Infrastructure 2023

Paul Gessinger
CERN
2023-11-08 - ACTS Workshop Orsay 2023

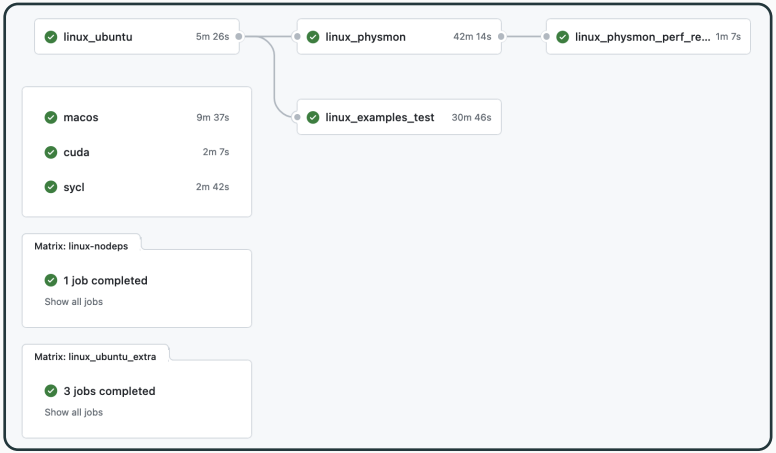
Build & Infrastructure

- Continuous integration
- Developer tools
- Release strategy

Continuous Integration

Continuous Integration jobs overview

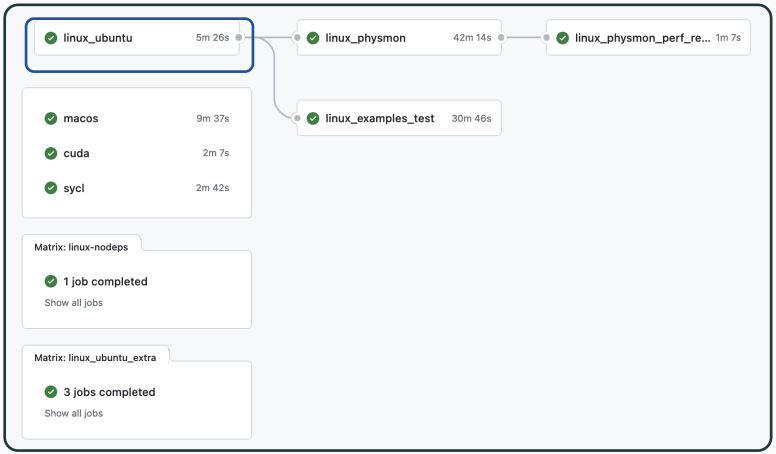
- Continuous integration tests our code automatically
- Is required to pass for PRs to be merged
- Note: **GitHub CI runs on merge result of PR into** `main`
  - That means jobs do not get triggered in case of merge conflicts



CI: Build workflow

linux_ubuntu

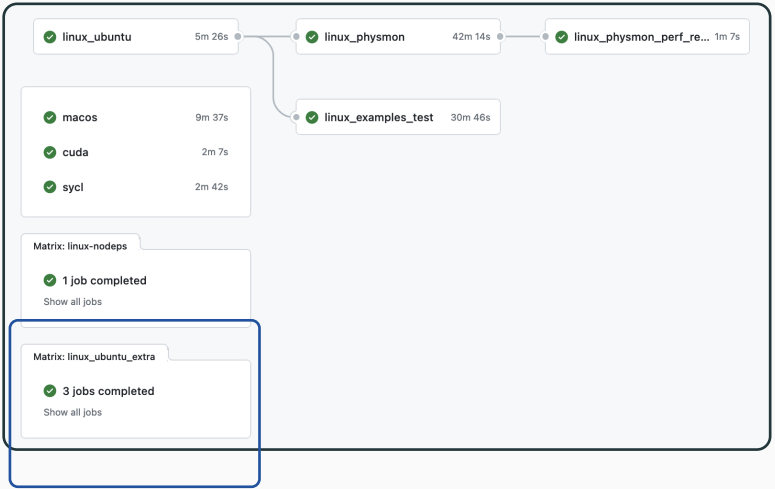
- The main CI build: this job builds everything!
- Runs unit tests, integration tests, downstream project usage
- Uses Ubuntu 22.04 image that bundles all dependencies



