



# Git Workflows

Patricia Grubel (she/her)  
Los Alamos National Laboratory

Better Scientific Software tutorial @ Improving Scientific Software 2022

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



See slide 2 for  
license details

LA-UR-21-29292



# License, Citation and Acknowledgements

## License and Citation

- This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/) (CC BY 4.0). 
- **The requested citation the overall tutorial is: David E. Bernholdt, Patricia A. Grubel, Rinku K. Gupta, and David M. Rogers, Better Scientific Software tutorial, in Improving Scientific Software conference, online, 2022. DOI: [10.6084/m9.figshare.19416767](https://doi.org/10.6084/m9.figshare.19416767)**
- 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.

# Content

- Goal using Version Control with Git
- Git Workflow Mechanisms for Collaboration
  - Branches
  - Pull Requests
  - Forks
- Code Review
- Exposure to workflows of different complexity
- Collaboration using Git Workflows for CSE projects
- What to think about when evaluating different workflows

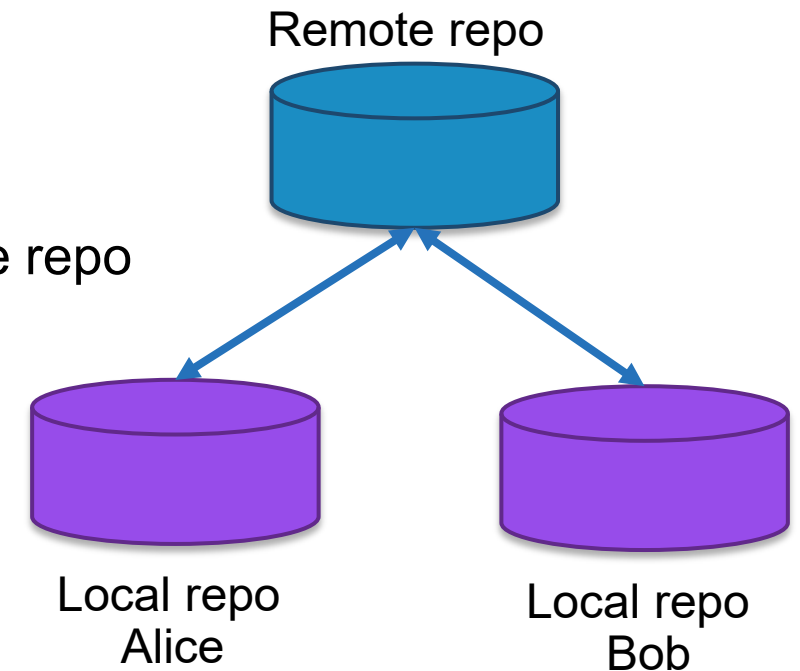
# Goal

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 the main branch



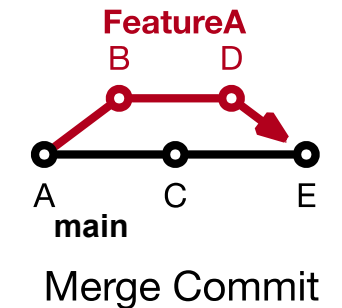
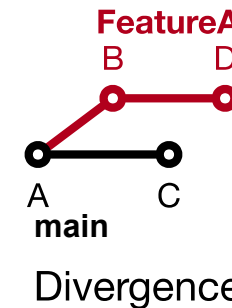
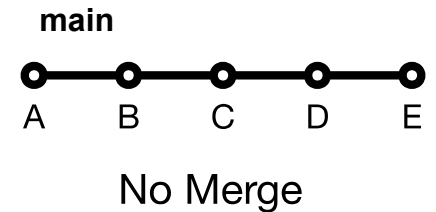
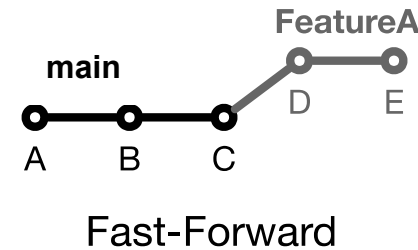
# Git Workflow Mechanisms for Collaboration

