Continuous Integration

Greg Watson (he/him)
Oak Ridge National Laboratory

Better Scientific Software tutorial @ ISC 2022

Contributors: David E. Bernholdt (ORNL), Mark C. Miller (LLNL), David M. Rogers (ORNL), James M. Willenbring (SNL)

License, Citation and Acknowledgements

License and Citation

- **The requested citation the overall tutorial is: Anshu Dubey and Gregory R. Watson, Better Scientific Software Tutorial, in ISC High Performance, 2022, Hamburg Germany. DOI: 10.6084/m9.figshare.19781752**
- Individual modules may be cited as *Speaker, Module Title*, in Better Scientific Software tutorial, ISC, 2022 …

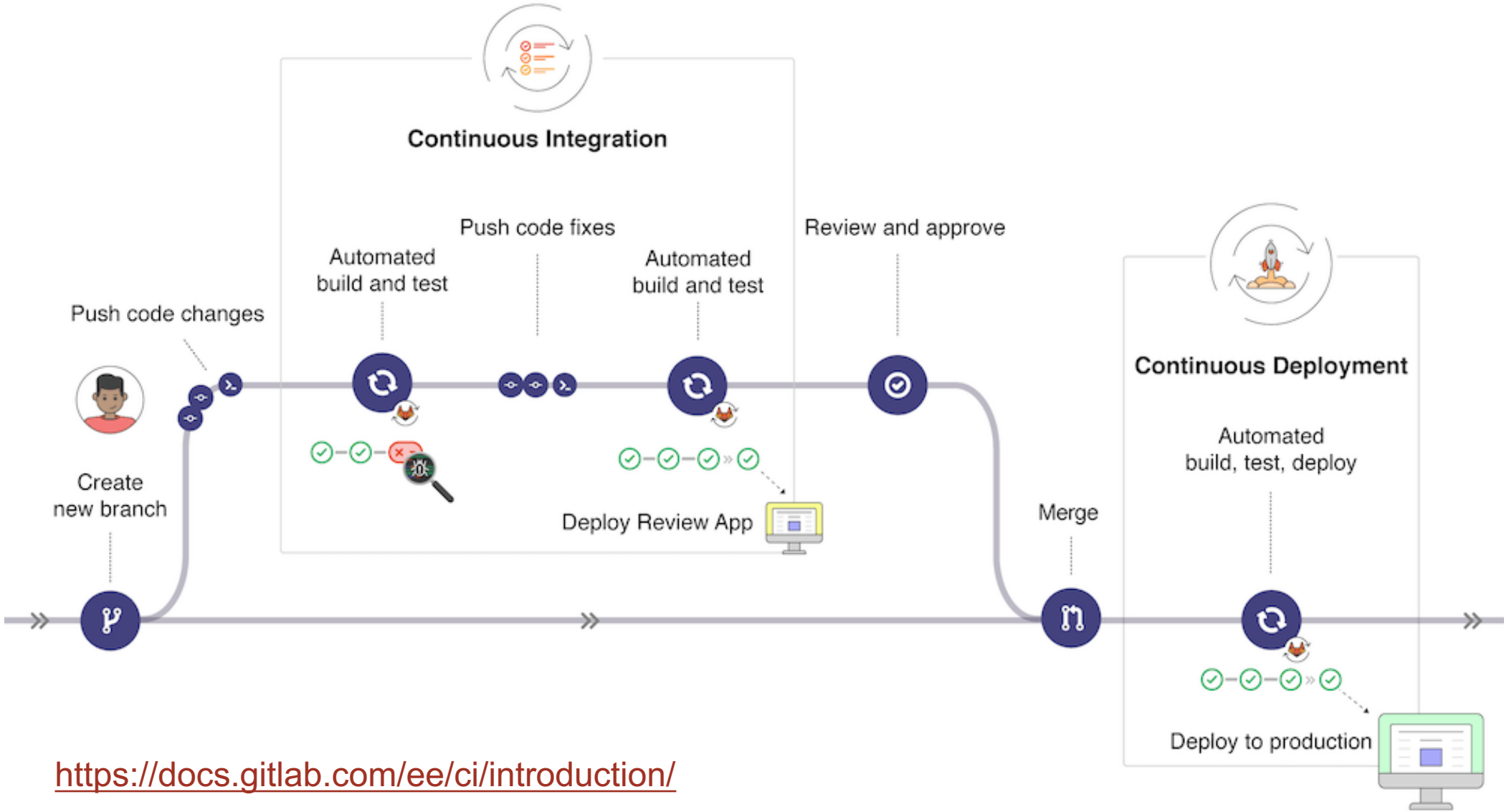
Acknowledgements

- 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

What is Continuous Integration (CI)