CI: Build workflow

linux_ubuntu_extra

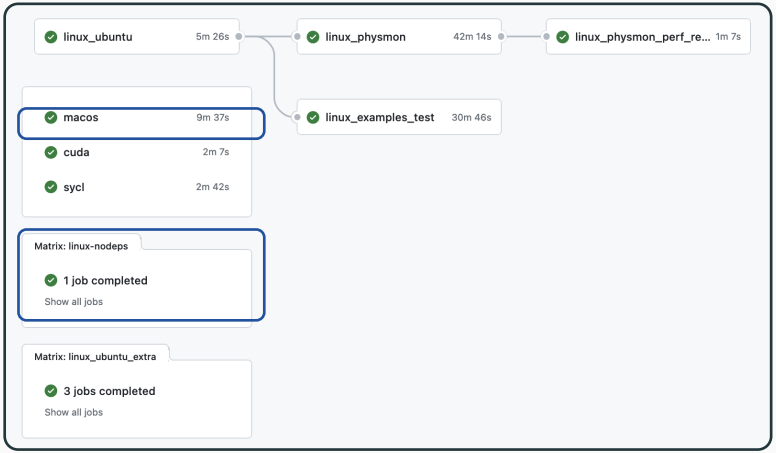
- Build using Ubuntu 22.04 with updated GCC, based on central Ubuntu test
- Ubuntu 22.04 test using clang (+libstdc++) instead of GCC



CI: Build workflow

lcg + macos

- LCG tests removed from GitHub CI
- macOS: All dependencies, tests + downstream test
- nodeps: build core without external dependencies



Unit tests

- Comprehensive unit test suite
- Quality of tests varies:
  - Some tests really test expectations
  - Some tests simply test something runs without crashing
- Tests are run on all CI jobs
- Special job with debug symbols to measure coverage
- Have let this slide a bit: **quick-turnaround unit tests are invaluable for development!**

```
...
290/293 Test #291: ConvertTrackPodio ........ Passed 0.01 sec
  Start 292: PodioTrackStateContainer
291/293 Test #292: PodioTrackStateContainer . Passed 0.02 sec
        Start 293: Alignment
292/293 Test #293: Alignment ................ Passed 0.01 sec
293/293 Test #287: TGeoParser ............... Passed 0.28 sec

100% tests passed, 0 tests failed out of 293

Total Test time (real) =  60.51 sec
```

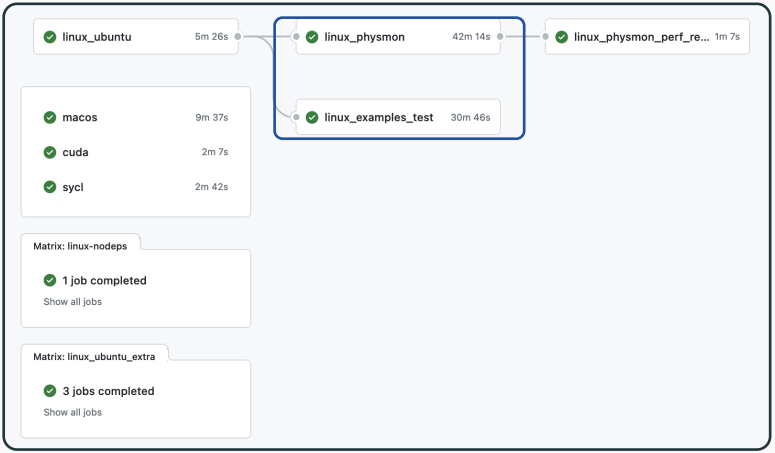
```
@@            Coverage Diff            @@
##             main    #2621    +/-   ##
=========================================
  Coverage   49.59%   49.60%
=========================================
  Files          473      473
  Lines        26824    26828    +4
  Branches     12355    12355
=========================================
+ Hits         13304    13308    +4
  Misses        4753     4753
  Partials      8767     8767
```