- 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 contributions from external collaborators that have read access only
  - Controls on original repo remains with the team

# Branches

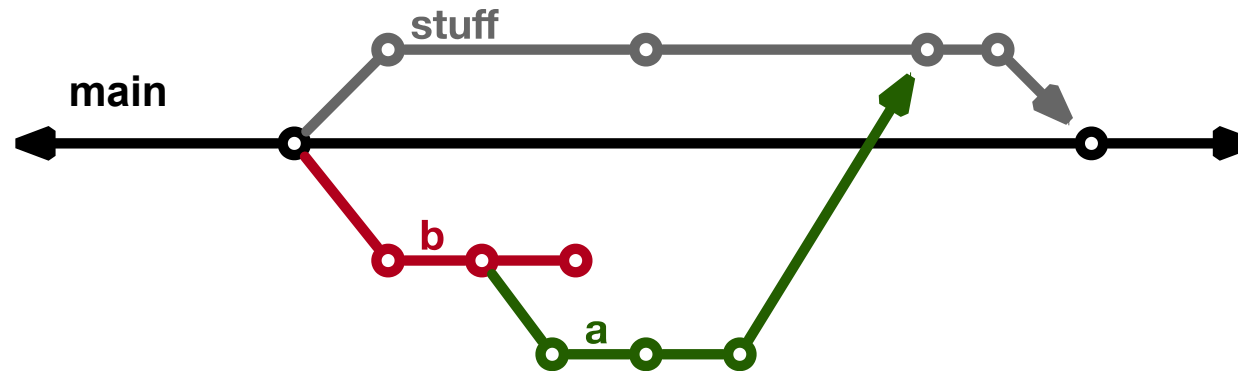
Branches are independent lines of development

- Use branches to protect main 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 main
- Integration occurs at merge commits



# 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?



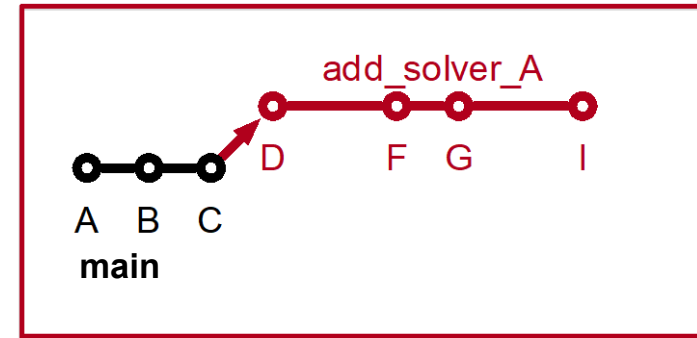


# Feature Branches

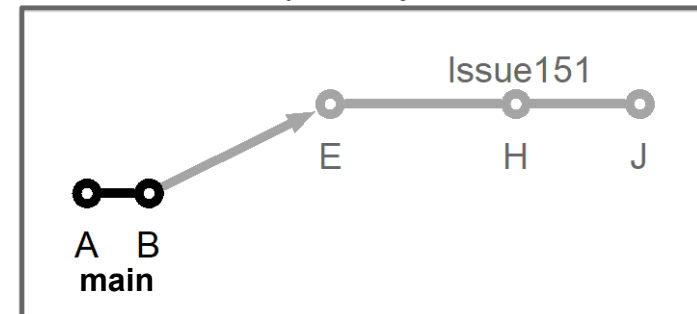
## Extend Centralized Workflow

- 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

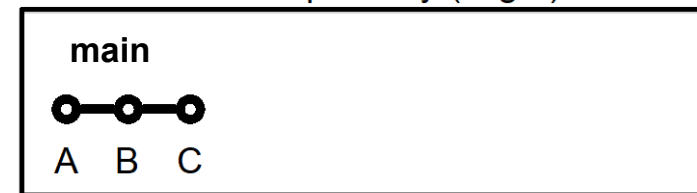
Alice's Local Repository



Bob's Local Repository



Main Remote Repository (origin)

