Collaborative Software Development

Patricia Grubel (she/her)
Los Alamos National Laboratory

Better Software for Reproducible Science tutorial @ SC23

Contributors: David E. Bernholdt (ORNL), Patricia A. Grubel (LANL), Rinku K. Gupta (ANL), Michael A. Heroux (SNL), Mark C. Miller (LLNL), Jared O'Neal (ANL), David M. Rogers (ORNL), James M. Willenbring (SNL)

LA-UR-22-31408

License, Citation and Acknowledgements

License and Citation

- **The requested citation the overall tutorial is:** David E. Bernholdt, Patricia A. Grubel, David M. Rogers, and Gregory R. Watson, Better Software for Reproducible Science tutorial, in The International Conference for High-Performance Computing, Networking, Storage, and Analysis (SC23), Denver, Colorado, 2023. DOI: 10.6084/m9.figshare.24226105.
- Individual modules may be cited as *Speaker, Module Title*, in *Tutorial Title*, …

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 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 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 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

Outline

- Why Develop Software Collaboratively?
- Tools
- Version Control - Using Git
- Git Workflows
- Using Agile for Scientific Software
- Kanban
- Code Review
- Software Licensing

Why Develop Software Collaboratively?

Why Develop Software Collaboratively?

- Most real-world projects involve teams rather than individuals
- Collaboration has advantages
  - Produces working code quicker
  - Better design from considering more points of view and experience
  - Can be more enjoyable, even a social experience
- Collaboration has challenges
  - Logistics (time zones etc.)
  - Communication
  - Ensuring everyone is working from the same version and can contribute equally
  - Understanding other's code and what they intended

Tools

Kanban

agile

Git

feature branch

lifetime branches

JIRA

code review

Git Workflows

Pull Requests

issue tracking

Scrum

policies

Trello

Github

version control

project boards

Gitlab

Why do we need tools?

- To keep organized when working on a team
- Stay on the same page, same knowledge of the project, work, goals etc.
- Keep track of, address and prioritize bug fixes, feature requests, etc.
- Tools capture and mediate information about the collaborative process

A major tool used for version control is Git …

Version Control – Using Git

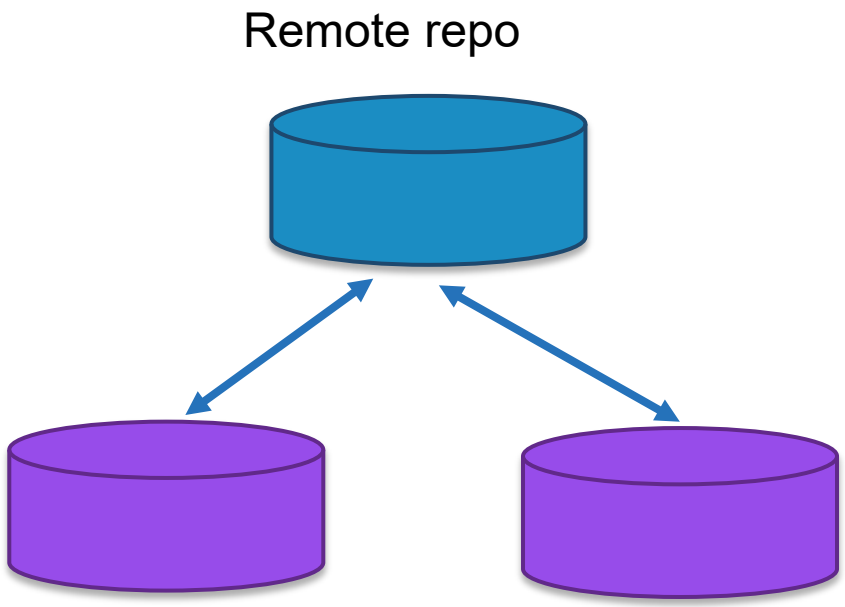
Version Control - Git

- Version control (or revision control) is a means of tracking changes made to source code
- Various *hosting services* are available - GitHub, GitLab, Bitbucket, etc.
- Main features of Git used for collaboration*
  - Branches - allows separate development for features or fixes on the same repo
  - Pull Requests (PRs) - Enables code review and testing before merge
  - Clones - allows each developer their own working copy
  - Forks - allows those outside the team to collaborate (open source software)

* This tutorial will cover the first two since knowledge of them is needed for the development workflow using Git, referred to as ”Git Workflow”