CI: Build workflow

Performance monitoring

linux_examples_test

- Uses Python bindings for the examples
- Python-level tests



Python level examples tests

- Comprehensive suite of `pytest` tests
- Directly tests the examples code, but indirectly also the core code
- Implements hash-based ROOT file-level reproducibility checks
  - CI notifies on PRs if outputs change
  - Manual follow up and reference update required
- Allows us to catch basic regressions early
- Ordering independent: reproducibility checks on MT code

CI: Build workflow

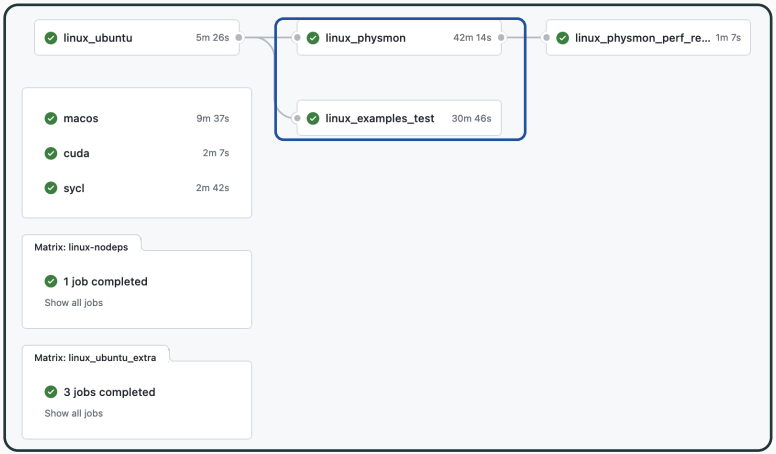
Performance monitoring

linux_examples_test

- Uses Python bindings for the examples
- Python-level tests

linux_physmon

- Runs tracking workflows and records performance metrics
- Compares to references

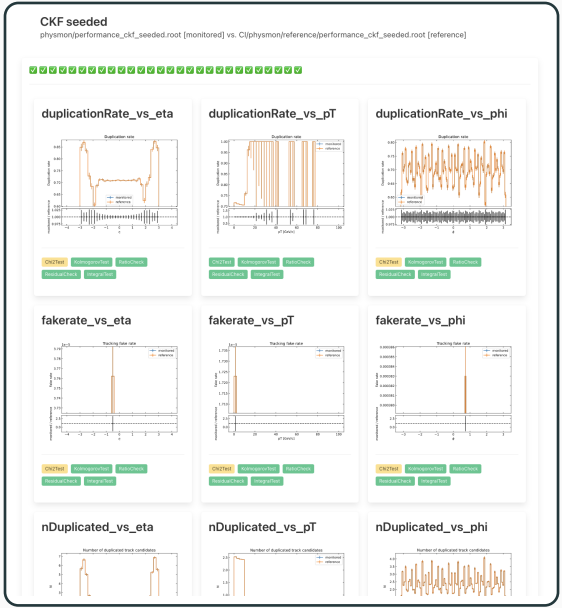


Physics performance monitoring

- Higher level automated performance checks for physics workflows
- Added only 2 years ago: but quickly expanded
- Has become one of the main ways for us to check impact of developments
- **Caveat:** this only runs workflows on OpenDataDectetor, and only in limited combinations (resource constraint)
- We are not sensitive to experiment-specific impacts with this (see next talk by Carlo)

