

# Introduction to Continuous-X services at NHR@KIT

Holger Obermaier, Michele Mesiti | 4. July 2024



# Table of Contents

## 1. Motivation

## 2. Continuous-X at NHR@KIT

## 3. GitLab: Examples and Exercises

## What is the X in Continuous-X (CX)

- Continuous Integration
- Continuous Testing
- Continuous Benchmarking
- Continuous Deployment

# Question

Have you ever worked with software that was hard (or even impossible) to change or improve?

What was the reason?

# Why continuous X?

Shortening the feedback loop between *design* and *use* of software (this includes *development*) increases the speed at which *useful* software can be produced.

This has been made relevant by the ability to deliver software multiple times a day (e.g., via Internet) compared to a couple of times a year (e.g., by *physically shipping* CDs or Floppy Disks). This typically gives a competitive advantage to businesses that adopt proper Cx practices <sup>1</sup>.

Here “continuous” means: in cycles that are much shorter (typically - less than one day) than they used to be. Cycles are made shorter thanks to *automation*.

---

<sup>1</sup>Forsgren, Humble, Kim - *Accelerate*

# Workflows in Software Development

## Git Flow

Introduced in 2010<sup>2</sup>, prescribes using these branches:

- `main` to release the code
- `release-*`, for release candidates (starting from `develop`)
- `develop`, the “main” branch for development
- feature branches - one per feature, merge to `develop` with `--no-ff`
- `hotfix-*` (starting from `main`)

This complexity might be necessary in some cases (e.g. to maintain multiple versions at the same time) but it is typically overkill for many projects.

The main problem is when feature branches live for long time.

The longer they live, the harder the merge.

---

<sup>b</sup>Original post by Vincent Driessen [↗](#)

# Workflows in Software Development

## GitHub Flow

A simpler branching model:

- Use the `main` branch to release the code
- Create a branch for any change
- open a pull/merge request, make changes and discuss them (*code review*)

## Trunk-Based Development

(Almost) No branches!

- `develop` (almost) only on `main`
- use pair programming instead of code reviews
- strong **automatic** tests are needed
- This is a key enable of *continuous integration* as a development practice

# Why automate?

- gatekeeping/enforcing coding standard on contributions from others
- parts of the development cycle that do not fit on the machine you use to develop code.

## Typically...

- Build and publish your work on the web (including documentation)
- Run *all* unit and integration tests, and static analysis tools
- Measure test coverage
- Check code style and practices (Linting, auto-formatting)
- Test compilation with different compilers and libraries (e.g., MPIs, CUDA/ROCm, ...)
- Test and benchmark on specific hardware ⇐
- Automated building/packaging for various compilers, MPI versions, architectures, ...
- ... all using standardized environments in conjunction with containers



# A few words on Automated Testing

Testing is one of the most important tasks that can be (at least partially) automated.

Having a test suite that runs quickly is important (even if you launch it asynchronously).

Test Coverage can be used to estimate how much code is tested (gives only an upper bound - remember Goodhart's Law [↗](#))

Having a test suite that runs quickly and has meaningful coverage is a problem that requires careful software design.

# Continuous-X Service at NHR@KIT


## 1. Motivation

## 2. Continuous-X at NHR@KIT

- Introduction
- Runner Setup
- CI levels in detail

## 3. GitLab: Examples and Exercises

# GitLab runner, pipelines and jobs

- GitLab CI/CD pipelines files are composed by jobs.
- The execution of the jobs is managed by an application called `gitlab-runner` .
- A GitLab server can be connected with `gitlab-runner` by *registering a runner* (there can be more than one *runner* per `gitlab-runner` application).
- The GitLab server (the *coordinator*) requests then *runner* to execute single jobs, providing all the necessary code and data and collecting the results.

SCC manages an installation of GitLab for KIT-affiliated and also externals (<https://gitlab.kit.edu>).  
For more information (e.g. access), see the documentation (English / Deutsch )

## Where runners live?

It is possible to request runners to be configured on `gitlab.kit.edu` (see how to request one).

In what follows, we assume you need to configure one yourself, because, for example:

- you want to do Continuous Benchmarking
- you want to test your code in the environment provided on a HPC cluster.

# Continuous-X Service Levels at NHR@KIT

Level	Number jobs/day	Runner on	Self-managed Runner	Permanent Runner	Full Hardware Access (e.g. GPUs)	Container support	Jobs with timing constraints
1	Low	Compute Node	✓	✗	✓	✓	✗
2	Medium	Dedicated Login Node	✓	✓	✓	✓	medium workloads
3	High	Dedicated Hardware	✗	✓	✓	✓	✓

See: [NHR@KIT User Documentation: Continuous Integration: Overview](#) ↗

# GitLab: Setting up a Runner

The `gitlab-runner` application is used to:

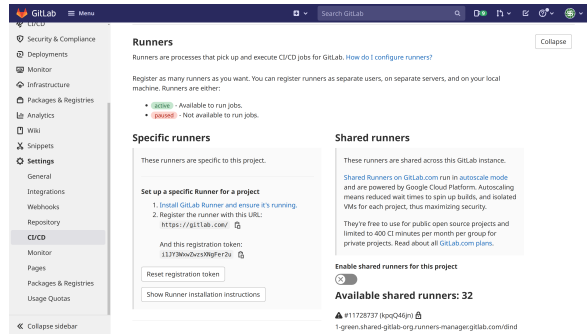
- *create* a runner from the web interface, which assigns a *token* to the runner;
- *register* runners for a GitLab instance or repository, with the `register` subcommand, passing the token you got from the web interface. Multiple runners can be registered, for multiple GitLab servers and repositories, on the same machine, but each runner needs its own token.
- *launch* all the registered runners for the user: this is done with the `run` subcommand. This command will eventually run the pipeline.

The capability of each registered runner depends on:

- The *executor* used by the runner (e.g., `shell`, `docker`, `custom`)
- The *host* the runner lives on (e.g. a login node or a compute node), i.e., on which host the `gitlab-runner run` command was called, or the CI level

# GitLab: Settings: CI/CD: Runner

Example Repository 



The screenshot shows the GitLab web interface for configuring CI/CD runners. The left sidebar contains navigation options like Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, Analytics, Wiki, Snippets, and Settings. The 'Settings' menu is expanded to show 'CI/CD' as the selected option. The main content area is titled 'Runners' and includes a 'Collapse' button. It explains that runners are processes that pick up and execute CI/CD jobs. Below this, there are two sections: 'Specific runners' and 'Shared runners'. The 'Specific runners' section provides instructions on how to set up a specific runner for a project, including a list of steps and a registration token. The 'Shared runners' section explains that these runners are shared across the entire GitLab instance and are powered by Google Cloud Platform. At the bottom, there is a toggle for 'Enable shared runners for this project' and a list of available shared runners, showing 32 available runners.

**Runners**

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

Register as many runners as you want. You can register runners as separate users, on separate servers, and on your local machine. Runners are either:

- **active** - Available to run jobs.
- **paused** - Not available to run jobs.

