Git Workflows

Rinku Gupta
Argonne National Laboratory

Software Productivity and Sustainability track, ATPESC 2021

Contributors: Patricia Grubel (LANL), Rinku K. Gupta (ANL), Jared O'Neal (ANL), James M. Willenbring (SNL)

LA-UR-21-25665

License, Citation and Acknowledgements

License and Citation

- **The requested citation the overall tutorial is: David E. Bernholdt, Anshu Dubey, Rinku K. Gupta, and David M. Rogers, Software Productivity and Sustainability track, in Argonne Training Program on Extreme-Scale Computing (ATPESC), online, 2021. DOI: 10.6084/m9.figshare.15130590**
- Individual modules may be cited as *Speaker, Module Title*, in Better Scientific Software tutorial…

Acknowledgements

- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.
- This work was performed in part at the Argonne National Laboratory, which is managed by UChicago Argonne, LLC for the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.
- This work was performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
- This work was performed in part at the Lawrence Livermore National Laboratory, which is managed by Lawrence Livermore National Security, LLC for the U.S. Department of Energy under Contract No. DE-AC52-07NA27344.
- This work was performed in part at the Los Alamos National Laboratory, which is managed by Triad National Security, LLC for the U.S. Department of Energy under Contract No.89233218CNA000001
- This work was performed in part at Sandia National Laboratories. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Content

- Brief explanation of Version Control with Git
- Git Workflow Mechanisms for Collaboration
  - Branches
  - Pull Requests
  - Forks
- Exposure to workflows of different complexity
- Collaboration using Git Workflows for CSE projects
- What to think about when evaluating different workflows
- Extra: Heat Equation Example Workflow

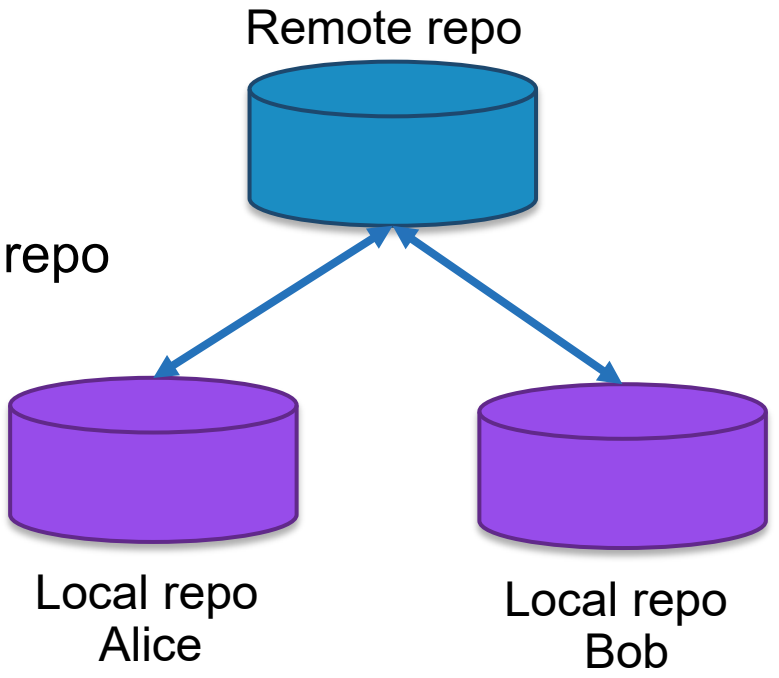
Goal

Development teams would like to use version control to collaborate productively and ensure correct code.

First Workflow

This process of collaborating *via* Git is called the **Centralized Workflow**

- See Atlassian/BitBucket for more information
- “Simple” to learn and “easy” to use
- Leverages local vs. remote repo dimension
  - Integration in local repo when local repos interact with remote repo
- What if you have many team members?
- What if developers only push once a month?
  - Lengthy development efforts without integrating
  - Occasional contributors
- What if team members works on different parts of the code?
- Working directly on master



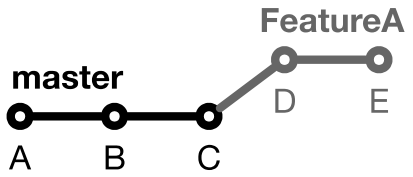
Git Workflow Mechanisms for Collaboration

- Branches
  - Enable separate development for features or fixes on the same repo
  - Enables different types of Workflows
- Pull Requests
  - Enables code review and testing before merge
- Forks
  - Enables outside contributors that have read access only
  - Controls on original repo remains with the team

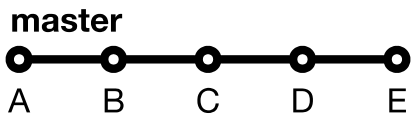
Branches

Branches are independent lines of development

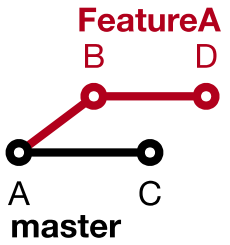
- Use branches to protect master branch
- Feature branches
  - Organize a new feature as a sequence of related commits in a branch
- Branches are usually combined or **merged**
- Develop on a branch, test on the branch, and merge into master
- Integration occurs at merge commits



