Use of C++ code formatting/linting tools

CMS Offline Software

Malik Shahzad MUZAFFAR
ROOT Team Meeting 26/07/2021

Intro

- clang-format/tidy
  - What are these tools
  - How to use
  - Examples
- Code checks for CMSSW Offline Software
  - CMSSW Offline Software
  - CMSSW CI system and code formatting/linting tools
- clang-format/tidy for ROOT project

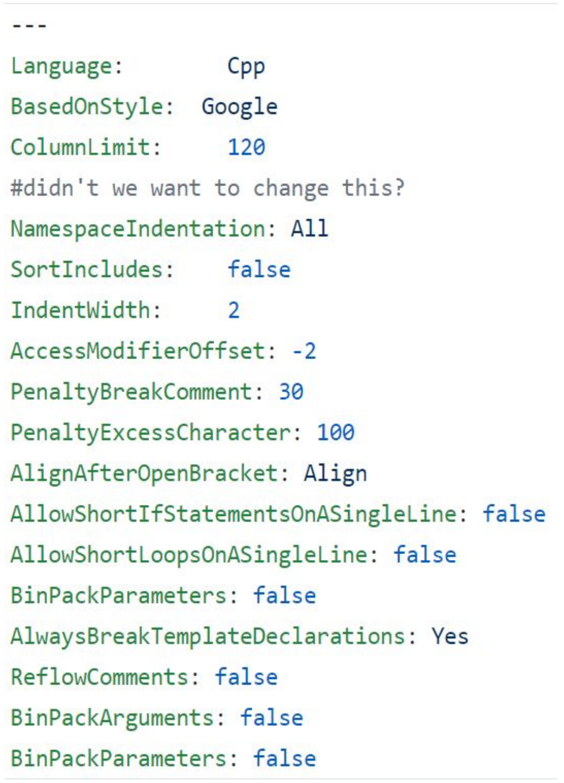
clang-format: What is it?

- A tool to automatically format C/C++ code but supports other languages (JavaScript, Proto, C-Sharpe)
  - Many predefined styles
    - LLVM, Google, Chromium, Mozilla, WebKit, Microsoft
    - Custom styles by configuring specific styles options
- Allows developers and code reviewers to spend less time on formatting and reviewing code style issue

Clang-format: How to use

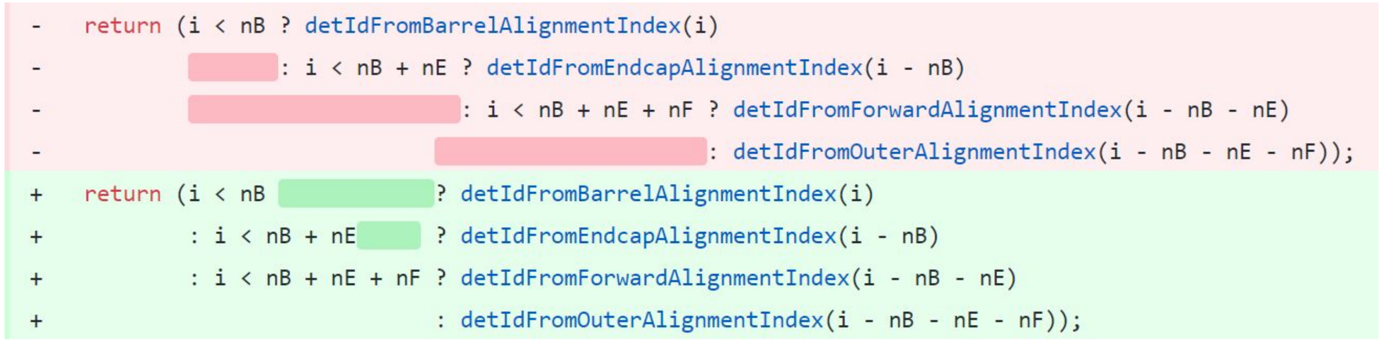
- Very simple to use, you just needed **clang-format**
  - Code style can be selected via **-style='{...}'** or **-style=file** command-line option
    - **.clang-format** yaml format file can be used to override specific style options
      - clang-format uses **.clang-format** from the closest parent directory
- **clang-format-diff.py** can be used to format only the changes (e.g git diff | clang-format-diff.py)
- Very fast and can run in parallel
  - Can do inplace edit (**-i** command line option)