https://docs.gitlab.com/ee/ci/introduction/

CI Components

- Testing
  - Focused, critical functionality (infrastructure), fast, independent, orthogonal, complete, …
  - Existing test suites often require re-design/refactoring for CI
- Integration
  - Changes across key branches merged & tested to ensure the “whole” still works
    - Integration can take place at multiple levels
      - Individual project
      - Spack
      - E4S
  - Develop, develop, develop, merge, merge, merge, test, test, test…NO!
  - Develop, test, merge, develop, test, merge, develop, test, merge…YES!
- Continuous
  - Changes tested every commit and/or pull-request (like auto-correct)

- CI generally implies a lot of automation

Test Driven Development vs. Automated Testing vs. CI

- ***Test Driven Development:*** A development methodology where functional test are written before the code
  - Works well with CI as tests are written and committed and are automatically run (failing)
  - Code that implements the functionality being tested retriggers the tests automatically
- ***Automated Testing***: Software that automatically performs tests on a regular basis and reliably detects and reports anomalous behaviors/outcomes.
  - Examples: Auto-test, CTest/CDash, nightly testing, etc.
  - May live “next to” your development workflow
  - Potential issues: change attribution, timeliness of results, multiple branches of development
- ***Continuous Integration (CI)***: automated testing performed at high frequency and fine granularity
  - Aimed at preventing code changes from breaking key branches of development (e.g. main)
  - Lives “within” your development workflow
  - Potential issues: extreme automation, test granularity, coverage, 3rd-party services/resources

Examples…

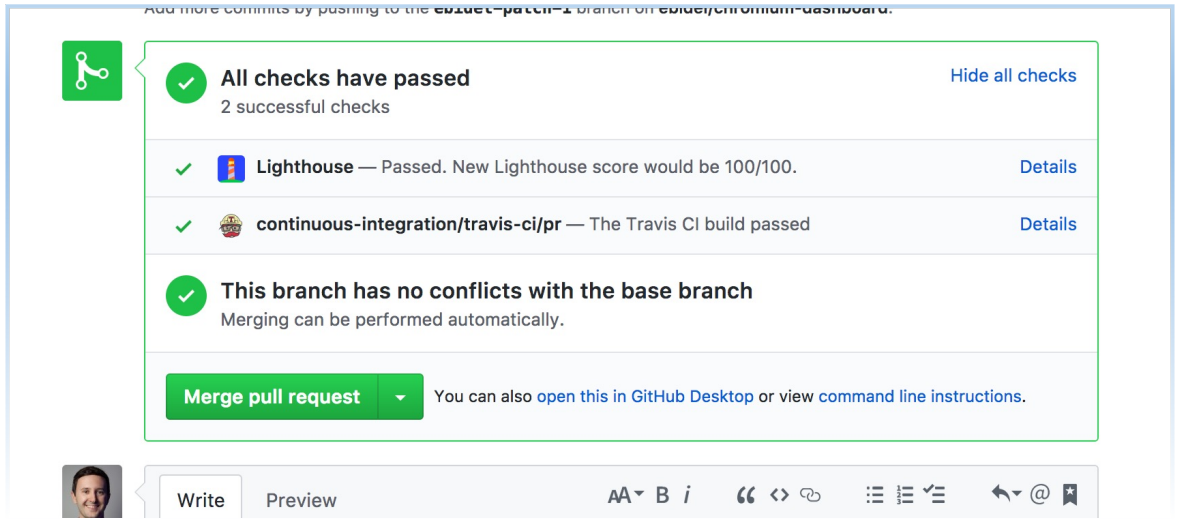
Automated Nightly Testing Dashboard
Lives "next to" your development work

Results of VisIt Regression Test (pascal,trunk,serial)

Test suite run started at 2020:07:09:22:49:46.
(Click on table header to sort)