**Specific runners**

These runners are specific to this project.

**Set up a specific Runner for a project**

1. Install [GitLab Runner](#) and ensure it's running.
2. Register the runner with this URL:  
`https://gitlab.com/`

And this registration token:  
`11Z730w2ez399fEr2y`

**Shared runners**


These runners are shared across this GitLab instance.

[Shared Runners on GitLab.com](#) run in [autoscale mode](#) and are powered by Google Cloud Platform. Autoscaling means reduced wait times to spin up builds, and isolated VMs for each project, thus maximizing security.

They're free to use for public open source projects and limited to 400 CI minutes per month per group for private projects. Read about all [GitLab.com plans](#).

**Enable shared runners for this project**

**Available shared runners: 32**

▲ #11728737 (kpcQ46jn)   
[1-green.shared-gitlab-org.runners-manager.gitlab.com/dind](#)

# Register a GitLab Runner on the HoreKa Cluster

- Login to dedicated CI node

```
ssh hk-ci-controller.scc.kit.edu
```

```
(bq0742@hk-ci-controller.scc.kit.edu) Your OTP: <OTP>  
(bq0742@hk-ci-controller.scc.kit.edu) Password: <Password>  
...  
Last login: Mon Oct 25 16:15:10 2021 from 2a00:1398:4:1801::810d:3bc6  
[ ] [bq0742@hkn1993] [Mon. 2021-11-08 15:07:19]  
[~] $
```



# Register a GitLab Runner on the HoreKa Cluster (continued)

- Register and configure GitLab Runner

```
gitlab-runner register
```

- Enter the GitLab instance URL: e.g. `https://gitlab.com/`
  - Enter the registration token: <Token from GitLab: Settings: CI/CD: Runners>
  - Enter a description for the runner: e.g. GitLab Runner at NHR@KIT
  - Enter an executor: shell
- Configuration is written to `${HOME}/.gitlab-runner/config.toml`
  - Execute GitLab Runner

```
gitlab-runner run
```

# Register a GitLab Runner on the HoreKa Cluster (continued)

Executors types supported by the GitLab Runner:

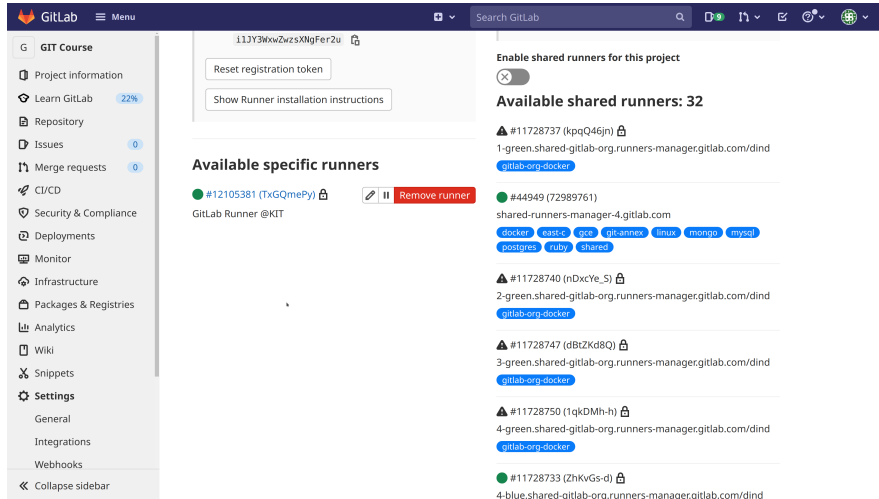
- Shell (simplest)
- Custom (allows customization to our system, e.g. using containers and/or Slurm)
- SSH
- Docker
- VirtualBox
- ...

# Register a GitLab Runner on the HoreKa Cluster (Containers on Slurm)

## Using Containers

- NHR@KIT User Documentation: Using Containers [↗](#)
- **Custom** executor instead of shell executor
- No native Docker support (security constraints)
- Enroot [↗](#): Root-less execution of Docker images
- Template folder: `/usr/share/gitlab-runner/custom-executor-enroot`
- Template GitLab Runner config: `config.toml`
- GitLab Runner config uses prepared scripts: `config.sh`, `prepare.sh`, `run.sh`, `cleanup.sh`
- `.gitlab-ci.yml` uses keyword `image` to configure Docker image


# GitLab Settings: CI/CD: Runners



The screenshot shows the GitLab web interface for configuring CI/CD runners. The left sidebar contains navigation options: GIT Course, Project information, Learn GitLab (22%), Repository, Issues (0), Merge requests (0), CI/CD, Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, Analytics, Wiki, Snippets, and Settings (General, Integrations, Webhooks, Collapse sidebar). The main content area is titled "Available specific runners" and shows one runner: #12105381 (TxGQmePy) with a "Remove runner" button. Above this, there are buttons for "Reset registration token" and "Show Runner installation instructions". On the right, the "Enable shared runners for this project" toggle is turned off, and "Available shared runners: 32" are listed. These include runners #11728737, #44949, #11728740, #11728747, #11728750, and #11728733, each with a "gitlab-org-docker" tag and a list of supported languages like docker, east-e, gce, git-annex, linux, mongo, mysql, postgres, ruby, and shared.

# CI Level 1 at NHR@KIT

## CI Level 1

- GitLab Runner is executed by a *batch job*
- GitLab Server is contacted by the runner, the batch job starts
- Requires access from compute node to GitLab Server
- GitLab Runner quits when all waiting CI jobs are executed
- Problem: Start of the GitLab Runner job is unknown in advance
- For repeating GitLab Runner jobs use `scrontab` (consider also scheduled pipelines  )
- ⇒ Best suited for *nightly builds* and *nightly integration tests*

## CI Level 1 at NHR@KIT (continued)

- Prerequisite: Register a GitLab Runner on HoreKa Cluster
- In CI Level 1 the GitLab Runner is executed as a batch job

```
sbatch \  
  --wrap="gitlab-runner run" \  
  --time="00:30:00" \  
  --partition="dev_cpuonly"
```

# CI Level 1 at NHR@KIT (continued)

- Alternative method: Create a batch script for submission:

```
#!/usr/bin/bash
#SBATCH --partition dev_cpuonly
#SBATCH --time 00:30:00

# Prepare your environment
module add compiler/intel mpi/impi numlib/mkl
module list

# Start GitLab Runner
gitlab-runner run
```

## CI Level 1 at NHR@KIT (continued)

- Use `scrontab -e` to set up regular GitLab Runner jobs:

```
#SCRON -p dev_accelerated
#SCRON -t 00:30:00
@midnight gitlab-runner run
```