cmssw/.clang-format

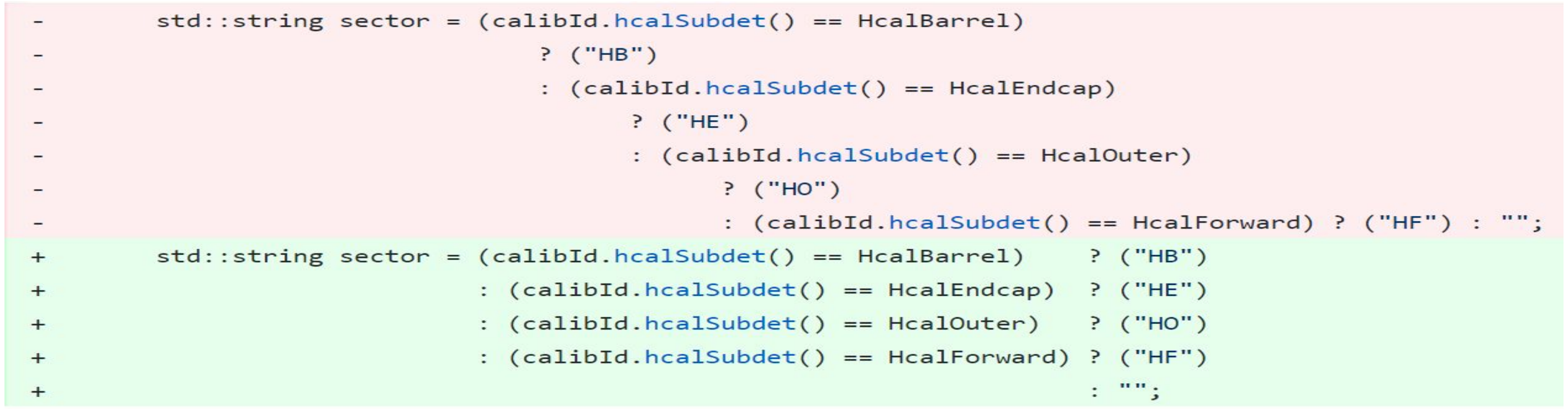


```
---
Language:        Cpp
BasedOnStyle:  Google
ColumnLimit:     120
#didn't we want to change this?
NamespaceIndentation: All
SortIncludes:     false
IndentWidth:     2
AccessModifierOffset: -2
PenaltyBreakComment: 30
PenaltyExcessCharacter: 100
AlignAfterOpenBracket: Align
AllowShortIfStatementsOnASingleLine: false
AllowShortLoopsOnASingleLine: false
BinPackParameters: false
AlwaysBreakTemplateDeclarations: Yes
ReflowComments: false
BinPackArguments: false
BinPackParameters: false
```

clang-format: Examples



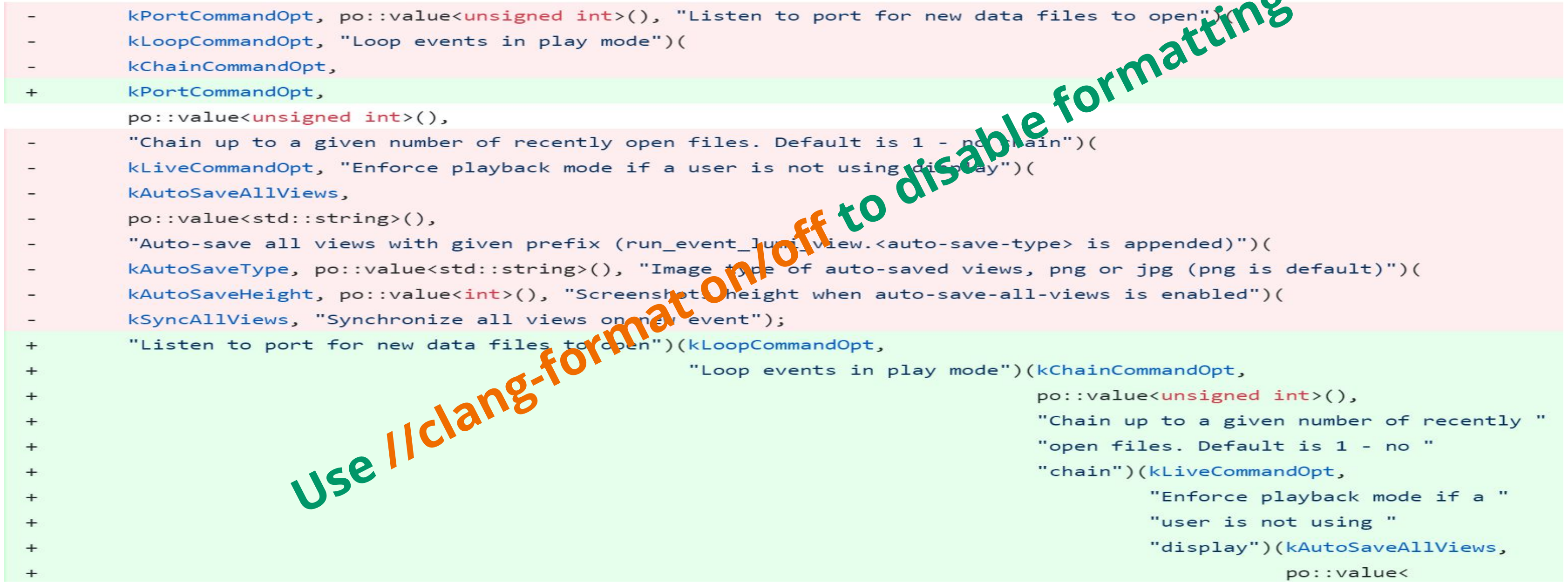
```diff
-   return (i < nB ? detIdFromBarrelAlignmentIndex(i)
-                  : i < nB + nE ? detIdFromEndcapAlignmentIndex(i - nB)
-                                : i < nB + nE + nF ? detIdFromForwardAlignmentIndex(i - nB - nE)
-                                                   : detIdFromOuterAlignmentIndex(i - nB - nE - nF));
+   return (i < nB             ? detIdFromBarrelAlignmentIndex(i)
+           : i < nB + nE      ? detIdFromEndcapAlignmentIndex(i - nB)
+           : i < nB + nE + nF ? detIdFromForwardAlignmentIndex(i - nB - nE)
+                              : detIdFromOuterAlignmentIndex(i - nB - nE - nF));
```



```diff
-       std::string sector = (calibId.hcalSubdet() == HcalBarrel)
-                                ? ("HB")
-                                : (calibId.hcalSubdet() == HcalEndcap)
-                                      ? ("HE")
-                                      : (calibId.hcalSubdet() == HcalOuter)
-                                            ? ("HO")
-                                            : (calibId.hcalSubdet() == HcalForward) ? ("HF") : "";
+       std::string sector = (calibId.hcalSubdet() == HcalBarrel)    ? ("HB")
+                            : (calibId.hcalSubdet() == HcalEndcap)  ? ("HE")
+                            : (calibId.hcalSubdet() == HcalOuter)   ? ("HO")
+                            : (calibId.hcalSubdet() == HcalForward) ? ("HF")
+                                                                    : "";
```

