COMP1531

🌏 Projects - Continuous Integration

Lecture 3.2

Author(s): Hayden Smith

(Download as PDF)

In This Lecture

- **Why? 🤔**
  - To scale multi-user software projects, we need automated ways to integrate and test code
- **What? 📰**
  - Continuous Integration
  - Pipelines
  - Runners

💑 Continuous Integration

Continuous Integration: Practice of automating the integration of code changes from multiple contributors into a single software project.

Or in more concrete terms: Helping make merges into master more **frequent** and **stable**.

💑 Continuous Integration

Typically continuous integration consists of a **series of operations that are executed on any commit that is pushed to the repository**, for example:

- Building (not applicable in JS)
- Testing
- More (in next lectures).

i.e. To oversimplify, continuous integration allows us to:

1. Automatically run `npm run test` (and more) on every commit.
2. Get a visual "OK"/"Not OK" summary of this on gitlab, including more details.

🔬 Setting It Up

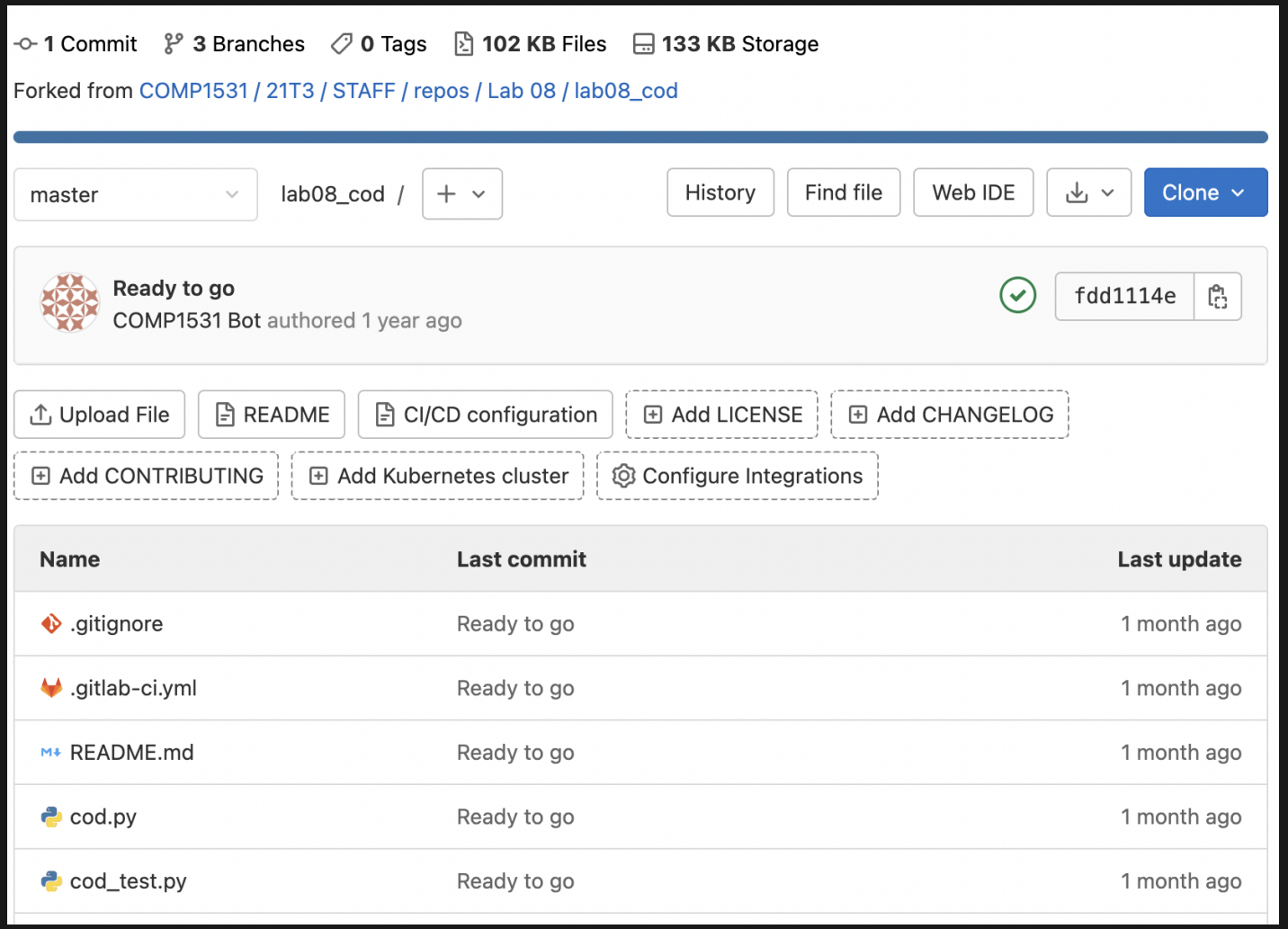
Every git website has it's own way of handling continuous integration. With gitlab, it's the addition of a `.gitlab-ci.yml` file within the root of your git repository. An example that just does testing would be:

```yaml
image: comp1531/basic:latest

cache:
  paths:
    - node_modules

stages:
  - checks

sanity:
  stage: checks
  script:
    - echo 'Hello!'
```