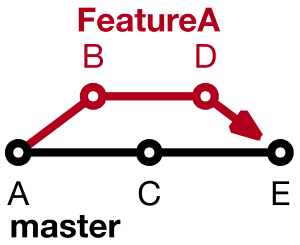
Fast-Forward



No Merge



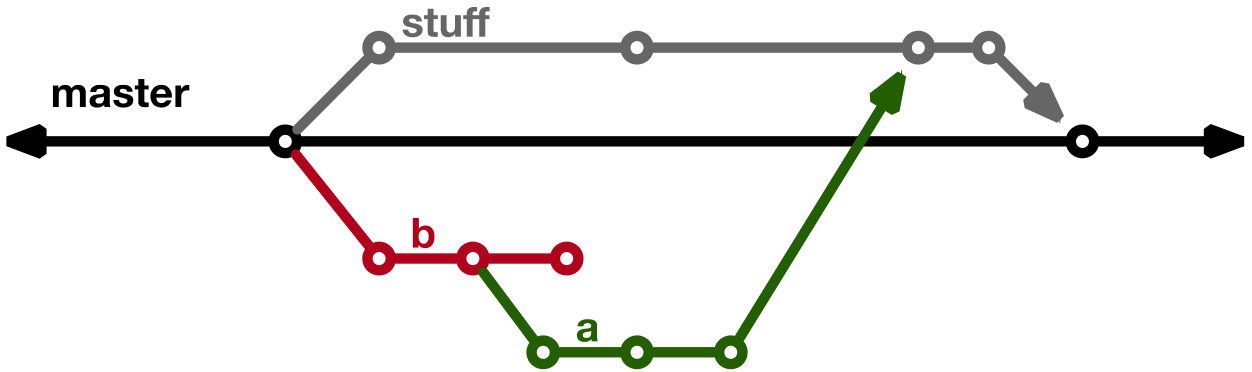
Divergence



Merge Commit

Control Project Branch Complexity

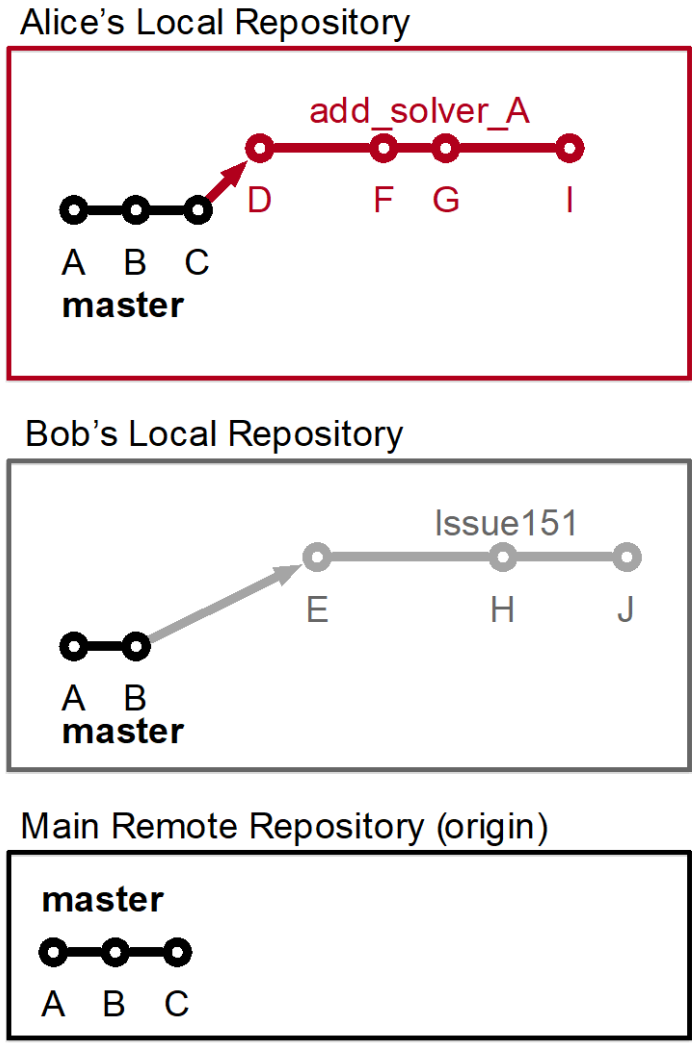
- Workflow policy is needed
  - Project supported branches and workflows should not be unnecessarily complex
  - Individuals and sub-teams can leverage more complex models when advantageous
  - Descriptive names or linked to issue tracking system
  - Where do branches start and end?



Feature Branches

Extend Centralized Workflow

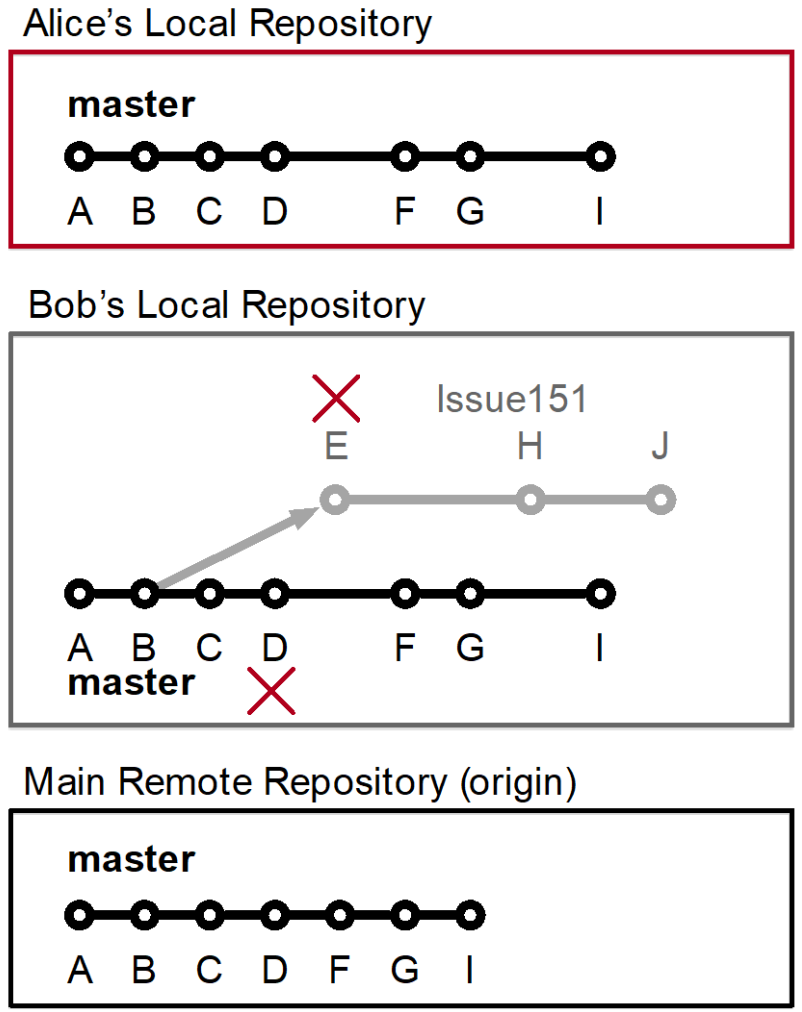
- Remote repo has commits A & B
- Bob pulls remote to synchronize local repo to remote
- Bob creates local feature branch based on commit B
- Commit C pushed to remote repo
- Alice pulls remote to synchronize local repo to remote
- Alice creates local feature branch based on commit C
- Both develop independently on local feature branches



Feature Branch Divergence

Alice integrates first without issue

- Alice does fast-forward merge to local master
- Alice deletes local feature branch
- Alice pushes master to remote
- Meanwhile, Bob pulls master from remote and finds Alice's changes
- Merge conflict between commits D and E