Index	Category	Test File	Status	Runtime (sec)
243	rendering	ospray.py	Unacceptable	5.0
273	simulation	batch.py	Unacceptable	38.0
24	databases	chgcar.py	Succeeded With Skips	11.0
32	databases	exodus.py	Succeeded With Skips	14.0
66	databases	silo.py	Succeeded With Skips	50.0
67	databases	silo_altdriver.py	Succeeded With Skips	87.0
75	databases	xdmf.py	Succeeded With Skips	14.0
109	hybrid	merge_tree.py	Succeeded With Skips	11.0
136	meshtype	emptydomains.py	Succeeded With Skips	7.0
256	rendering	view.py	Succeeded With Skips	17.0
275	simulation	curve.py	Succeeded With Skips	8.0
281	simulation	life.py	Succeeded With Skips	8.0
296	simulation	zerocopy.py	Succeeded With Skips	32.0
0	databases	ANALYZE.py	Succeeded	10.0
1	databases	ANSYS.py	Succeeded	9.0
2	databases	CGNS.py	Succeeded	11.0
3	databases	Cale.py	Succeeded	6.0
4	databases	Chombo.py	Succeeded	7.0
5	databases	EnSight.py	Succeeded	9.0
6	databases	FITS.py	Succeeded	8.0
7	databases	Fluent.py	Succeeded	7.0
8	databases	GDAL.py	Succeeded	20.0
9	databases	NASTRAN.py	Succeeded	15.0

CI Testing
Lives embedded in your development work



What can make CI difficult

Common situations

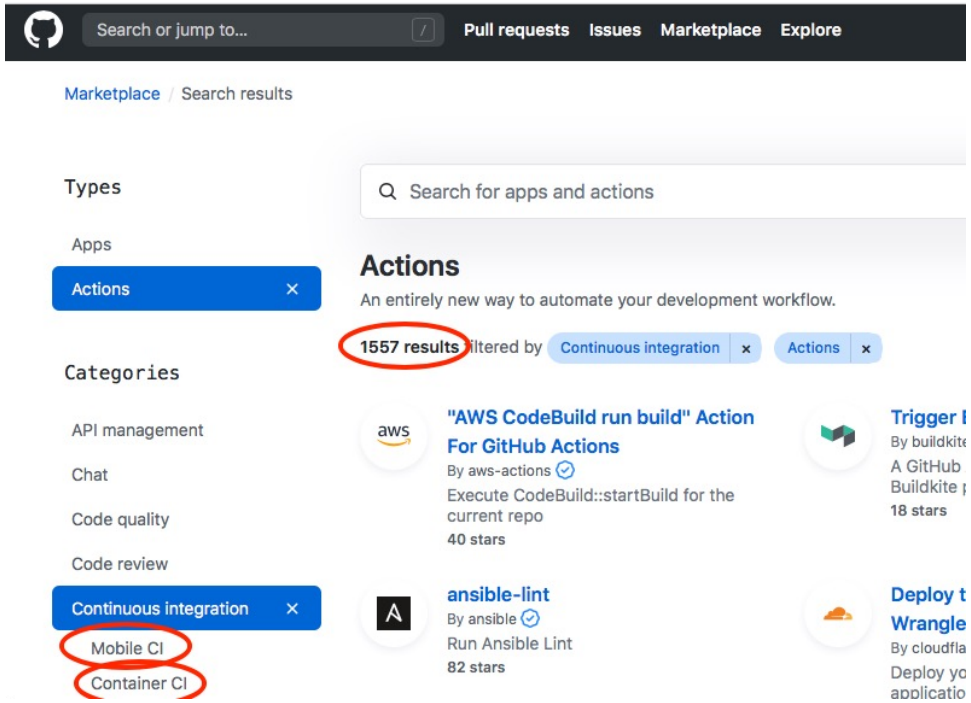
- Just getting started
  - Many technologies/choices; often in the ”cloud”
  - Solution: start small, simple, build up
- Developing suitable tests
  - Many project’s existing tests not suitable for CI
  - CI testing is a balance of thoroughness and responsiveness
  - Solution: Simplify/refactor and/or sub-setting test suite
- Ensuring sufficient coverage
  - Some changes to code never get tested – CI can provide a false sense of security
  - Solution: tools to measure it, enforce always increasing

Advanced situations

- Defining failure for *many* configurations / inconsistent failures
  - Bit-for-bit (exact) match vs. fuzzy match
  - Solution: absolute/relative tolerances → AI/ML
- Numerous 3rd party libraries (TPLs)
  - Compiling takes too long
  - Solution: cache pre-built TPLs, containers
- Performance testing
  - Avoid time-, space-, scaling-performance degradation
  - Solution: Performance instrumentation and *scheduled* testing

CI Resources (Where do jobs run?)