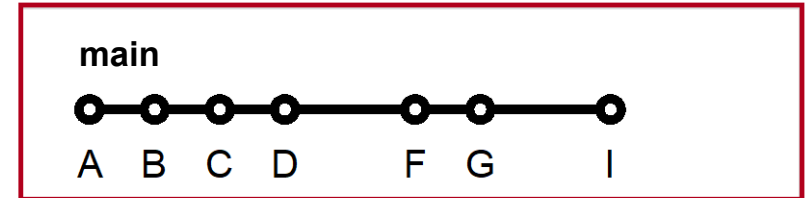


# Feature Branch Divergence

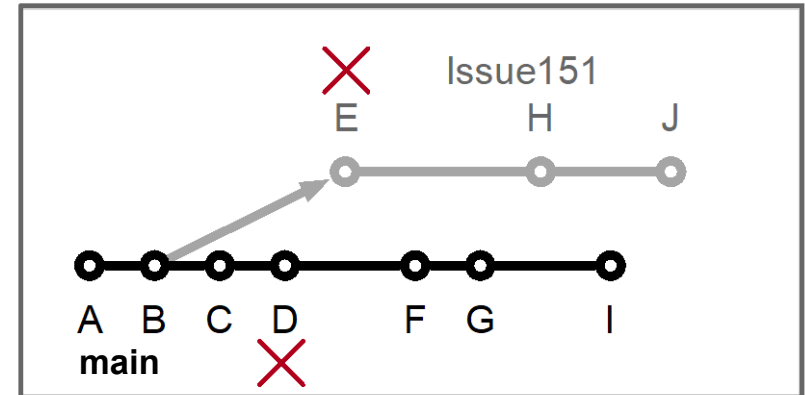
Alice integrates first without issue

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

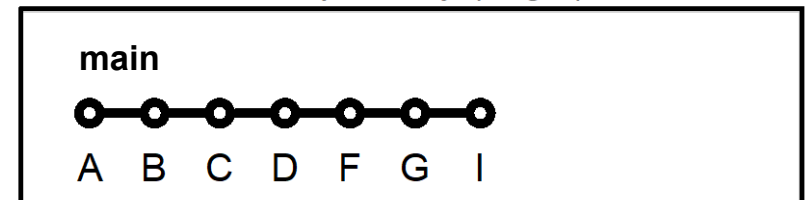
Alice's Local Repository



Bob's Local Repository



Main Remote Repository (origin)

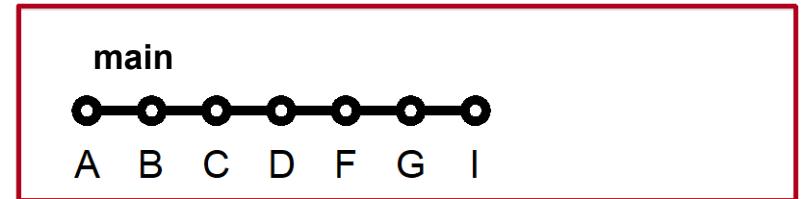


# Feature Race Condition

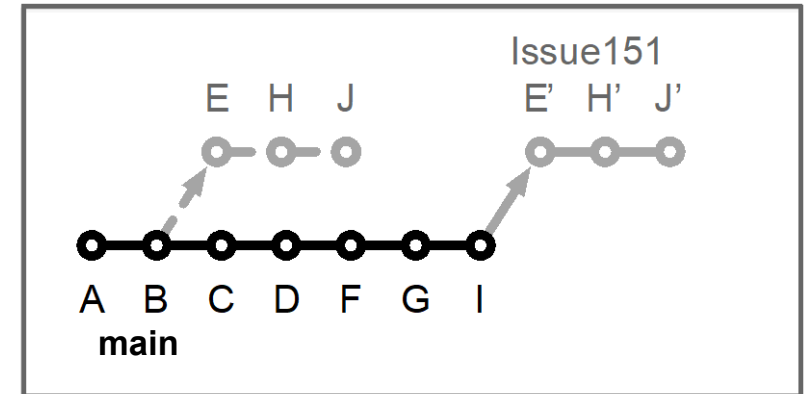
Integration occurs on Bob's local repo

- Bob laments not having fast-forward merge
- Bob **rebases** local feature branch to latest commit on main
  - 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