- Jobs are not guaranteed to execute at the preferred time!
- Jobs are regularly queued in the batch system:

```
squeue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
1503545	dev_accel	gitlab-r	bq0742	PD	0:00	1	(BeginTime)

- You can make the slurm job trigger a pipeline.



# CI Level 2 at NHR@KIT

- Prerequisite: Register a GitLab Runner on HoreKa Cluster
- In CI Level 2 the GitLab Runner
  - is executed as systemd user service (you need *linger* enabled - contact us!)
  - runs on dedicated login node (e.g. `hk-ci-controller.scc.kit.edu`)
    - Limited / shared resources ⇒ Runtime variations
    - Only suited for medium workloads
    - No access to special hardware
  - is self-managed
    - Systemd management
    - CI jobs can start immediately

## CI Level 2 at NHR@KIT (continued)

- Start GitLab Runner Service

```
systemctl --user start gitlab-runner.service
```

- Get status of GitLab Runner Service

```
systemctl --user status gitlab-runner.service
```

```
gitlab-runner.service - GitLab Runner for bq0742
Loaded: loaded (/etc/systemd/user/gitlab-runner.service; disabled; vendor
↳ preset: enabled)
Active: active (running) since Tue 2021-11-09 11:45:41 CET; 3s ago
Main PID: 1167130 (gitlab-runner)
CGroup:
↳ /user.slice/user-8946.slice/user@8946.service/gitlab-runner.service
    1167130
```

## CI Level 2 at NHR@KIT (continued)

- Read log output of GitLab Runner Service

```
journalctl --user --unit gitlab-runner.service
```

```
WARNING: Running in user-mode.  
WARNING: Use sudo for system-mode:  
WARNING: $ sudo gitlab-runner...
```

```
Configuration loaded                               builds=0  
listen_address not defined, metrics & debug endpoints disabled builds=0  
[session_server].listen_address not defined, session endpoints disabled  
↪ builds=0
```

## CI Level 2 at NHR@KIT (continued)

- Stop GitLab Runner Service

