UTSouthwestern Medical Center
Lyda Hill Department of Bioinformatics | BioHPC

GitLab CI/CD
(Continuous Integration / Continuous Delivery)

[web] portal.biohpc.swmed.edu
[email] biohpc-help@utsouthwestern.edu

Updated for 2021-11-10

Basic Overview

- **Git crash course**
- **What is CI? What is CD?**
- **Basic testing principles**
- **Gitlab CI itself**
- **Runners**
  - How your CI gets run!
- **Demonstration**

Disclaimer

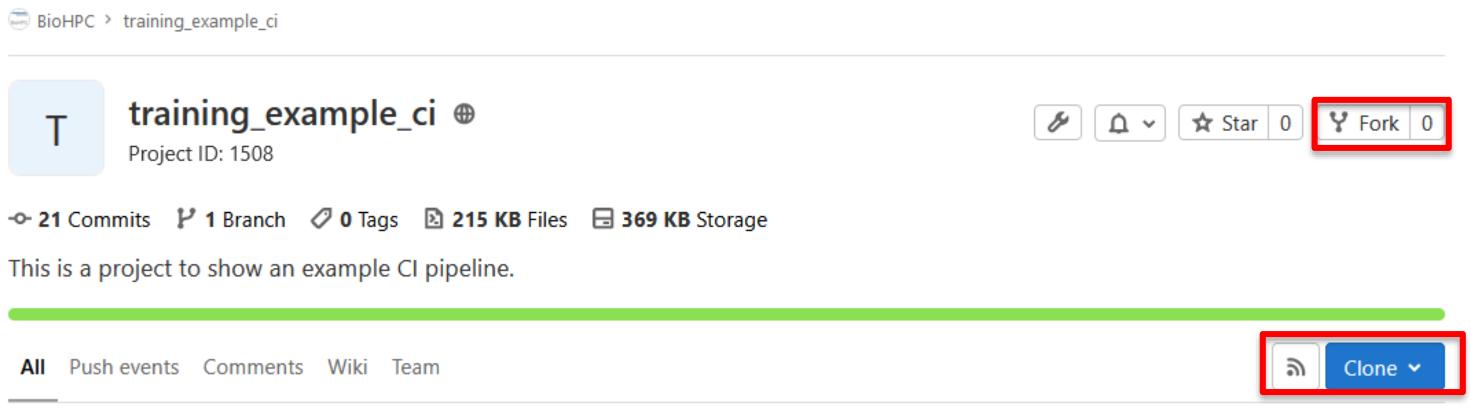
- **This is not a comprehensive survey, but is meant to get you exposed to CI concepts and familiar with the infrastructure BioHPC has available.**
  - You may want to explore the documentation a bit.

- **Your science, your software, and your team might be better suited to some variation/subset of what's presented here.**
  - CI is meant to *lessen the overall amount of work* that you have to do to maintain your codebase.
  - A smaller script-based project may need much less CI than a more complex, compiled application.
  - CI is a *spectrum* of different tools and approaches, and you can mix/match as you please.

- **As always, we are available at biohpc-help@UTSouthwestern.edu if you need further assistance or pointing in the right direction.**

Practical example

- https://git.biohpc.swmed.edu/biohpc/training-example-ci
- Contains several files (heavily commented) to illustrate different parts of the GitLab CI.
- Clone or fork the repository to a project of your own and play around!


BioHPC > training_example_ci

training_example_ci

Project ID: 1508

21 Commits 1 Branch 0 Tags 215 KB Files 369 KB Storage

This is a project to show an example CI pipeline.

Understanding of Git is necessary to use CI to its fullest capability

- The concepts of CI are very closely tied to concepts in version control, specifically Git.
  - Repositories
  - Commits
  - Branches
  - Tags

- Slides from previous training:
  - https://portal.biohpc.swmed.edu/media/filer_public/21/ad/21adc5e7-f5e8-467c-830e-b40df31a3935/gitintro.pdf
  - https://portal.biohpc.swmed.edu/content/training/training-slides/ - search for ‘Git'

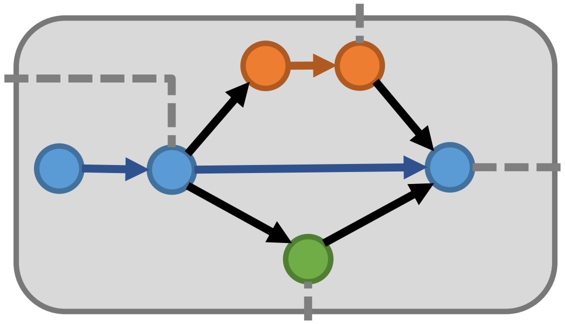
Quick overview of Git

- Git is a **version control system**, designed to **track changes** to your codebase.
- A **git repository** is a collection of code, tracked by git.
- A **git commit** is a set of changes, applied to some previous repository state, that updates the repository to some new state.
- A **git push** is an action that migrates those changes to some other repository (e.g. Gitlab)
- A **git branch** is a series of related commits distinct from other branches.
- A **git merge** is a process of bringing changes from one branch to your current one.
- A **git tag** is a ‘special name’ given to a particular commit.

Git can be thought of as a graph of ‘repository states’

```
# ...some code...
thisVar = 5;
thatVar =
newFunction(thisVar);
```

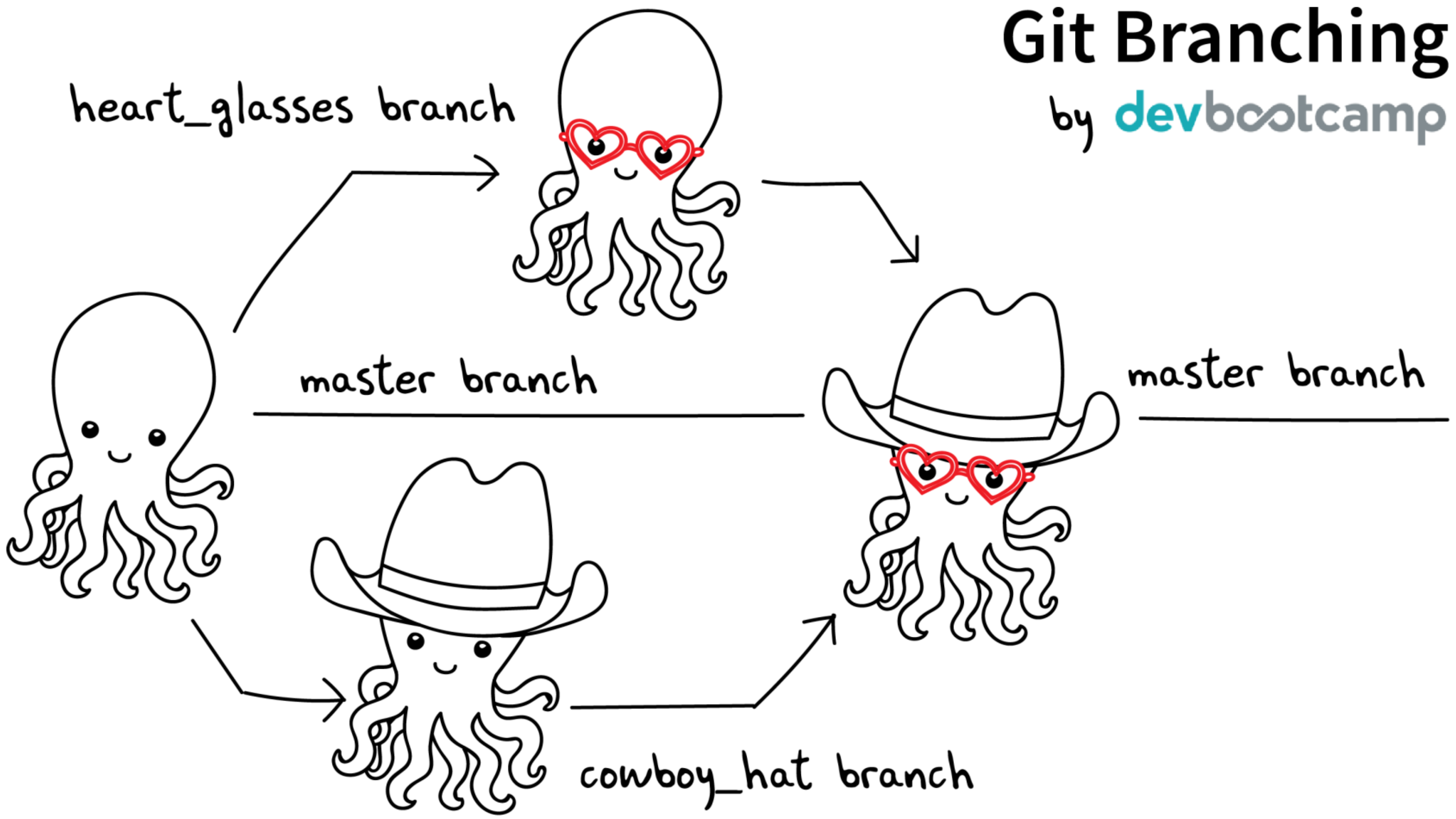
```
# ...some code...
thisVar = 5;
thatVar =
oldFunction(thisVar);
```



