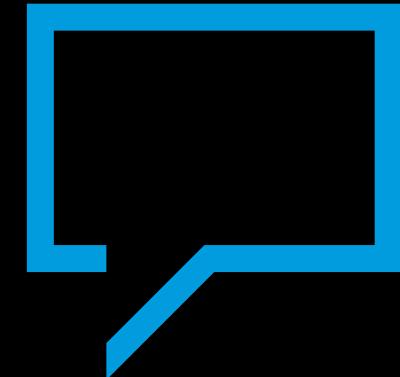




CONTRIBUTING TO OPEN-SOURCE SOFTWARE



HOW TO BE PART OF A LARGE SOFTWARE PROJECT

Eric A. Borisch, MS
MR Laboratory, Mayo Clinic

ISMRM & SMRT Annual Meeting & Exhibition
May 19, 2021



15-20 May 2021

Declaration of Financial Interests or Relationships

Speaker Name: Eric A. Borisch

I have no financial interests or relationships to disclose with regard to the subject matter of this presentation.



DOWNLOAD THESE SLIDES

<https://github.com/eborisch/ISMRM>



<https://github.com/eborisch/ISMRM/blob/main/2021/Open-source.pdf>

SOFTWARE ENGINEERING FOR MR SCIENTISTS

How to use version control:

#E5882 : Megan Poorman

How to be part of a large software project:

#E5883 : Eric Borisch

Wednesday, May 19

17:00 - 18:00 UTC

LEARNING OBJECTIVES

- Who can contribute to Open-Source Software (OSS)?
- Ways to contribute to OSS.
- How to provide useful feedback / contributions to a project.
- Using *git* to contribute to OSS projects.

AGENDA

1. Why you should contribute
2. Getting started – working with a project
3. Ways to contribute
4. Collaborating with *git*: key concepts
5. Contributing to a GitHub project
6. “Live” demonstration
7. Final thoughts, references & further reading

1

WHY YOU SHOULD CONTRIBUTE

CONTRIBUTING TO OPEN-SOURCE SOFTWARE

- Open-Source Software (OSS) is fundamentally a community endeavor
- You bring your own expertise and experiences to a project, which can be at any level:
 - as an end user
 - or as developer using a library
 - or as part of the project's team
- In the research context:
 - Helps support reproducible research
 - Visibility to future collaborators or employers

WHO CAN CONTRIBUTE TO OPEN-SOURCE SOFTWARE?

ANYONE

WHO CAN CONTRIBUTE TO OPEN-SOURCE SOFTWARE?

YOU



QUICK PREVIEW OF CONTRIBUTING A CHANGE

The screenshot shows a GitHub repository page for `poormanme / exampleRepo1`. The repository has 1 watch, 0 stars, 0 forks, and 2 commits. The most recent commit was made by Megan Poorman, 6 days ago, with 2 commits. The commit message is "first person update". The files modified are `README.md` (initial commit) and `peopleData.csv` (first person update). The `README.md` file contains the text "This is an example repo. yay.".

Code Issues Pull requests Actions Projects Wiki Security Insights

master ▾ 2 branches 0 tags Go to file Add file ▾ Code ▾

Megan Poorman first person update 7e98867 6 days ago 2 commits

README.md initial commit 6 days ago

peopleData.csv first person update 6 days ago

README.md

This is an example repo. yay.

About

No description, website, or topics provided.

Readme

Releases

No releases published

Packages

No packages published

TIP: *.md : [Markdown format](#), frequently used for formatted documentation on GitHub

ERRATA

- Collaborative repository location updated!
- Be sure to use the *itsASmallMagneticWorld* repository if you are following the video examples to contribute changes.

The screenshot shows a GitHub repository page. At the top, there is a navigation bar with a search bar, pull requests, issues, marketplace, and explore links. Below the navigation bar, the repository name 'poormanme / itsASmallMagneticWorld' is displayed, with 'itsASmallMagneticWorld' highlighted by a yellow box. To the right of the repository name are 'Unwatch' and '2' buttons. Below the repository name, there are tabs for Code, Issues, Pull requests, Actions, Projects, Wiki, and Security. The 'Code' tab is selected and underlined in red. A dropdown menu for the 'master' branch is open. To the right of the dropdown are 'Go to file', 'Add file', and 'Code' buttons. The main content area shows a list of files: 'README.md', 'jupyter notes', 'findCommonInstitutions...', and 'name_institution_data.csv'. Each item has a timestamp of '11 hours ago' and a circular icon with the number '7'. At the bottom of the page, the repository name 'itsASmallMagneticWorld' is prominently displayed in a large white font.

2

GETTING STARTED

Working with a project



WORKING WITH A PROJECT

- A good place to start is the project's page on GitHub
- All code on GH is organized by *project/repository*, here *mrirecon/bart*
 - *project* is sometimes an individual *user*
- Check for two things right away:
 - Readme
 - License



Screenshot of a GitHub repository page for `mrirecon/bart`.

The page shows the following details:

- Code tab** is selected.
- Issues**: 8
- Pull requests**: 1
- Actions**
- Projects**
- Wiki**
- Code dropdown**: master
- Buttons**: Go to file, Add file, Code
- Commits** (Recent):
 - aTrotier and uecker traj test default vs custo... 24 minutes ago 1,740
 - Makefiles Typo 15 months ago
 - doc remove CMake documentation 14 months ago
 - lib add empty lib directory (using the .gitign... 7 years ago
 - matlab Simple MATLAB/Octave Unit Test (#227) 9 months ago
 - python white space fixes 3 months ago
 - rules use generic rule (function) for building libs 15 months ago
 - save Save FFTW wisdom when TOOLBOX path... 2 years ago
 - scripts Conway's game of life 3 months ago
 - src POSIX shared memory files 41 minutes ago
 - tests traj test default vs custom angles 24 minutes ago
 - utests add ubsan option 14 days ago
- About** section:
 - BART: Toolbox for Computational Magnetic Resonance Imaging
 - mrirecon.github.io/bart/
 - Tags: compressed-sensing, mri, iterative-methods, computational-imaging, bart-toolbox
 - Readme** (highlighted)
 - BSD-3-Clause License** (highlighted)
- Releases**: 24
 - version 0.7.00 (Latest) on Mar 1
 - + 23 releases

WORKING WITH A PROJECT

- The *Readme* file will frequently have a section on contributing.
- Read and adhere to a project's contribution guidelines.
- Given a project, and knowing how/where they would like contributions, let's explore different types of contributions.

pydicom / pydicom

Sponsor Watch 69 Star 1.1k Fork 371

Code Issues 44 Pull requests 7 Discussions Actions Projects ...

README.md

Used by 2.3k + 2,294

circleci passing codecov 95% python 3.6 | 3.7 | 3.8 | 3.9 pypi package 2.1.2 DOI 10.5281/zenodo.4197955 chat on gitter

pydicom

pydicom is a pure Python package for working with DICOM files. It lets you read, modify and write DICOM data in an easy "pythonic" way.