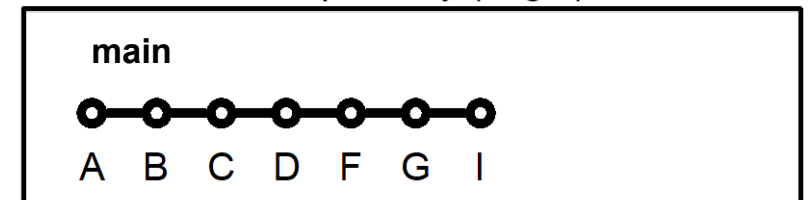
Alice's Local Repository



Bob's Local Repository



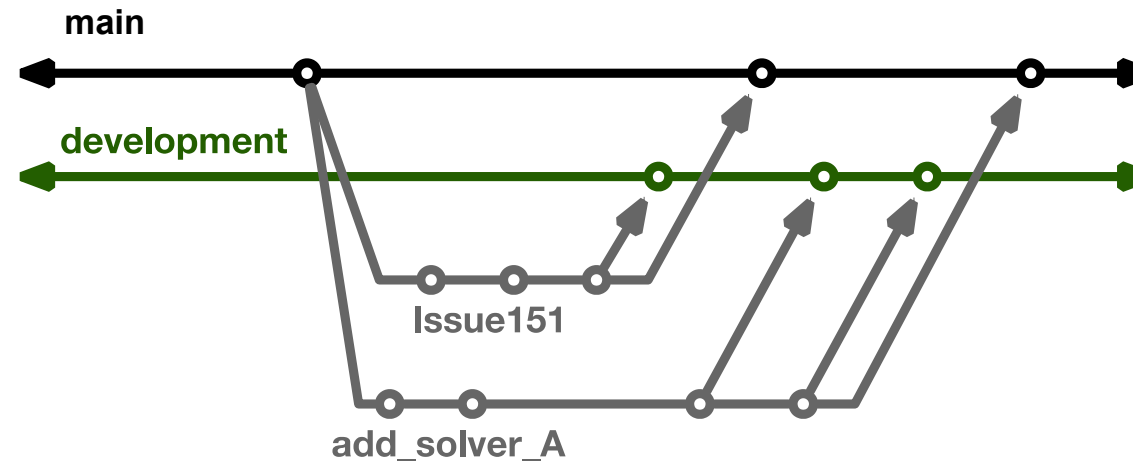
Main Remote Repository (origin)



# More Branches

## Branches with infinite lifetime

- Base off of main 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 team and 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 *upstream* repository
    - Forks of public repositories are public
    - External collaborators 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 main)
  - 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

# Code Review – What Peer Code Review Can Provide

- Allows discussion of proposed changes
  - Iterations for better code
  - Discussions and reviewing allow more understanding of the code
- Ensures requested change/feature met
- Evaluates impact of the change
  - Breakages
  - Interactions with other parts of code
- Ensures coding guidelines are met
- Improves practices by learning
  - About other parts of the code
  - Helpful coding techniques by others

Blog: **How to code review in a Pull Request**

Author: Hugo Sousa - March 17, 2021

<https://blog.codacy.com/how-to-code-review-in-a-pull-request/>

# Code Review - Improvement and Practices

- Helpful practices for scientific research software
  - Make code review process formal with structured guidelines
  - Allocate sufficient time in the development process to perform code review
  - Try to ensure at least one science review and one technical review
  - Timely reviews - provide quick feedback to incoming review requests
  - Train reviewers on how to phrase good feedback
  - Train developers to accept comments to improve their code
  - Include automatic code review tool and train reviewers in best use practice of the tool