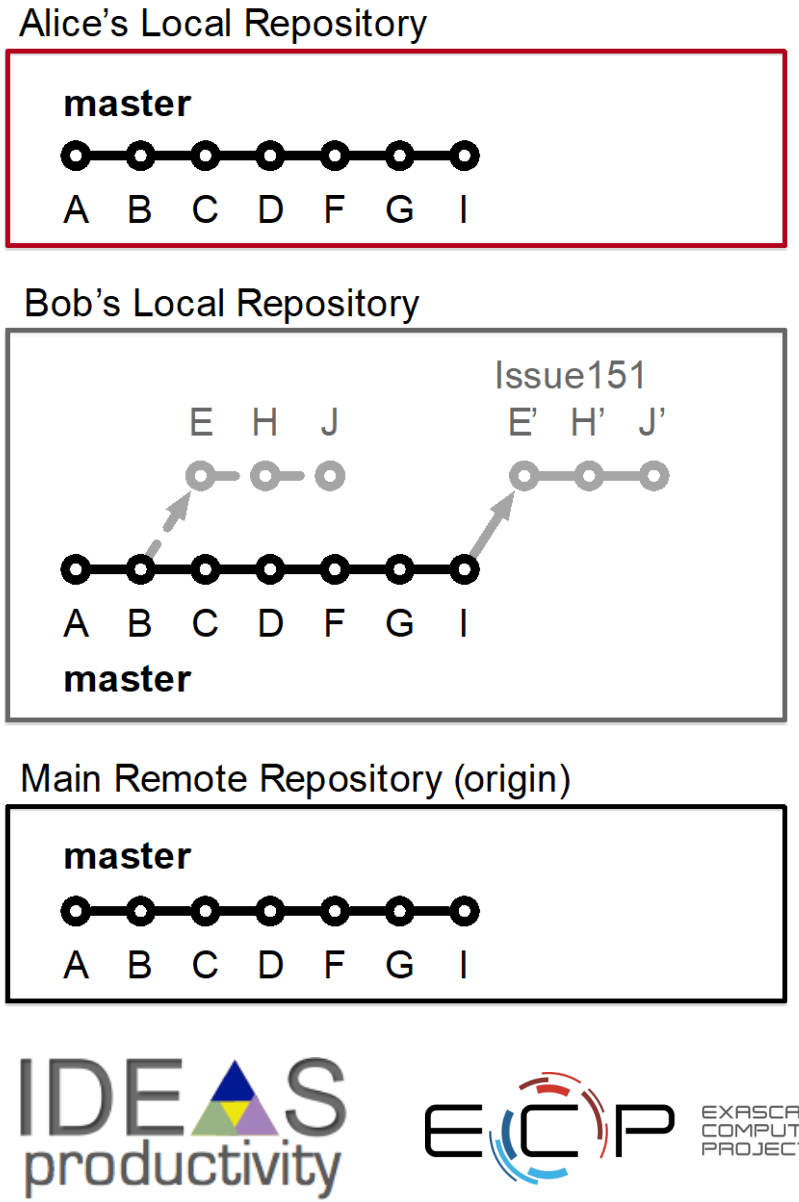


Feature Race Condition

Integration occurs on Bob's local repo

- Bob laments not having fast-forward merge
- Bob **rebases** local feature branch to latest commit on master
  - E based off of commit B
  - E' based off of Alice's commit I
  - E' is E integrated with commits C, D, F, G, I
- Merge conflict resolved by Bob & Alice on Bob's local branch when converting commit E into E'
- Can test on feature branch and merge easily and cleanly

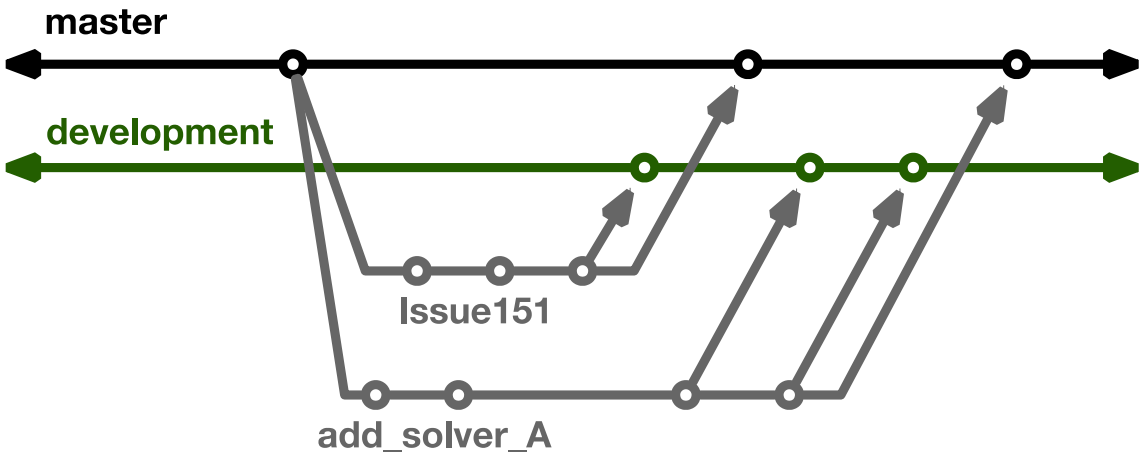
- See Atlassian/BitBucket for a richer Feature Branch Workflow



More Branches

Branches with infinite lifetime

- Base off of master branch
- Exist in all copies of a repository
- Each provides a distinct **environment**
  - Development vs. pre-production



Pull Requests

- Review and testing before merge
  - Alerts others about changes in branch before merge
  - Discussions ensue with possible follow up commits
  - Can request reviewer
- Set policies for merge

GitHub Forks

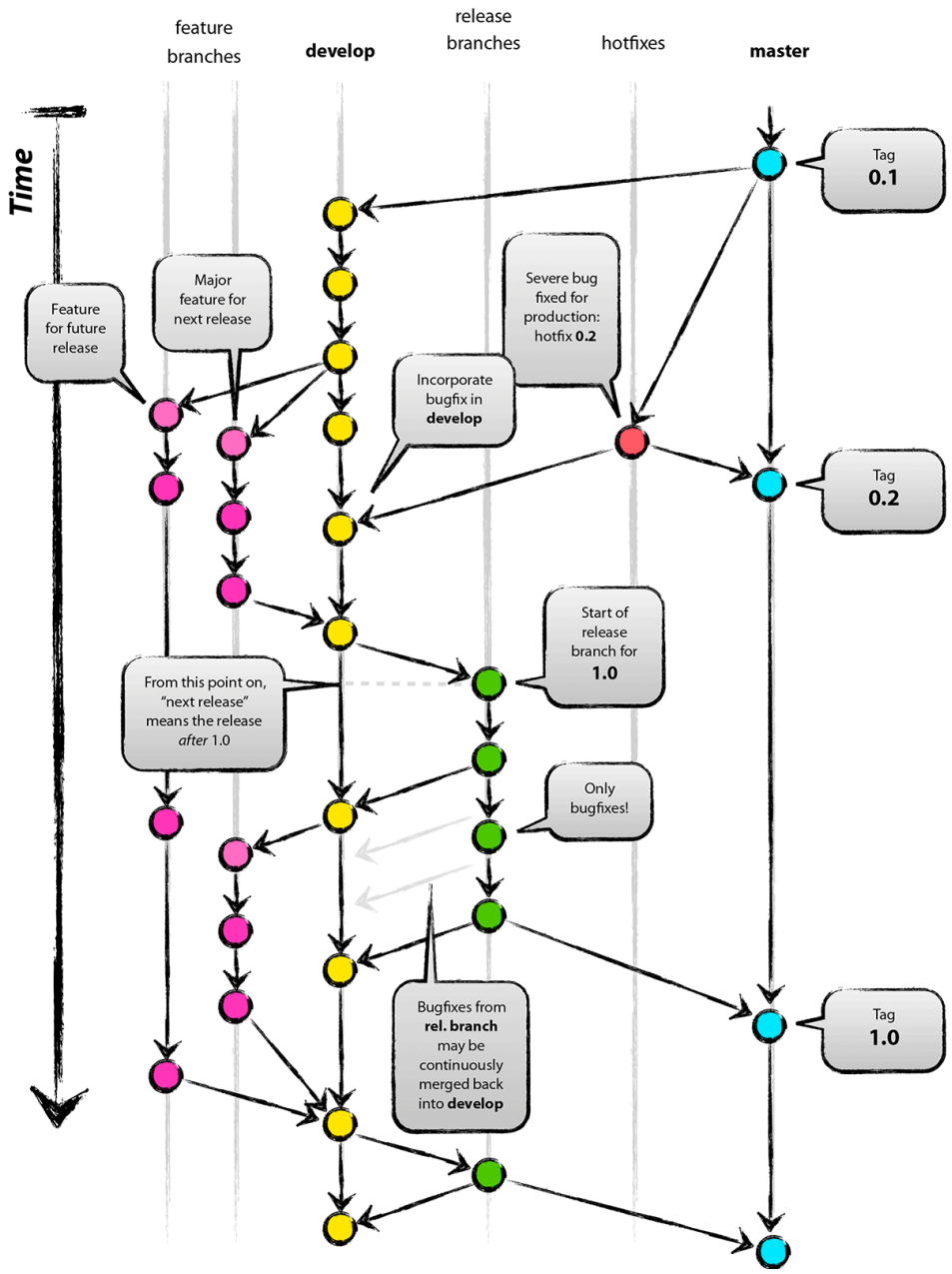
- A “fork” of a repository is a complete copy of another repository, inside a different GitHub account.
  - Forking requires read access to the main (often referred to as “upstream”) repository
    - Forks of public repositories are public
    - Other users can be granted write access to your fork
    - You cannot fork a fork
  - Does not copy issues or pull requests
  - Use branches within your fork (do not modify master)
  - A pull request (GitLab uses “merge request”) can be used to suggest changes to the upstream repository
    - Added benefit: pull requests from forks prevent huge numbers of branches on the upstream repository

Git Workflow Models of Different complexity

Commonly Known Workflows

- Git Flow
- Github Flow
- Gitlab Flow

Git Flow



- Full-featured workflow
- Increased complexity
- Designed for SW with official releases
- Feature branches based off of develop
- Git extensions to enforce policy
- How are develop and master synchronized?
- Where do merge conflicts occur and how are they resolved?