Version Control - Git

- Allows collaboration while ensuring everyone works from the same version

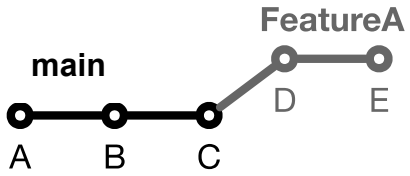


Use of Branches

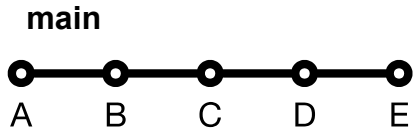
- Enables independent development for features or fixes on the same repo
- Enables concurrent development by multiple developers
- Provides different types of Workflows
- Protects main branch
- Develop on a branch, test on the branch, and merge into main
- Integration occurs at merge commits

Feature Branches

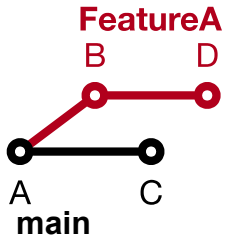
- Organize a new feature as a sequence of related commits in a branch



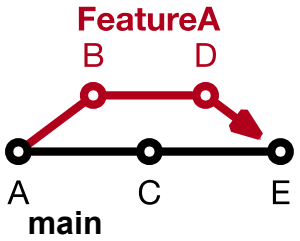
Fast-Forward



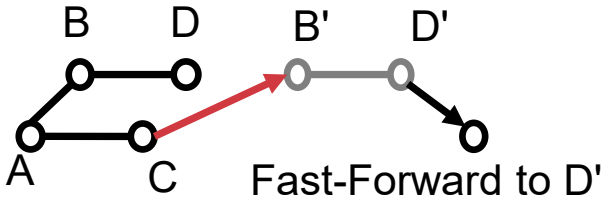
No Merge



Divergence



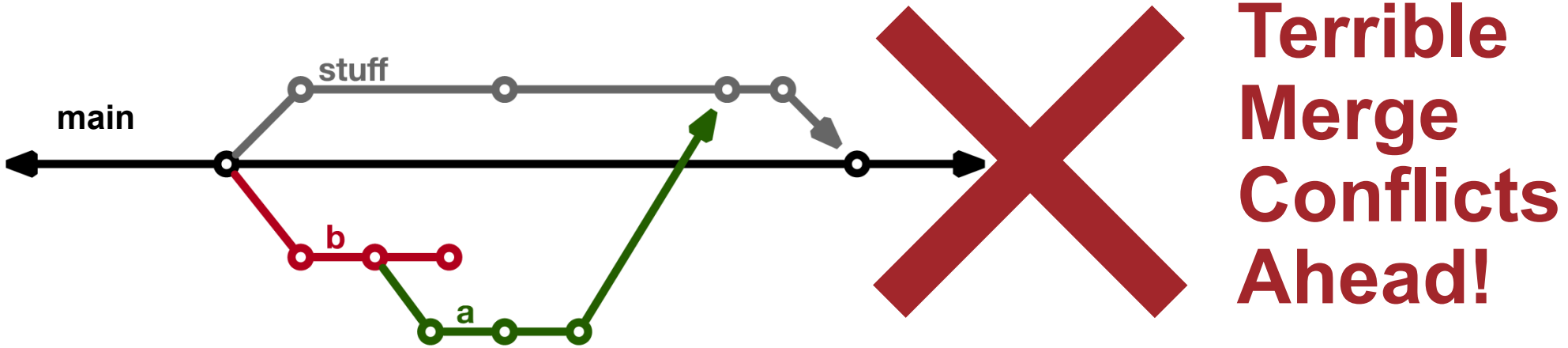
Merge Commit



Rebase D onto C

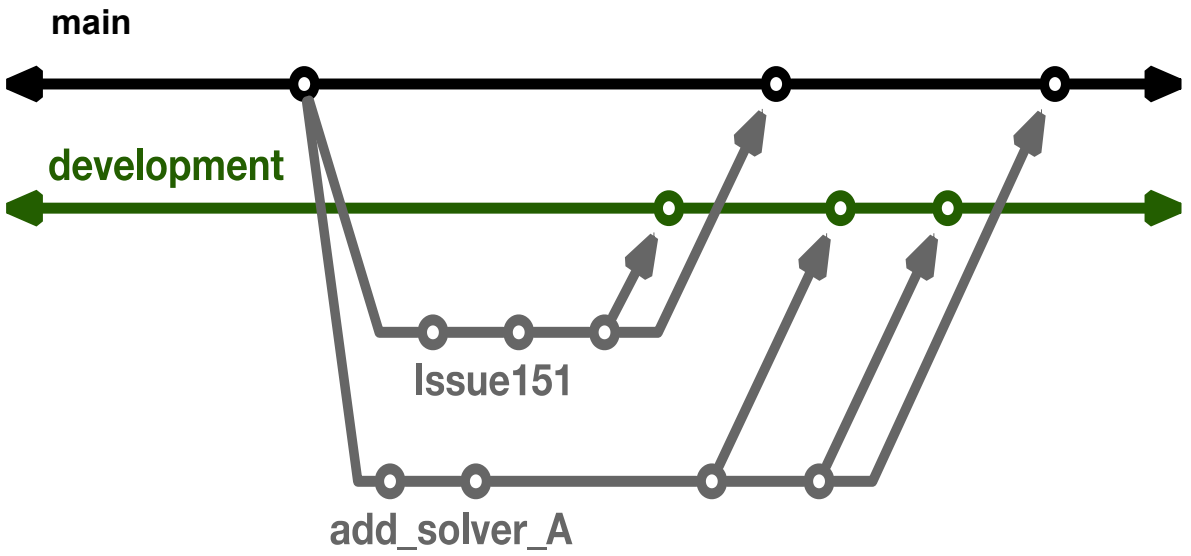
How Do We Control Project Branch Complexity?

- 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 names linked to issue tracking system
  - Where do branches start and end?



Infinite Lifetime Branches

- Base off main branch
- Exist in all copies of a repository
- Each provides a distinct **environment**
  - Development, pre-production, release etc.



Branching Strategies

- What's stable, under what conditions?
- What's tested?
- Branch Lifetimes
  - Indefinite - main, development, stable
  - Short term – feature, fixes
  - Longer Period – based on release schedules
- Establish Workflow Policies

Pull Requests

Why use Pull Requests?

- Allows code 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 (may want particular expertise, may want to give someone knowledge)
- Set policies for merge
  - Enforce rules such as coding standards
  - Minimum number of reviewers
  - Protected branches – who can merge to certain branches etc.

What makes a good Pull Request?

- Covers “one thing”
  - One body of work
  - Independently manageable
- Avoid large PR’s - keep them small
- Break them up if they get too large
- Merge frequently (goes with keep them small)
- Good description – helps reviewers and users

Git Workflows

What is a Git Workflow?

- A Git workflow consists of the structure of the branches and policies
  - Designated lifetime branches
    - main, production, pre-production, development
  - Feature Branches
    - New features, bug fixes etc.
  - Schedules for releases (if applicable)
  - Testing
  - Branch Protections

Why use a Git Workflow?

- Provides collaboration in a consistent and productive manner
- Team members are on the same page for development
- Policies make it clear
  - How to use the branching structure
  - What branches are protected
  - What is tested and when
  - What branches are stable
- Helps team members understand the current state of the code