3.2_gitlab-ci_basic.yml

Let's try and add this to a repo.

Setting It Up



See the tick? This tick indicates that some process was run from the last commit. It uses `.gitlab-ci.yml` to figure out what to run. You can click the tick.

Setting It Up

passed **Job pytest** triggered 1 month ago by Rani Jiang

This job is archived. Only the complete pipeline can be retried.

```
1  Running with gitlab-runner 13.8.0 (775dd39d)
2    on COMP1531 Primary Runner tBsSVHYJ
3  Preparing the "docker" executor                    00:03
4  Using Docker executor with image comp1531/basic:20T3
...
5  Using locally found image version due to "if-not-present" pull policy
6  Using docker image sha256:b9bde8920c13862905ac628e8eda7d1fa59a4760287992cad96899d1de965954 for comp1531/basic:20T3 with digest comp1531/basic@sha256:bf744bea3285c3238463fbc90f0b97e3bd3ba905b9a83b6d1486aa96b558c02d ...
8  Preparing environment                              00:01
9  Running on runner-tbssvhyj-project-221082-concurrent-0 via cashewbread...
11 Getting source from Git repository                 00:03
12 Fetching changes with git depth set to 50...
13 Reinitialized existing Git repository in /builds/COMP1531/22T1/STAFF/repos/lab08/lab08_cod/.git/
14 Checking out fdd1114e as master...
15 Removing .pytest_cache/
16 Removing __pycache__/
17 Skipping Git submodules setup
19 Executing "step_script" stage of the job script     00:04
20 $ pytest
```

Duration: 13 seconds
Finished: 1 month ago
Timeout: 15m (from runner)
Runner: #49 (tBsSVHYJ) COMP1531 Primary Runner