Author: Vincent Driessen
Original Blog: https://nvie.com/posts/a-successful-git-branching-model/
License: Creative Commons

GitHub Flow

http://scottchacon.com/2011/08/31/github-flow.html

- Published as viable alternative to Git Flow
- No structured release schedule
- Continuous deployment & continuous integration allows for simpler workflow

Main Ideas

1. All commits in master are **deployable**
2. Base feature branches off of master
3. Push local repository to remote constantly
4. Open Pull Requests early to start dialogue
5. Merge into master after Pull Request review

GitLab Flow

https://docs.gitlab.com/ee/workflow/gitlab_flow.html

- Published as viable alternative to Git Flow & GitHub Flow
- Semi-structured release schedule
- Workflow that simplifies difficulties and common failures in synchronizing infinite lifetime branches

Main Ideas

- Master branch is staging area
- Mature code in master flows downstream into pre-production & production infinite lifetime branches
- Allow for release branches with downstream flow
  - Fixes made upstream & merged into master.
  - Fixes cherry picked into release branch

Collaboration using Git Workflows for CSE projects

- Trilionos Workflow
- Open MPI Workflow
- Flecsi Workflow

Current Trilinos Workflow

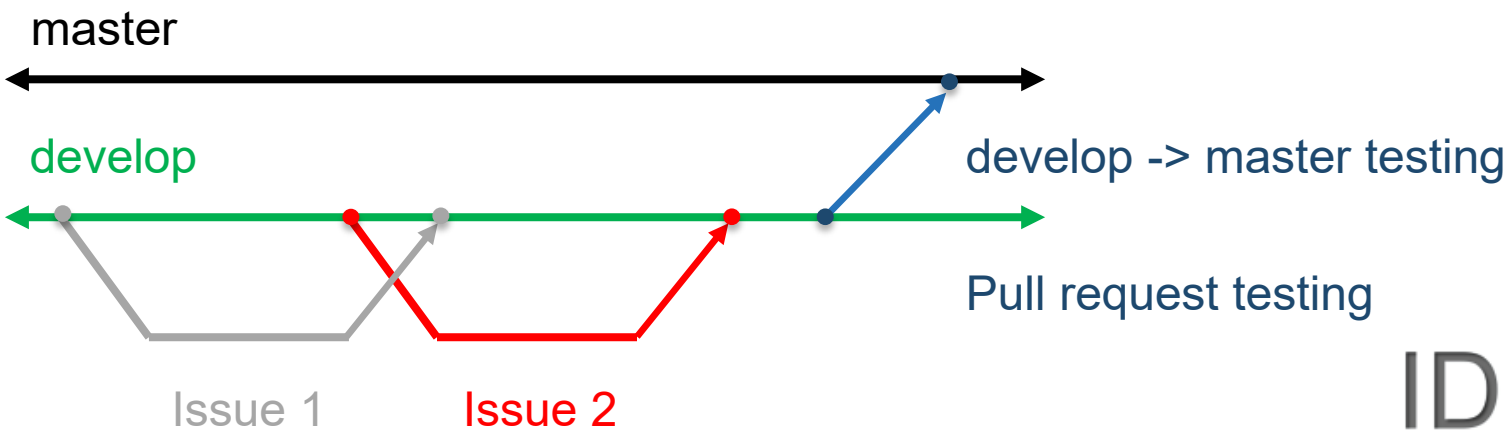
https://trilinos.github.io/

Test-driven workflow

- Feature branches start and end with develop
- All changes to develop must come from GitHub pull requests
- Feature branches are merged into develop only after passing pull request test suite
- Change sets from develop are tested daily for integration into master

Workflow designed so that

- All commits in master are in develop
- Merge conflicts exposed when integrating into develop
- Merge conflicts never occur when promoting to master



Current Open MPI Workflow

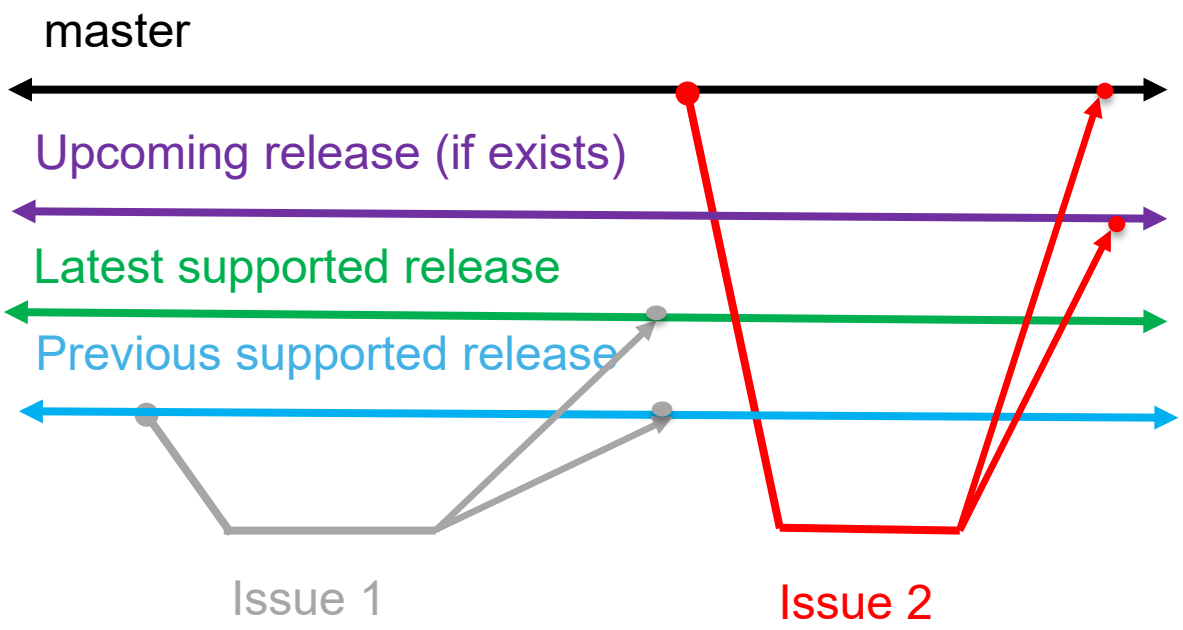
https://www.open-mpi.org

Versioning:

Major versions - break compatibility

Minor versions – visible

Releases correct issues



Workflow designed so that

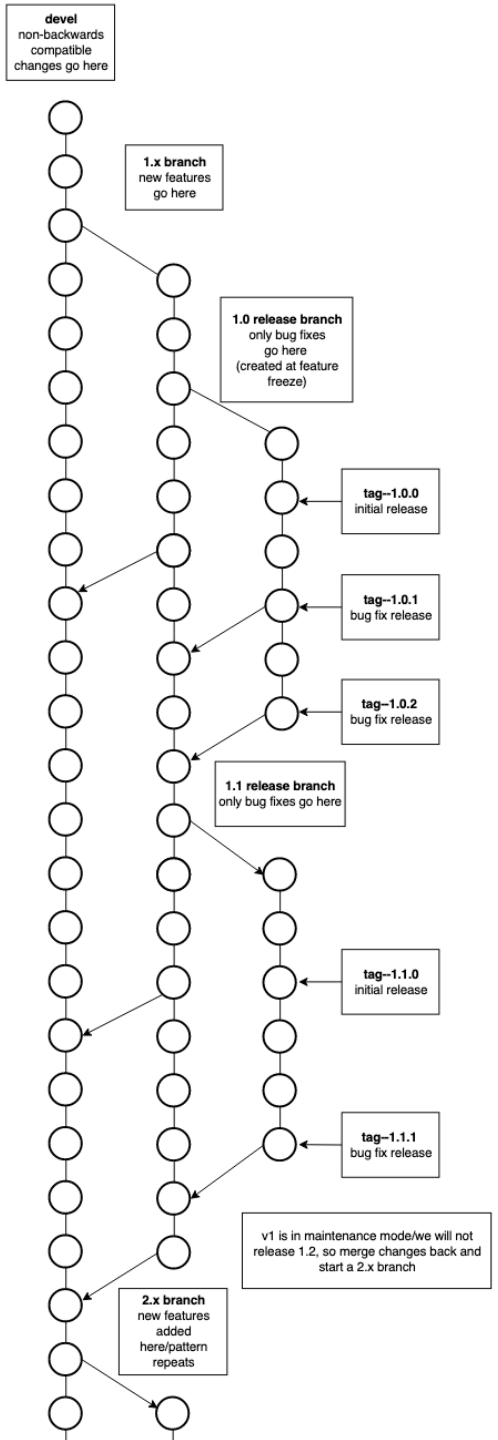
- Support two most recent releases
- Issues are addressed on all applicable branches
- All PR's reviewed by at least one core developer
- Master and supported branches work at all times
- Developers work on master or feature branches depending on complexity of the changes

Testing

- CI testing on PR's for any branch using Jenkins (limited set of compilers, hardware, tests)
- Nightly testing on all branches using community-build MTT framework (more complex set of compilers, hardware, tests)
- Additional testing for release candidates

Current FleCSI Workflow

https://flecsi.github.io/flecsi



Versioning:

Incompatible - **devel** branch breaks compatibility with previous versions

Feature (1, 2 …) named for major version

Release - **(1.x, 2.x …)** named for major.minor version, correct issues, tags used for bug fixes.

Workflow designed so that

- All supported branches work at all times
- Merge Requests are tested and reviewed

Testing

- Customized unit-testing framework based on Google Test
- Special *gitlab-ci* branch - images and configuration files

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Considerations for Choosing a Git Workflow

Want to establish a clear set of polices that

- results in correct code on a particular branch (usually master),
- ensures that a team can develop in parallel and communicate well,
- minimizes difficulties associated with parallel and distributed work, and
- minimizes overhead associated with learning, following, and enforcing policies.

Adopt what is good for your team

- Consider team culture and project challenges
- Assess what is and isn't feasible/acceptable
- Start with simplest and add complexity where and when necessary

Extra: Demo for Heat Equation Example Workflow

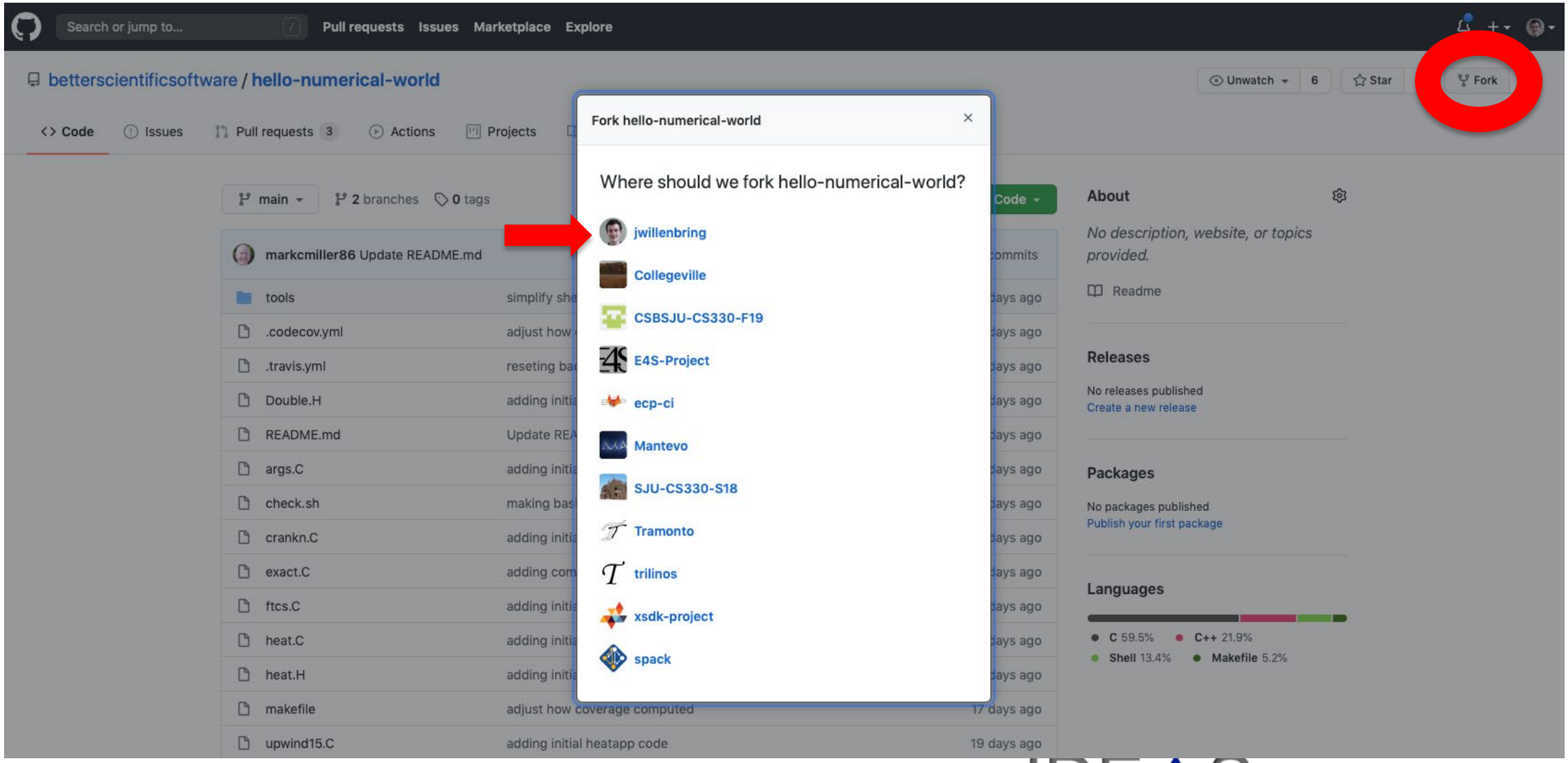
- Fork repository (once)
- Clone the fork (once)
- Create and checkout branch
  - Base branch on current development or other appropriate version for each feature
- Modify and commit code
- Push change to fork
- Issue pull request to upstream repository
- Review pull request
- CI testing (covered in upcoming module)