- Free Resources
  - GitHub, BitBucket, GitLab, etc. provide shared runners
  - AWS, Azure Pipelines have free tiers that can be used
  - All launch a VM (Linux variants, Windows and OSX)
    - Constrained in time/size, hardware (e.g. GPU type/count)
    - Not a complete solution for many HPC/scientific codes, but a useful starting point.



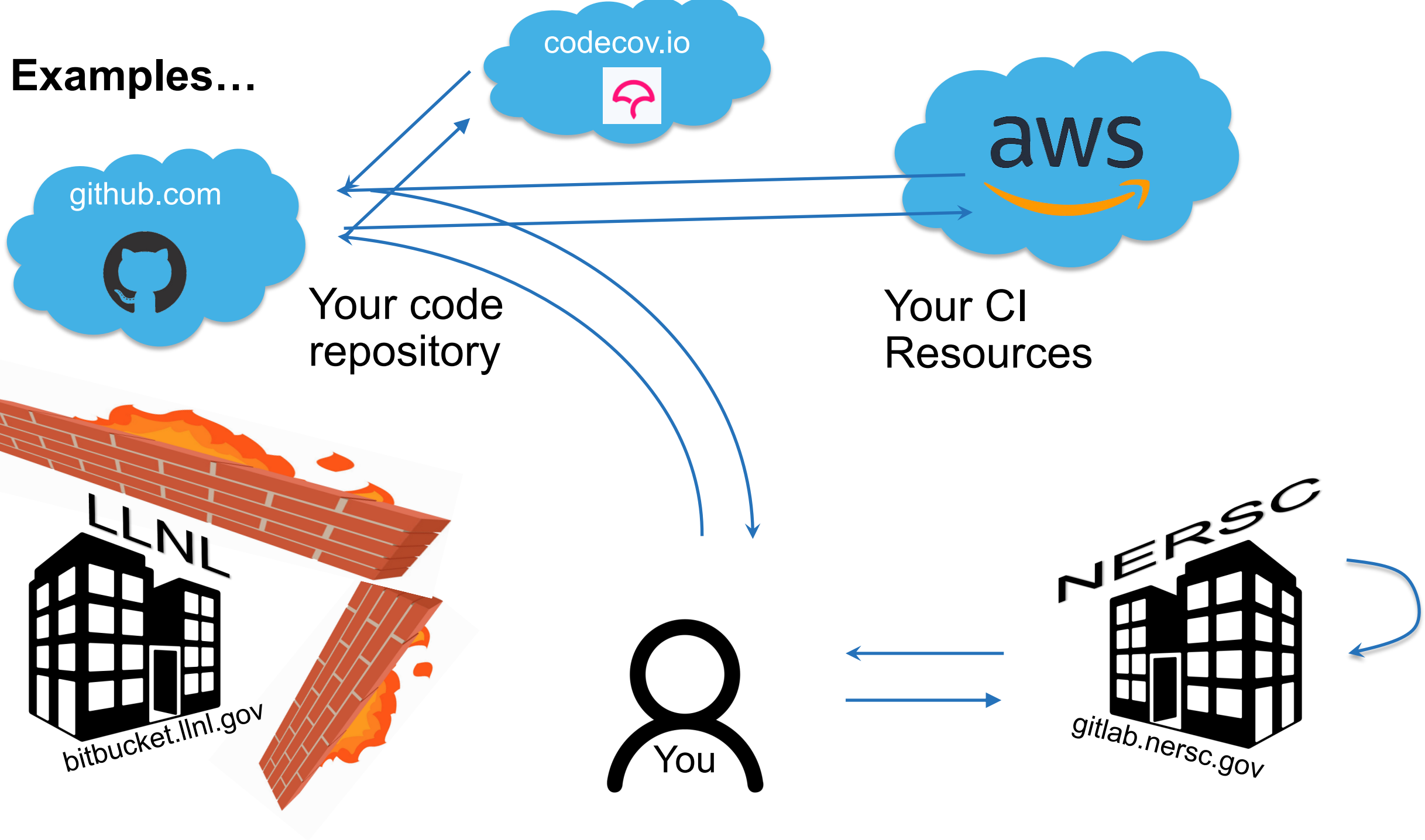
- Site-local Resources
  - Group, department, institution, computing facility
  - Examples: CADES @ ORNL, Bamboo @ LLNL, Jenkins @ ANL, Travis+CDash @ NERSC
  - ECP Program: GitLab-CI @ ANL, LANL, LLNL, NERSC, ORNL, SNL