Commit fdd1114e
Ready to go

Pipeline #922720 for `master`

checks

→ pytest

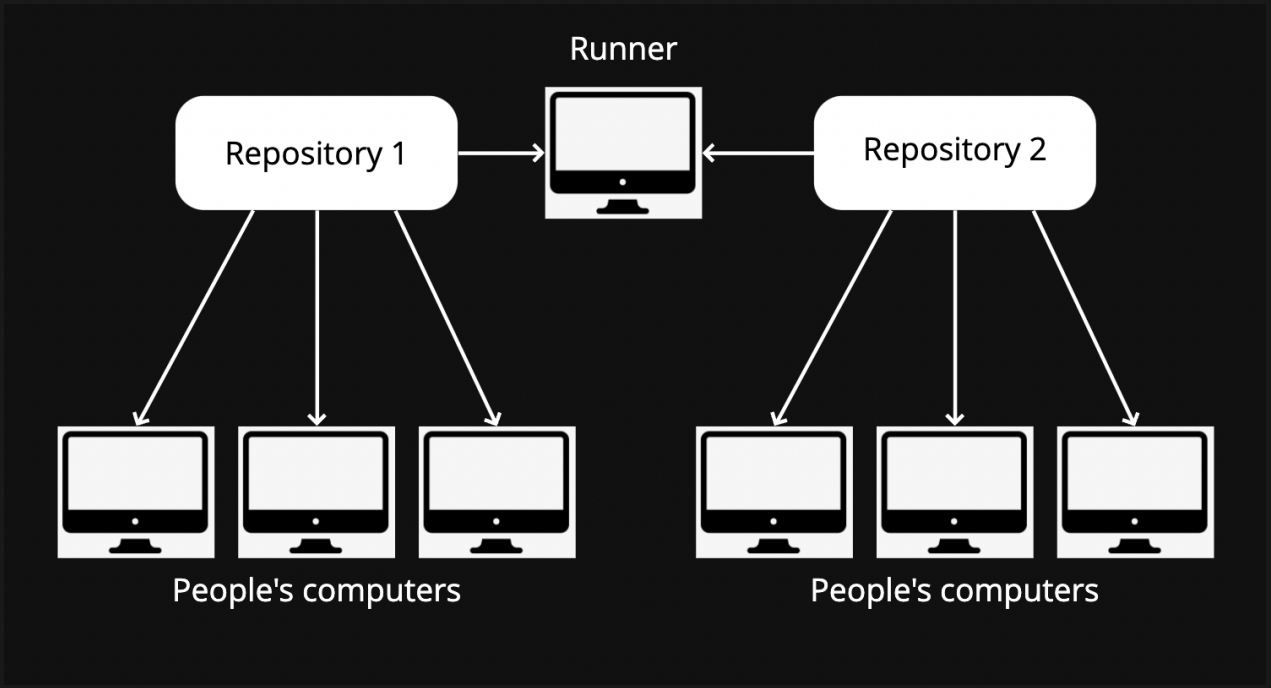
This is what we call the **pipeline** - a summary of what was run.

🔨 How It Works

When a commit is pushed, all of the code in that commit is taken by another computer (or "runner") and has the `.gitlab-ci.yml` instructions run on it.

🔨 How It Works

Architecture



🔨 How It Works

A runner really is just another computer whose sole job it is to run these "pipelines".

For more commercial products **github** and **bitbucket**, they have an array of runners that are used for people with git repositories. These tend to have free usage limits and then they start charging.

For **gitlab**, runners are not build in, but we've setup a runner for you. This runner runs on any `.gitlab-ci.yml` configuration that is pushed within the COMP1531 repos on gitlab.

⚙️ Configuring

Now let's add **jest** to the pipeline!

```yaml
image: comp1531/basic:latest

cache:
  paths:
    - node_modules

stages:
  - checks

testing:
  stage: checks
  script:
    - npm run test
```

3.2_gitlab-ci_test.yml

⚙️ Configuring

Normally you would have an extra step here for `npm install`!

```
1 testing:
2   stage: checks
3   script:
4     - npm install
5     - npm run test
```