```
systemctl --user stop gitlab-runner.service
```

# CI Level 2 at NHR@KIT: A special executor

A Custom Executor has been developed for HoreKa and bwUniCluster 2.0 that can reside on the login node (as for CI level 2) and launch jobs on the compute nodes via Slurm, while using Enroot or Singularity (now Apptainer) for containers.

This helps to have the same performance reliability as CI level 1 but without having to manually start the `gitlab-runner`, among other things.

Installation/Registration steps:

- 1 Clone this repository [↗](#) to HoreKa/bwUniCluster 2.0/Future Technologies Partition
- 2 To register a new runner, execute the script in the repository `utils/gitlab-runner-register-wrapper.sh` and follow the instructions (launch it before with the `--help` option to see what is available)
- 3 for more information, refer to the README [↗](#)

# CI Level 3 at NHR@KIT

- Dedicated hardware
- For projects that
  - generate many CI jobs per day
  - need predictable runtimes and performance
  - require privileged access to special resources
- Available hardware: see e.g. the documentation [↗](#) for the Future Technologies Partition, and for HoreKa [↗](#)
- Get in contact with CI Operations Team

# Register a GitLab Runner on the HoreKa Cluster: recap

## Options:

- Wrap `gitlab-runner run` in a time-limited slurm job (CI level 1)
- Schedule a pipeline - either manual launch or with `scrontab`, possibly in conjunction with Scheduled Pipelines [↗](#), or trigger a pipeline [↗](#) from inside the slurm job (CI level 1)
- Run permanently on the login node, using a shell executor (CI level 2)
- Run permanently on the login node, using a custom executor that submits jobs automatically to a compute node (CI level 2)

# GitLab: Examples and Exercises

## 1. Motivation

## 2. Continuous-X at NHR@KIT

- Introduction
- Runner Setup
- CI levels in detail


## 3. GitLab: Examples and Exercises



# GitLab: the CI/CD interface

- GitLab CI/CD Documentation:
  - Get started with GitLab CI/CD [↗](#)
  - Keyword reference for the `.gitlab-ci.yml` file [↗](#)
  - GitLab CI templates [↗](#)
- YAML-config file `gitlab-ci.yml`
- GitLab offers integrated CI-editor with visualization and linting

# GitLab: the CI/CD interface (continued)

- Each push triggers a new **pipeline** (unless skipped )
- Each pipeline consist of **stages**
  - Predefined stages: `.pre`, `build`, `test`, `deploy`, `.post`
  - Stages run in sequence
  - Working tree is typically cleaned up between stages: Use **artifacts** to keep files
- Each stage consist of **jobs**
  - Jobs in the same stage can run in parallel
- If a job fails then the stage fails, and all subsequent stages are skipped
- If a stage fails then the pipeline fails

# Exercises and Examples

Repository with examples and exercises [↗](#)

- Tip: when trying the exercises, disable email notifications regarding pipeline events (click on the bell icon in the main page of the repository and select “Disabled”)
- For more examples: the official GitLab CI/CD template collection [↗](#)

## Disclaimer

The selection of features listed here is based on my personal experience. It is only a selection: to have a broader view of what is available, look at the templates and at the official documentation.

# Exercise: A Basic Pipeline

A single-stage pipeline that compiles and run a C program.

- 1 “Fork” repository on a GitLab server
- 2 Configure runner for the cloned repository
- 3 Briefly consider the security aspects (what code is the runner going to run? Where? When?)
- 4 Trigger the pipeline on the `main` branch

## Exercise: Failures

A failing pipeline gives an important message about the status of our code, so it is important that when there is a failure in the code we run (e.g., test cases or benchmarks)

- 1 Run the pipeline on the `failure-01` branch. It fails: why? Is it expected?
- 2 Run the pipeline on the `fail-failed-01` branch. It does not fail: why? Is this expected?
- 3 Run the pipeline on the `fails-correctly-again` branch. It fails: why? Is it expected? What was changed?

How much code you write in your `.gitlab-ci.yml` and how much code should you write in bash scripts?

# Exercise: Environment Variables

To customize the behaviour of the test code, GitLab offers a few ways to set environment variables with different degrees of secrecy.

Checkout the `environment-variables` branch and follow the instruction in `README.md`.

## Example: Artifacts


Artifacts are the main way to transfer information between jobs in a pipeline, and from the runner to the GitLab web interface.

- 1 Checkout the `artifacts` branch
- 2 have a look at the `.gitlab-ci.yml` file
- 3 Try to run the pipeline. Does the behaviour depend on the executor type (e.g., `docker` vs `shell`)?

# Example: Pipeline code reuse with templates and extends

Don't Repeat Yourself: reuse code in pipeline definition

- 1 Checkout branch templates
- 2 have a look at the `.gitlab-ci.yml` file, notice the use of `.greet-base` and of the `extends` key
- 3 How can the base job be customized?

See also: [yaml anchors](#) 



## Example: Managing complexity with include:

- 1 Checkout branch `include`
- 2 have a look at the `.gitlab-ci.yml` file, and the use of `include`:
- 3 How can you include multiple files? How is the content merged?

# GitLab CI/CD: Exercise: Mirroring

It is possible to periodically mirror the content of a repository to and from another one.

- Push mirror: the mirror is updated when a push to the repo is made (docs) [↗](#)
- Pull mirror: the mirror is updated periodically (polling) (docs) [↗](#)

**Note:** Setting up a pull mirror on GitLab requires GitLab Premium (at the time of writing, this is available on `gitlab.kit.edu`)

# GitLab CI/CD: Exercise: Mirroring from GitHub to GitLab

Depending on the GitLab tier available, it might be impossible to set up a pull mirror. As a workaround, it is possible to use GitHub actions to push.

- 1 Create an empty repository on GitHub and push the example repo there (tip: disable notifications)
- 2 Create an empty repository on a GitLab server (tip: disable notifications)
- 3 Checkout the `github-to-gitlab-mirror` branch in the example repo
- 4 Follow the instructions in the `README.md` to set up the mirroring

# Basic Git: Exercise: Git Hooks

The basic features of CI can be replicated without GitLab or GitHub, but just setting up a bare repository and the relevant hooks.

- 1 Checkout branch hooks
- 2 Follow instructions in `README.md`

# GitLab CI/CD: Test results and coverage

GitLab can show the results [↗](#) of a test suite in a more convenient way.

Coverage reports produced as static websites [↗](#) can also be uploaded on GitLab pages.

If the GitLab pages feature is not active, one can use third party services to host the coverage reports [↗](#).

Example on [gitlab.com](https://gitlab.com) [↗](#)

**Exercise:** fix the tests and reach 100% code coverage.

# HPC: a MPI job in a pipeline

Create a job that launches the program `hostname` on 2 different nodes, using:

- 1 The shell executor (suggestion: either use `srun`, or prepare a shell script to launch with `salloc`).
- 2 The custom executor (suggestion: use `srun hostname` in the job body, set the `COMMAND_OPTIONS_SBATCH` variable appropriately).

# HPC: Run the NVidia HPL benchmark as part of a CI pipeline


NVidia provides a container image [↗](#) for its HPC benchmarks.


The exercise: run the HPL benchmark on a GPU node on HoreKa.

It requires registering an account at [ngc.nvidia.com](https://ngc.nvidia.com), setting up authentication for Enroot, and registering a runner for the repository of the exercise.

The exercise is described/sketched here [↗](#).

# Conditionally running a job in a pipeline

It is possible to use job rules  to limit the execution of some jobs to some events, e.g.:

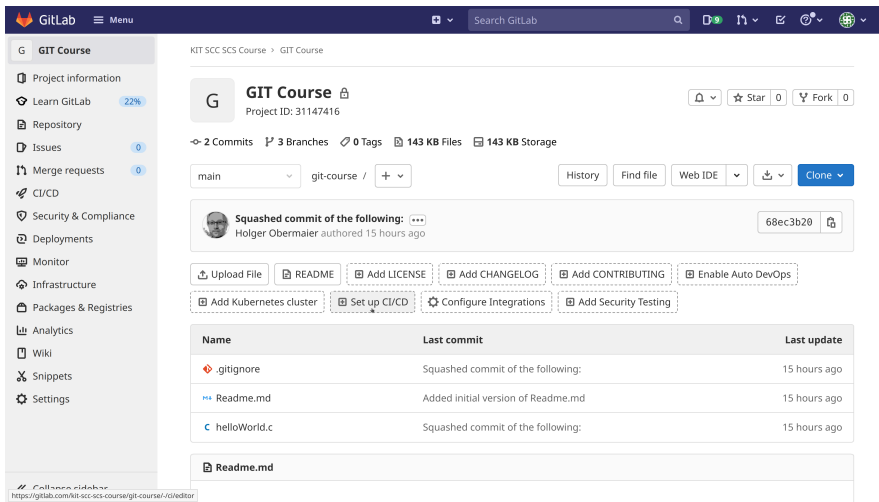
- when a tag is pushed, to create a release on a registry/data repository (and optionally obtain a DOI to use when citing the software/dataset)
- when a merge request is processed, or when a scheduled run happens
- only on specific branches
- only when some files change 



# Screenshot Gallery

A set of reference pictures (a substitute for the live demonstration)

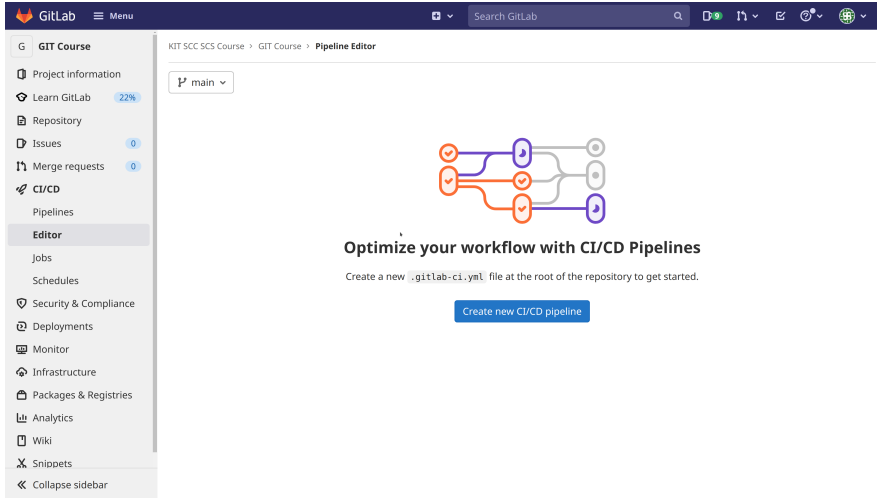
# GitLab: the CI/CD interface (Set up CI/CD)



The screenshot displays the GitLab interface for a repository named 'GIT Course'. The left sidebar contains navigation options such as 'Project information', 'Learn GitLab', 'Repository', 'Issues', 'Merge requests', 'CI/CD', 'Security & Compliance', 'Deployments', 'Monitor', 'Infrastructure', 'Packages & Registries', 'Analytics', 'Wiki', 'Snippets', and 'Settings'. The main content area shows the repository details, including the project ID '31147416', commit count '2', branch count '3', and storage usage '143 KB'. A 'Squashed commit of the following' section shows a commit by Holger Obermaier. Below this, there are buttons for 'Upload File', 'README', 'Add LICENSE', 'Add CHANGELOG', 'Add CONTRIBUTING', 'Enable Auto DevOps', 'Add Kubernetes cluster', 'Set up CI/CD', 'Configure Integrations', and 'Add Security Testing'. A table lists files with their names, last commit messages, and update times. The 'Set up CI/CD' button is highlighted in the file list.

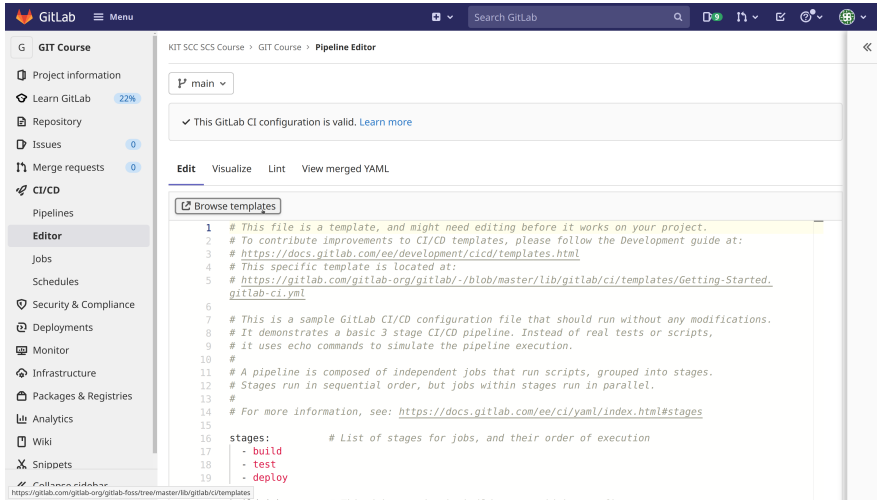
Name	Last commit	Last update
.gitignore	Squashed commit of the following:	15 hours ago
README.md	Added initial version of README.md	15 hours ago
helloWorld.c	Squashed commit of the following:	15 hours ago

# GitLab: the CI/CD interface (CI/CD Editor)



The screenshot shows the GitLab web interface for editing CI/CD pipelines. The top navigation bar includes the GitLab logo, a menu icon, and a search bar. The left sidebar contains a navigation menu with categories like 'Project information', 'Learn GitLab', 'Repository', 'Issues', 'Merge requests', 'CI/CD', 'Editor', 'Jobs', 'Schedules', 'Security & Compliance', 'Deployments', 'Monitor', 'Infrastructure', 'Packages & Registries', 'Analytics', 'Wiki', 'Snippets', and 'Collapse sidebar'. The main content area displays the 'Pipeline Editor' for a project named 'KIT SCC SCS Course'. It shows a dropdown menu for the 'main' branch and a central graphic of a pipeline with nodes. Below the graphic, the text reads 'Optimize your workflow with CI/CD Pipelines' and 'Create a new .gitlab-ci.yml file at the root of the repository to get started.' A blue button labeled 'Create new CI/CD pipeline' is positioned below the text.

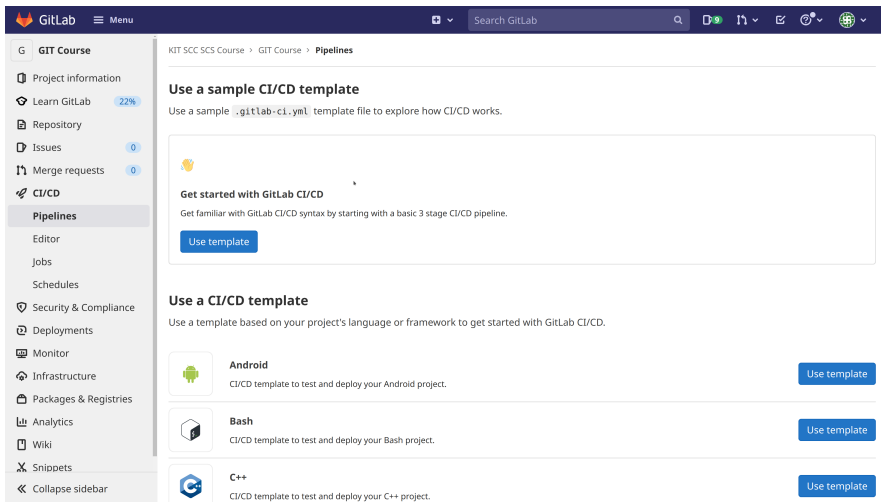
# GitLab: the CI/CD interface (Browse Templates)



The screenshot displays the GitLab Pipeline Editor interface. The top navigation bar includes the GitLab logo, a menu icon, and a search bar. The left sidebar contains a navigation menu with categories like 'GIT Course', 'CI/CD', and 'Security & Compliance'. The main content area is titled 'Pipeline Editor' and shows a dropdown menu for 'main'. A message indicates that the GitLab CI configuration is valid. Below this, there are tabs for 'Edit', 'Visualize', 'Lint', and 'View merged YAML'. The 'Browse templates' section is active, displaying a list of template files with their contents. The first template is highlighted in yellow.