## Testing and Code Review Practices in Research Software Development

Presenter: Nasir Eisty – September 9, 2020

<https://ideas-productivity.org/events/hpc-best-practices-webinars/#webinar044>

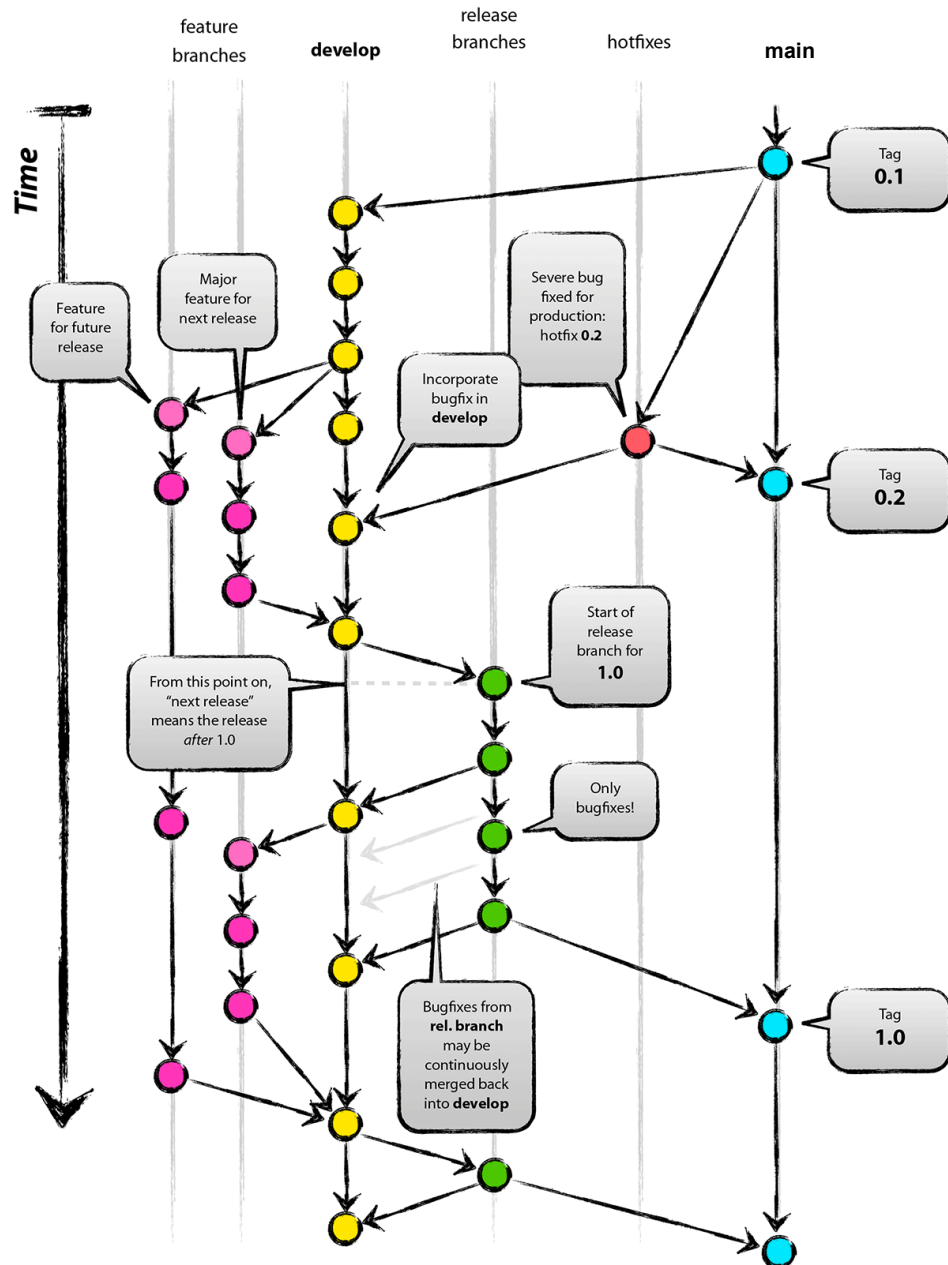


# Git Workflow Models of Different complexity

## 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 main 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

## Key Ideas

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

# GitLab Flow

[https://docs.gitlab.com/ee/workflow/gitlab\\_flow.html](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