- Create your own by setting up resources/services

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Examples…


codecov.io
github.com
aws
Your code repository
Your CI Resources
LLNL
bitbucket.llnl.gov
You
NERSC
gitlab.nersc.gov

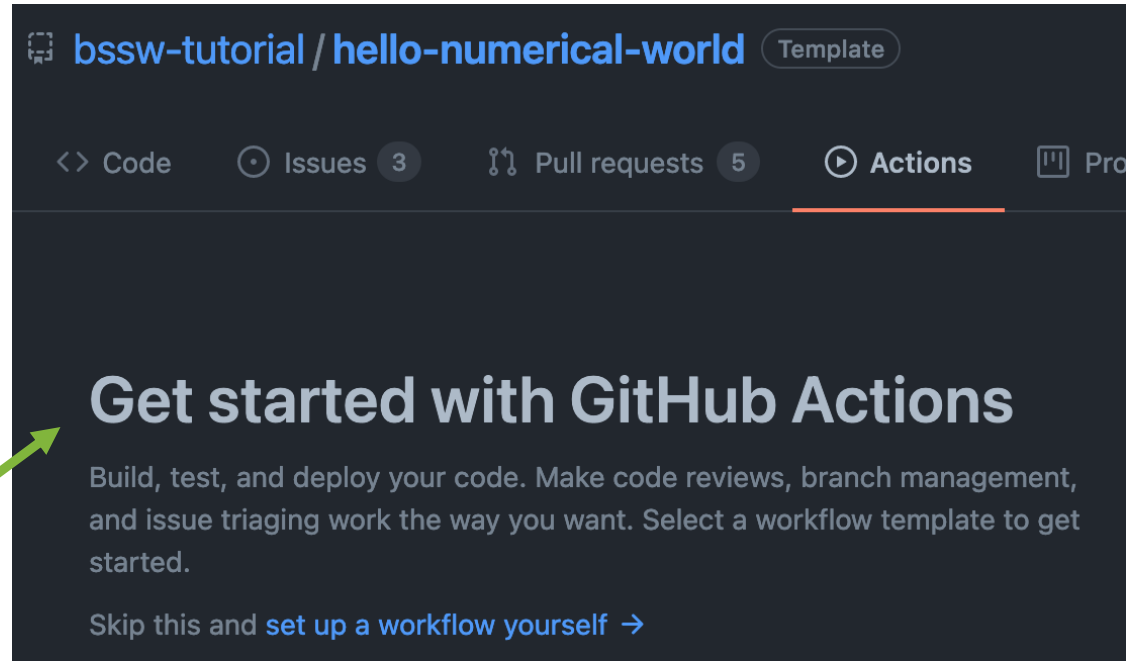
Getting started with CI

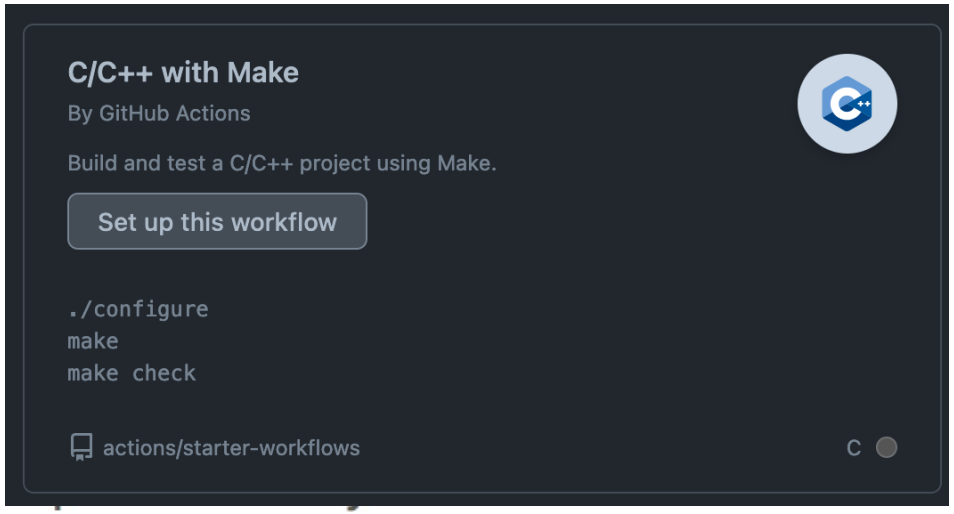
- What *configuration* is most important?
  - Examples: gcc, icc, xlc? MPI-2 or MPI-3? Python 2, 3 or 2 & 3?
- What *functionality* is most important?
  - Examples: vanilla numerical kernels? OpenMP kernels? GPU kernels? All of these?
- Good candidates…
  - A “hello world” example for your project
  - At a minimum, even just building the code can be a place to start!
  - Once you’ve got the basics working, its easy to build up from there