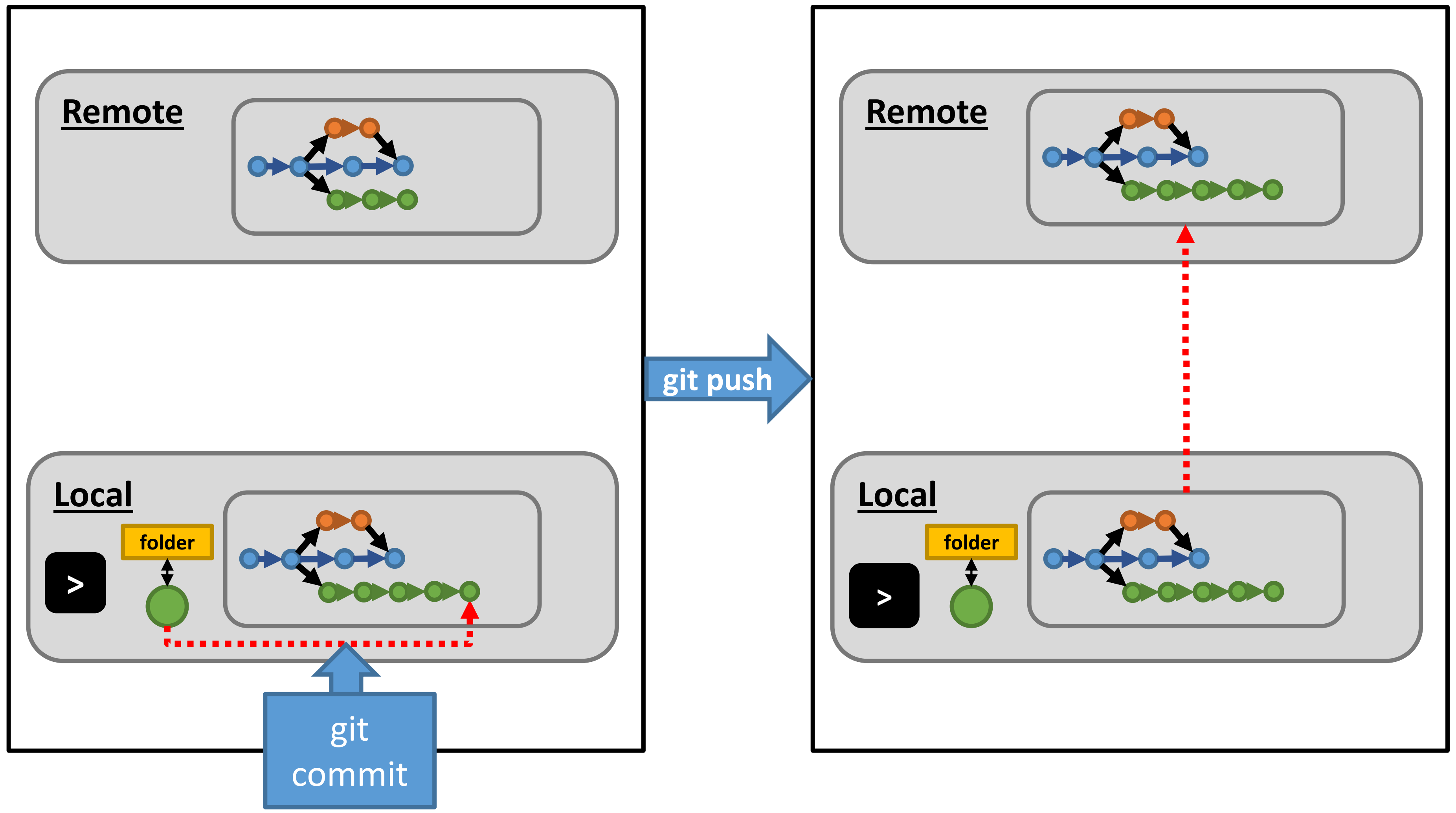
```
# ...some code...
thisVar = 7;
thatVar =
newFunction(thisVar);
```

```
# ...some code...
thisVar = 7;
thatVar =
oldFunction(thisVar);
```

Put another way…

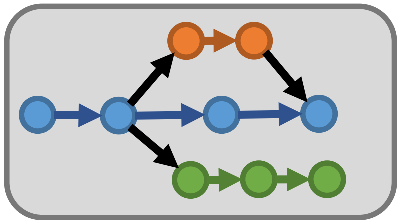


Git commits and pushing to remote



Lastly…

A **project** in GitLab is a repository + all additional supporting ‘stuff’



+

Wiki
Issues/Bug Tracking
Container/Package Registries
CI/CD configuration

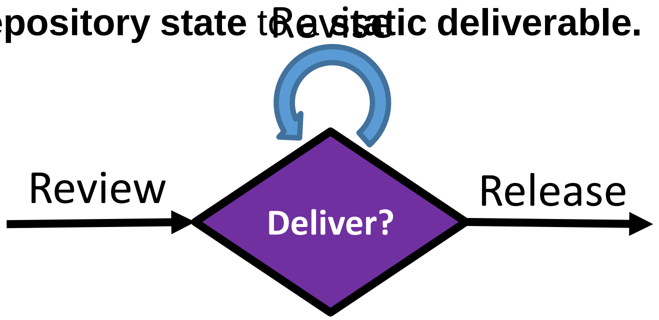
What is CI/CD?

Both are ideal practices / philosophies

CI/CD tools are meant to help you attain these ideals through automation.

- Continuous Integration (CI) is the general practice of trying to **frequently integrate code changes** into a central repository while making sure the codebase is 'good'.
  - Develop, test, merge.
  - Only include changes when tests are passed.
  - Testing the **code**

- Continuous Delivery (CD) is the practice of going from **repository state to a static deliverable.**
  - Build, test, release.
  - Only releasing when tests are passed.
  - Testing the **application**



Why CI?

- It <u>automates tedious, time-consuming, or repetitive tasks</u> and <u>reduces human error</u>
- It <u>checks your work for you</u> – CI pipelines provide a record of successes/failures, which is <u>valuable debug information</u> and <u>verifies functionality</u>
- Frequently checking whether or not your code works lets you <u>make consistent progress</u> with your code – fewer surprises, fewer headaches.
- If you have many people working on the same codebase, they can agree on the tests that will be run, and then <u>work independently</u> while ensuring the code continues to work as they all expect.

A few examples use cases for CI/CD

- If you have a software package you want to publish…
  - CI can automatically build and test your code on one branch before you merge it to production.
  - Whenever anything is merged to production, CD can then automatically package and publish the resulting software package.

- If you have simulation code you want to benchmark…
  - CI can run a battery of performance tests and provide you with detailed information about run-time, memory usage, etc.