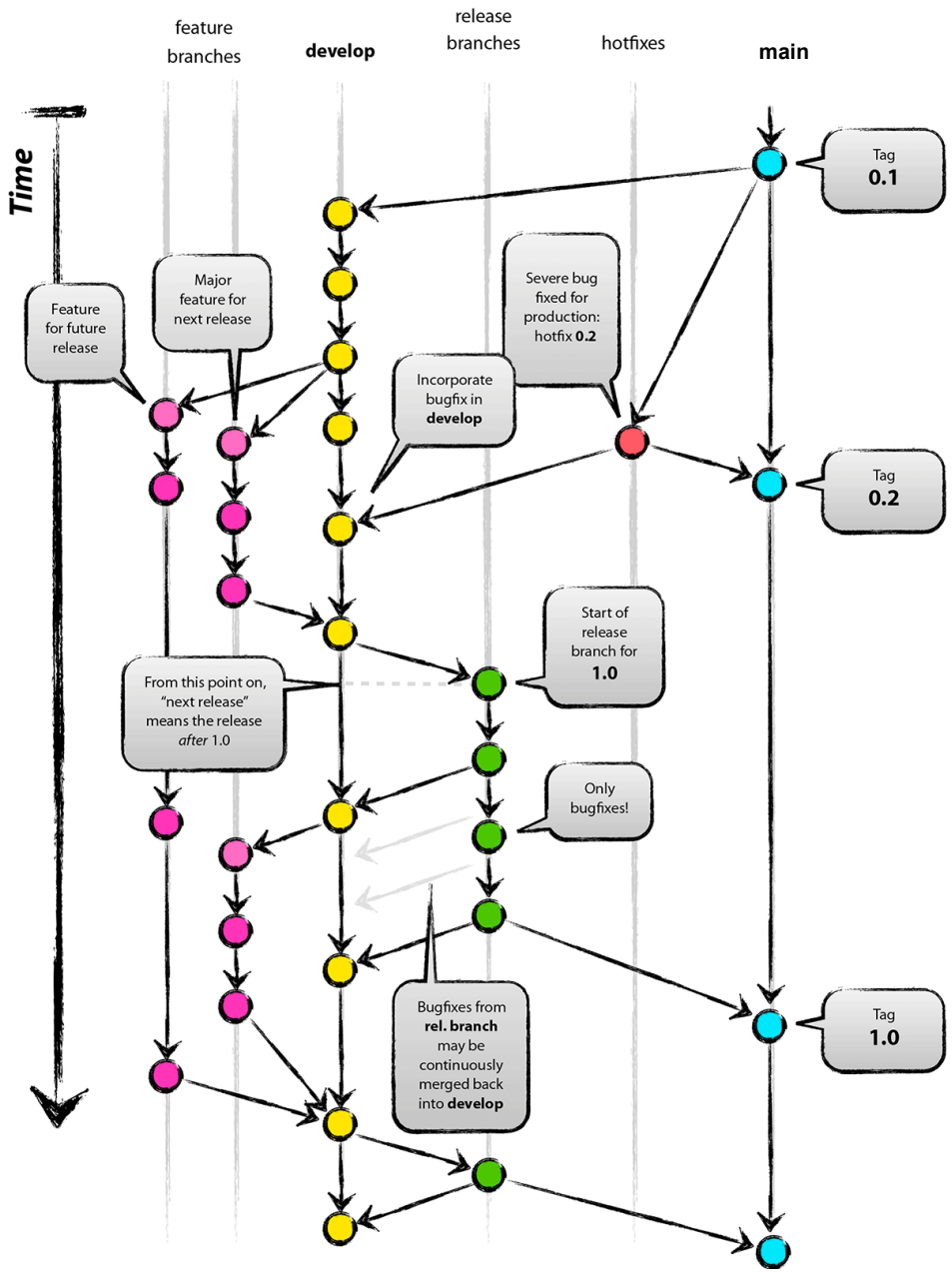
Commonly Known Git Workflows

- Git Flow
- GitHub Flow
- GitLab Flow

Design Patterns for Git Workflows by Roscoe A. Bartlett

provides an extensive discussion of the concepts behind git workflows

Git Flow



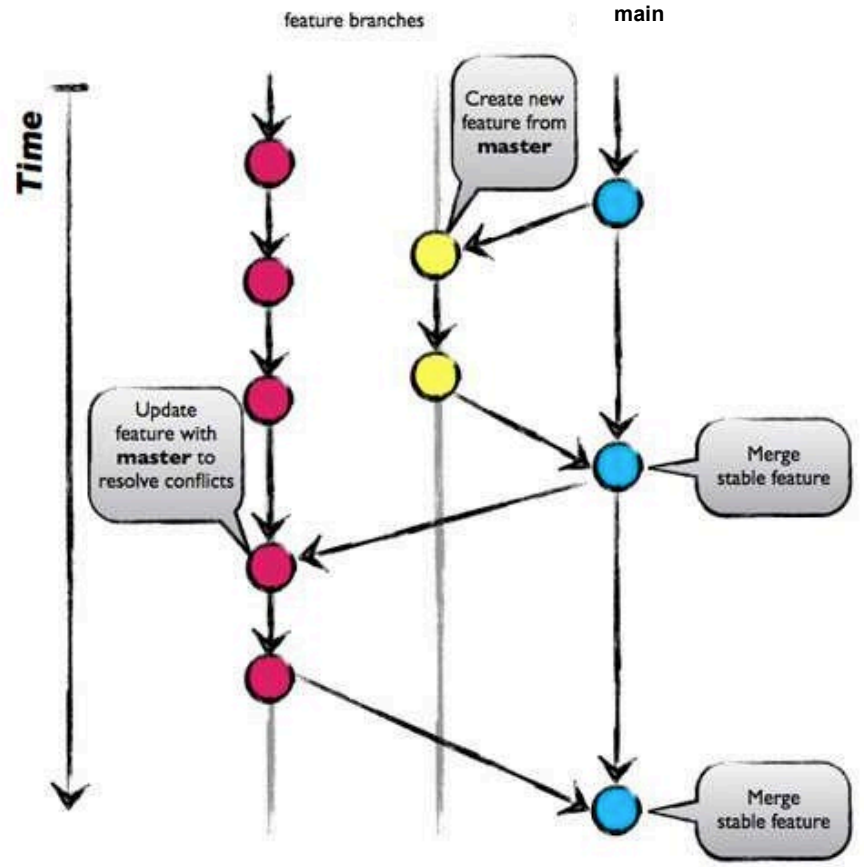
- Full-featured workflow
- Increased complexity
- Designed for Software with official releases
- Feature branches based off of develop
- Git extensions to enforce policy
- How are develop and main synchronized?
- Where do merge conflicts occur and how are they resolved?

