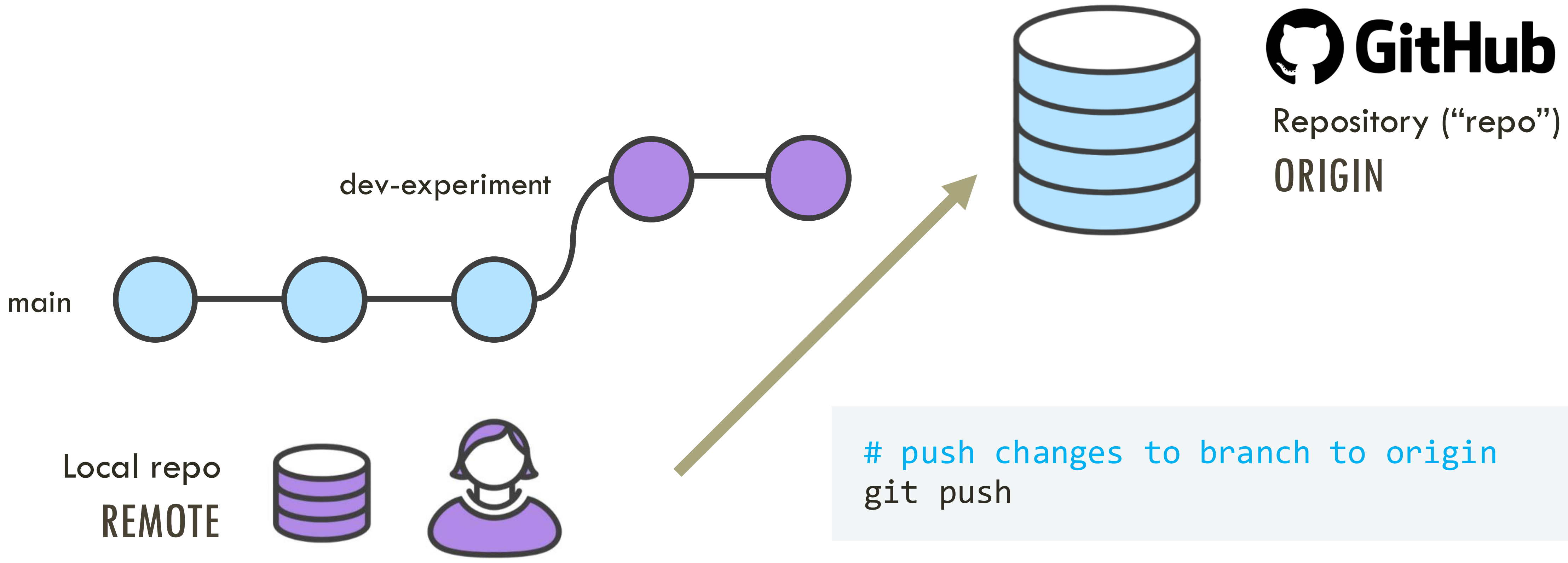
1. When working on a change to the code, make a new branch rather than working directly on `main`.
2. `pull` before making changes – be sure you are updating the latest development.
3. Ready to merge your branch into `main`? Run your unit tests, install checks, etc. first.
4. Found a bug? Write a test.
5. Merge into `main` and push to `origin`.

Using branches allows you to develop features for your code without breaking the functionality of the `main` branch for users.

When merging your new feature into the `main` branch, it’s good to put in place some tests (e.g. unit tests, installation checks, dependency checks, etc.) to ensure the new feature doesn’t break existing code.

This isn’t a feature of git, but of your own continuous integration routine or of the services with which git interacts (e.g. GitHub).