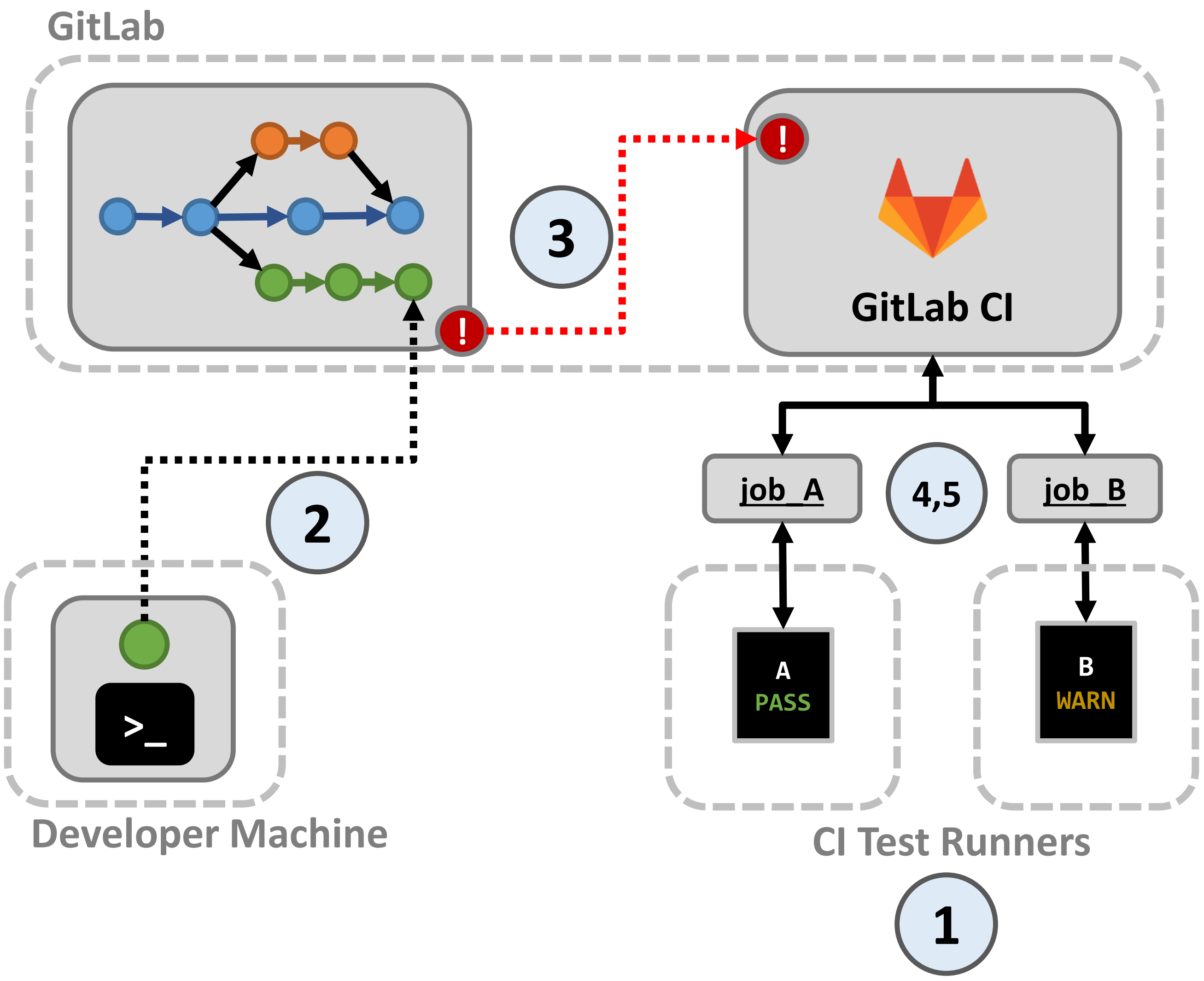
- If you have data analysis code you want to consistently prove…
  - CI can run your code on a series of test datasets to show classification accuracy.

CI can do all this automatically, every time you push a change to your project (you can also customize it to trigger off various conditions)

More on CI/CD and testing (documentation)

- Good overview of CI:
  - https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment
  - https://www.atlassian.com/continuous-delivery/continuous-integration/how-to-get-to-continuous-integration
- Overview of testing (broadly):
  - https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing
- GitLab-specific documentation:
  - https://docs.gitlab.com/ee/ci/

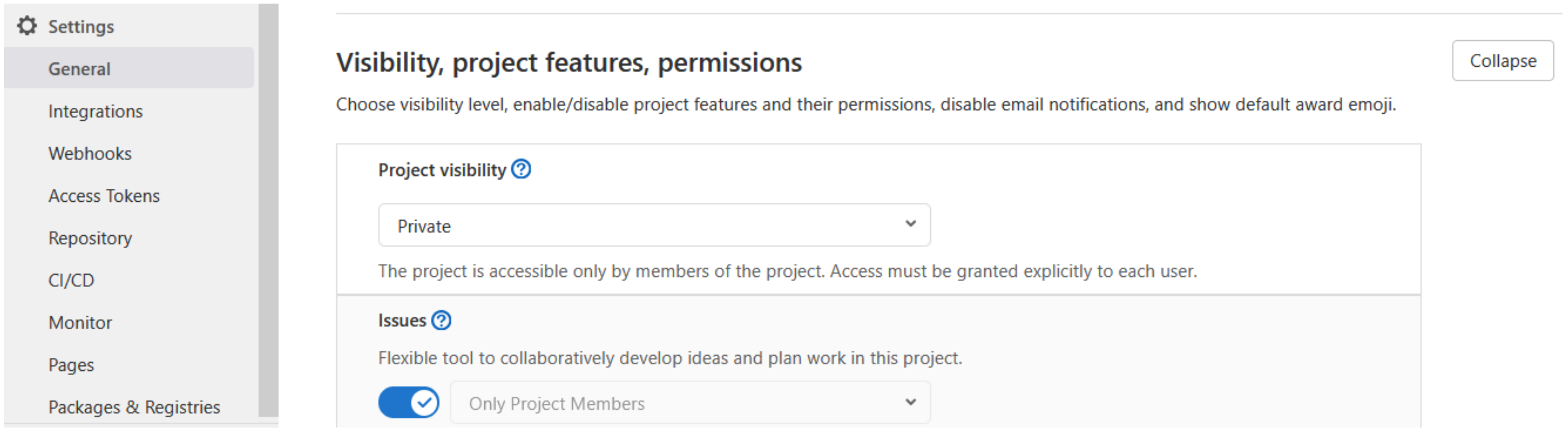
The general CI process on GitLab



1. User sets up runner(s) on suitable host(s)
2. User submits commit containing **.gitlab-ci.yml**
3. GitLab triggers CI pipeline, schedules jobs.
4. Jobs sent to runners
5. Runners send results back to GitLab

Enabling CI in your project and controlling who can run pipelines

CI must be enabled via a settings switch in your project.



Visibility, project features, permissions

Choose visibility level, enable/disable project features and their permissions, disable email notifications, and show default award emoji.

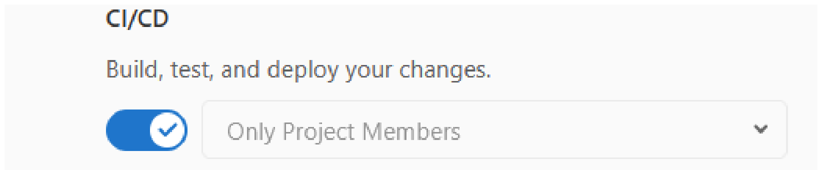
Project visibility: Private

The project is accessible only by members of the project. Access must be granted explicitly to each user.

Issues

Flexible tool to collaboratively develop ideas and plan work in this project.

✓ Only Project Members



CI/CD

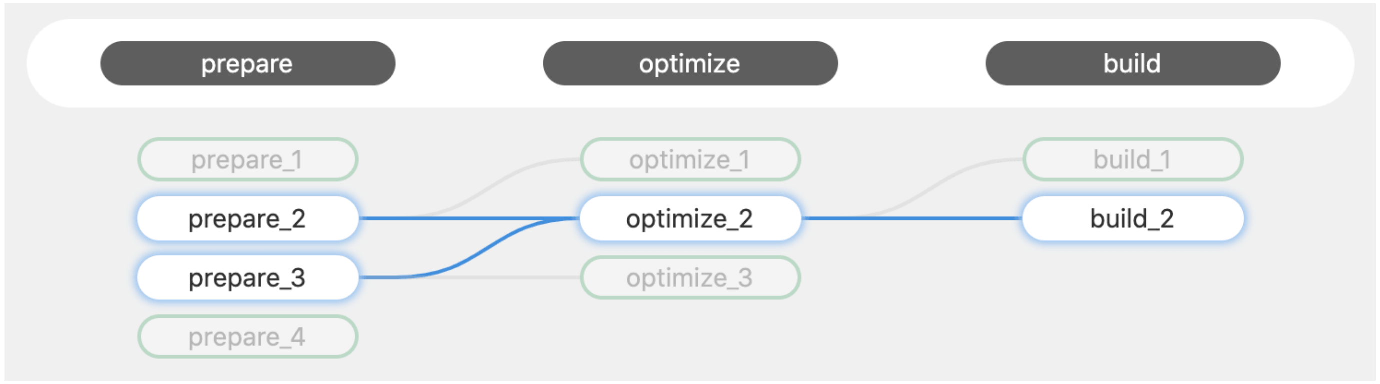
Build, test, and deploy your changes.