```
1 # This file is a template, and might need editing before it works on your project.
2 # To contribute improvements to CI/CD templates, please follow the Development guide at:
3 # https://docs.gitlab.com/ee/development/cicd/templates.html
4 # This specific template is located at:
5 # https://gitlab.com/gitlab-org/gitlab/-/blob/master/lib/gitlab/ci/templates/Getting-Started.
  gitlab-ci.yml
6
7 # This is a sample GitLab CI/CD configuration file that should run without any modifications.
8 # It demonstrates a basic 3 stage CI/CD pipeline. Instead of real tests or scripts,
9 # it uses echo commands to simulate the pipeline execution.
10 #
11 # A pipeline is composed of independent jobs that run scripts, grouped into stages.
12 # Stages run in sequential order, but jobs within stages run in parallel.
13 #
14 # For more information, see: https://docs.gitlab.com/ee/ci/yaml/index.html#stages
15
16 stages:           # List of stages for jobs, and their order of execution
17   - build
18   - test
19   - deploy
```

# GitLab: the CI/CD interface (CI/CD Templates)



The screenshot shows the GitLab web interface. The top navigation bar includes the GitLab logo, a menu icon, and a search bar. The left sidebar contains a navigation menu with categories like 'GIT Course', 'Project information', 'Learn GitLab', 'Repository', 'Issues', 'Merge requests', 'CI/CD', 'Pipelines', 'Editor', 'Jobs', 'Schedules', 'Security & Compliance', 'Deployments', 'Monitor', 'Infrastructure', 'Packages & Registries', 'Analytics', 'Wiki', 'Snippets', and 'Collapse sidebar'. The main content area is titled 'KIT SCC SCS Course > GIT Course > Pipelines'. It features a section 'Use a sample CI/CD template' with a sub-section 'Get started with GitLab CI/CD' and a 'Use template' button. Below this, there is another section 'Use a CI/CD template' with three options: 'Android', 'Bash', and 'C++', each with a 'Use template' button.

KIT SCC SCS Course > GIT Course > Pipelines

## Use a sample CI/CD template

Use a sample `.gitlab-ci.yml` template file to explore how CI/CD works.




### Get started with GitLab CI/CD

Get familiar with GitLab CI/CD syntax by starting with a basic 3 stage CI/CD pipeline.

[Use template](#)

## Use a CI/CD template

Use a template based on your project's language or framework to get started with GitLab CI/CD.

-  **Android**  
CI/CD template to test and deploy your Android project. [Use template](#)
-  **Bash**  
CI/CD template to test and deploy your Bash project. [Use template](#)
-  **C++**  
CI/CD template to test and deploy your C++ project. [Use template](#)

# GitLab: the CI/CD interface (`.gitlab-ci.yml`)

```
stages:
- build
- test

build-hello-world-job:
  stage: build
  script: cc helloWorld.c -o helloWorld
  artifacts:
    paths:
    - helloWorld
    expire_in: 1 day

