Testing and testing infrastructure

Andrey Alekseenko

SciLifeLab, KTH Royal Institute of Technology, Stockholm

2024-09-10

Testing

GROMACS

- “In simple terms, it’s a process of checking something if it does what it intended to do”
  https://keencoder.dev/unit-testing-for-absolute-beginners

Testing

GROMACS

- Unit tests: verify individual methods and functions
- Integration tests: verify that different modules work well together
- Functional tests: verify the final output of the application
- End-to-end tests: replicate a user behavior
- Acceptance testing: verify entire application for end goals
- Performance testing: verify reliability, speed, scalability
- Smoke testing: basic checks of major features

One of the possible nomenclatures; many others exists

Testing

GROMACS

- CI: practice of merging all developers' working copies to a shared mainline several times a day
- CD: software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time and, following an automated testing pipeline

+ Less conflicts between team members
+ Less need for manual testing
+ No need to follow later with (external) contributors

https://en.wikipedia.org/wiki/Continuous_integration

https://en.wikipedia.org/wiki/Continuous_delivery

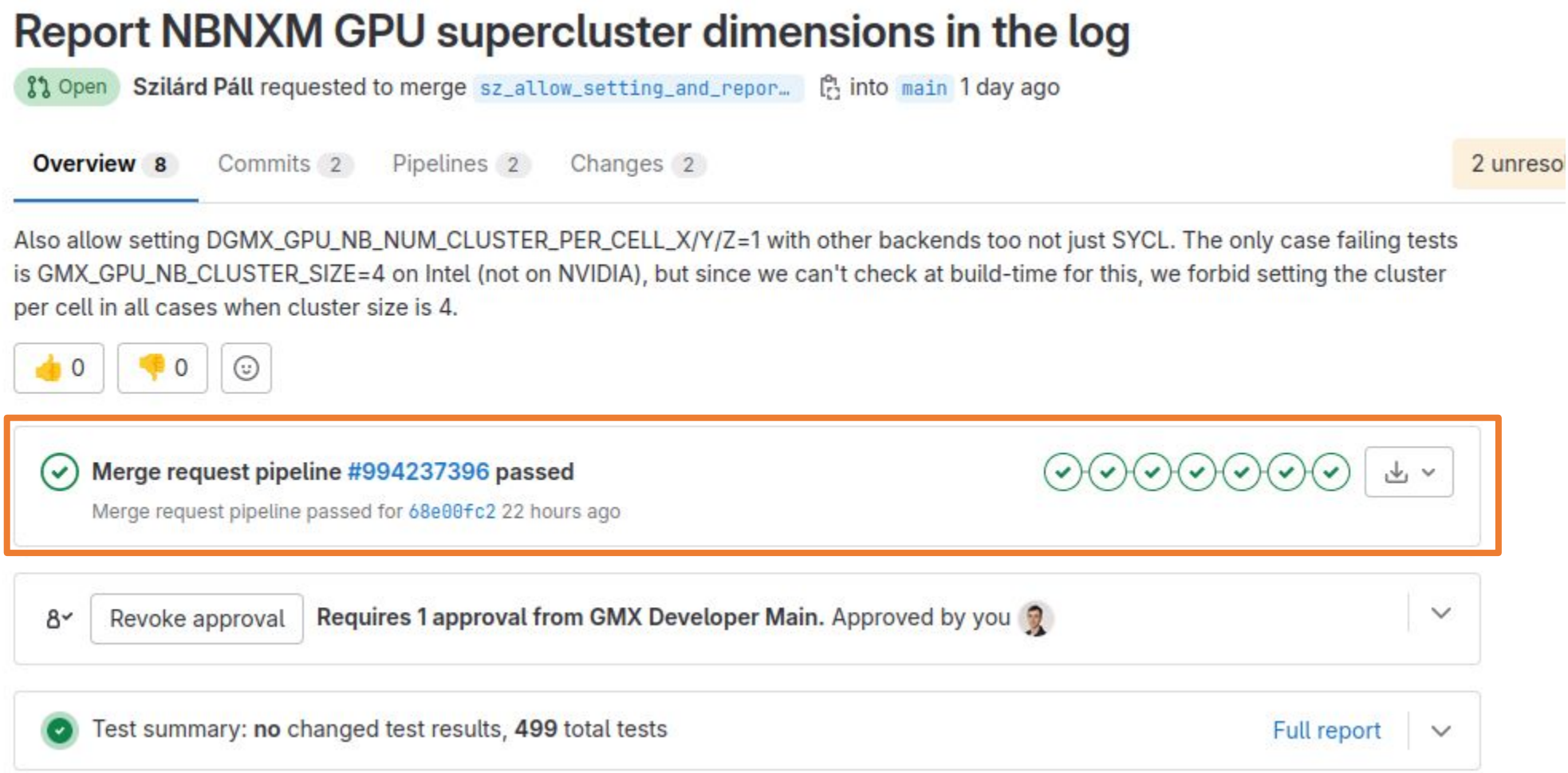
Testing

GROMACS

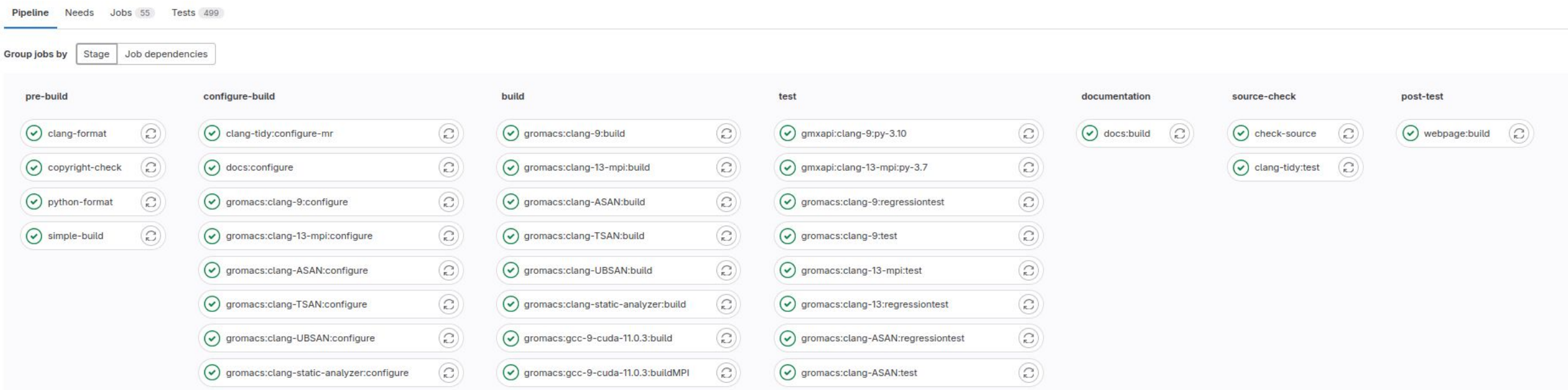
- CI: practice of merging all developers' working copies to a shared mainline several times a day
- CD: software engineering approach in which teams ~~produce software in short cycles~~, ensuring that the software can be reliably released at any time and, following an automated testing pipeline
- Manual testing: not everything can be (easily) automated

GROMACS CI pipeline

GROMACS



GROMACS CI pipeline



Parts of GROMACS automated testing

- Pre-build: code formatting, copyrights, simple build
- Configure-build: CMake in different configurations
- Build: Build the configurations above
- Test: Run the tests for the builds above
- Documentation: Check that documentation is building
- Source-check: Static analysis checks
- Post-test: Manual webpage can be built