Git Workflow for the Heat Equation Example

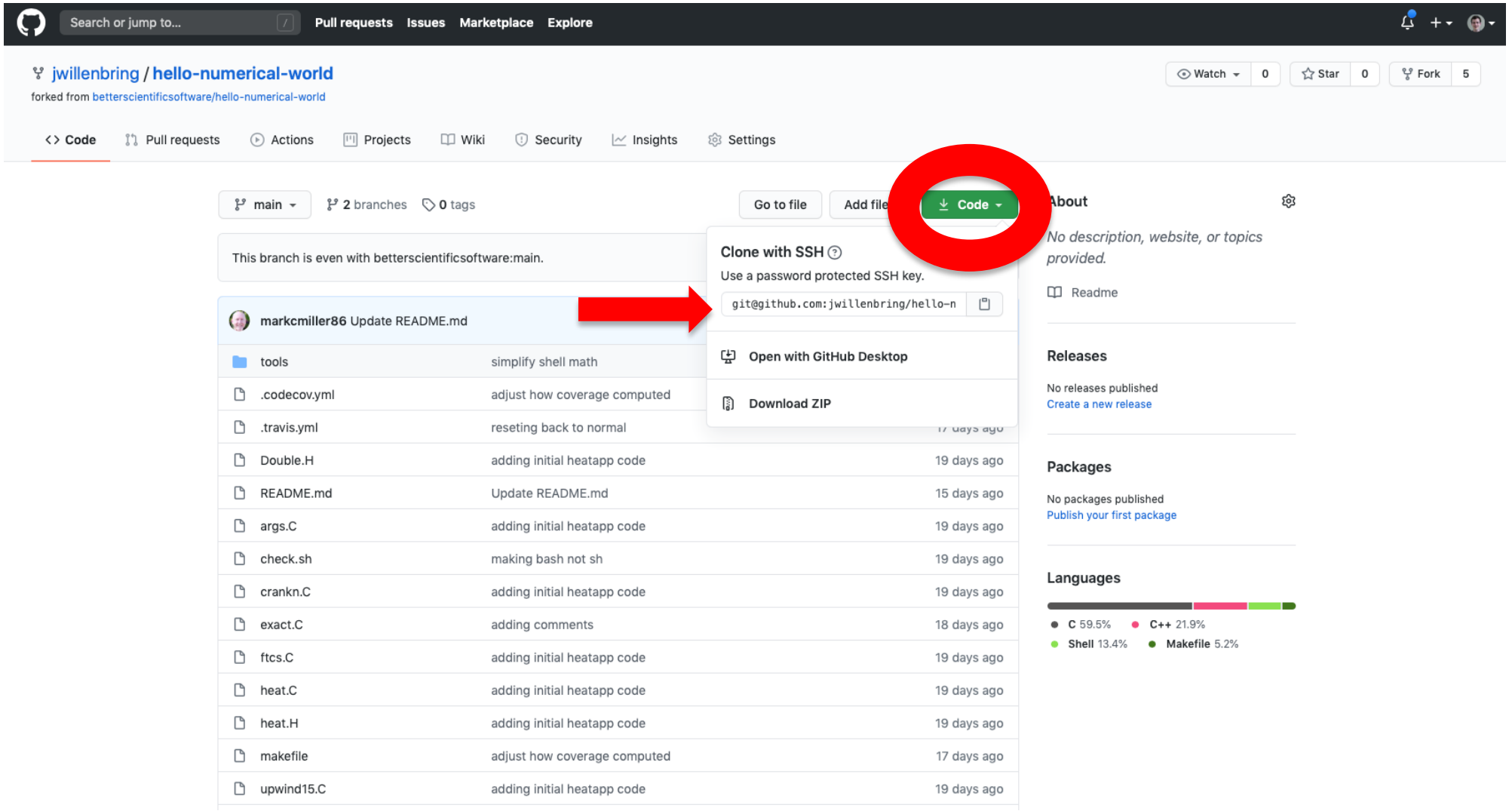
- Developers
  - Work on feature branches in their forks
    - Using forks requires contributors to have only read-access to primary repository
  - Issue pull requests for changes
    - Natural opportunity to review and test all changes

- Pull requests
  - Are reviewed by at least one developer (not the author)
  - Undergo CI testing prior to merging

Fork the Repository



Find the Path to Clone



Clone the fork; Create and Checkout a New Branch

```
s988335:repos jmwille$ git clone git@github.com:jwillenbring/hello-numerical-world.git
Cloning into 'hello-numerical-world'...
Enter passphrase for key '/Users/jmwille/.ssh/id_rsa':
remote: Enumerating objects: 102, done.
remote: Counting objects: 100% (102/102), done.
remote: Compressing objects: 100% (52/52), done.
remote: Total 102 (delta 54), reused 94 (delta 50), pack-reused 0
Receiving objects: 100% (102/102), 21.69 KiB | 120.00 KiB/s, done.
Resolving deltas: 100% (54/54), done.
s988335:repos jmwille$
```

```
s988335:repos jmwille$ cd hello-numerical-world/
s988335:hello-numerical-world jmwille$ git checkout -b issue-1000
Switched to a new branch 'issue-1000'
s988335:hello-numerical-world jmwille$
```


IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Modify and Commit Code

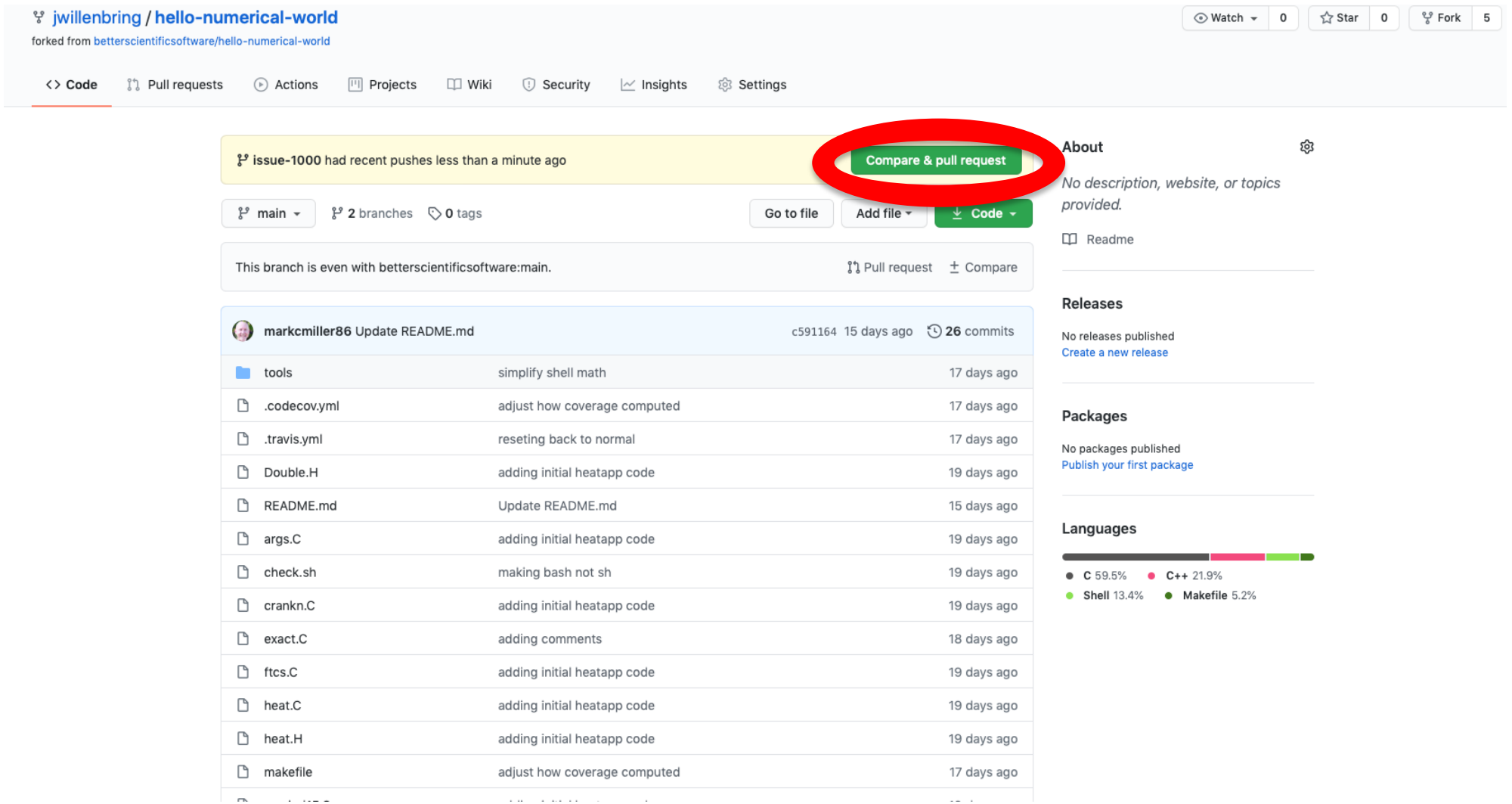
```
s988335:hello-numerical-world jmwille$ vi README.md
s988335:hello-numerical-world jmwille$ git diff
diff --git a/README.md b/README.md
index 3cd1a3c..b44c57e 100644
--- a/README.md
+++ b/README.md
@@ -22,7 +22,7 @@ is known as the _Diffusion Equation_ and also the [_Heat Equation_](https://en.w

 ### Simplifying Assumptions

-To make the problem tractable for this lesson, we make some simplifying assumptions...
+To make the problem tractable for this lesson, we make some simplifying assumptions:

 1. The thermal diffusivity, ![](http://latex.codecogs.com/gif.latex?%5Calpha),
    is constant for all _space_ and _time_.
s988335:hello-numerical-world jmwille$
```

```
s988335:hello-numerical-world jmwille$ git add README.md
s988335:hello-numerical-world jmwille$ git commit -m "replaced ... with :"
[issue-1000 1c3a901] replaced ... with :
 1 file changed, 1 insertion(+), 1 deletion(-)
s988335:hello-numerical-world jmwille$
```

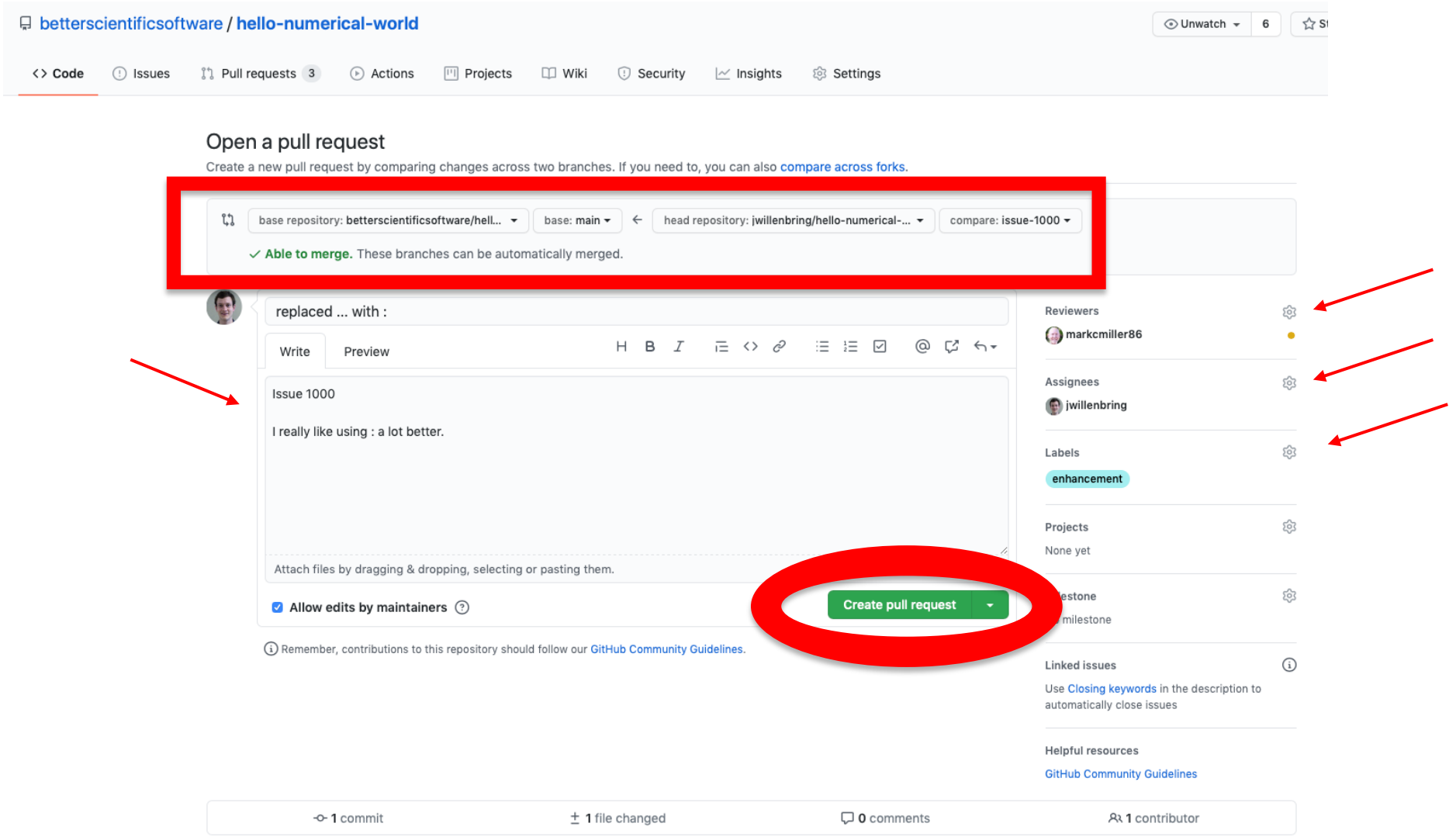
Push Change to Fork

```
s988335:hello-numerical-world jmwille$ git remote -vv
origin  git@github.com:jwillenbring/hello-numerical-world.git (fetch)
origin  git@github.com:jwillenbring/hello-numerical-world.git (push)
s988335:hello-numerical-world jmwille$ git branch
* issue-1000
  main
s988335:hello-numerical-world jmwille$ git push origin issue-1000
Enter passphrase for key '/Users/jmwille/.ssh/id_rsa':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 310 bytes | 310.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote:
remote: Create a pull request for 'issue-1000' on GitHub by visiting:
remote:      https://github.com/jwillenbring/hello-numerical-world/pull/new/issue-1000
remote:
To github.com:jwillenbring/hello-numerical-world.git
 * [new branch]      issue-1000 -> issue-1000
s988335:hello-numerical-world jmwille$
```