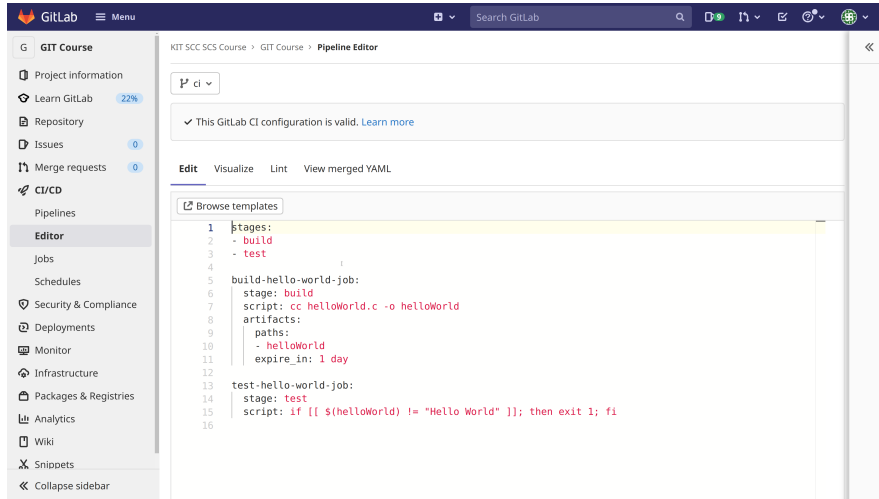
test-code-job1:
  stage: test
  script: if [[ $(helloWorld) != "Hello World" ]]; then exit 1; fi
```

# GitLab: the CI/CD interface (`.gitlab-ci.yml`)

- Test stage can use compute resources

```
...  
  
test-code-job2:  
  stage: test  
  script:  
    - srun -p dev_cpuonly      -t 20 -N 1 -n 76          CPU_Test.sh  
    - srun -p dev_accelerated -t 20 -N 1 -n 76 --gres=gpu:4 GPU_Test.sh
```

# GitLab: the CI/CD interface (Edit)

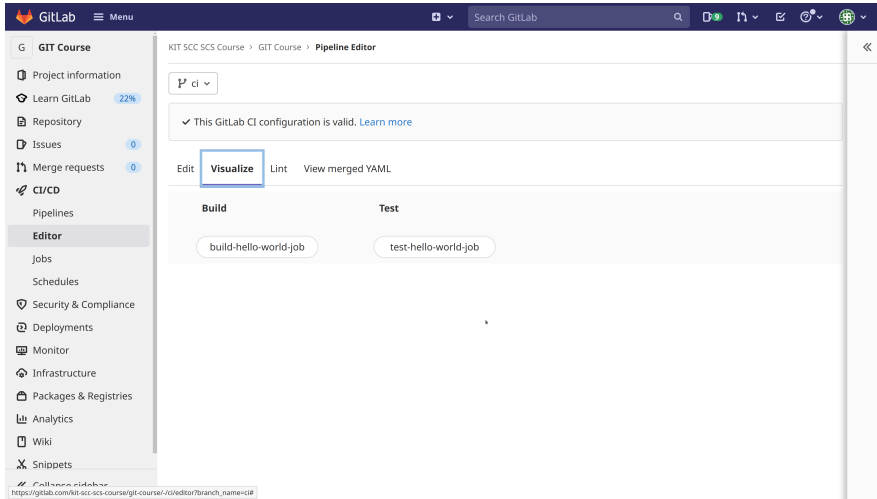


The screenshot shows the GitLab Pipeline Editor interface. The top navigation bar includes the GitLab logo, a menu icon, and a search bar. The left sidebar contains a navigation menu with categories like 'GIT COURSE', 'CI/CD', and 'Editor'. The main content area displays the pipeline configuration for 'KIT SCC SCS Course > GIT Course > Pipeline Editor'. A confirmation message states 'This GitLab CI configuration is valid. Learn more'. Below this, there are tabs for 'Edit', 'Visualize', 'Lint', and 'View merged YAML'. A 'Browse templates' button is visible. The pipeline configuration is shown in a code editor with line numbers 1 through 16. The configuration includes a 'stages' section with 'build' and 'test' stages, and two jobs: 'build-hello-world-job' and 'test-hello-world-job'.