Contributors 63

Contributing

To contribute to *pydicom*, read our [contribution guide](#).

To contribute an example or extension of *pydicom* that doesn't belong with the core software, see our contribution repository: [contrib-pydicom](#).

3

WAYS TO CONTRIBUTE

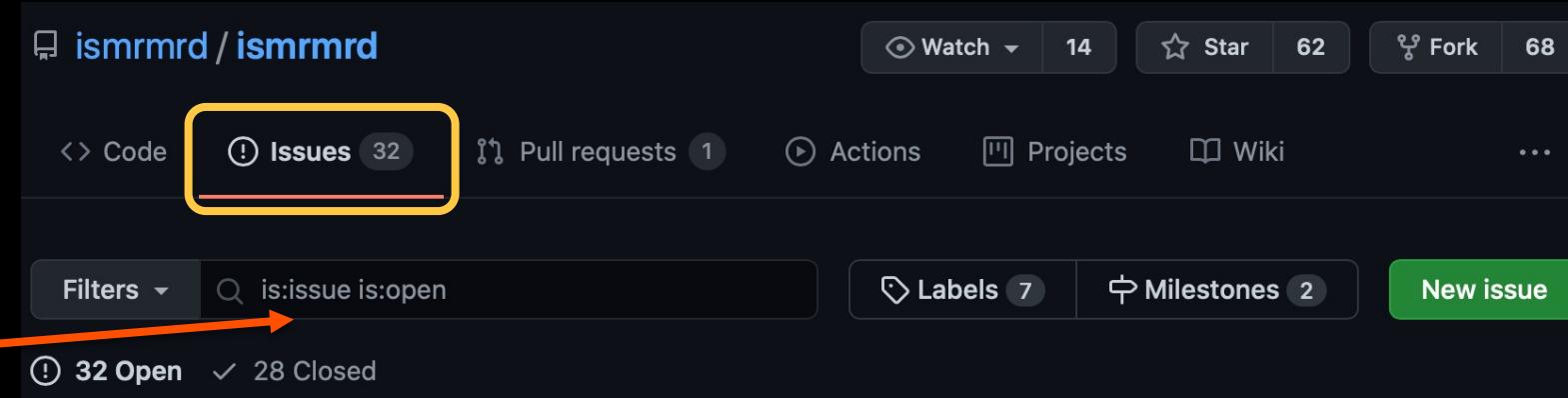
CONTRIBUTING



- Any level of user can meaningfully contribute to an open-source project.
- End users (not necessarily programmers)
 - Documentation improvements or additions [Pull requests]
 - Report issues, or provide feature requests [Issues or Discussions]
- Developers (users of a software library, for example)
 - Provide bug fixes [Pull requests]
 - Implement new features [Pull requests]
- Team members (Originators or collaborators on the project)
 - Review and address bug reports / feature requests [Issues & Pull requests]
 - Discuss and plan future developments [Discussions / Wiki]

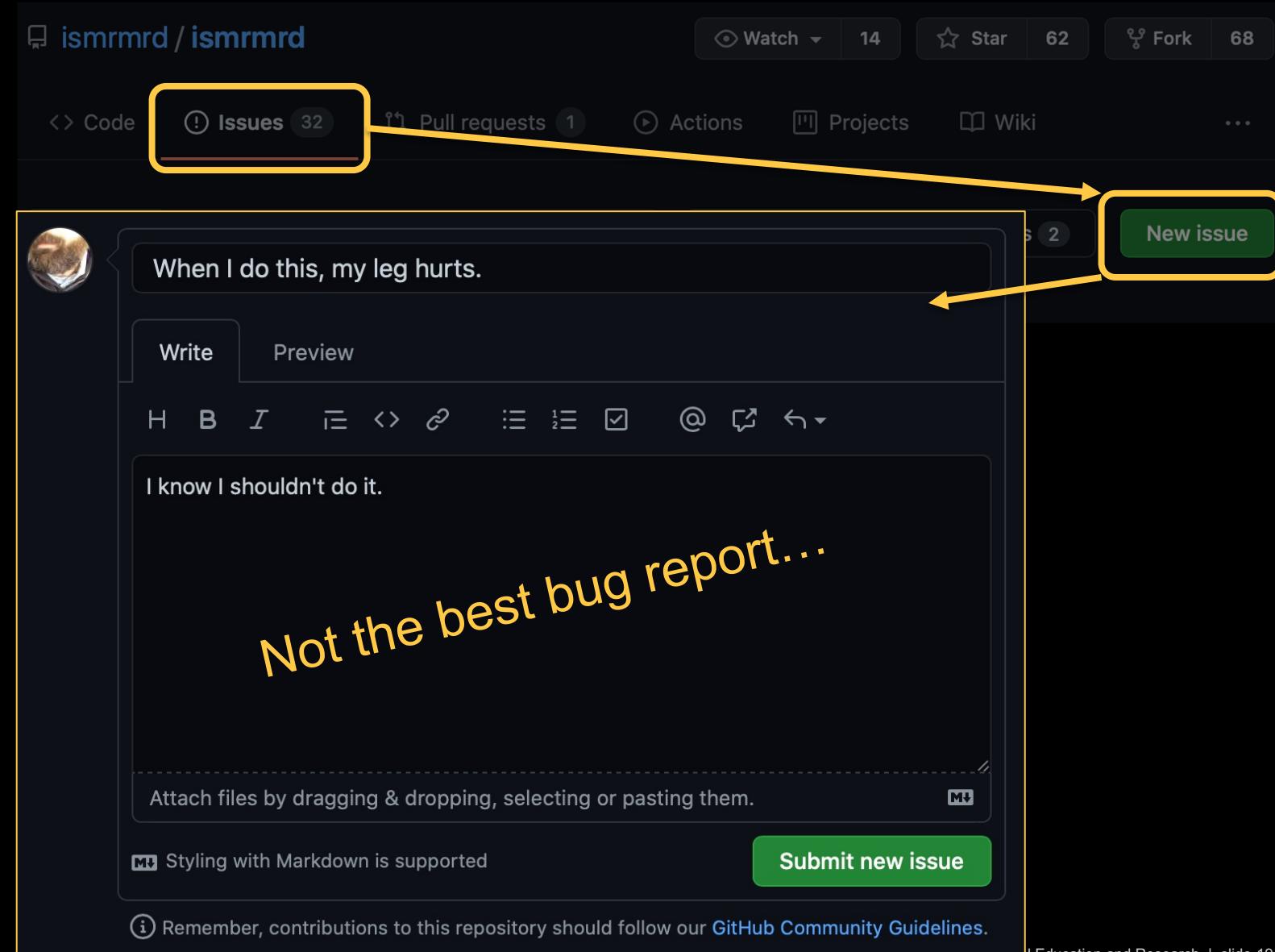
SUBMITTING ISSUES OR ENHANCEMENTS

- Issues are typically used both for *bug reports* and *enhancements* (feature requests)
- Always search for *existing* issues or suggestions that are similar or duplicates of your own.
 - Even adding a “me too” note to an existing issue helps inform the project that multiple users are impacted
 - Avoiding duplicates makes prioritizing and addressing issues easier