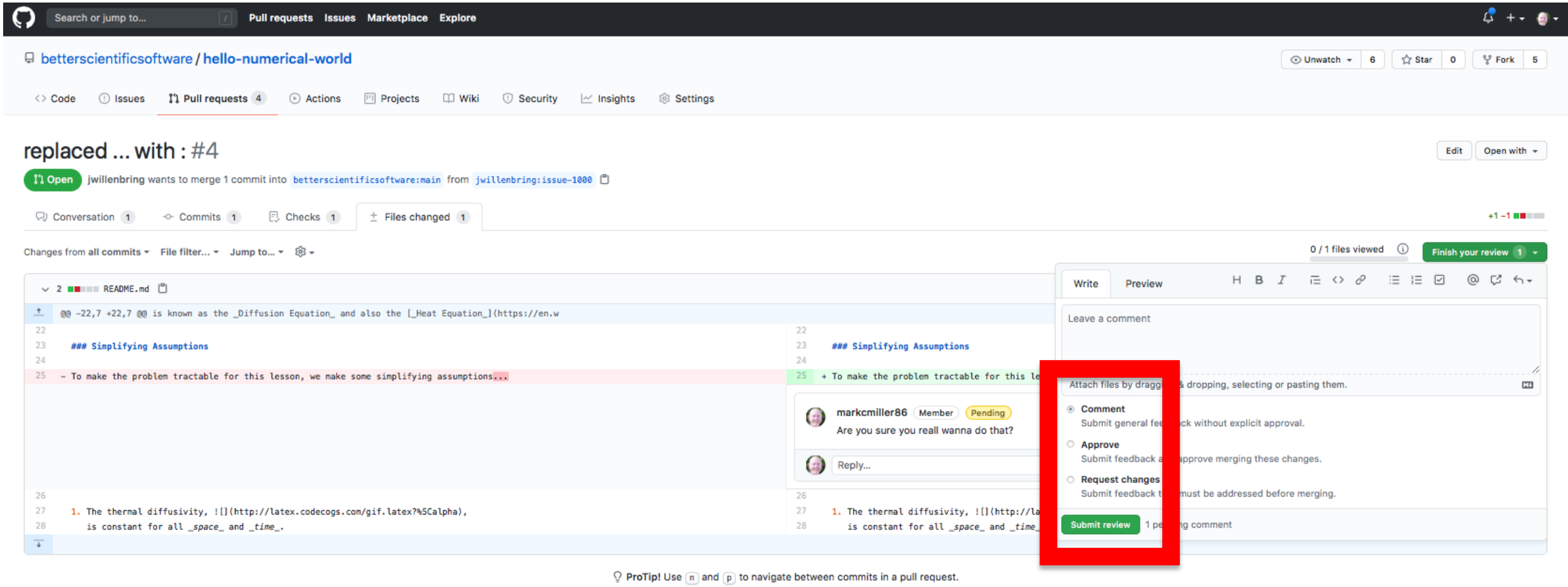
Issue Pull Request to Upstream Repository



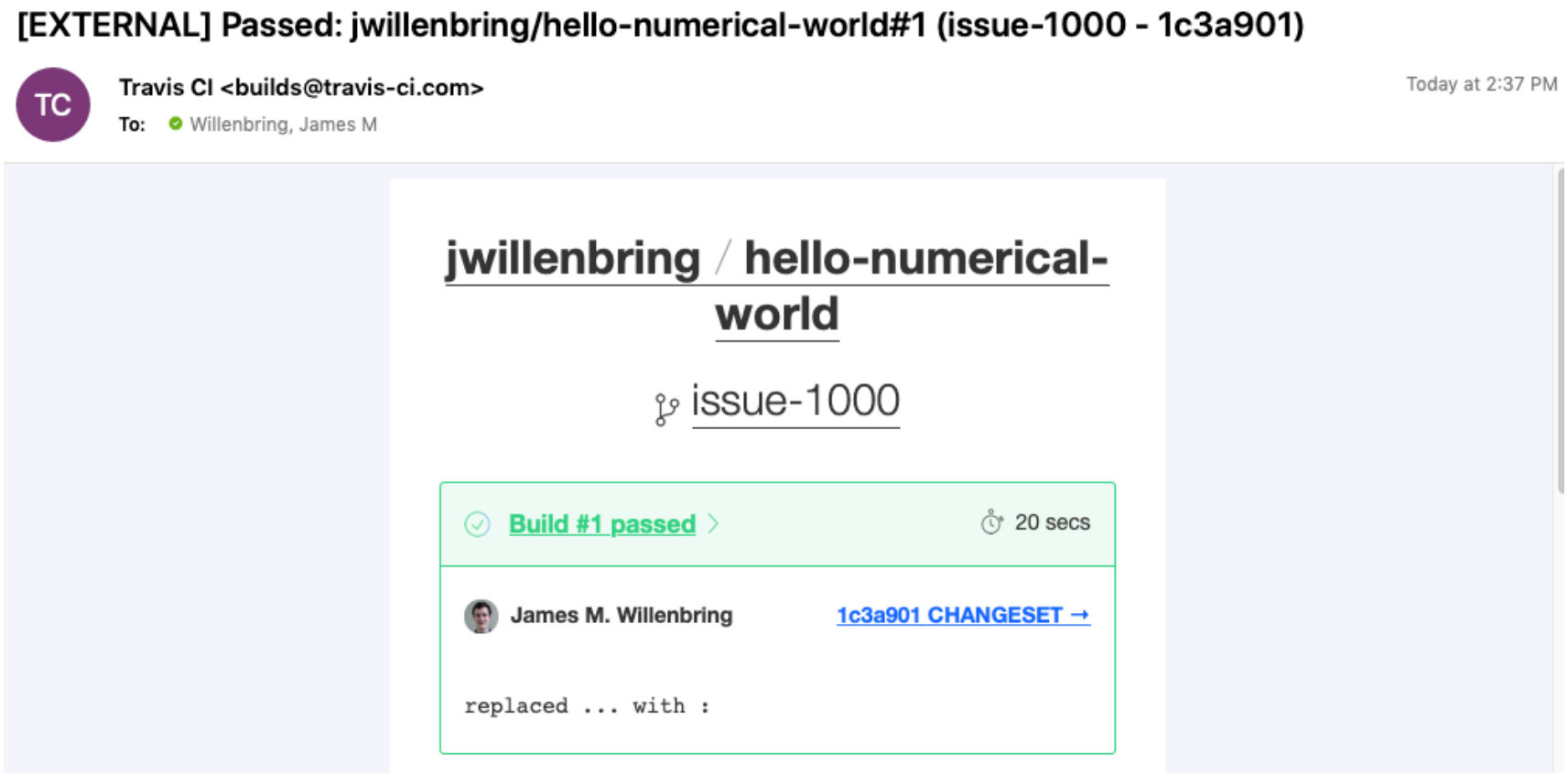
Issue Pull Request to Upstream Repository



Review Pull Request



CI Testing for PR



This will be covered in the CI module