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

GitHub Flow

https://docs.github.com/en/get-started/quickstart/github-flow

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



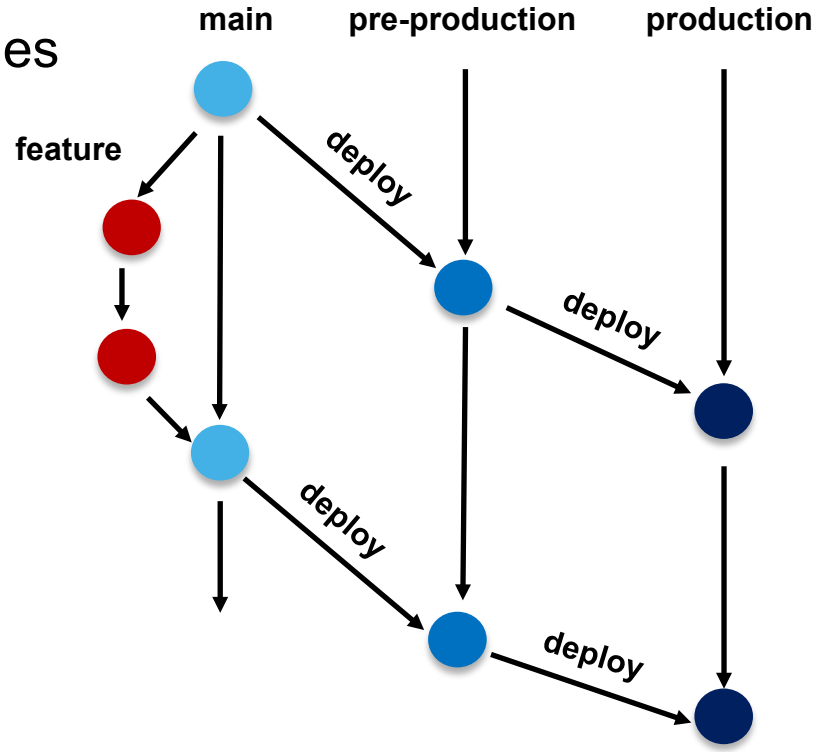
Key Ideas

1. All commits in the main branch are **deployable**
2. Base feature branches off 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/topics/gitlab_flow.html

- Published as viable alternative to Git Flow & GitHub Flow
- Semi-structured release schedule
- Simplifies difficulties 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