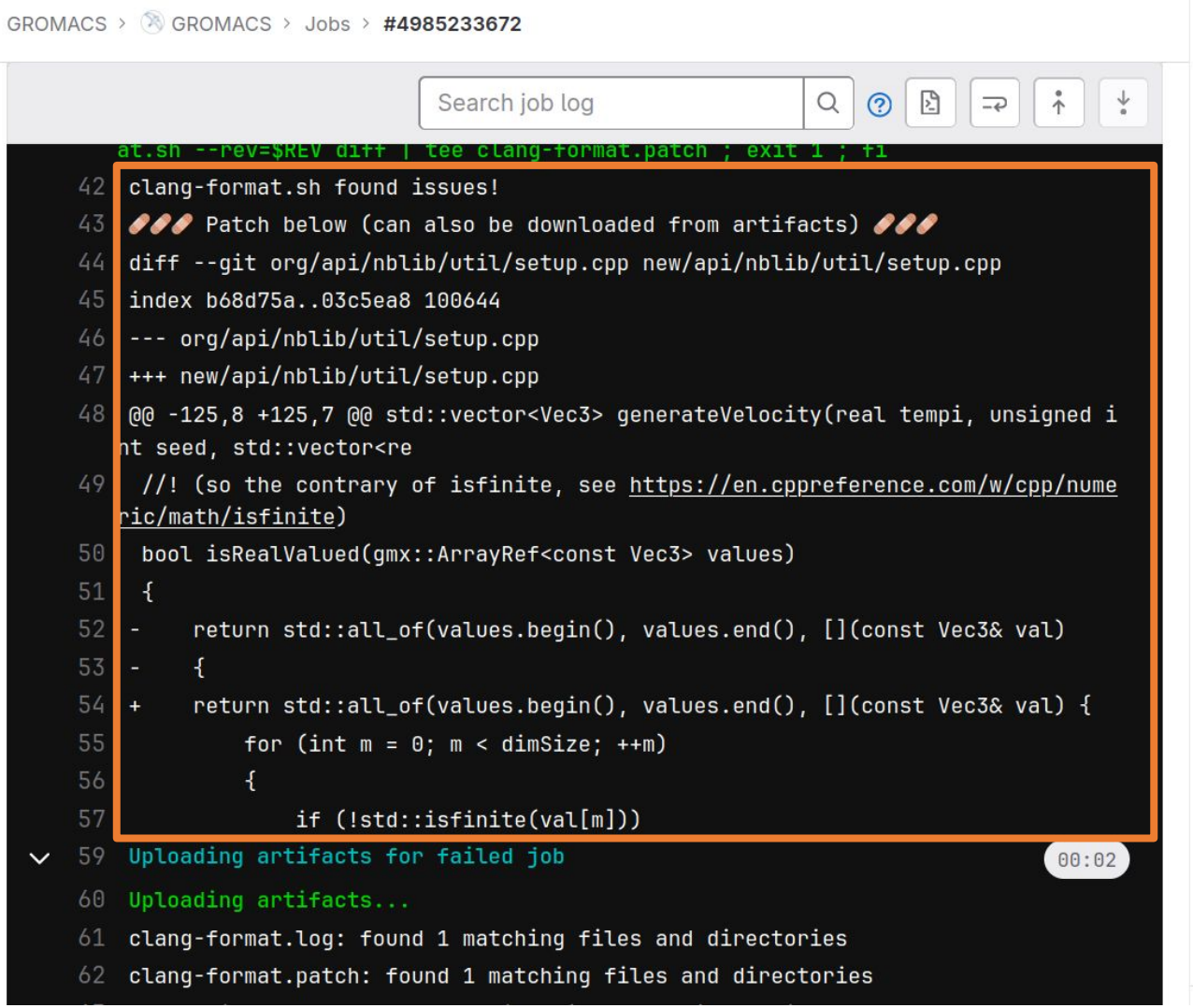
Code style

GROMACS

- Clang-format: C++ code style
  - *Very* sensitive to clang-format version, needs to be 11.0.1
  - `sudo apt install clang-format-11` (Ubuntu 22.04)
  - `pipx install clang-format==11.0.1` (Ubuntu 24.04)
- Python-format: Python code style (`black`)
- Copyright-check: Copyright headers

Code style

GROMACS



clang-format

Duration: 28 seconds

Finished: 1 week ago

Queued: 2 seconds

Timeout: 1h (from project)