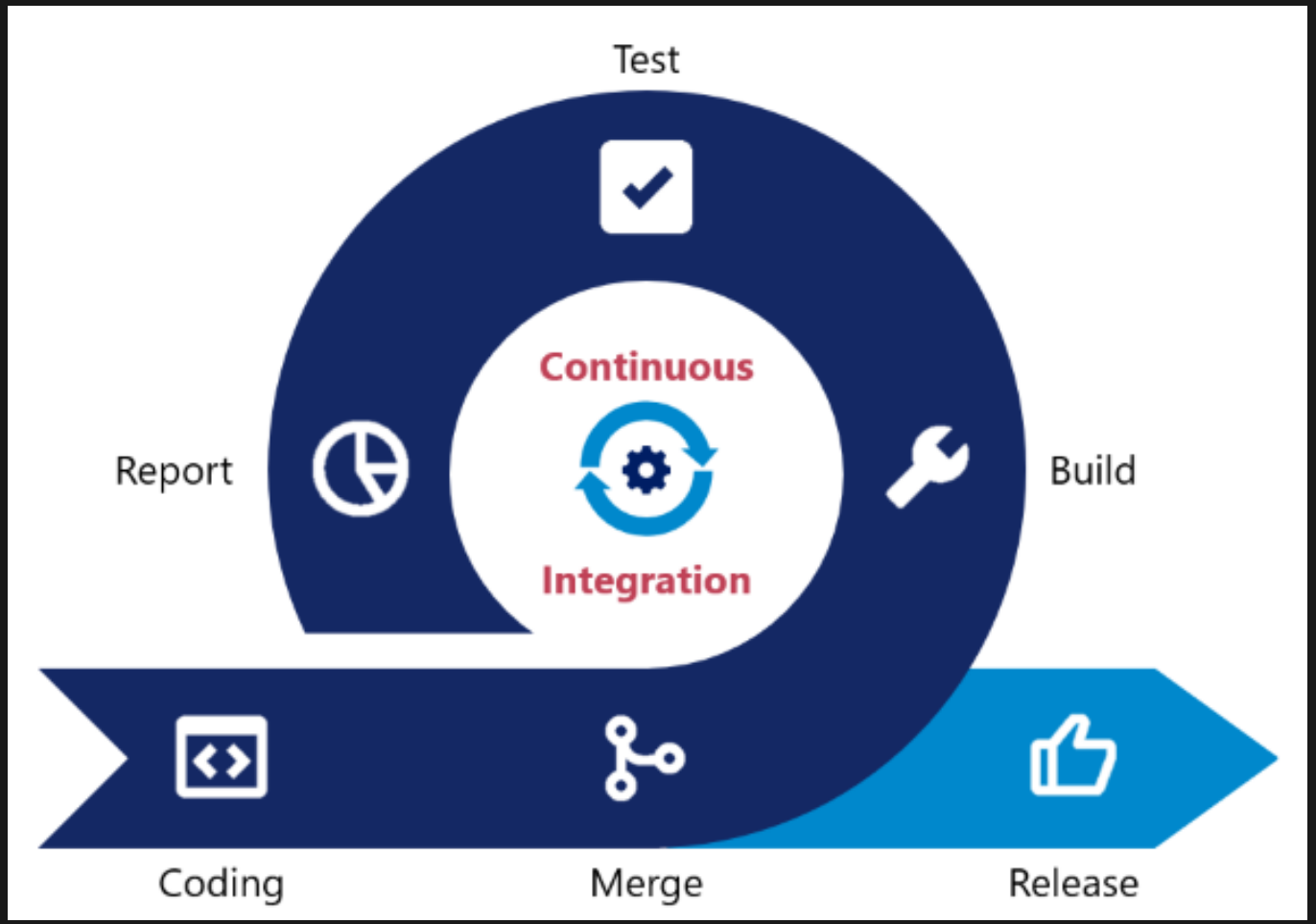
💑 Continuous Integration

In summary, continuous integration assists us in making frequent code changes, because we can:

1. Write tests
2. Write implementation
3. Push to gitlab + add merge request
4. Make sure we have the green tick
5. Merge in

Confidence!

💑 Continuous Integration



💑 Continuous Integration

An important rule to follow is that your **master** branch should ALWAYS be green. No code should be merged into it unless you're getting the green tick.

📚 Further Reading

You should definitely read the following:

- Gitlab Continuous Integration
- Atlassian Continuous Integration

👂 Feedback

Or go to the form here.