Git Workflows from CSE projects

- Trilinos
- OpenMPI
- Flecsi

Current Trilinos Workflow

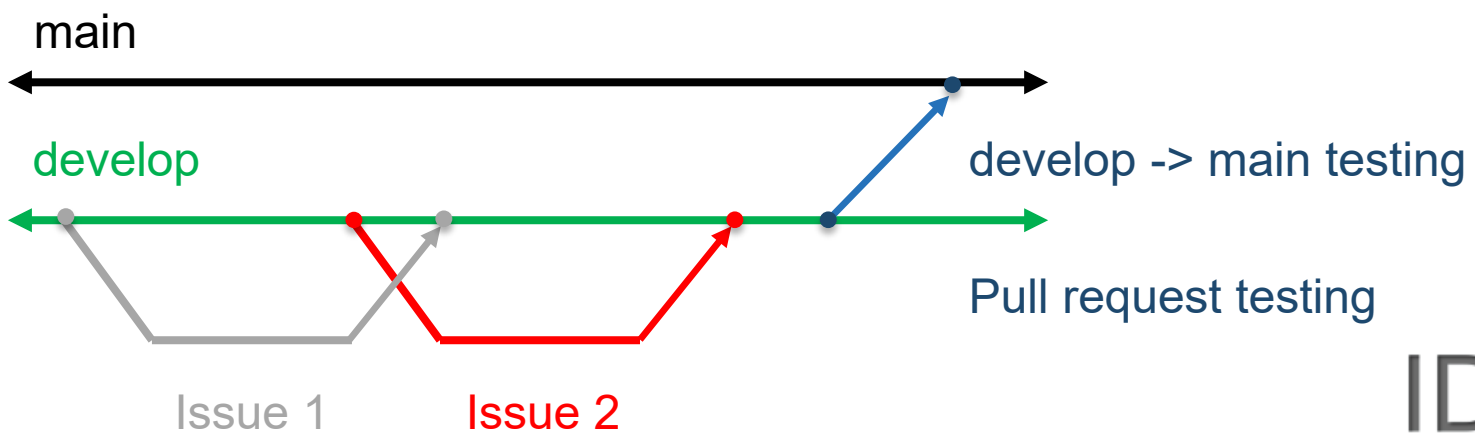
https://trilinos.github.io/

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

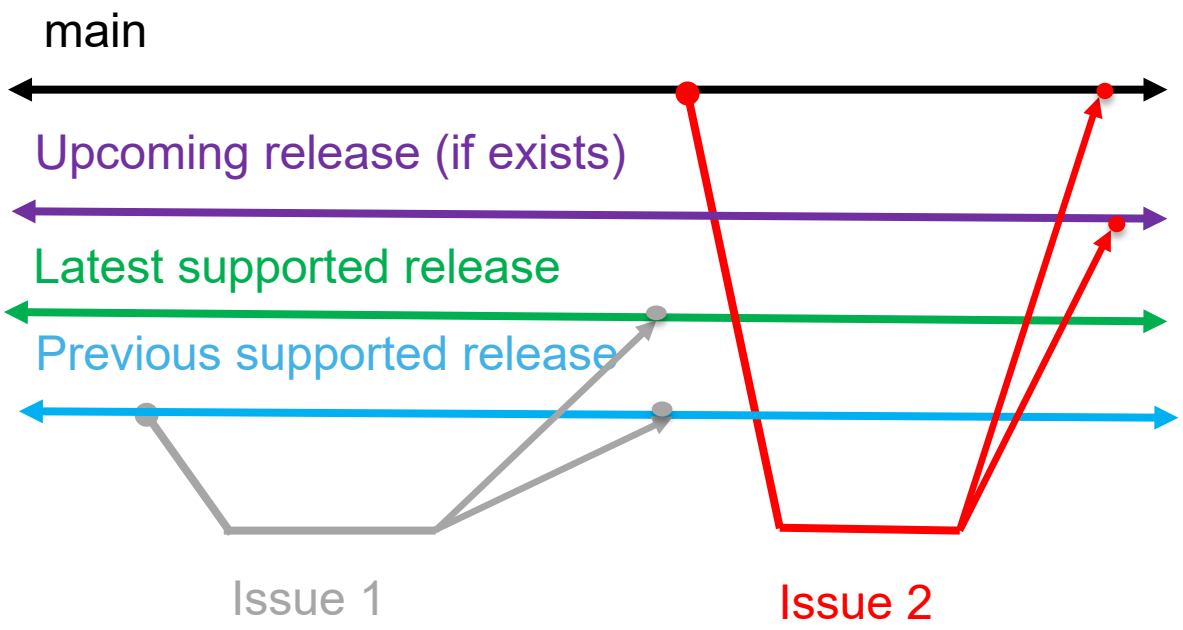
https://www.open-mpi.org

Versioning:

Major versions - break compatibility

Minor versions – visible

Releases correct issues



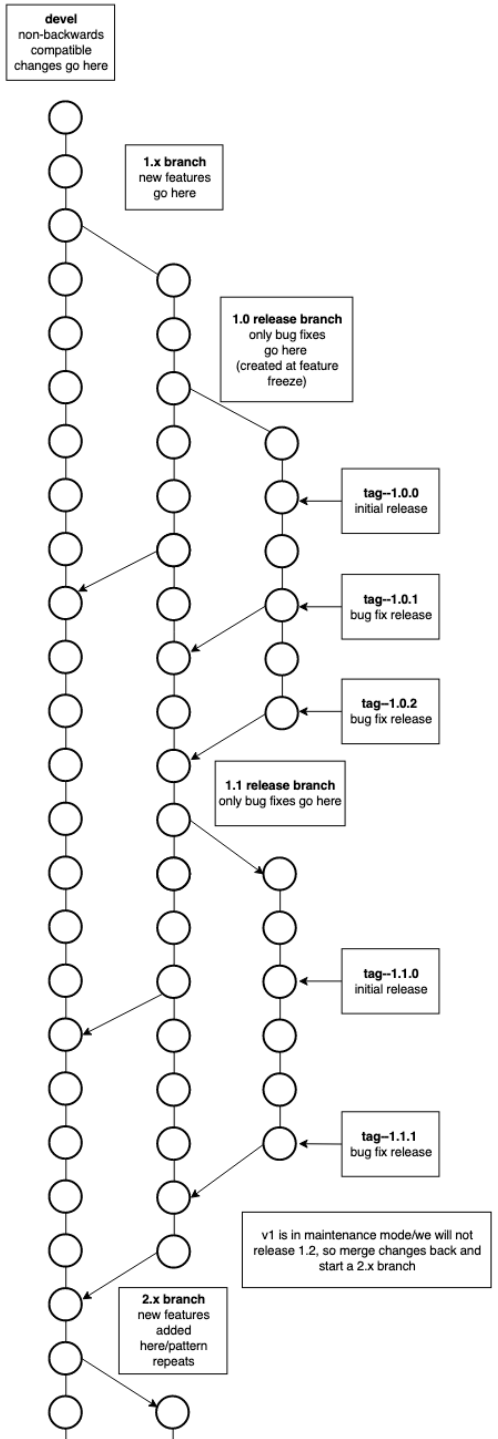
Workflow designed so that

- 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

Develop -> Feature -> Release Workflow

Develop (incompatible) 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

Guidelines for establishing a Git Workflow

- Communicating it to your collaborators
  - Include workflow in Contributing Guide
  - Establish conventions for branch naming (issues, major/minor versions)
- Enforce workflow
  - Branch protections
  - Limiting who can push/merge
  - Testing & review requirements
- **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

Using Agile for Scientific Software

Why Agile?

- Fits the research experience better than heavier-weight approaches
  - Aligns more naturally with how scientific progress is made
- Well-suited for scientific software efforts (when tailored correctly)
  - Works well for small teams
  - Provides meaningful, beneficial structure that promotes
    - Productivity
    - Productization
    - Sustainability
    - Flexibility in requirements
    - Communication

What is Agile?

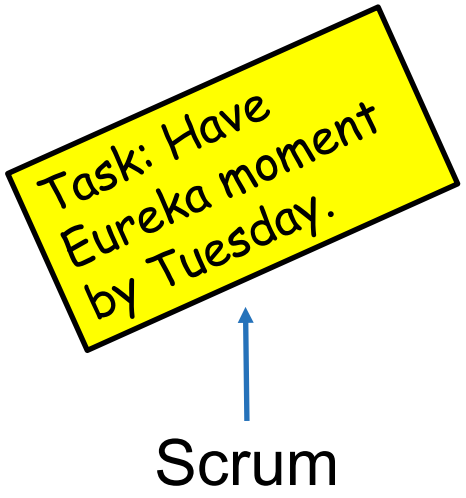
- Agile is not a software development lifecycle model
- Some “common misconceptions” about agile
  - I don’t write documentation
  - I don’t do formal requirements, design, or really test…
- Agile is not an excuse to do sloppy work
- Some people consider agile to be synonymous with Scrum
  - From Atlassian: Scrum is a framework that helps teams work together
  - Scrum is Agile, Agile is not (only) Scrum
  - A square is a rectangle, not all rectangles are squares
  - Agile is not Kanban either

What is Agile?

http://agilemanifesto.org/

Getting Started with Agile

- Agile principles are not hard and fast rules
- Try adopting a few Agile practices
  - Following a rigid, ill-fit framework usually leads to failure



- Kanban is a good starting framework
  - Follow basic principles
  - Add practices when advantageous
  - Better than removing elements from Scrum

Kanban

Basic Kanban

Backlog	Ready	In Progress	Done
• Any task idea • Trim occasionally • Source for other columns	• Task + description of how to do it. • Could be pulled when slot opens. • Typically comes from backlog.	• Task you are working on *right now.* • **The only Kanban rule: Can have only so many "In Progress" tasks.** • Limit is based on experience, calibration. • **Key: Work is *pulled*. You are in charge!**	• Completed tasks. • Record of your life activities. • Rate of completion is your "velocity".