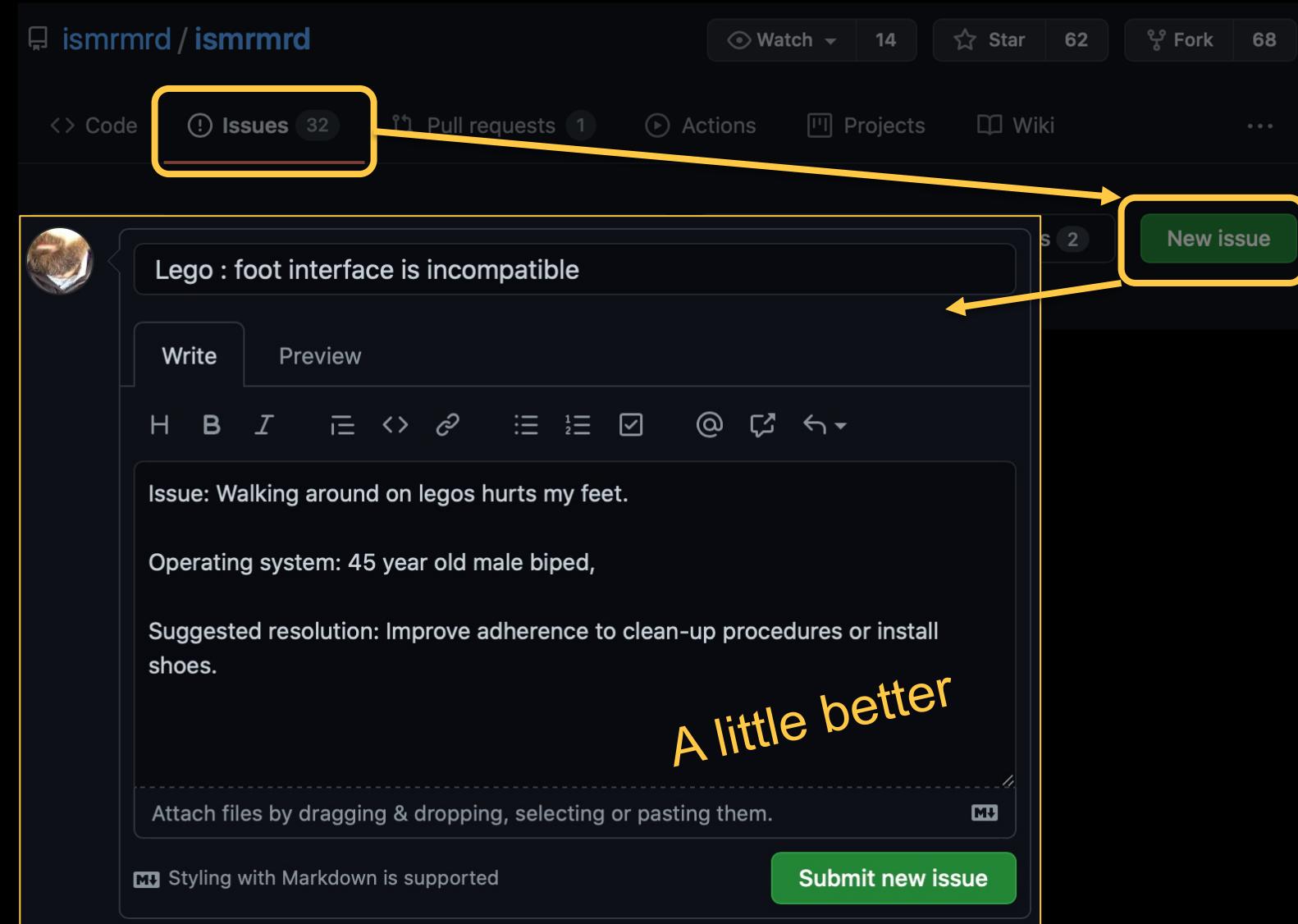
SUBMITTING ISSUES OR ENHANCEMENTS

- When you are satisfied this is a new issue or enhancement, create a new issue.
- Make your subject line as descriptive as possible (while keeping it short)