Getting started with CI:

Setting up CI

Service	Interface	
GitHub Actions	Repo YAML file	.github/workflows/<test_name>.yml
GitLab	Web page configurator + repo YAML file [& repo scripts]	/.gitlab-ci.yml in root of repo
Bamboo	Web page configurator + repo scripts	
Travis	repo YAML file [& repo scripts]	/.travis.yml in root of repo





Getting started with GitHub Actions:

19 lines (15 sloc) | 359 Bytes

```yaml
name: Check Results

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: make coverage
      run: make CXXFLAGS=--coverage LDFLAGS="--coverage -lm" check_all
    - name: upload coverage
      run: bash <(curl -s https://codecov.io/bash)
```

github.com

Code Issues Pull requests **Actions** Projects ...

Added workflows.

Check Results #1

build

succeeded 4 minutes ago in 7s

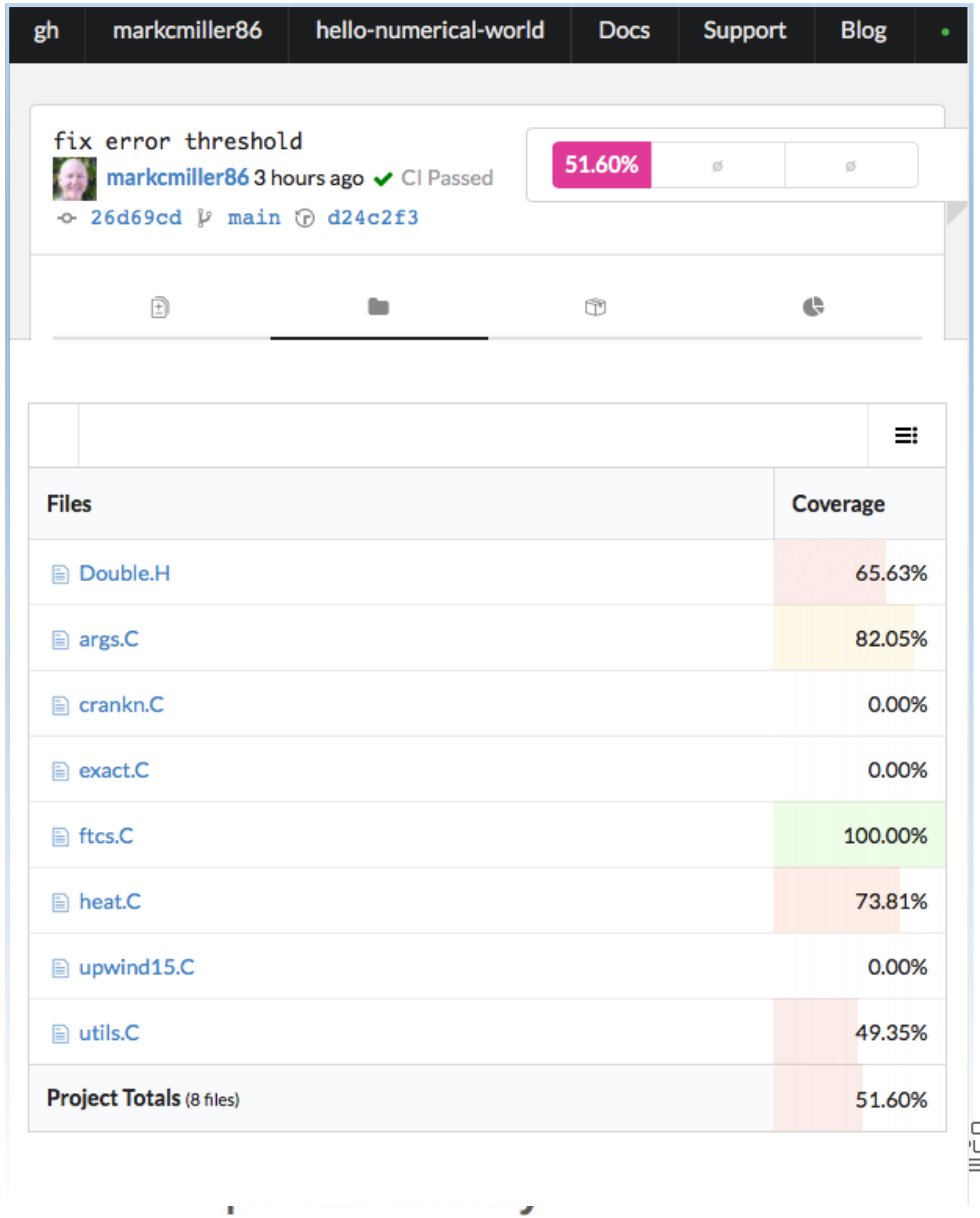
Set up job 2s

Run actions/checkout@v2 1s

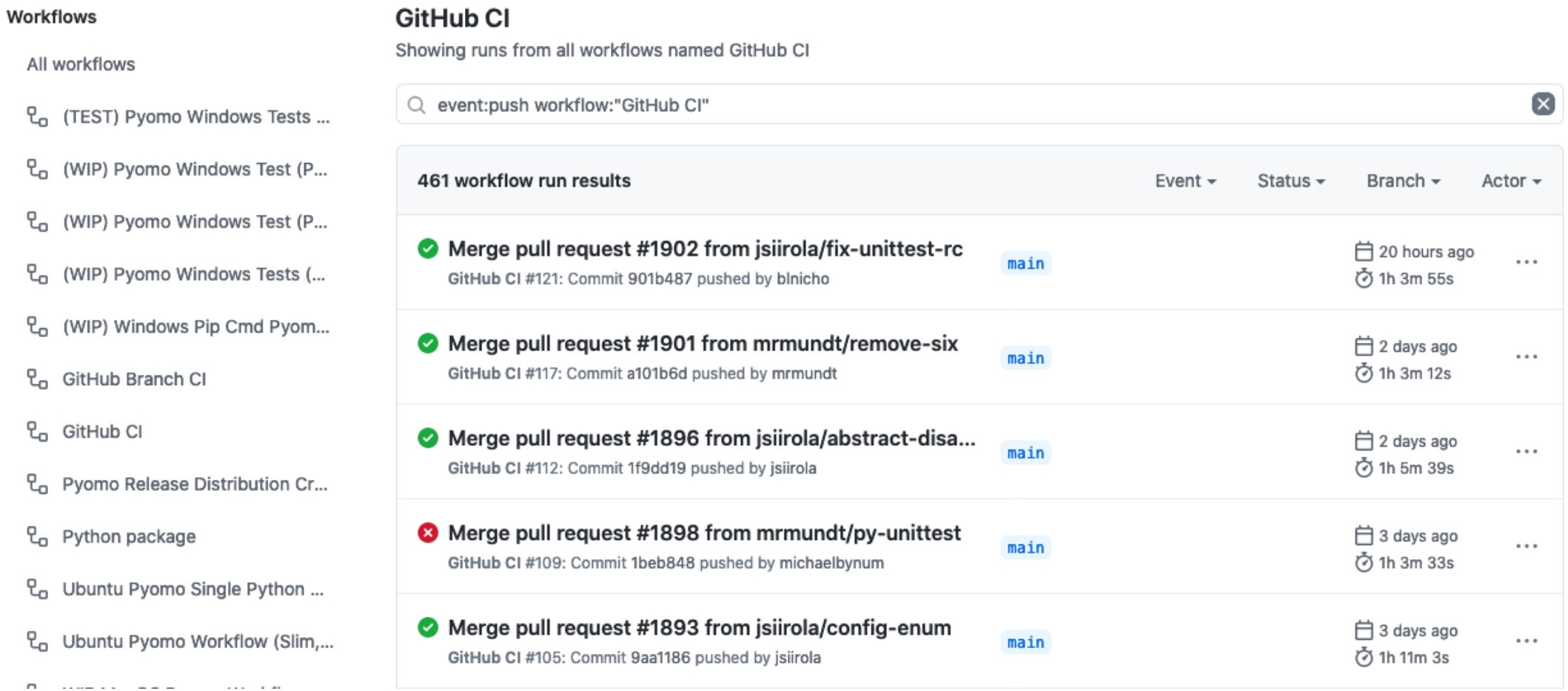
make coverage 3s

```
1  ▶ Run make CXXFLAGS=--coverage LDFLAGS="--coverage -lm"
   check_all
4  g++ -c --coverage  heat.C -o heat.o
5  g++ -c --coverage  utils.C -o utils.o
6  g++ -c --coverage  args.C -o args.o
7  g++ -c --coverage  exact.C -o exact.o
```