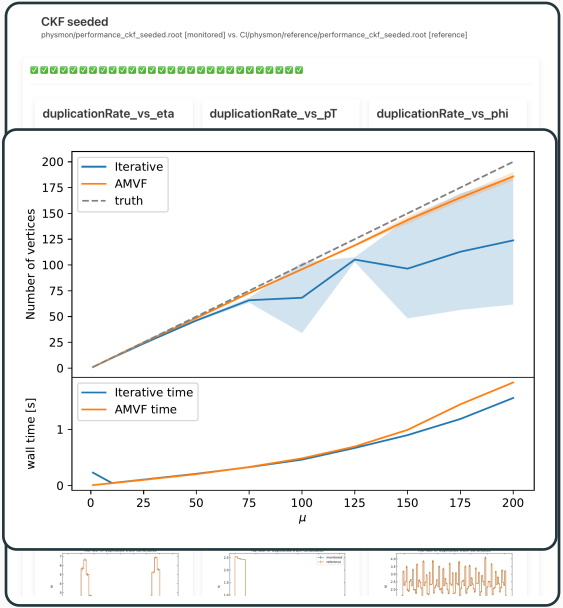
Physics performance monitoring

- Implemented workflows:
  - Truth tracking with KF and GSF
  - CKF + seeding, truth estimation, truth smearing + vertexing
  - CKF on $t\bar{t}$ event at pile-up 200
  - Fatras and G4 simulation
  - Dedicated vertexing workflow
- Runs histogram comparisons using histcmp and produces a neat single-HTML-file report with embedded plots
- Implements basic compatibility checks: KS, χ^2, ratio, residual, integral
  - Looking into replacing this with `dcube`



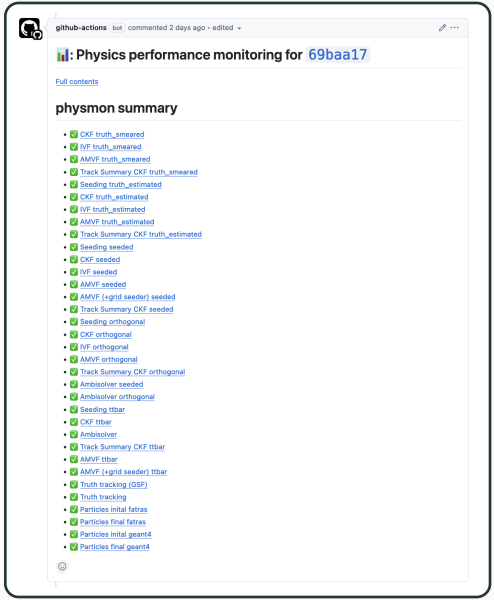
Physics performance monitoring

- Implemented workflows:
  - Truth tracking with KF and GSF
  - CKF + seeding, truth estimation, truth smearing + vertexing
  - CKF on $t\bar{t}$ event at pile-up 200
  - Fatras and G4 simulation
  - Dedicated vertexing workflow
- Runs histogram comparisons using histcmp and produces a neat single-HTML-file report with embedded plots
- Implements basic compatibility checks: KS, χ^2, ratio, residual, integral
  - Looking into replacing this with `dcube`



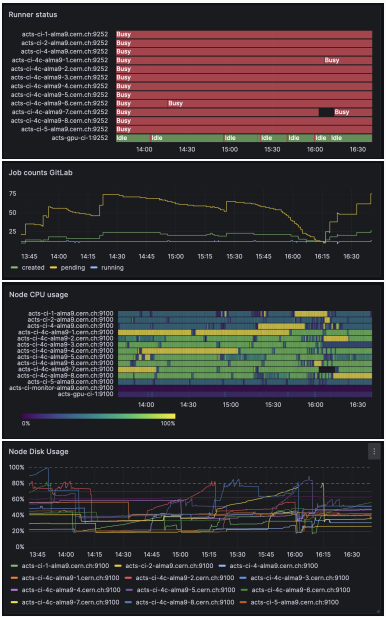
Herald and report comment

- Report job posts a summary of the physmon run to open Pull Requests
  - Contains links to Herald, which allows browsing artifact contents without downloading
- Comment is updated when physmon reruns