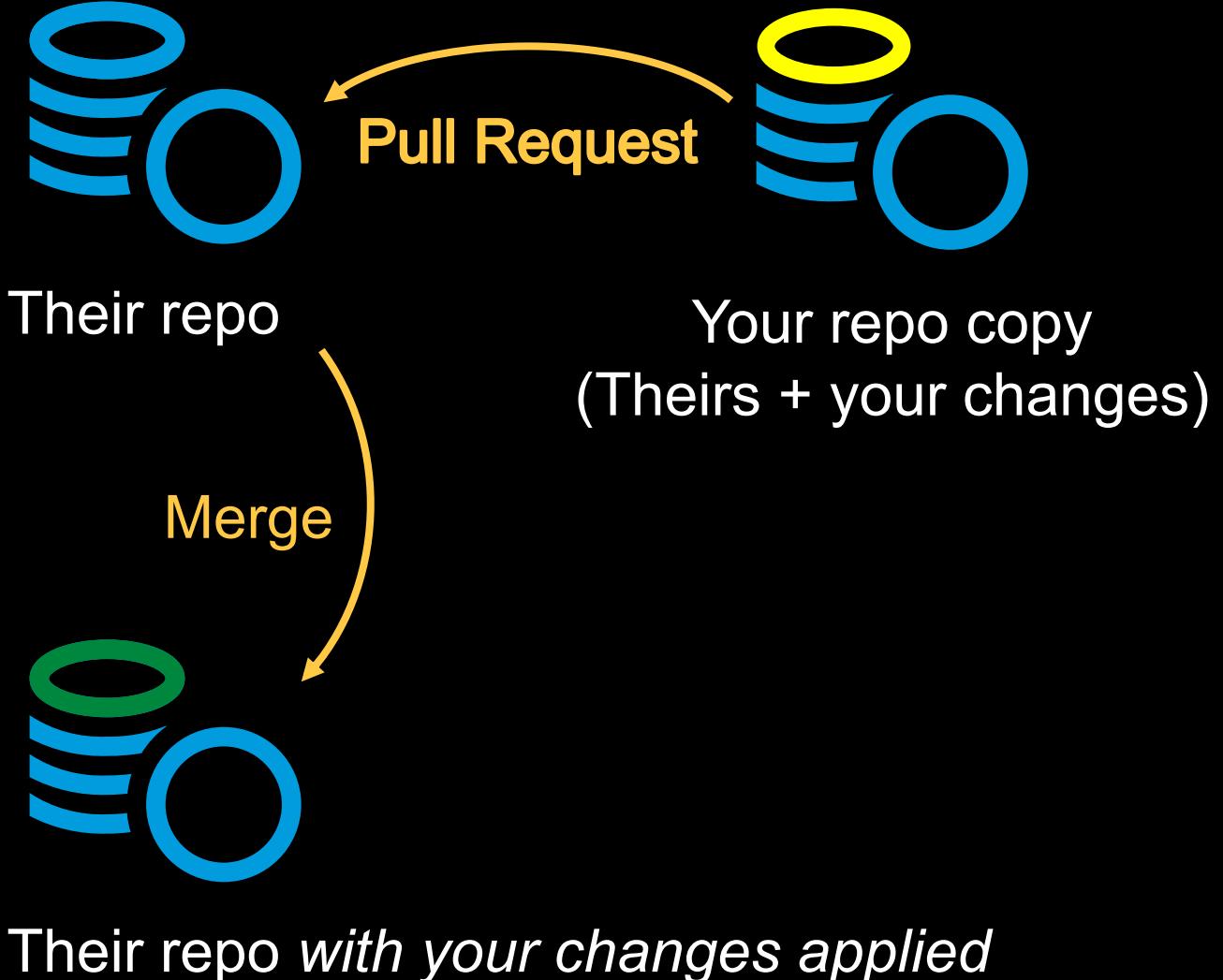
SUBMITTING ISSUES OR ENHANCEMENTS

- When you are satisfied this is a new issue or enhancement, create a new issue.
- Make your subject line as descriptive as possible (while keeping it short)
- For bug reports:
 - try to include a *reproducer*
 - Include the specifics of how you are using the software. Operating system (and version), software version, etc.
 - If you have thoughts on how the bug can be resolved, include those, too!
 - Many projects will have an issue template to fill out



PULL REQUEST : PREVIEW

- We will come back to this...



4

COLLABORATING WITH GIT: KEY CONCEPTS & PRACTICES

KEY GIT TERMS AND RELATED COMMANDS

Repository : `git clone <remote>`

A set of files with a recorded history of states; stored in a `.git/` directory at the top-level of the on-disk directory tree.

[A] commit : `git log` shows history

An atomic record (snapshot*) of file state / contents at a point in time.

[To] commit : `git commit -m <msg>`

The act of creating a commit from a (typically staged via `git add`) selected state.

branch : `git branch -v` lists branches

One of potentially multiple views of a repository state. The primary branch is typically named either `main` or `master`, and you can switch branches with `git checkout <name>`

A new branch with can be created with `git checkout -b <new_name> <origin>`, where `origin` can be a `remote/branch`, or omitted to refer to the current (HEAD) state.

* Internally, a git commit describes the state of files at a point in time (a snapshot); diffs (changes) can be calculated between different snapshots with related content.

remote : `git remote -v` lists remotes

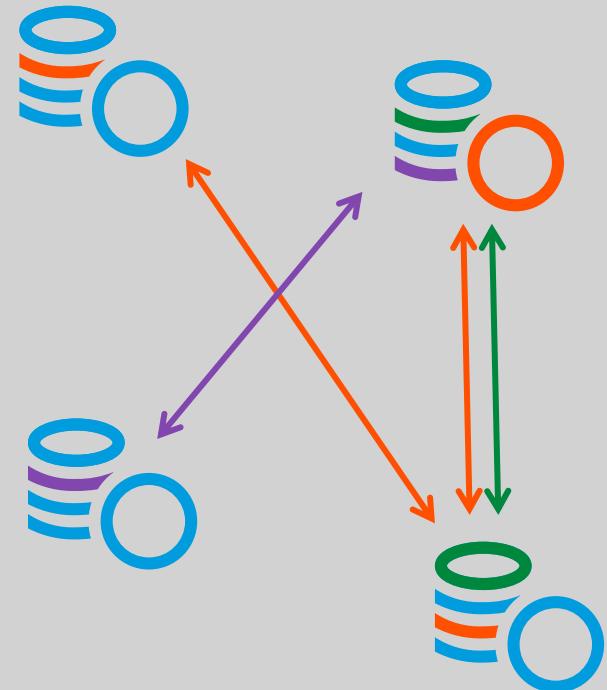
A repository hosted somewhere other than the current directory tree; frequently on github or gitlab. Common names: `origin` or `upstream`. Using `git clone <remote>` to initialize a repository automatically sets `origin` to the source URL.

A repository can have multiple remotes set via: `git remote add <name> <URL>`

A `remote` can be updated to match the latest version via `git fetch <name>`

GIT: DISTRIBUTED VERSION CONTROL

- Built from the ground-up for distributed version control
- *git* helps you manage and distribute your changes, but doesn't impose many restrictions
- Some best practices when collaborating with others:
 - Perform development on a local branch rather than *main (master)*
 - Rebase your development branch against *upstream* frequently
 - Stage commits to upstream in a *feature branch* (the first step to a *pull request*)



GIT COLLABORATION BEST PRACTICES: WORK IN A LOCAL DEVELOPMENT BRANCH

- Avoid working directly in the *main (master)* branch
 - The *main* branch should be updated on GitHub via *pull requests*, which *merge* in *commits* from a *feature branch*
- After cloning the remote repository, immediately checkout a local development branch with `git checkout -b <branch name> <starting point>`
 - -b: Checkout the requested starting point (*origin/master* below) as a local branch with the provided name (*lcl_dev*)
 - Note that the branch will be set to *track* the provided starting point (here *origin/master*)

```
~/s/external $ git clone https://github.com/ismrrmrd/ismrrmrd.git
Cloning into 'ismrrmrd'...
<snip>
~/s/external $ cd ismrrmrd/
~/s/e/ismrrmrd (master|✓) $ git checkout -b lcl_dev origin/master
Branch 'lcl_dev' set up to track remote branch 'master' from 'origin'.
Switched to a new branch 'lcl_dev'
```

GIT COLLABORATION EXAMPLE: WORK IN A LOCAL DEVELOPMENT BRANCH

```
~/s/e/ismrrmrd (lcl_dev|✓) $ echo 'THIS IS A CHANGE' >> README.md
~/s/e/ismrrmrd (lcl_dev|+1) $ git add README.md
~/s/e/ismrrmrd (lcl_dev|●1) $ git commit -m "Testing a change"
[lcl_dev 515b6f4] Testing a change
 1 file changed, 1 insertion(+)
```

```
~/s/e/ismrrmrd (lcl_dev↑1|✓) $ git log
commit 515b6f43a5a700630f6cec7bc7c0b7382bfa83b2 (HEAD -> lcl_dev)
Author: Eric A. Borisch <borisch.eric@mayo.edu>
Date:   Mon Apr 12 11:39:19 2021 -0500
```