✓ Only Project Members

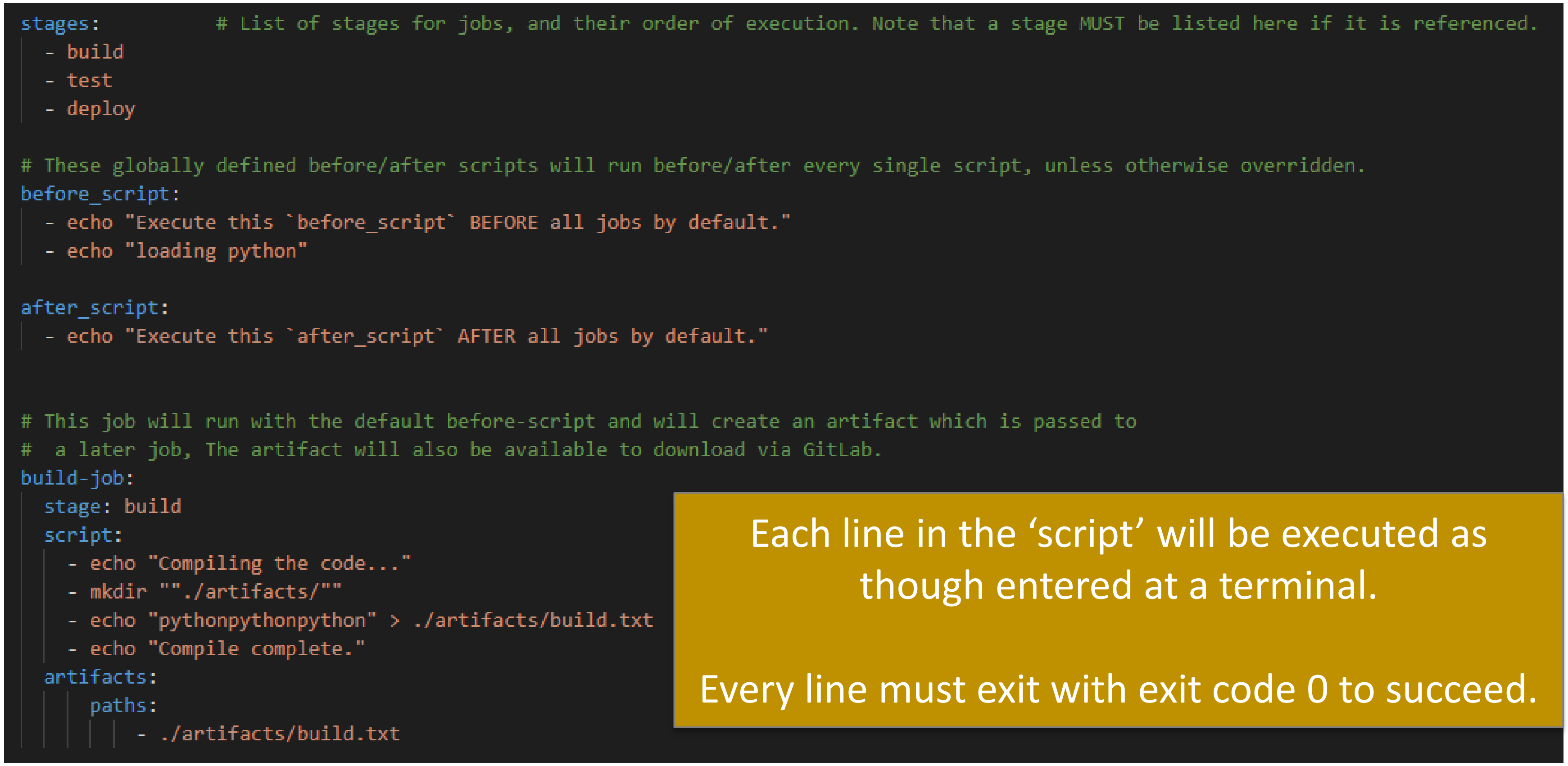
GitLab-specific CI/CD structure

- GitLab's CI scheduling only triggers if there is a '**.gitlab-ci.yml'** file in the root/base of your project.
  - Exactly as written, including leading period '.' – it is a hidden file.
  - This file completely specifies how GitLab should run your CI pipeline.

- Pipelines are composed of **jobs** which are organized into **stages.**
  - All jobs in a stage must pass (or be allowed to fail) before the next stage is run.
- **Artifacts** are files which can be passed between stages – a compiled library can be used by a later job.
  - Artifacts can also be downloaded via the GitLab web interface.



Partial .gitlab-ci.yml file



```yaml
stages:          # List of stages for jobs, and their order of execution. Note that a stage MUST be listed here if it is referenced.
  - build
  - test
  - deploy

# These globally defined before/after scripts will run before/after every single script, unless otherwise overridden.
before_script:
  - echo "Execute this `before_script` BEFORE all jobs by default."
  - echo "loading python"

after_script:
  - echo "Execute this `after_script` AFTER all jobs by default."


# This job will run with the default before-script and will create an artifact which is passed to
#  a later job, The artifact will also be available to download via GitLab.
build-job:
  stage: build
  script:
    - echo "Compiling the code..."
    - mkdir ""./artifacts/""
    - echo "pythonpythonpython" > ./artifacts/build.txt
    - echo "Compile complete."
  artifacts:
    paths:
      - ./artifacts/build.txt
```

Each line in the 'script' will be executed as though entered at a terminal.

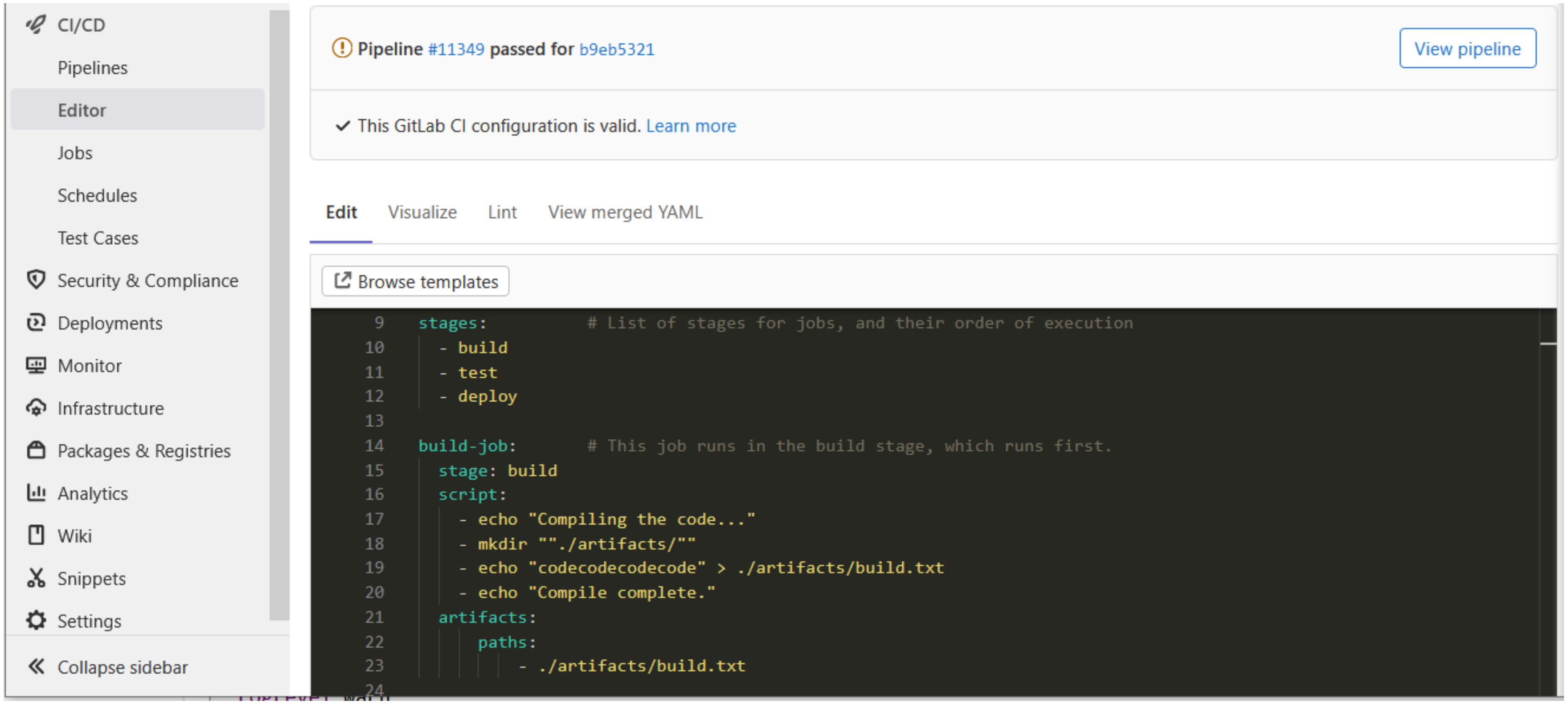
Every line must exit with exit code 0 to succeed.