Runner: #22213073 (Coj35txY9) gitlab-runner-gitlab-runner-c94b4dbff-cg7lj

Job artifacts

The artifacts will be removed in 3 weeks

Keep | Download | Browse

Commit 171f228c in !3782

Micro-optim isRealValued by using std::all_of + isfinite

Pipeline #987039987 for !3782 with micro_optim

pre-build

Configure + Build + Test

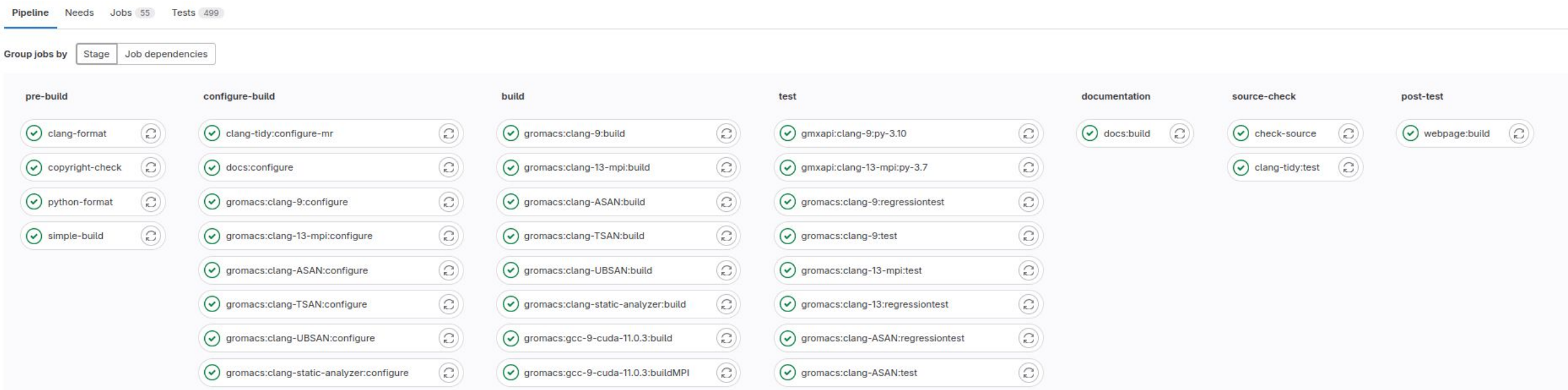
GROMACS

```
cmake .. && make && make check
```

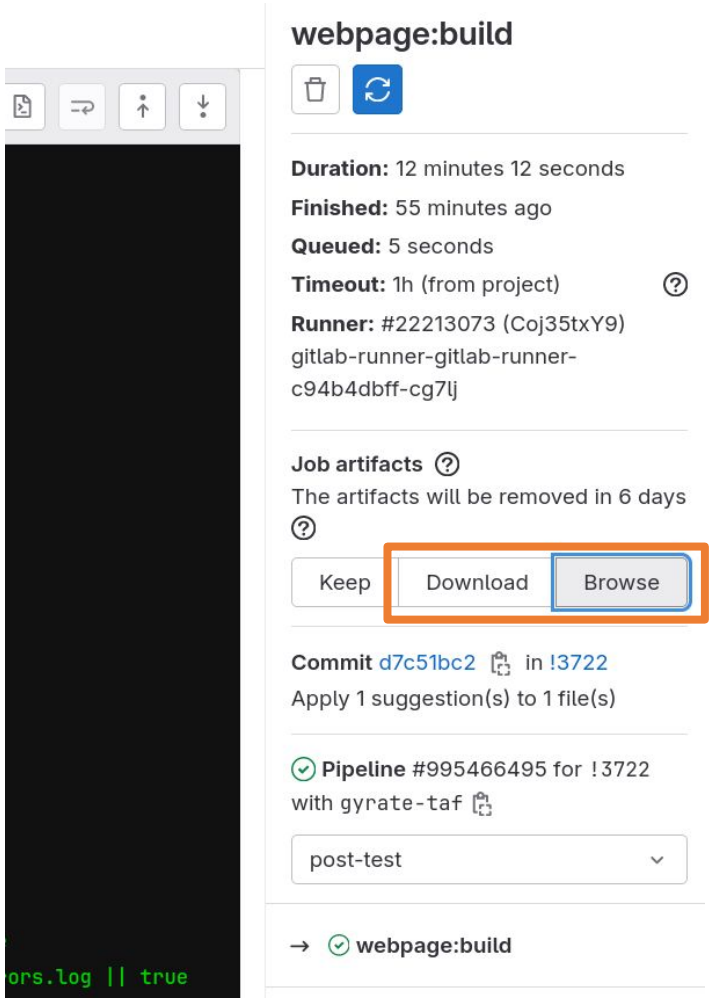
- Compiler: Clang/GCC
- MPI: on/off
- GPU: off/CUDA/OpenCL/SYCL
- Sanitizers: none/TSAN/ASAN/UBSAN

- “-test” and “-regressiontests”: two different test harnesses
- A few `gmxapi`-specific jobs

GROMACS CI pipeline



Documentation + webpage: preview



Artifacts / build-docs / docs / html:

- index.html
- manual-2024-dev.pdf

Source-check

GROMACS

- check-source: basic documentation format checks
- clang-tidy: advanced code static analysis

Source-check

GROMACS

- check-source: basic documentation format checks
- clang-tidy: advanced code static analysis

```
if (i == 1 && i == 2) { // correct?
}
```