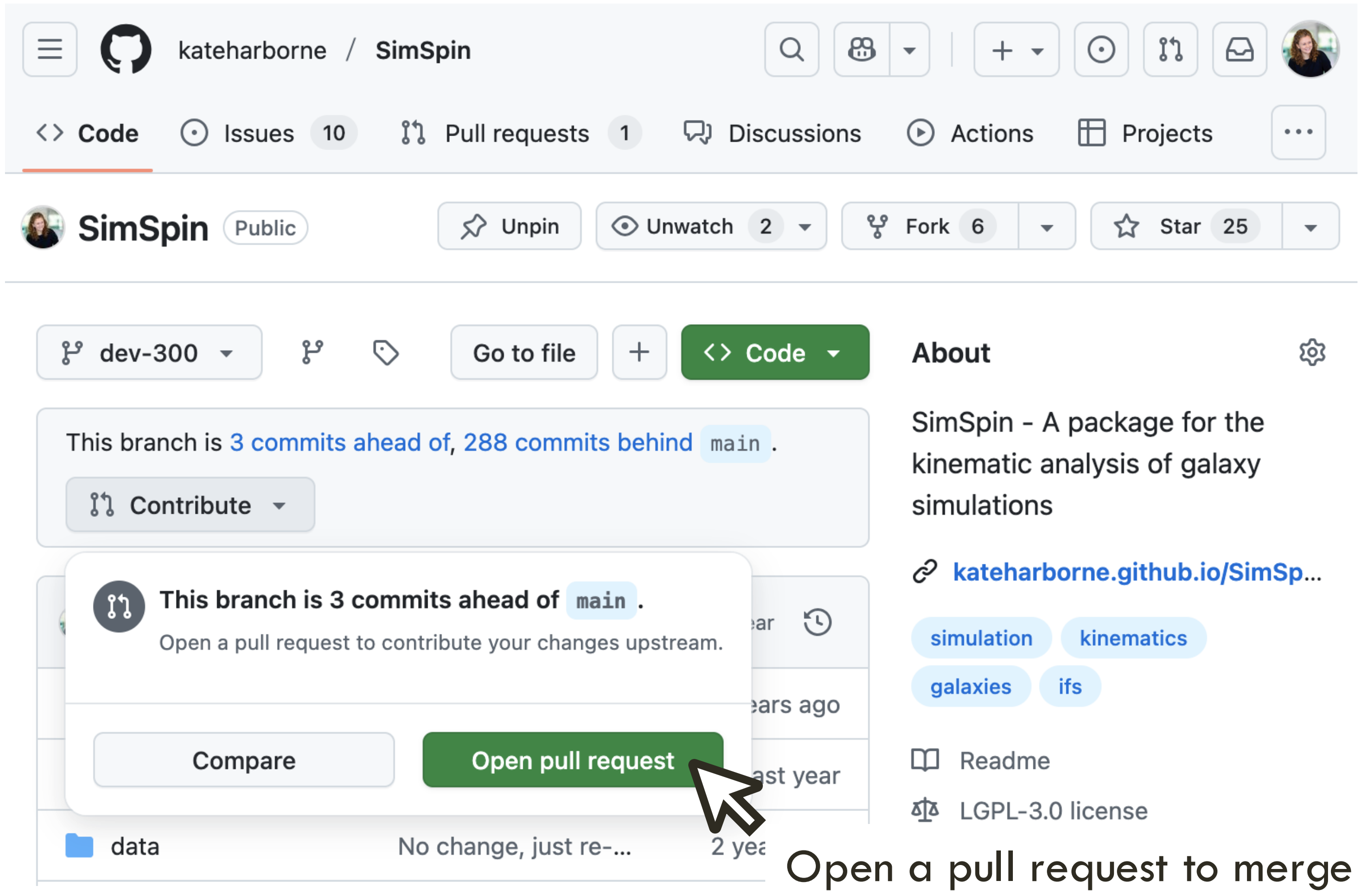
Some definitions – Merging a “BRANCH” <u>at origin</u>



```
# push changes to branch to origin
git push
```

This works because we have set up branch tracking (on Slide 28)

Some definitions – Merging a "BRANCH" at origin



kateharborne / SimSpin

Code · Issues 10 · Pull requests 1 · Discussions · Actions · Projects

SimSpin Public · Unpin · Unwatch 2 · Fork 6 · Star 25

dev-300 · Go to file · + · Code

This branch is 3 commits ahead of, 288 commits behind main.

Contribute

This branch is 3 commits ahead of main.

Open a pull request to contribute your changes upstream.

Compare · Open pull request

data · No change, just re-... · 2 yea

About

SimSpin - A package for the kinematic analysis of galaxy simulations

kateharborne.github.io/SimSp...