Bridged CI jobs

- Have custom setup to run CI job on CERN resources
  - Currently $\mathcal{O}(10)$ slots running in VMs + 1 physical machine with a GPU attached
  - Machines managed using ansible: trivial to add / recreate machines from scratch
  - Monitoring implemented using Prometheus and Grafana
  - Allows diagnosing problems in $\sim$real time
- Access steered by a custom allow-list: if you are a new contributor and are getting ”Pipeline refused” on these jobs, **get in touch!**

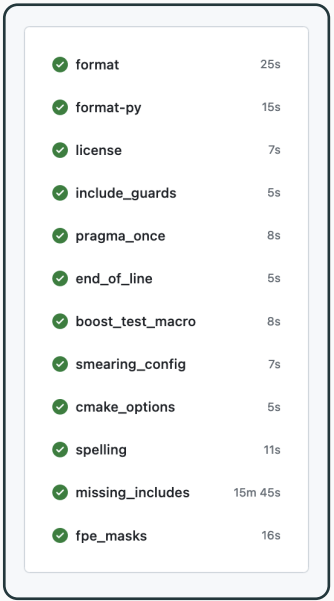


Bridged CI jobs

- LCG test jobs: LCG `102b`, `104` × CentOS7, Alma9 × GCC 11, 12, 13, clang 16 (with some combinations skipped)
- clang-tidy job (static code analysis, details here)
- GPU CI job: build jobs runs on CPU runner, test job runs on GPU runner under my desk
- Ubuntu 20.04/22.04 build on C++17 compiler (to ensure we're compatible) + 1 C++20 build
- Ubuntu 22.04 build on Clang
- Also runs physmon and python tests **with FPE monitoring** (not enough resources on GitHub CI)

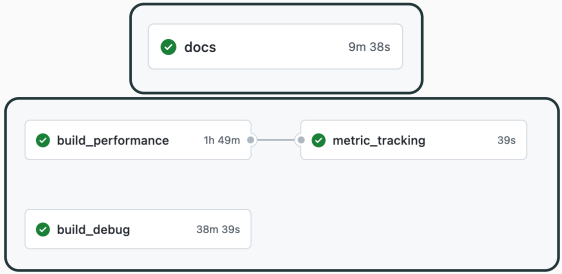
CI: Checks workflow

- Formatting: C++ (clang-format) and python (black)
  - Can run locally, or apply patch from CI artifact. Details here
- License check: validate that all source files start with the correct license block
- Include guards: check for `#ifndef MY_HEADER`: use `#pragma once` instead
- Check consistent enf od lines
- Check against usage of `BOOST_TEST()`
- Check smearing config
- Ensure CMake options are documented
- Spell-check in the code
- Check for missing includes
- Check FPE masks are documented (more on that later)



CI: Code analysis and docs workflow

- Docs job builds the documentation, checks for missing comments etc.
- Build performance runs cmakeperf and measures compilation unit memory consumption
  - Report on the main branch here
- Build debug: similar to `linux_ubuntu`, but without examples
  - Uses debug build to run unit tests and generat code coverage report
  - Coverage report is posted to each PR



Additional bots and workflows

- **merge-sentinel**: manages which jobs / checks are required (custom bot)
- **review-required**: checks approval from one of the reviewers (self-hosted policybot instance)

- **kodiakhq**: enables auto-merging (central instance)
- **WIP**: blocks merging PRs with label ”WIP” (central instance)
  - Discouraged in favor of using GitHub’s draft PR feature
- **codecov**: Coverage reporting (central instance)
- **readthedocs.org**: Preview build of the documentation, deployed to RTD
- **milestone-set**: Blocks merging without a milestone