## Key Ideas

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

# Collaboration using Git Workflows for CSE projects

- Trilinos Workflow
- Open MPI Workflow
- Flecsi Workflow

# Current Trilinos Workflow

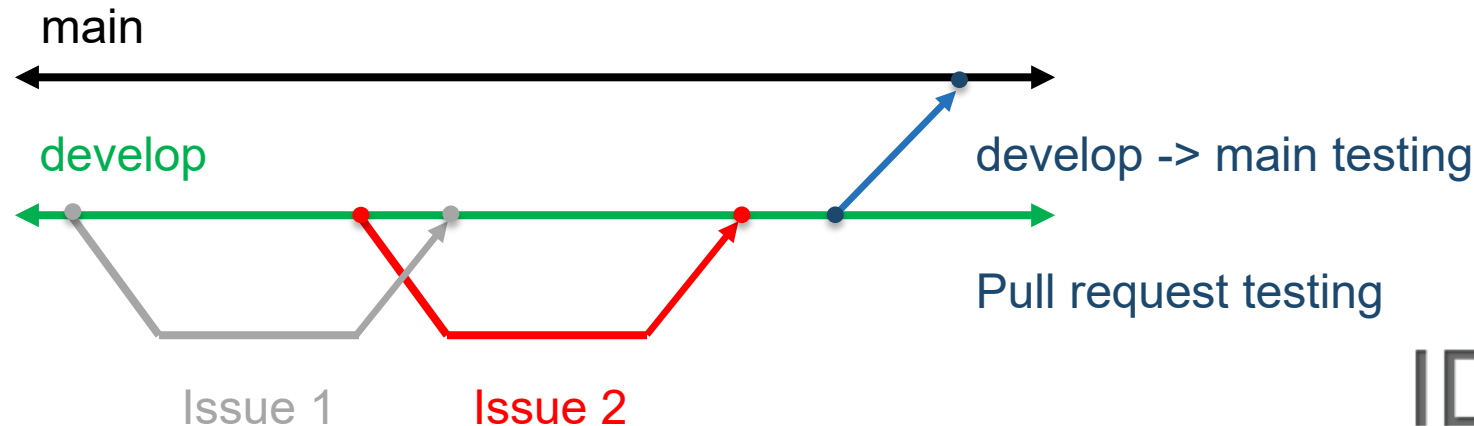
<https://trilinos.github.io/>

## 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 main

Workflow designed so that

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



# Current Open MPI Workflow

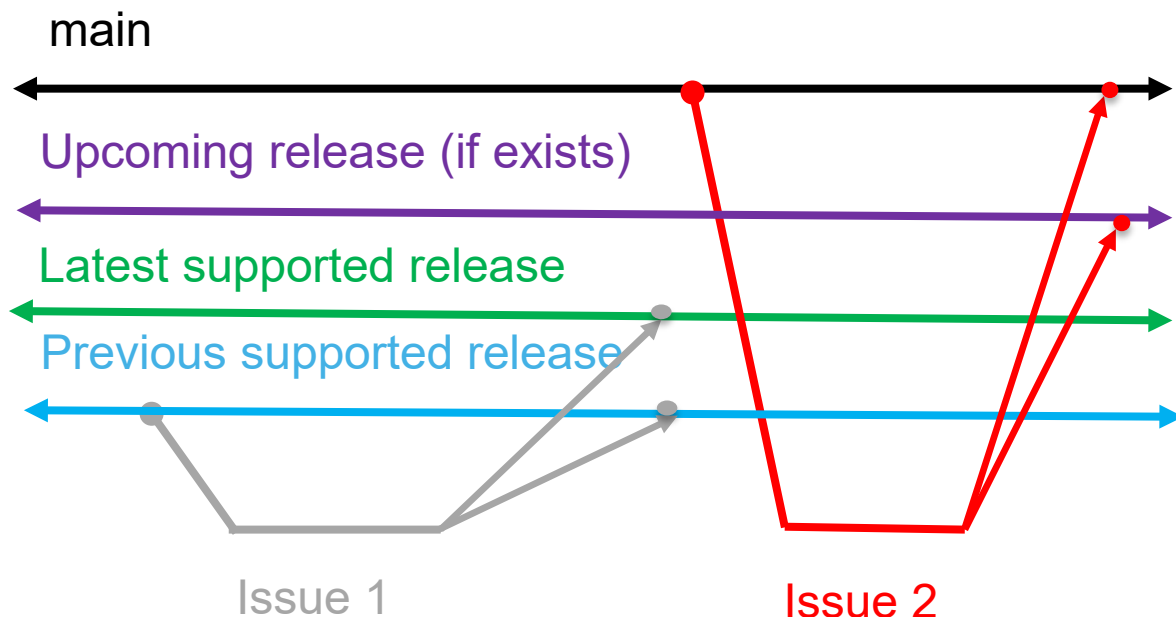
<https://www.open-mpi.org>

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