simulation · kinematics · galaxies · ifs

Readme

LGPL-3.0 license

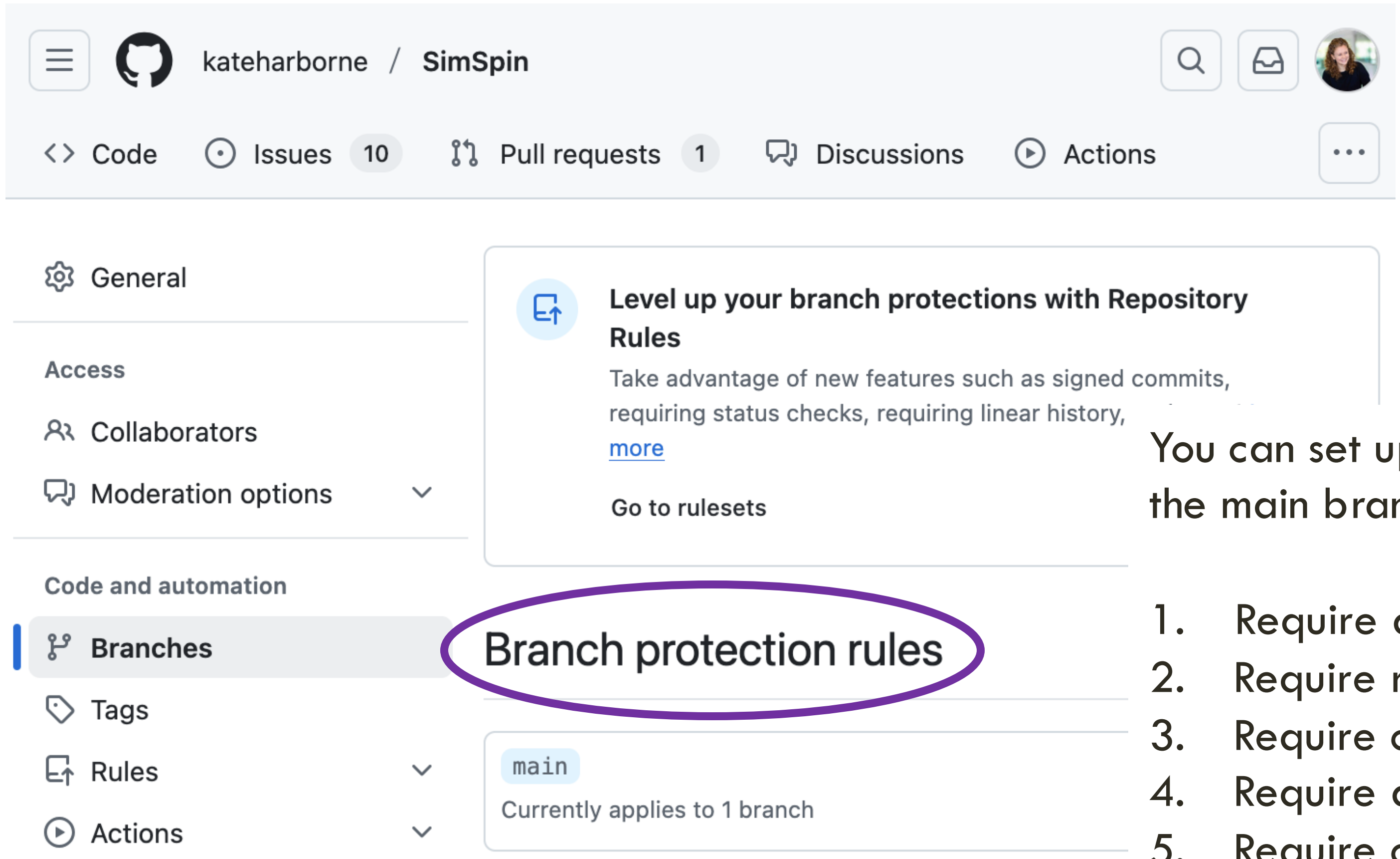
Open a pull request to merge changes into `main`

GitHub

Repository ("repo")

ORIGIN

Some definitions – Merging a "BRANCH" at origin


kateharborne / SimSpin

Code | Issues 10 | Pull requests 1 | Discussions | Actions

General

Access
- Collaborators
- Moderation options

Code and automation
- Branches
- Tags
- Rules
- Actions

Level up your branch protections with Repository Rules

Take advantage of new features such as signed commits, requiring status checks, requiring linear history, more

Go to rulesets

Branch protection rules

`main`
Currently applies to 1 branch

GitHub

Repository ("repo")

ORIGIN

You can set up a variety of different protections to ensure that the main branch is consistently executable, e.g.

1. Require a pull request before a merge.
2. Require review/approval from other developers.
3. Require checks to pass before merge.
4. Require a linear history
5. Require conversation resolution before merge.

Set up branch protections on your repo

As a minimum, you should ensure the "Require a pull request before merging" is ticked such that your team cannot commit directly to the main branch of the code.

When you have tests, you may also want to "Require status checks to pass before merging". This will force your team members to wait until the tests have passed before a PR can be approved (which is also a good way to ensure you never break the code for users!) *See here for GitHub Actions and James' talk this afternoon.*

+ many more options should your team decide!

Collaborative version management

1. When working on a change to the code, make a new branch rather than working directly on `main`.
2. `pull` before making changes – be sure you are updating the latest development.
3. Ready to merge your branch into `main`? Run your unit tests, install checks, etc. first.
4. Found a bug? Write a test.

~~Merge into `main` and push to `origin`.~~

5. `push` branch to `origin`
6. Open a pull request with branch protections

Any tips from the audience?

Additional Resources:

- Atlassian git tutorials: https://www.atlassian.com/git/tutorials/
- Look to example repos: https://github.com/astropy/astropy
- Some background: https://www.datacamp.com/blog/all-about-git
- Git conventional commit messages: https://www.conventionalcommits.org/en/v1.0.0/

Dr Kate Harborne
ICC, Durham University
katherine.e.harborne@durham.ac.uk