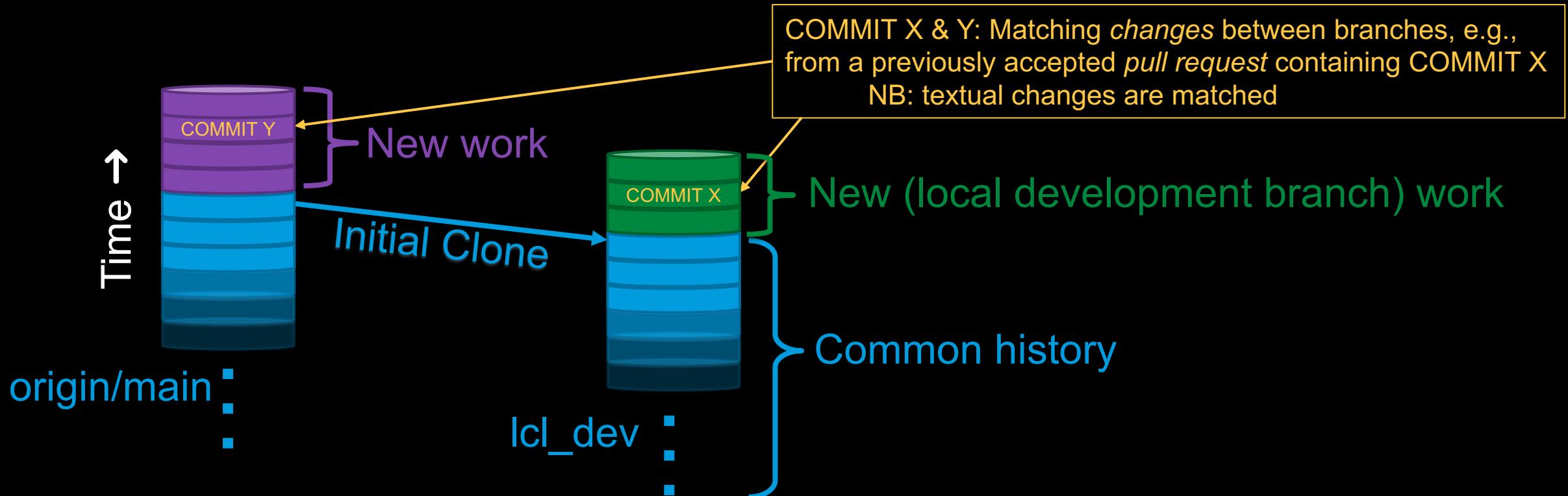
Testing a change

```
commit c86c89cfcee43e4ce1c146f6e4817dc2e06b2e63 (origin/master, origin/HEAD, master)
Author: Michael Hansen <mihansen@microsoft.com>
Date:   Sat Feb 20 09:54:28 2021 -0800
```

Fixed all warnings and treating warnings as errors on Windows (#171)

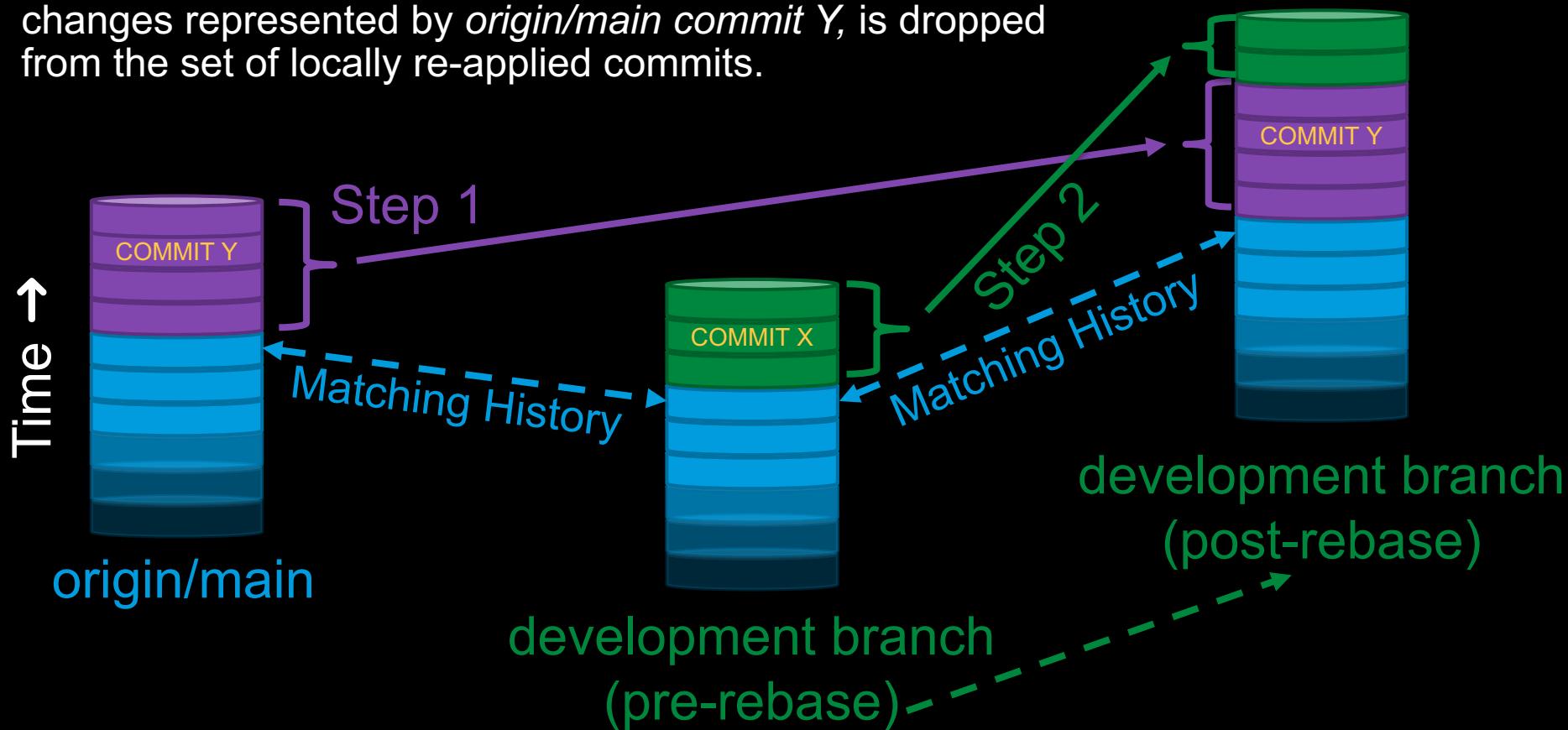
GIT COLLABORATION BEST PRACTICES: REBASE DEVELOPMENT BRANCH FREQUENTLY

- Rebasing pulls new changes from the upstream repository, and replays local changes *on top of* them, in order.
- Matching sets of textual changes that exist in the upstream branch are skipped during replay



GIT COLLABORATION BEST PRACTICES: REBASE DEVELOPMENT BRANCH FREQUENTLY

- `git fetch origin && git rebase origin/main or
git pull --rebase origin main`
- During step 2, the *local commit X*, which matches the changes represented by *origin/main commit Y*, is dropped from the set of locally re-applied commits.



GIT COLLABORATION BEST PRACTICES: REBASE DEVELOPMENT BRANCH FREQUENTLY

- `git fetch origin && git rebase origin/main` or
`git pull --rebase origin main`
- After completion, your local changes are still present, but with only the necessary differences (commits) from origin/main retained.
- Any new changes from upstream will also be present.