Full file at: https://git.biohpc.swmed.edu/biohpc/training-example-ci/-/blob/master/.gitlab-ci.yml

File format docs at: https://docs.gitlab.com/ee/ci/yaml/gitlab_ci_yaml.html

Every project with CI/CD enabled has access to a Pipeline Editor

- Automatically checks syntax as you write your CI script. **Highly recommend using this tool.**
  - – This will save you a huge amount of time – YAML is a fairly strict syntax.
- Whenever you commit changes here, it will trigger a pipeline like any commit would.



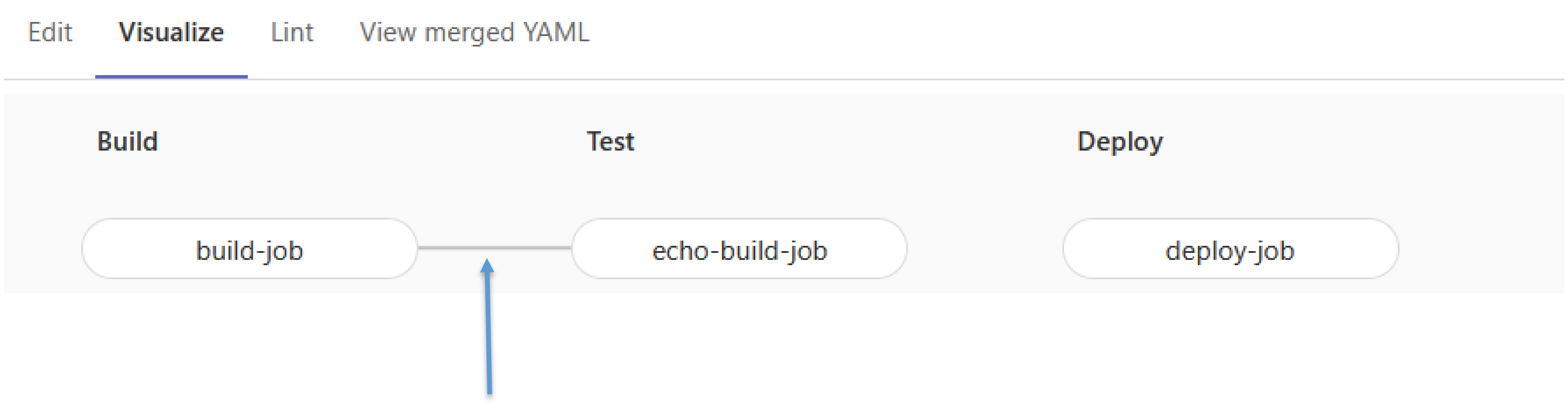
Pipeline #11349 passed for b9eb5321

✓ This GitLab CI configuration is valid. Learn more

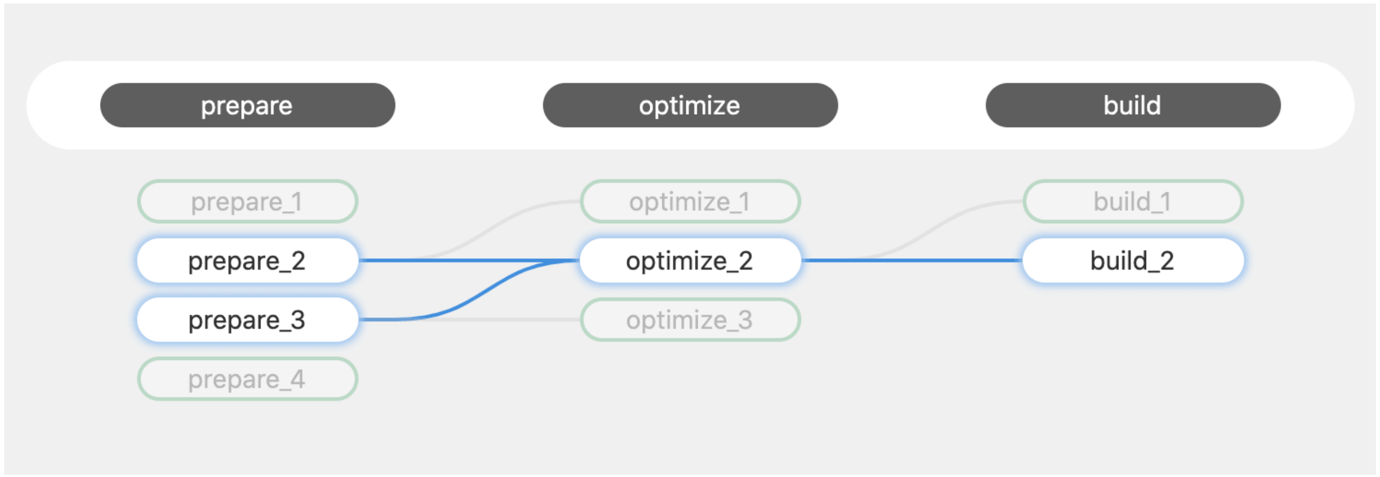
```
stages:          # List of stages for jobs, and their order of execution
  - build
  - test
  - deploy

build-job:       # This job runs in the build stage, which runs first.
  stage: build
  script:
    - echo "Compiling the code..."
    - mkdir "./artifacts/"
    - echo "codecodecodecode" > ./artifacts/build.txt
    - echo "Compile complete."
  artifacts:
    paths:
      - ./artifacts/build.txt
```

Pipeline Editor lets you visualize relationships within pipelines

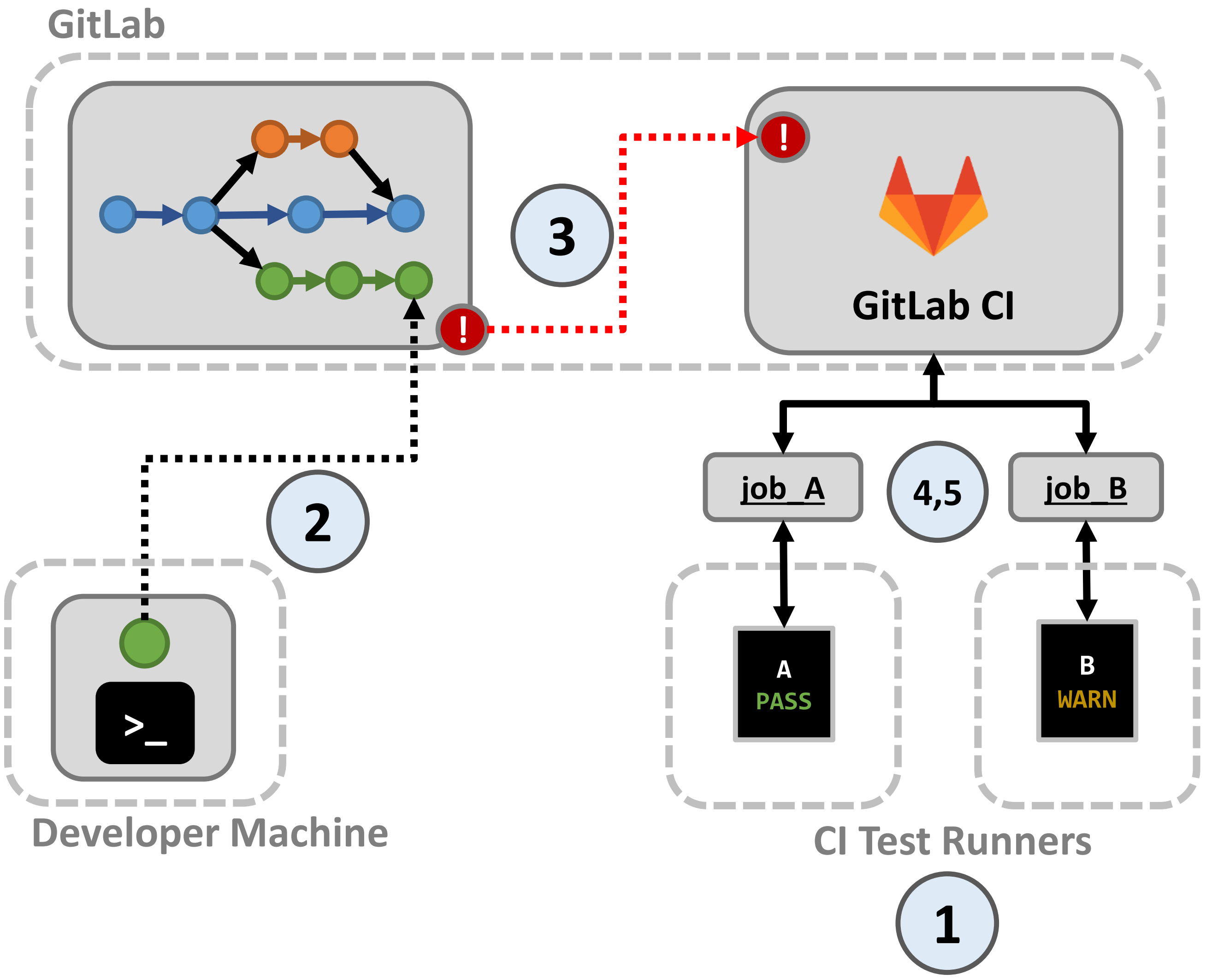


Connections show a 'needs' relationship – later job needs artifact from earlier job.