Developer tools

Developer tools

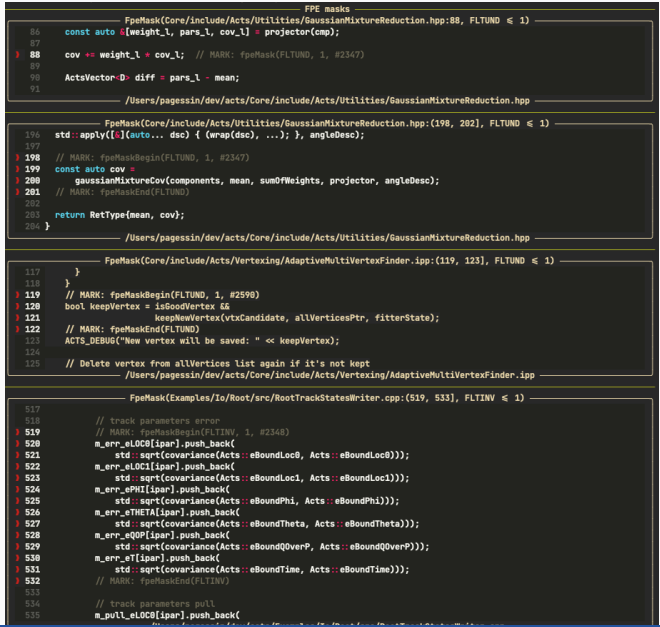
- CMake option: `ACTS_FORCE_ASSERTIONS` force-enables `assert`
  - Allows having checks in ACTS code + 3rd party code even in Release builds
  - Affects performance, be aware when benchmarking
- CMake options `ACTS_LOG_FAILURE_THRESHOLD=<LEVEL>` and `ACTS_ENABLE_LOG_FAILURE_THRESHOLD` steer log level failure threshold (details)
  - If enabled, logging at or above `<LEVEL>` will cause a job to fail
  - Can be set at compile-time or at runtime via env variable or set from Python
  - Allows us to check workflows covered by CI do not produce WARNINGs and ERRORs

FPE monitoring

- FPE monitoring introduced as a plugin
- Uses custom infrastructure to enable FPE trap signal and handle
- Signal handler collects stack trace in async-safe way and records it
  - FPE locations are deduplicated based on the top-most stack frame **source code location!**
- Sequencer can be configured to ignore (**mask**) FPEs
- If configured: Sequencer terminates job when FPE is encountered (and not masked)
- At end of job: FPE are accumulated per algorithm and reported
  - Job is still failed if unmasked FPE are encountered

FPE masking

- Mask is a combination of source file and line range
  - Matching is performed from bottom to top in stack frame: if any frame matches mask, FPE is considered masked
  - Limit number of FPEs per event (but keep in mind that FPE state has to be reset manually)
- Typically want to **fix** the FPE, masking does not mask them outside of Examples
- Statistical process: if you get them depends on the workflow / inputs