```
1 stages:
2   - build
3   - test
4
5 build-hello-world-job:
6   stage: build
7   script: cc helloWorld.c -o helloWorld
8   artifacts:
9     paths:
10      - helloWorld
11     expire_in: 1 day
12
13 test-hello-world-job:
14   stage: test
15   script: if [[ $(helloWorld) != "Hello World" ]]; then exit 1; fi
16
```

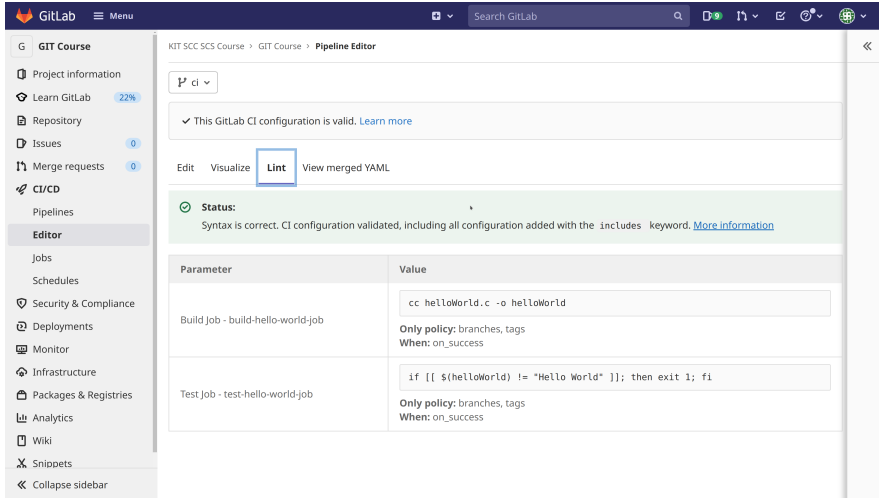


# GitLab: the CI/CD interface (Visualize)



The screenshot displays the GitLab Pipeline Editor interface. The top navigation bar includes the GitLab logo, a menu icon, and a search bar. The left sidebar contains a navigation menu with categories like 'GIT Course', 'Project information', 'Learn GitLab', 'Repository', 'Issues', 'Merge requests', 'CI/CD', and 'Editor'. The main content area shows the 'Pipeline Editor' for a project named 'KIT SCC SCS Course'. A dropdown menu is open, showing 'ci' as the selected pipeline. A confirmation message states 'This GitLab CI configuration is valid. Learn more'. Below this, there are three tabs: 'Edit', 'Visualize' (which is highlighted with a blue border), and 'Lint'. The 'Visualize' tab displays a visual representation of the pipeline with two jobs: 'build-hello-world-job' under the 'Build' stage and 'test-hello-world-job' under the 'Test' stage. The URL at the bottom of the page is [https://gitlab.com/kit-scc-scs-course/git-course/ci/editor?branch\\_name=ci#](https://gitlab.com/kit-scc-scs-course/git-course/ci/editor?branch_name=ci#).

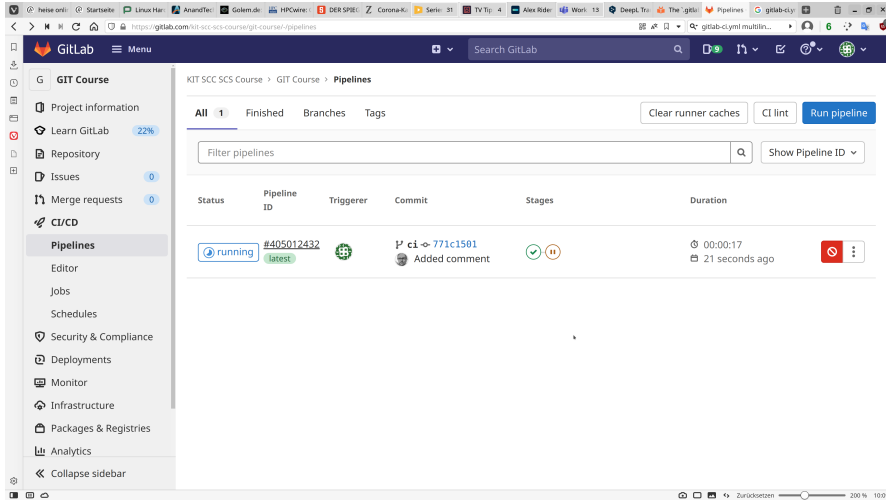
# GitLab: the CI/CD interface (Lint)



The screenshot shows the GitLab Pipeline Editor interface. The left sidebar contains navigation options: GIT Course, Project information, Learn GitLab (22%), Repository, Issues (0), Merge requests (0), CI/CD, Pipelines, Editor (selected), Jobs, Schedules, Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, Analytics, Wiki, Snippets, and Collapse sidebar. The main content area is titled "KIT SCC SCS Course > GIT Course > Pipeline Editor". It shows a dropdown menu with "ci" selected. A green message box indicates: "This GitLab CI configuration is valid. [Learn more](#)". Below this are tabs for "Edit", "Visualize", "Lint" (highlighted with a blue box), and "View merged YAML". A green status bar shows a checkmark and the text: "Status: Syntax is correct. CI configuration validated, including all configuration added with the includes keyword. [More information](#)". Below the status bar is a table with two columns: "Parameter" and "Value".

Parameter	Value
Build Job - build-hello-world-job	<pre>cc helloWorld.c -o helloWorld</pre> <p><b>Only policy:</b> branches, tags <b>When:</b> on_success</p>
Test Job - test-hello-world-job	<pre>if [[ \$(helloWorld) != "Hello World" ]]; then exit 1; fi</pre> <p><b>Only policy:</b> branches, tags <b>When:</b> on_success</p>

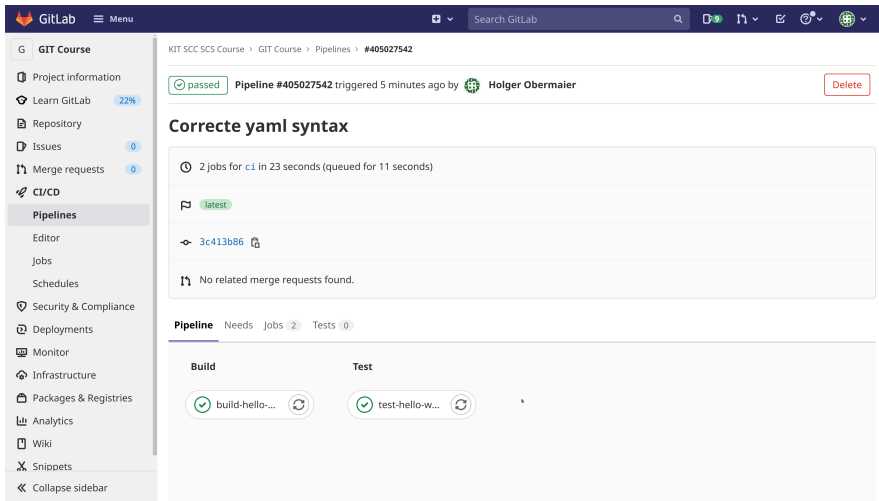
# GitLab: the CI/CD interface (CI/CD: Pipelines)



The screenshot shows the GitLab web interface for a project named 'GIT Course'. The left sidebar contains navigation options: Project Information, Learn GitLab (22%), Repository, Issues (0), Merge requests (0), CI/CD, Pipelines (selected), Editor, Jobs, Schedules, Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, Analytics, and Collapse sidebar. The main content area displays the 'Pipelines' section for the 'GIT Course' project. It includes tabs for 'All' (1), 'Finished', 'Branches', and 'Tags'. There are buttons for 'Clear runner caches', 'CI lint', and 'Run pipeline'. A search bar labeled 'Filter pipelines' and a dropdown for 'Show Pipeline ID' are also present. Below this is a table of pipeline runs:

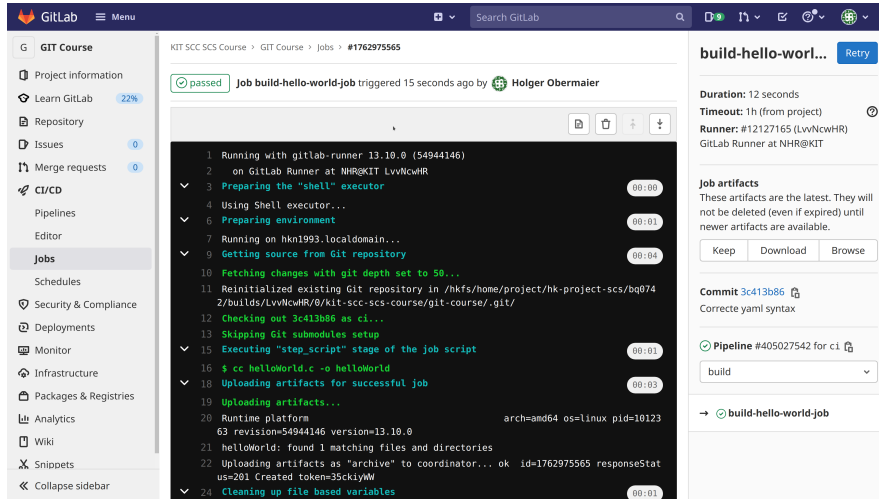
Status	Pipeline ID	Triggerer	Commit	Stages	Duration	
running	#405012432 latest		ci -> 771c1501 Added comment		00:00:17 21 seconds ago	

# GitLab: the CI/CD interface (Pipeline passed)



The screenshot displays the GitLab web interface for a pipeline. The top navigation bar includes the GitLab logo, a menu, and a search bar. The left sidebar contains navigation options such as 'GIT Course', 'Project information', 'Learn GitLab', 'Repository', 'Issues', 'Merge requests', 'CI/CD', 'Pipelines', 'Editor', 'Jobs', 'Schedules', 'Security & Compliance', 'Deployments', 'Monitor', 'Infrastructure', 'Packages & Registries', 'Analytics', 'Wiki', 'Snippets', and 'Collapse sidebar'. The main content area shows the pipeline details for 'KIT SCC SCS Course > GIT Course > Pipelines > #405027542'. The pipeline status is 'passed', triggered 5 minutes ago by 'Holger Obermaier'. The pipeline title is 'Correcte yaml syntax'. Below the title, it shows '2 jobs for ci in 23 seconds (queued for 11 seconds)'. The 'latest' commit is '3c413b86'. There are no related merge requests found. The pipeline summary shows 'Build' and 'Test' stages, both with '1' job completed successfully. The 'Build' job is 'build-hello-...' and the 'Test' job is 'test-hello-w...'. Each job has a refresh icon next to it.

# GitLab: the CI/CD interface (Build Job)



The screenshot displays the GitLab CI/CD interface for a job named "build-hello-world-job". The job status is "passed", triggered 15 seconds ago by Holger Obermaier. The interface is divided into three main sections: a left sidebar with navigation options, a central job log, and a right-hand summary panel.

**Left Sidebar:** Contains navigation links for Project information, Learn GitLab (22%), Repository, Issues (0), Merge requests (0), CI/CD (selected), Pipelines, Editor, Jobs (selected), Schedules, Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, Analytics, Wiki, and Snippets. A "Collapse sidebar" button is at the bottom.

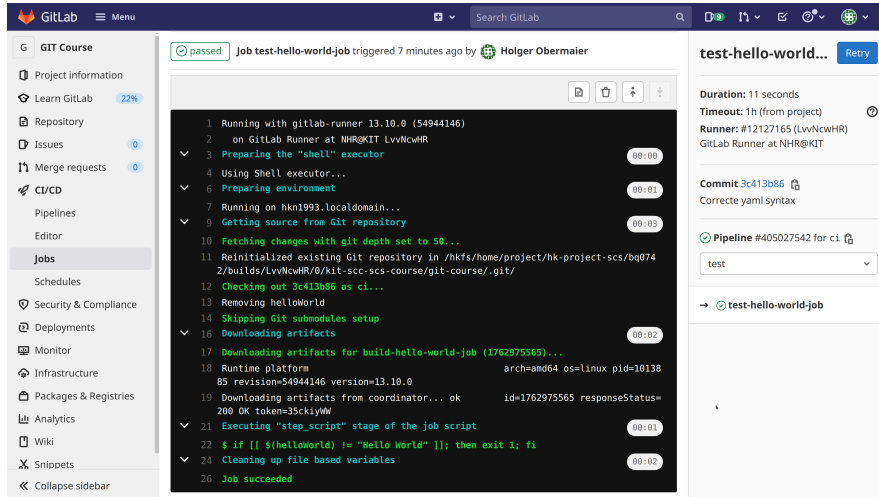
**Central Job Log:** Shows a list of 24 steps with their durations:

- 1 Running with gitlab-runner 13.10.0 (54944146)
- 2 on GitLab Runner at NHR@KIT LvNcwHR
- 3 **Preparing the "shell" executor** (00:00)
- 4 Using Shell executor...
- 5 **Preparing environment** (00:01)
- 6 Running on hkn1993.localdomain...
- 7 **Getting source from Git repository** (00:04)
- 8 **Fetching changes with git depth set to 50...**
- 9 Reinitialized existing Git repository in /hkfs/home/project/hk-project-scs/bq0742/builds/LvNcwHR/0/kit-scc-scs-course/git-course/.git/
- 10 **Checking out 3c413b86 as ci...**
- 11 **Skipping Git submodules setup**
- 12 **Executing "step\_script" stage of the job script** (00:01)
- 13 `$ cc helloWorld.c -o helloWorld`
- 14 **Uploading artifacts for successful job** (00:03)
- 15 **Uploading artifacts...**
- 16 Runtime platform arch=amd64 os=linux pid=10123
- 17 63 revision=54944146 version=13.10.0
- 18 helloWorld: found 1 matching files and directories
- 19 Uploading artifacts as "archive" to coordinator... ok id=1762975565 responseStat
- 20 us=201 Created token=35ckiywW
- 21 **Cleaning up file based variables** (00:01)

**Right-hand Summary Panel:**

- build-hello-worl...** with a "Retry" button.
- Duration:** 12 seconds
- Timeout:** 1h (from project)
- Runner:** #12127165 (LvNcwHR) GitLab Runner at NHR@KIT
- Job artifacts:** These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available. Buttons: Keep, Download, Browse.
- Commit:** 3c413b86 (Correct yaml syntax)
- Pipeline:** #405027542 for ci (build selected)
- Job:** → build-hello-world-job

# GitLab: the CI/CD interface (Test Job)



The screenshot displays the GitLab CI/CD interface. At the top, the status is "passed" and the job "test-hello-world-job" is noted as triggered 7 minutes ago by Holger Obermaier. The main area shows a log of the job's execution steps, including preparing the shell executor, environment, and source, followed by artifact downloading and script execution. The final step is "Job succeeded". On the right, job details are provided, such as a 11-second duration, a 1-hour timeout, and the runner information. A dropdown menu shows the pipeline ID and the job name.