Even more automated tests

GROMACS

- Post-merge (multi-GPU tests)
  - Heavy tests not to run on every minor change to the MR
- Nightly (exotic hardware)
  - https://gitlab.com/gromacs/gromacs/-/pipelines
- GitHub actions (Windows, macOS)
  - https://github.com/gromacs/gromacs/actions

Not covered by automated testing

- Performance testing
- Many analysis tools
- Large-scale runs (> 2 GPUs, > 4 ranks)
- Rare devices (POWER9, ARM, high-end GPUs, etc)
- Long-running physical validation tests

Test frameworks

GROMACS

- GoogleTest for most tests
  - Modern and convenient
  - We call them “unit tests”, but that’s not always the case
- Old Perl scripts for regression tests
  - `cmake -DREGRESSIONTEST_DOWNLOAD=ON`
  - Don’t touch it unless you’re changing `mdrun` behavior :)
  - https://gitlab.com/gromacs/gromacs-regressiontests/

Running tests

GROMACS

- All tests:
  - `make tests && make check`
  - `ctest .`
- Specific test set:
  - `ctest -R MdrunIOTests`
  - `./bin/mdrun-io-test`
- Specific test case:
  - `./bin/mdrun-io-test --gtest_filter=GromppTest.*`
  - Some tests might also need to have `-ntomp N` or `-ntmpi N` set
- Run multiple times (flaky test):
  - `ctest --repeat-until-fail 100 --output-on-failure -R MdrunIOTests`

Finding tests

GROMACS

- File: `src/gromacs/utility/logger.cpp`
- Tests: `src/gromacs/utility/tests/logger.cpp`
- Reference data: `src/gromacs/utility/tests/refdata/`
- Also in: `src/programs/*/tests/`

Understanding tests

GROMACS

```
TEST(EnergyTermTest, AddFrameWorks)
{
    EnergyTerm term(0, true, "test", "test");
    term.addFrame(2, 1000, 10, 50, 5, 255);
    term.addFrame(4, 2000, 10, 100, 10, 155);
    EXPECT_EQ(term.numFrames(), 2);
    auto errorEstimate = term.errorEstimate(1);
    ASSERT_TRUE(errorEstimate.has_value());
    EXPECT_REAL_EQ(errorEstimate.value(), 0);
}
```

A lot of boilerplate code omitted

https://google.github.io/googletest/

Parametrized tests

GROMACS

```
TEST_P(HbondModuleTest, Works)
{
    const auto params = GetParam();
    std::string name  = std::get<0>(params);
    int startingValue = std::get<1>(params);
    // Do something with name and startingValue
}

INSTANTIATE_TEST_SUITE_P( //...
     ::testing::Combine(::testing::Values("name1", "name2"),
                        ::testing::Range(4, 8, 1)), // ...
```

A lot of boilerplate code omitted

https://google.github.io/googletest/

Reference data

GROMACS

```
TEST_F(WrapperTest, WrapsCorrectly)
{
    std::vector<std::string> wrapped = doThings();
    checker().checkSequence(wrapped.begin(), wrapped.end(), "Wrapped");
}
```

checker() will compare with data in
src/gromacs/*/tests/**refdata/WrapperTest_WrapsCorrectly**.xml

```
$ ./bin/utility-test -ref-data update-all
```

A lot of boilerplate code omitted

https://google.github.io/googletest/

Input data

GROMACS

If you need some input data (trajectories, etc):

`src/testutils/simulationdatabase/`

```
class TrjconvWithDifferentInputs: public gmx::test::CommandLineTestBase{};

TEST_F(TrjconvWithDifferentInputs, WithIndexGroupSubset) {
    auto& cmdline = commandLine();
    setInputFile("-s", "spc2.gro");
    ASSERT_EQ(0, gmx_trjconv(cmdline.argc(), cmdline.argv()));
}
```

Writing tests

GROMACS

- When fixing a bug: write a test first
  - Helps replicating
  - Helps verifying that your fix works
  - Helps prevent it from happening again

- When adding a new feature: think first about how to test it
  - Test behavior, not implementation

- Some things are very hard to test. Such is life.

Further information

- Testing in GROMACS:
  - https://manual.gromacs.org/current/dev-manual/testutils.html
- More on reference data:
  - https://manual.gromacs.org/current/doxygen/html-lib/page_refdata.xhtml
- GoogleTest Primer:
  - https://google.github.io/googletest/primer.html