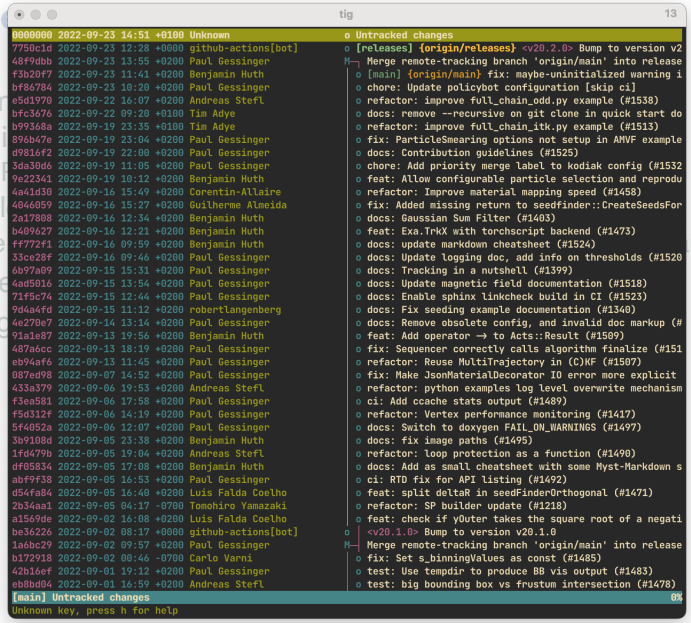
Release strategy

Release strategy

- We follow semantic versioning: vMAJOR.MINOR.PATCH
- We currently limit MAJOR bumps to once a month
  - Target MINOR/PATCH release about once a week
- General development targets the `main` via PRs
  - Requires one approval from one of our reviewers (get in touch if you want to review!)
- Mainline tagged releases go onto the `releases` branch
- Before making a release, `main` gets merged into `releases`

Release strat

- We follow sem
- We currently li
  - Target MINO
- General devel
  - Requires one
- Mainline tagge
- Before making

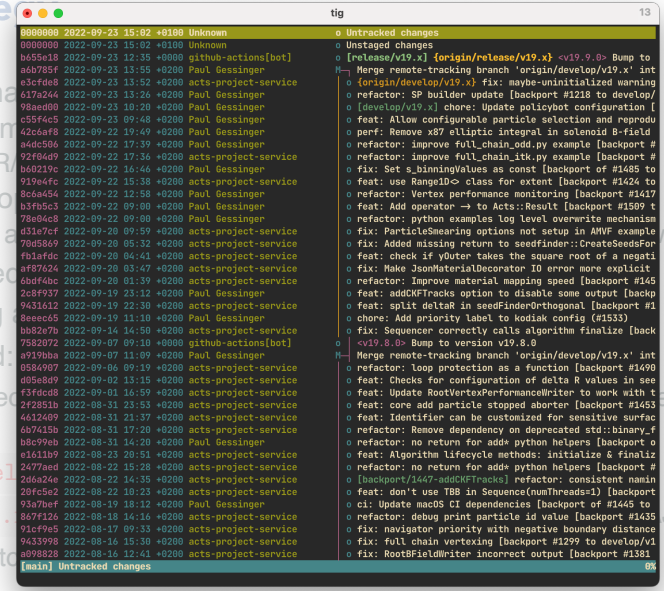


Release strategy

- We follow semantic versioning: vMAJOR.MINOR.PATCH
- We currently limit MAJOR bumps to once a month
  - Target MINOR/PATCH release about once a week
- General development targets the `main` via PRs
  - Requires one approval from one of our reviewers (get in touch if you want to review!)
- Mainline tagged releases go onto the `releases` branch
- Before making a release, `main` gets merged into `releases`
- When required: create `develop/vX.Y.Z` branch as needed
  - Should only receive backports, no approval needed since no new developments
- Example:
  - Created `develop/v19.x` branch after `v20.0.0` was tagged
  - `develop/v19.x` can receive MINOR and PATCH updates, but no MAJOR updates
  - Additional matching `release/vX.Y.Z` created which receives tags

Release strate

- We follow sema
- We currently lin
  - Target MINOR
- General develo
  - Requires one a
- Mainline tagged
- Before making
- When required:
  - Should only re
- Example:
  - Created `devel`
  - `develop/v19.`
  - Additional mat



Questions?

Backup

License block

```
// This file is part of the Acts project.
//
// Copyright (C) 2016-2020 CERN for the benefit of the Acts project
//
// This Source Code Form is subject to the terms of the Mozilla Public
// License, v. 2.0. If a copy of the MPL was not distributed with this
// file, You can obtain one at http://mozilla.org/MPL/2.0/.
```