The general CI process on GitLab

1. **User sets up runner(s) on suitable host(s)**
2. User submits commit containing **.gitlab-ci.yml**
3. GitLab triggers CI pipeline, schedules jobs.
4. Jobs sent to runners
5. Runners send results back to GitLab



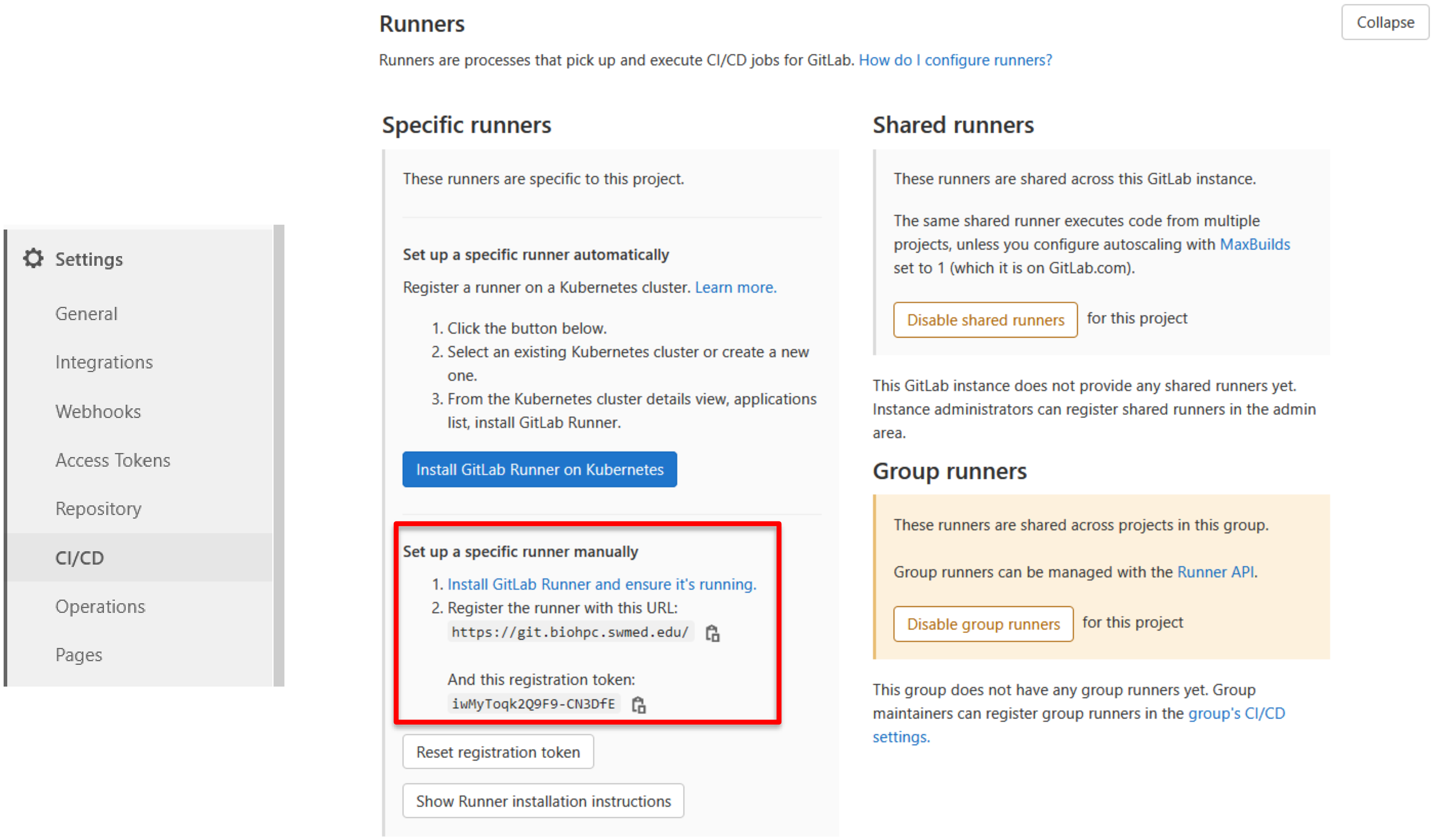
UTSouthwestern Medical Center
Lyda Hill Department of Bioinformatics
BioHPC

How runners run jobs

- When you push to a Gitlab project and CI is active…
  - GitLab checks to see if the repo has a **.gitlab-ci.yml** file at the root of the repository.
  - GitLab parses the CI file and uses the result to dispatch jobs to the runners.
- For each job, runners **will execute each line of the script as though it was entered at a Bash prompt**.
- The runner executes with a large number of CI variables in its environment
  - Lots of information about the job itself, various tokens…
  - **You can add additional variables in your .gitlab-ci.yml**

Each CI job runs on a runner. The machine the runner runs on should have all necessary dependencies installed/available.

Registering new runners



Runners

Runners are processes that pick up and execute CI/CD jobs for GitLab. How do I configure runners?

Specific runners

These runners are specific to this project.

Set up a specific runner automatically

Register a runner on a Kubernetes cluster. Learn more.

1. Click the button below.
2. Select an existing Kubernetes cluster or create a new one.
3. From the Kubernetes cluster details view, applications list, install GitLab Runner.

Set up a specific runner manually

1. Install GitLab Runner and ensure it's running.
2. Register the runner with this URL:
   `https://git.biohpc.swmed.edu/`

   And this registration token:
   `iwMyToqk2Q9F9-CN3DfE`

Shared runners

These runners are shared across this GitLab instance.

The same shared runner executes code from multiple projects, unless you configure autoscaling with MaxBuilds set to 1 (which it is on GitLab.com).

Disable shared runners for this project

This GitLab instance does not provide any shared runners yet. Instance administrators can register shared runners in the admin area.

Group runners

These runners are shared across projects in this group.

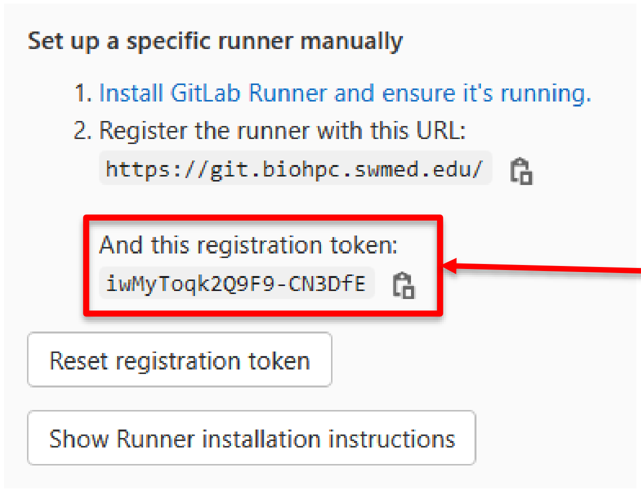
Group runners can be managed with the Runner API.

Disable group runners for this project

This group does not have any group runners yet. Group maintainers can register group runners in the group's CI/CD settings.