```
1 Running with gitlab-runner 13.10.0 (54944146)
2 on GitLab Runner at NHR@KIT LvNcwHR
3 Preparing the "shell" executor 00:00
4 Using Shell executor...
5
6 Preparing environment 00:01
7 Running on hkn1993.localdomain...
8
9 Getting source from Git repository 00:03
10 Fetching changes with git depth set to 50...
11 Reinitialized existing Git repository in /hkfs/home/project/hk-project-scs/bq074
12 2/builds/LvNcwHR/0/kit-scc-scs-course/git-course/.git/
13 Checking out 3c413b86 as ci...
14 Removing helloWorld
15 Skipping Git submodules setup
16 Downloading artifacts 00:02
17 Downloading artifacts for build-hello-world-job (1762975565)...
18 Runtime platform arch=amd64 os=linux pid=10138
19 85 revision=54944146 version=13.10.0
20 Downloading artifacts from coordinator... ok id=1762975565 responseStatus=
21 200 OK token=35ckiywW
22 Executing "step_script" stage of the job script 00:01
23 $ if [[ $(helloWorld) != "Hello World" ]]; then exit 1; fi
24 Cleaning up file based variables 00:02
25
26 Job succeeded
```

test-hello-world... [Retry](#)

Duration: 11 seconds  
Timeout: 1h (from project) ⓘ  
Runner: #12127165 (LvNcwHR)  
GitLab Runner at NHR@KIT

Commit 3c413b86 [🔗](#)  
Correcte yaml syntax

✓ Pipeline #405027542 for ci [🔗](#)

test

→ test-hello-world-job