Notes:

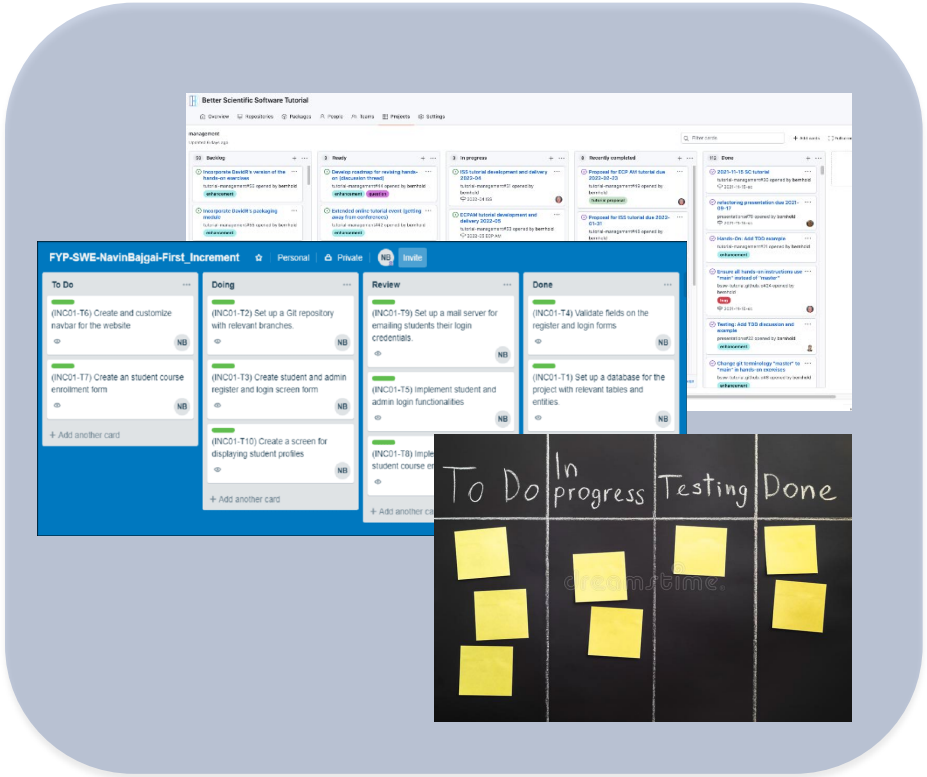
- Ready column is not strictly required, sometimes called "Selected for development".
- Other common column: In Review
- Can be creative with columns:
  - Waiting on Advisor Confirmation
  - Under review
  - Blocked

Kanban principles

- Limit number of “In Progress” tasks
  - Must be tuned by each team
  - Common convention: 2n-1 tasks where n = # team members
- Productivity improvement:
  - Optimize “flexibility vs swap overhead” balance. No overcommitting.
  - Productivity weakness exposed as bottleneck. Team must identify and fix the bottleneck.
  - Effective in R&D setting. Avoids a deadline-based approach. Deadlines are dealt with in a different way.
- Provides a board for viewing and managing issues

Kanban tools

- Wall, whiteboard, blackboard: Basic approach.
- Software, cloud-based:
  - Trello, JIRA, GitHub or GitLab Issues & Project Boards
  - Many more.
- Trello (browser, Android, iPhone, iPad).
  - Can add, view, update, anytime, anywhere.
  - Different boards for different contexts
    - Effective when people are split on multiple projects



Big question: How many “active” tasks?

- No single answer. Choose something and adjust from there.
- Personal Kanban approach: Start with 2 or 3.
- Use a freeway traffic analogy:
  - Does traffic flow best when fully packed? No.
  - Same thing with your effectiveness.
- Spend time consulting board regularly.
  - Brings focus.
  - Enables reflection, retrospection.
  - Use slack time effectively.
  - When you get out of the habit, start up again.
  - Steers towards previously started tasks

Code Review

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

Investing in Code Reviews for Better Research Software (2022-10-12)
Presenters : Thibault Lestang, Dominik Krzemiński and Valerio Maggio
https://ideas-productivity.org/events/hpcbp-068-codereview

Testing and Code Review Practices in Research Software Development (2020-09-09)
Presenter: Nasir Eisty
https://ideas-productivity.org/events/hpcbp-044-testingpractices

Software Licensing

Software Licensing

- Any software you write is “born copyrighted”
  - Copyright holder is usually you (as author) or your employer (more likely)
- By default, that copyright is “all rights reserved” to the holder of the copyright
- Specifying a license controls how others can use or contribute to the software
  - Treat the license as a *tool* to help you accomplish your goals for the software
  - Potentially many considerations in choosing a license
- For open source licenses
  - Choose an Open Source Initiative approved license (don’t make up your own)
  - Try a tool like https://choosealicense.com/
- Make sure your chosen license is clearly expressed in your code repository
- For considerably more information see: Introduction to Software Licensing from Best Practices for HPC Software Developers webinar series