On BioHPC…

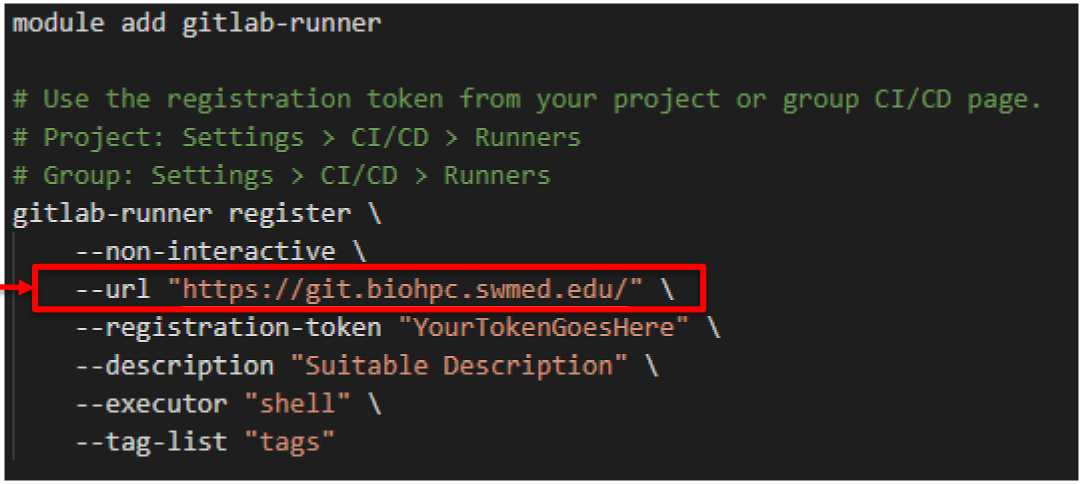
- **<u>Please only run this on BioHPC workstations or thin clients – NOT on Nucleus nodes</u>**
  - Acceptable to use via **WebGUI** or **WebGPU** jobs to test, but **not** for production use.
- **module add gitlab-runner**
- **gitlab-runner register**
  - follow the prompts – kind of finicky about backspaces.
- **gitlab-runner register --non-interactive …**



Set up a specific runner manually

1. Install GitLab Runner and ensure it's running.
2. Register the runner with this URL:
   `https://git.biohpc.swmed.edu/`

   And this registration token:
   `iwMyToqk2Q9F9-CN3DfE`



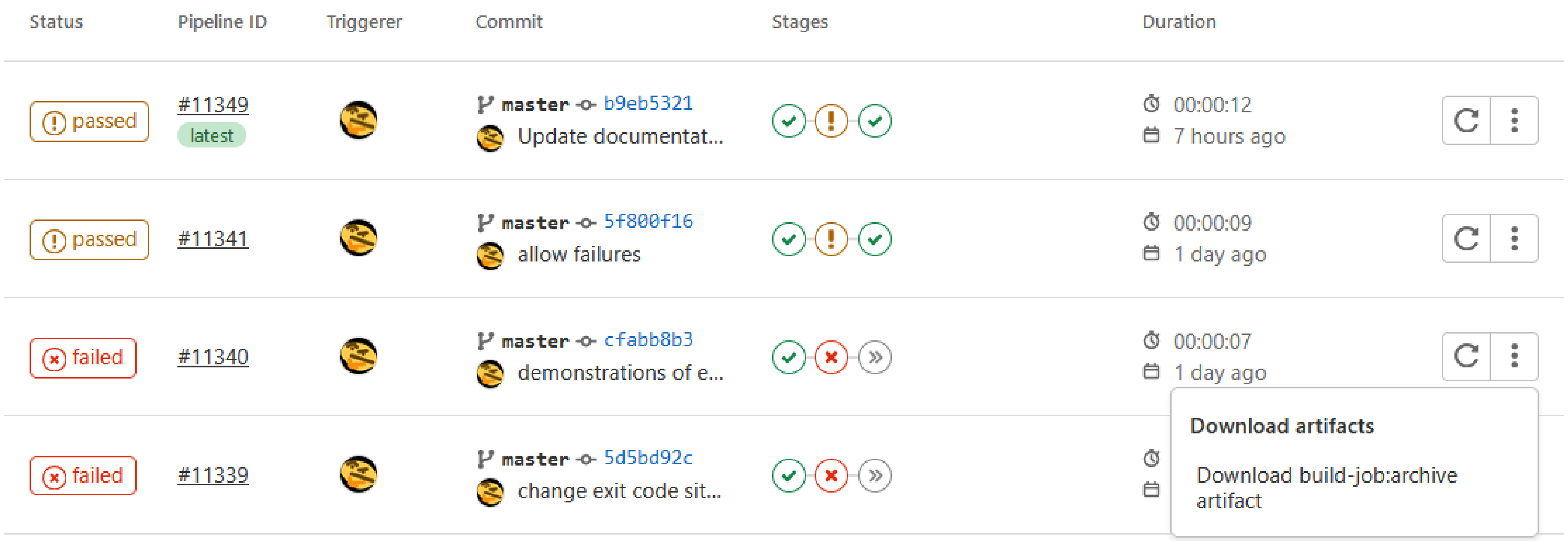
```
module add gitlab-runner

# Use the registration token from your project or group CI/CD page.
# Project: Settings > CI/CD > Runners
# Group: Settings > CI/CD > Runners
gitlab-runner register \
    --non-interactive \
    --url "https://git.biohpc.swmed.edu/" \
    --registration-token "YourTokenGoesHere" \
    --description "Suitable Description" \
    --executor "shell" \
    --tag-list "tags"
```

More on Runners (Documentation)

- General documentation:
  - https://docs.gitlab.com/runner/
- Advanced configuration:
  - https://docs.gitlab.com/runner/configuration/advanced-configuration.html
- Letting runners authenticate to GitLab (e.g. for publishing code)
  - https://docs.gitlab.com/ee/ci/jobs/ci_job_token.html
- Debugging runners:
  - https://docs.gitlab.com/runner/faq/#run-in---debug-mode

The Pipelines tab



Status	Pipeline ID	Triggerer	Commit	Stages	Duration
passed	#11349 latest		master b9eb5321 Update documentat...	✓ ! ✓	00:00:12 7 hours ago
passed	#11341		master 5f800f16 allow failures	✓ ! ✓	00:00:09 1 day ago
failed	#11340		master cfabb8b3 demonstrations of e...	✓ ✗ »	00:00:07 1 day ago
failed	#11339		master 5d5bd92c change exit code sit...	✓ ✗ »	