GIT COLLABORATION BEST PRACTICES: STAGE COMMITS IN A FEATURE BRANCH

- When preparing to submit a set of local changes related to a common feature, create a new branch named for the feature, and stage commits within the branch.
 - You will *push* this branch to GitHub and create a *pull request* (into *main*, for example) on GitHub when it is ready.
- There are multiple ways to do this, the simplest is to create a new branch based on origin/master and implement the feature directly in a set of commits.
- If you've already implemented the changes as a set of commits in */cl_dev*, either:
 - Create a new branch based on origin/master, and *git cherry-pick* desired commits into it. [Shown on next slide.]
 - Create a new branch based on the */cl_dev* and perform an *interactive rebase* onto origin/master, retaining only those commits related to the feature.

GIT COLLABORATION EXAMPLE: STAGE COMMITS IN A FEATURE BRANCH

```
~/s/e/ismrrmrd (lcl_dev↑2|✓) $ git log --pretty=oneline --abbrev-commit  
0287081 (HEAD -> lcl_dev) Unrelated change  
515b6f4 Testing a change  
c86c89c (origin/master, origin/HEAD, master) Fixed all warnings and treating warnings as  
errors on Windows (#171)  
1d042ab Added windows workflow (#170)
```

```
~/s/e/ismrrmrd (lcl_dev↑1|✓) $ git fetch origin  
<snip>  
~/s/e/ismrrmrd (lcl_dev↑2|✓) $ git checkout -b new_feature origin/master  
Branch 'new_feature' set up to track remote branch 'master' from 'origin'.  
Switched to a new branch 'feature'
```

```
~/s/e/ismrrmrd (new_feature|✓) $ git cherry-pick 515b6f4 ←  
[new_feature 27249af] Testing a change  
Date: Mon Apr 12 11:39:19 2021 -0500  
1 file changed, 1 insertion(+)
```

```
~/s/e/ismrrmrd (new_feature↑1|✓) $ git log --pretty=oneline --abbrev-commit  
27249af (HEAD -> new_feature) Testing a change  
c86c89c (origin/master, origin/HEAD, master) Fixed all warnings and treating warnings as  
errors on Windows (#171)  
1d042ab Added windows workflow (#170)
```

GIT COLLABORATION EXAMPLE: STAGE COMMITS IN A FEATURE BRANCH [+PUSH?]

- We've done our work in a development branch
- We've rebased onto (latest; after a `git fetch origin`) `origin/master`
- We've created a feature branch with just the commits we want.
- Time to push now, RIGHT?

```
~/s/e/ismrmrd (new_feature↑1|✓) $ git log --pretty=oneline --abbrev-commit
27249af (HEAD -> new_feature) Testing a change
c86c89c (origin/master, origin/HEAD, master) Fixed all warnings and treating warnings as
errors on Windows (#171)
1d042ab Added windows workflow (#170)

~/s/e/ismrmrd (new_feature↑1|✓) $ git push origin new_feature
ERROR: Permission to ismrmrd/ismrmrd.git denied to eborisch.
fatal: Could not read from remote repository.
```

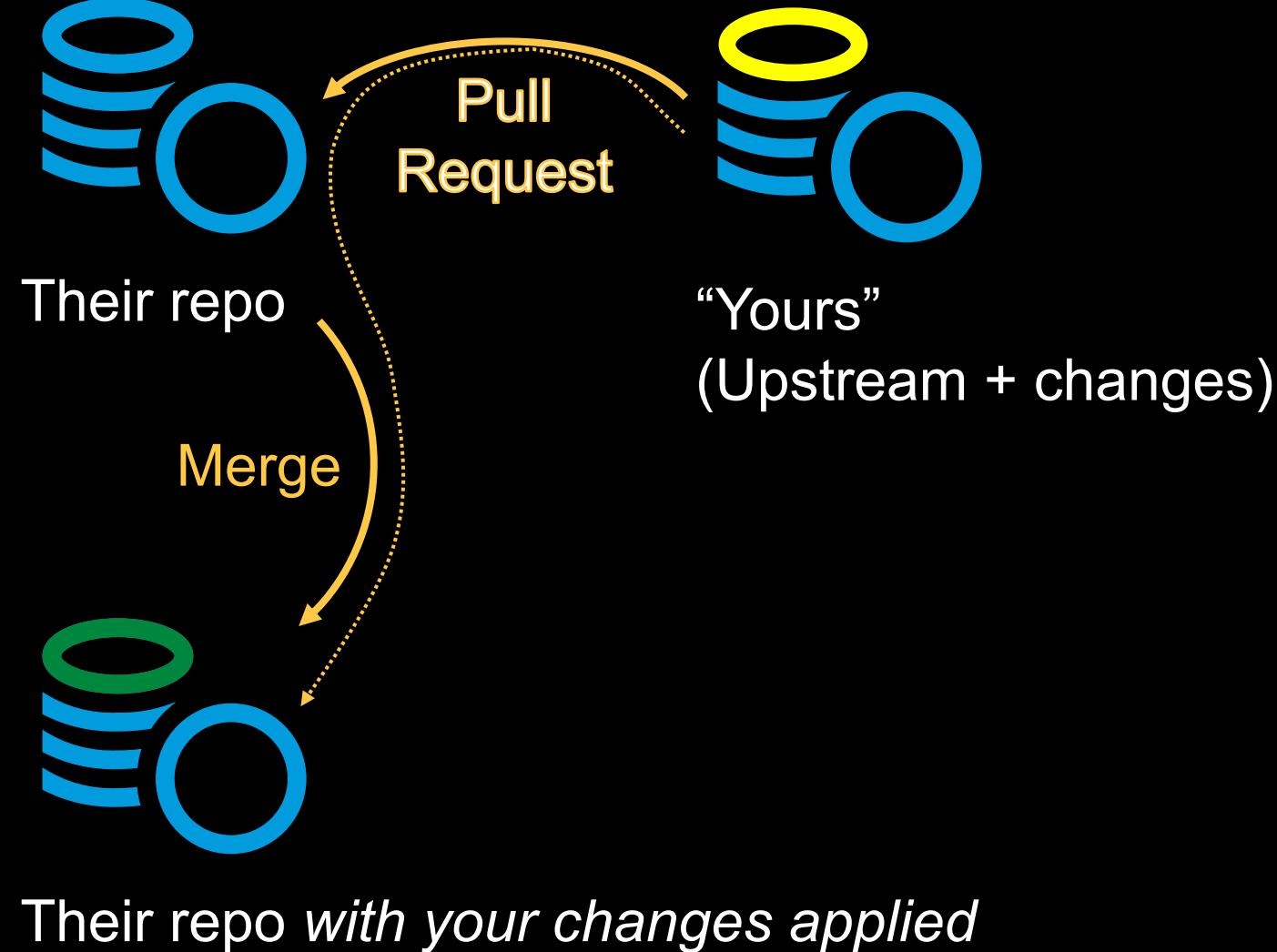
Please make sure you have the correct access rights
and the repository exists.