- Main and supported branches work at all times
- Developers work on main 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-built 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

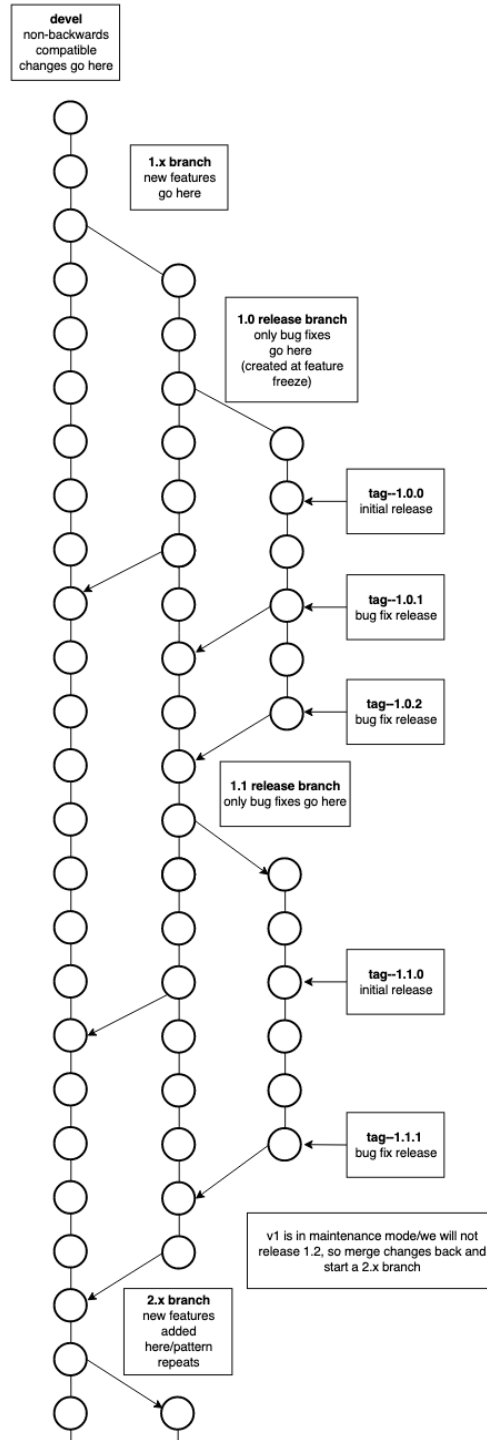
**Release** - (1.x, 2.x ...) named for major.minor version, correct issues, tags used for bug fixes.

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





# Considerations for Choosing a Git Workflow

Want to establish a clear set of policies that

- results in correct code on a particular branch (usually main),
- 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