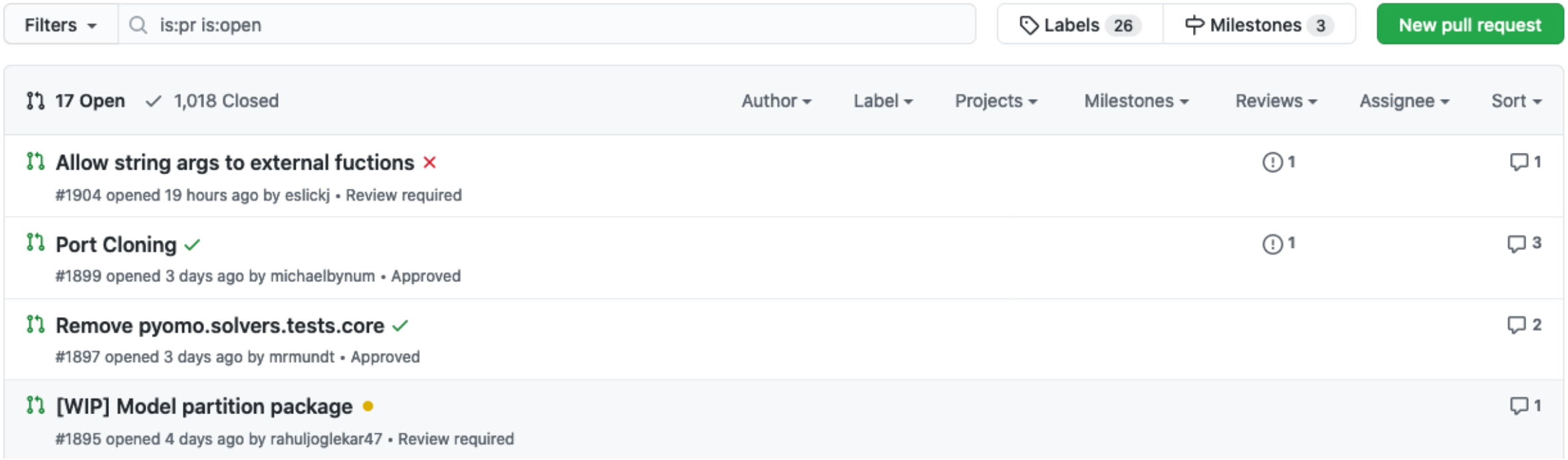
codecov.io



GitHub Actions – results of workflow test runs



GitHub Pull Request Status Indicators



Summary

- The purpose of CI is to identify problems early
  - Prevent code that would “break the build” or adversely impact other developers being introduced
  - Need to provide sufficient confidence, but run quickly – balance varies by project
- CI should complement (not replace) more extensive automated testing
  - Use scheduled testing for more and more detailed tests, more configurations and platforms, performance testing, etc.
- CI for TDD is a natural fit
  - Writing tests before the code works well with CI
- Many options for where to execute CI tests
  - Free services are a good (easy) place to start
  - But may not be sufficient in the long run (especially large HPC/scientific codes)
- Start simple to get automation working, then build out what you need
  - Focus initially on key software configurations and aspects of the code to be tested
  - Make sure your testing expands to cover new code, use TDD