5

CONTRIBUTING TO A GITHUB PROJECT

PULL REQUEST : OVERVIEW

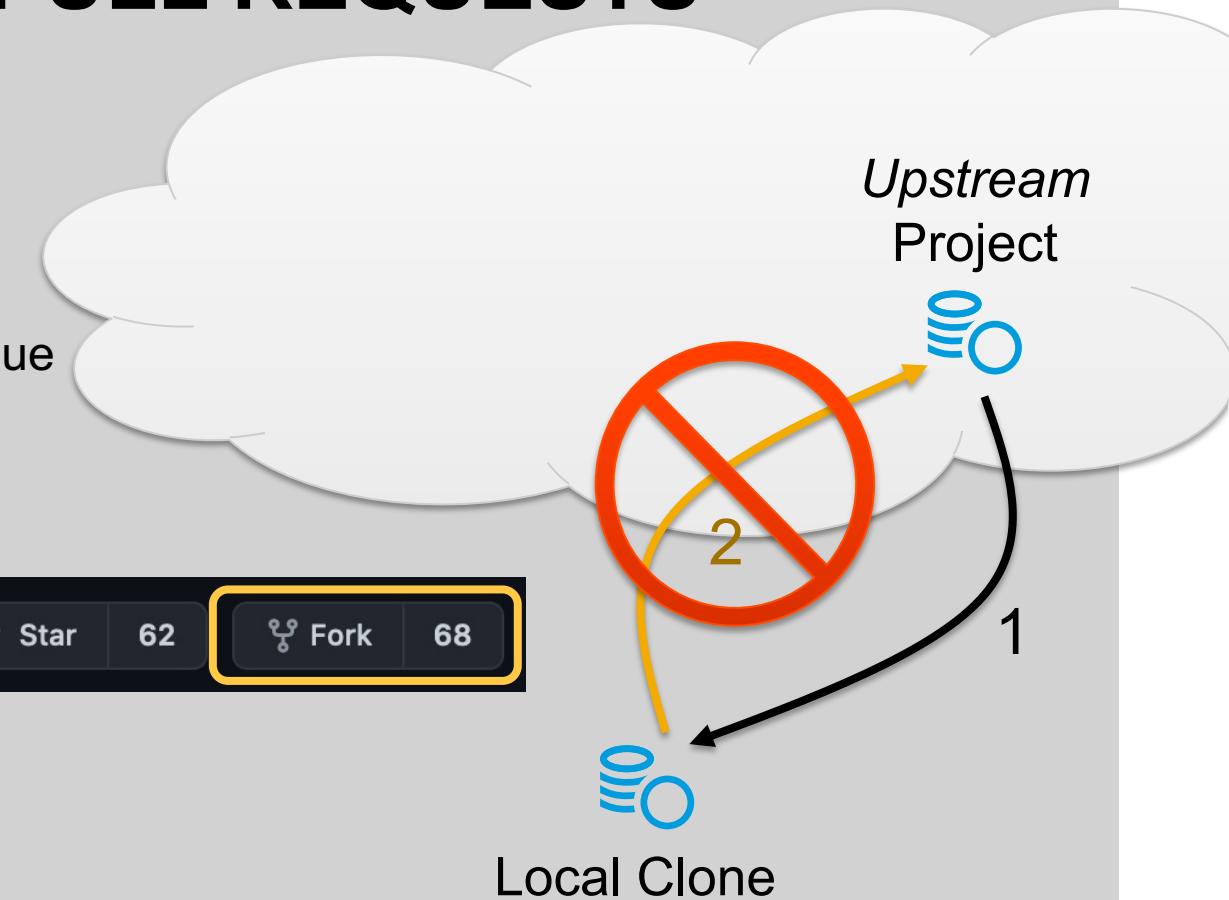
- *Pull requests* are the fundamental unit of collaboration for code modifications on GitHub.
- Encapsulates a set of changes (*commits*) that you have made to *your copy* of the repository along with a *request* to *pull* those changes into the *upstream* (the project's) repository, all with an online location to:
 - track the request
 - discuss the changes
 - modify the requested changes
 - etc.
- If the project accepts your changes, a *merge* is performed to apply your changes to the project's repository.



GITHUB CONTRIBUTIONS: PULL REQUESTS

1. We've started with a clone of a GitHub project
2. We prepared our local *feature branch* and tried to push it back to the project, but that was rejected due to permissions. (We don't have write access.)

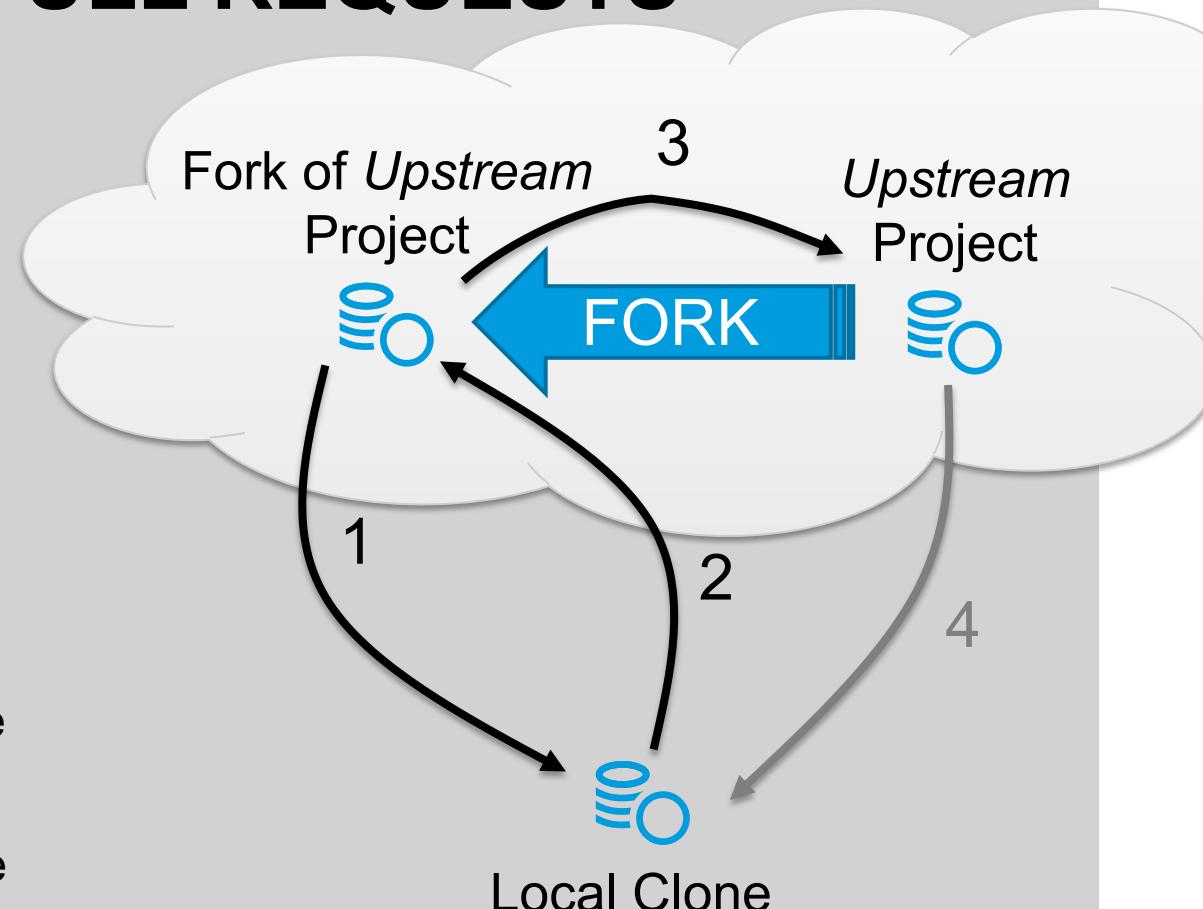
Solution? A “Fork”.