Download artifacts

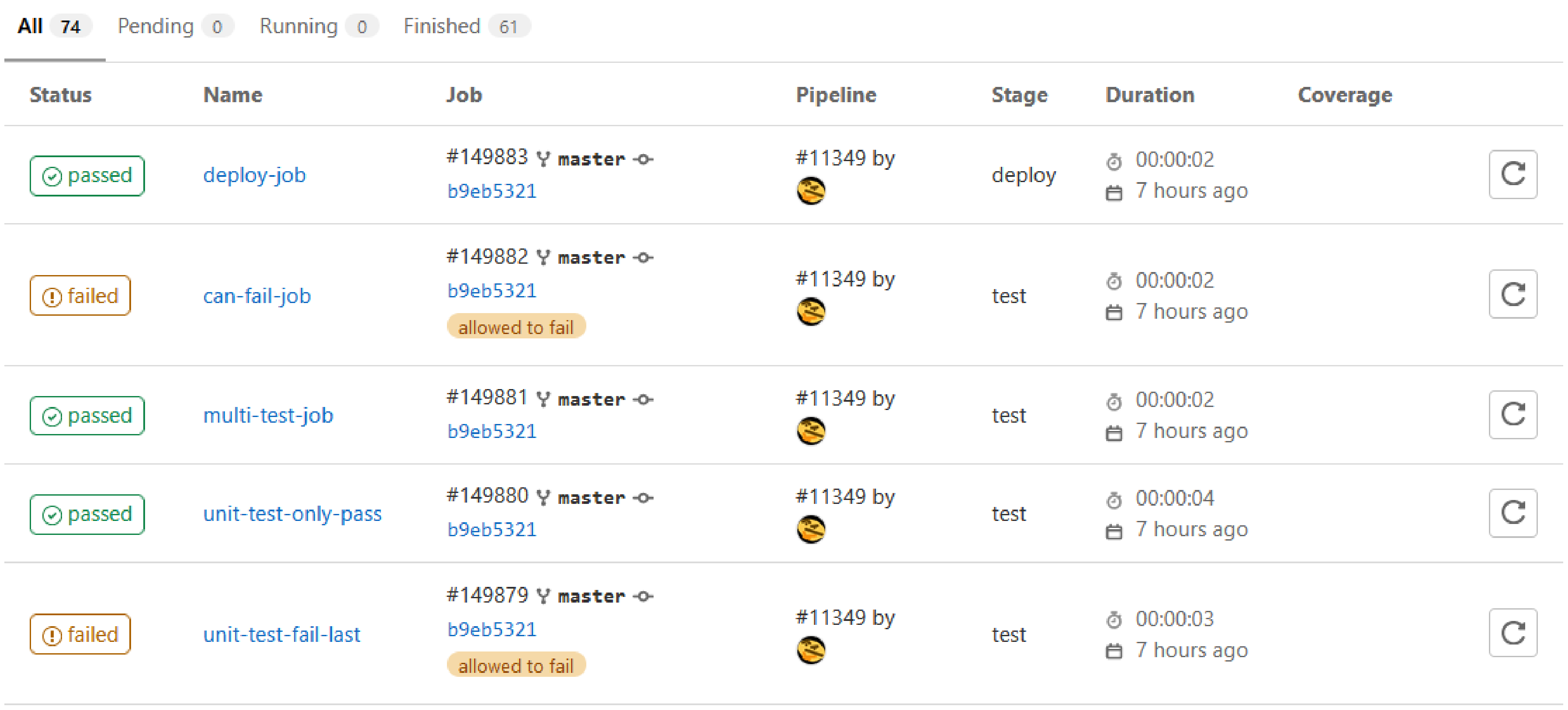
Download build-job:archive artifact

Quickly see which stages failed

Download artifacts from various jobs

UTSouthwestern Medical Center
Lyda Hill Department of Bioinformatics
BioHPC

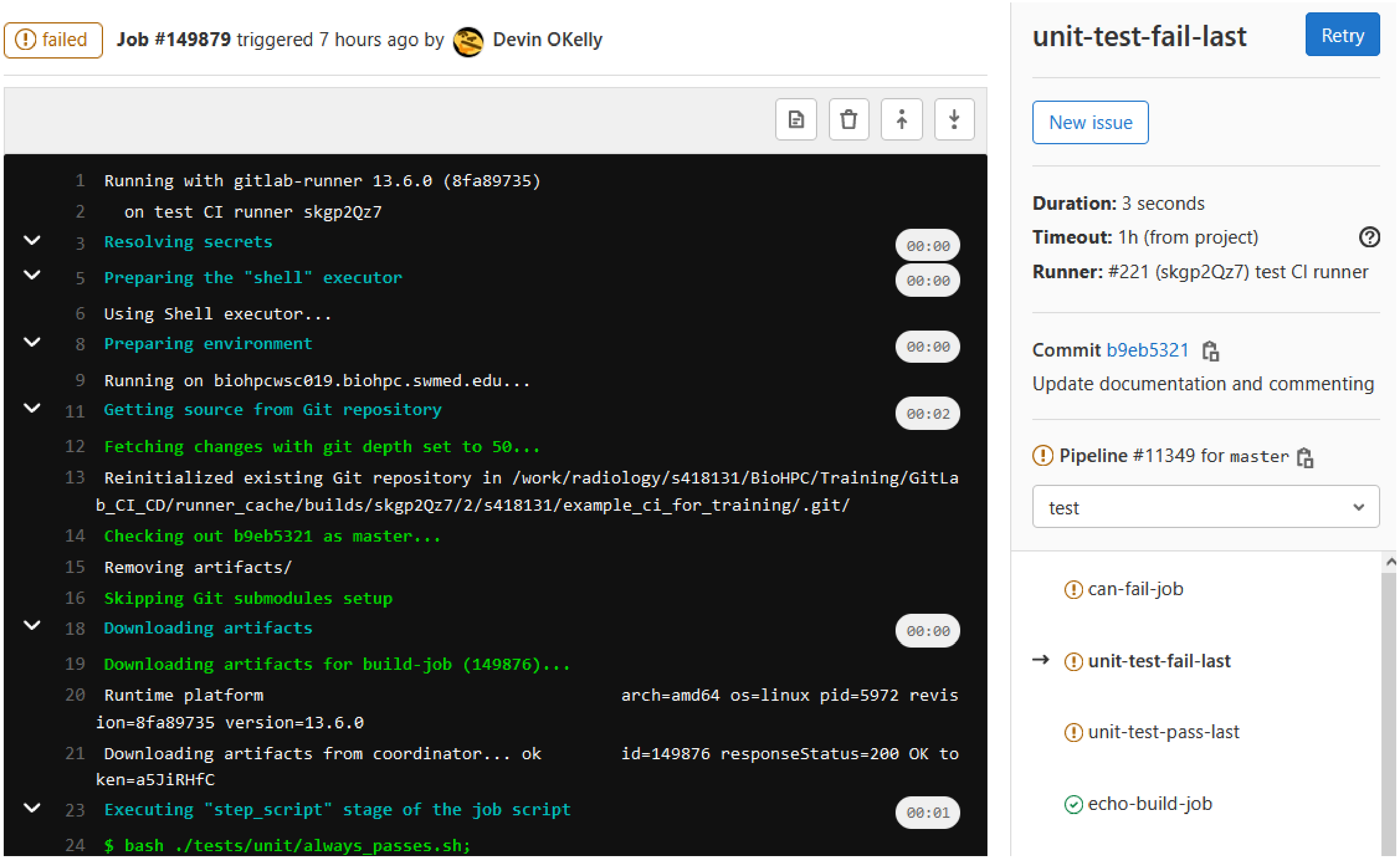
The Jobs tab – more fine-grained detail



All 74 | Pending 0 | Running 0 | Finished 61

Status	Name	Job	Pipeline	Stage	Duration	Coverage
passed	deploy-job	#149883 master b9eb5321	#11349 by	deploy	00:00:02 7 hours ago	
failed	can-fail-job	#149882 master b9eb5321 allowed to fail	#11349 by	test	00:00:02 7 hours ago	
passed	multi-test-job	#149881 master b9eb5321	#11349 by	test	00:00:02 7 hours ago	
passed	unit-test-only-pass	#149880 master b9eb5321	#11349 by	test	00:00:04 7 hours ago	
failed	unit-test-fail-last	#149879 master b9eb5321 allowed to fail	#11349 by	test	00:00:03 7 hours ago	

Each Job has a report page – first place to go when debugging CI



failed **Job #149879** triggered 7 hours ago by Devin OKelly

```
Running with gitlab-runner 13.6.0 (8fa89735)
  on test CI runner skgp2Qz7
Resolving secrets                                                    00:00
Preparing the "shell" executor                                       00:00
Using Shell executor...
Preparing environment                                                00:00
Running on biohpcwsc019.biohpc.swmed.edu...
Getting source from Git repository                                   00:02
Fetching changes with git depth set to 50...
Reinitialized existing Git repository in /work/radiology/s418131/BioHPC/Training/GitLa
b_CI_CD/runner_cache/builds/skgp2Qz7/2/s418131/example_ci_for_training/.git/
Checking out b9eb5321 as master...
Removing artifacts/
Skipping Git submodules setup
Downloading artifacts                                                00:00
Downloading artifacts for build-job (149876)...
Runtime platform                                    arch=amd64 os=linux pid=5972 revis
ion=8fa89735 version=13.6.0
Downloading artifacts from coordinator... ok        id=149876 responseStatus=200 OK to
ken=a5JiRHfC
Executing "step_script" stage of the job script                      00:01
$ bash ./tests/unit/always_passes.sh;
```

unit-test-fail-last

Duration: 3 seconds
Timeout: 1h (from project)
Runner: #221 (skgp2Qz7) test CI runner

Commit b9eb5321
Update documentation and commenting

Pipeline #11349 for master

test

- can-fail-job
- → unit-test-fail-last
- unit-test-pass-last
- echo-build-job

UTSouthwestern Medical Center
Lyda Hill Department of Bioinformatics
BioHPC

More on GitLab CI

- .gitlab-ci.yml syntax:
  - https://docs.gitlab.com/ee/ci/yaml/gitlab_ci_yaml.html
- CI variables (lots!)
  - https://docs.gitlab.com/ee/ci/variables/
- Pipeline editor
  - https://docs.gitlab.com/ee/ci/pipeline_editor
- Script syntax:
  - https://docs.gitlab.com/ee/ci/yaml/script.html
- Various examples:
  - https://docs.gitlab.com/ee/ci/examples/

Testing and CI

- Testing is the cornerstone of any effective codebase
  - Easier to debug when something breaks (“These features are affected”)
  - “Proof” of functionality

- Write tests that give you a lot of information without taking up too much computational resource or time.
- If you find yourself doing the same things over and over, consider using some form of CI/CD to automate the process.

Thank you for your attention!

- Slides will be available at:
  - https://portal.biohpc.swmed.edu/content/training/training-slides/
- Example CI repository available at:
  - https://git.biohpc.swmed.edu/biohpc/biohpc-training/example_ci_for_training

Contact us at biohpc-help@UTSouthwestern.edu if you need more assistance