GITHUB CONTRIBUTIONS: PULL REQUESTS

“Fork” the project to your account on GitHub, then start with the same steps ...

1. Clone (the fork) to your local system
 - Create *feature* branch; modify; commit. (repeat modify/commit as needed)
2. Push *feature* branch to our fork
3. Create a *pull request* to merge your *feature branch* into the *upstream* project
 - Will become part of *upstream* when they choose to *merge* it
4. Add *upstream* remote to facilitate future *pulls* of the latest changes :
`git remote add upstream <URL>`

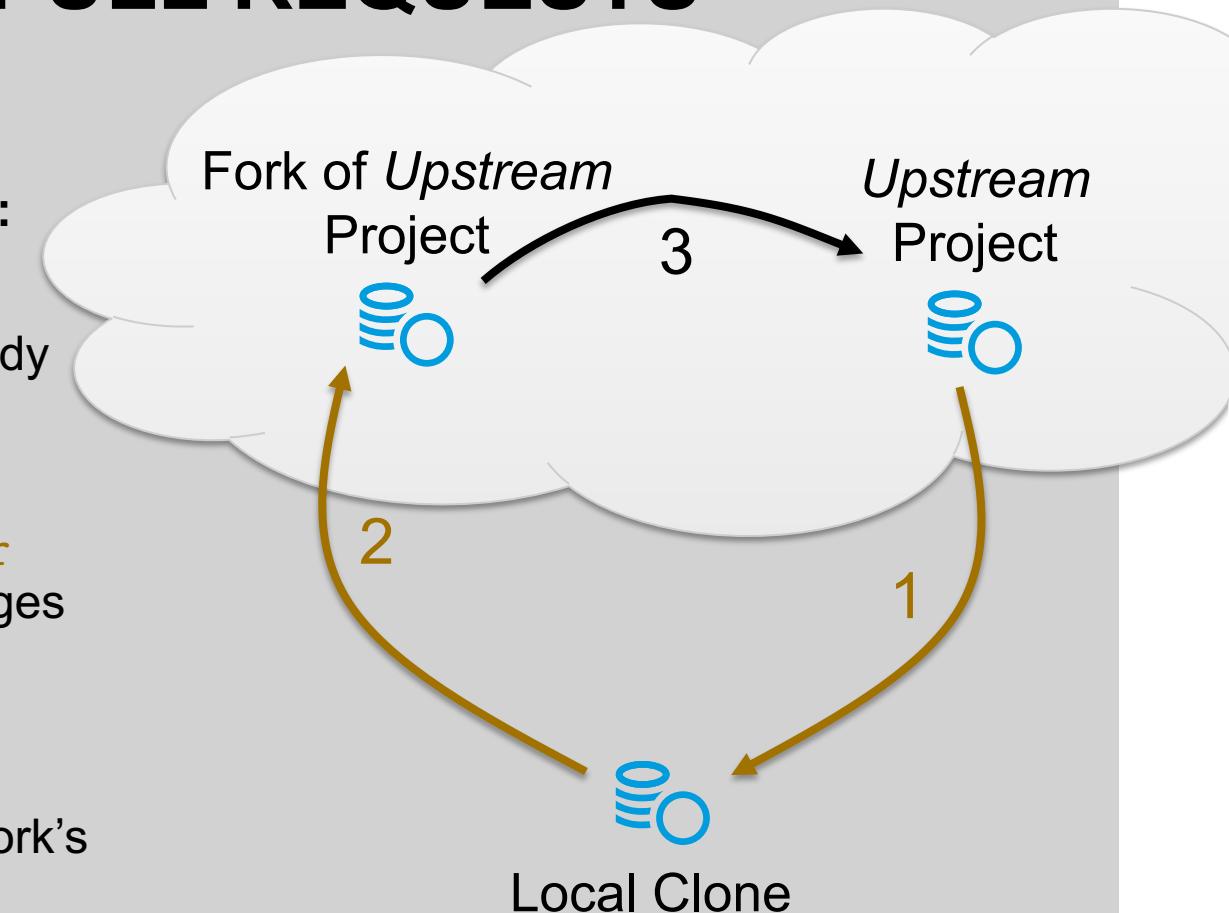


GITHUB CONTRIBUTIONS: PULL REQUESTS

Once this is established, new work is more direct:

Create a new local *feature branch*; modify; commit.
(Repeat modify/commit as needed.) When you're ready
to generate a pull request:

1. Perform:
`git pull --rebase upstream master`
within the *feature branch* to *pull* in the latest changes
2. Push the *feature branch* to your fork with:
`git push origin feature`
3. Create (on GitHub) a *pull request* to *merge* your fork's
feature branch into the *upstream project*



6

“LIVE” DEMONSTRATION

7

FINAL THOUGHTS & FURTHER READING

SOFTWARE ENGINEERING FOR MR SCIENTISTS

How to use version control:

#E5882 : Megan Poorman

How to be part of a large software project:

#E5883 : Eric Borisch

Wednesday, May 19

17:00 - 18:00 UTC

FURTHER READING

- These slides:
<https://github.com/eborisch/ISMRM/blob/main/2021/Open-source.pdf>
- Configuring SSH keys for working with GitHub
<https://docs.github.com/en/github/authenticating-to-github/connecting-to-github-with-ssh>
- Online git reference documentation:
<https://git-scm.com/docs>
- A nice overview on different ways to contribute to OSS:
<https://opensource.guide/how-to-contribute/>
- The git “choose your own adventure” or “oops I did X, how do I fix it?”:
<http://sethrobertson.github.io/GitFixUm/fixup.html>
- One of many “cheat-sheets” out there:
<https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet>
- Markdown format:
<https://guides.github.com/features/mastering-markdown/>
- GUI interfaces to git:
<https://git-scm.com/downloads/guis>



Questions?

 @eborisch

borisch.eric@mayo.edu