Clang-format: Example ...

https://github.com/cms-sw/cmssw/pull/34054/commits/aa2e97946048fbb1529eea04a25086639cf52af4

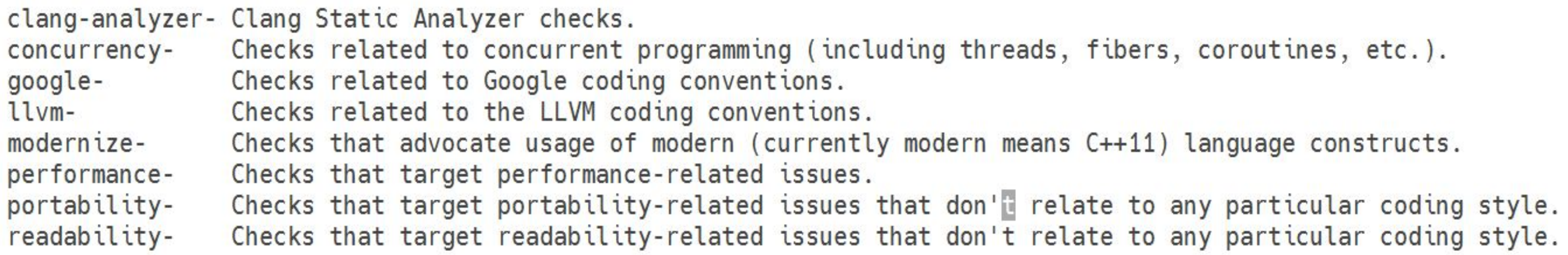


```
-        kPortCommandOpt, po::value<unsigned int>(), "Listen to port for new data files to open")(
-        kLoopCommandOpt, "Loop events in play mode")(
-        kChainCommandOpt,
+        kPortCommandOpt,
         po::value<unsigned int>(),
-        "Chain up to a given number of recently open files. Default is 1 - no chain")(
-        kLiveCommandOpt, "Enforce playback mode if a user is not using display")(
-        kAutoSaveAllViews,
-        po::value<std::string>(),
-        "Auto-save all views with given prefix (run_event_lumi_view.<auto-save-type> is appended)")(
-        kAutoSaveType, po::value<std::string>(), "Image type of auto-saved views, png or jpg (png is default)")(
-        kAutoSaveHeight, po::value<int>(), "Screenshot height when auto-save-all-views is enabled")(
-        kSyncAllViews, "Synchronize all views on new event");
+        "Listen to port for new data files to open")(kLoopCommandOpt,
+                                                     "Loop events in play mode")(kChainCommandOpt,
+                                                                                 po::value<unsigned int>(),
+                                                                                 "Chain up to a given number of recently "
+                                                                                 "open files. Default is 1 - no "
+                                                                                 "chain")(kLiveCommandOpt,
+                                                                                          "Enforce playback mode if a "
+                                                                                          "user is not using "
+                                                                                          "display")(kAutoSaveAllViews,
+                                                                                                     po::value<
```

Use //clang-format on/off to disable formatting

clang-tidy: What is it?

- A clang based C++ "Linter" that can identify
  - style violations
  - interfaces misuse
  - bugs that can be deduced via static analysis
- Contains a lot of checks and can also run clang static analyzer checks
  - Also provides easy interface for writing new checks



```
clang-analyzer- Clang Static Analyzer checks.
concurrency-    Checks related to concurrent programming (including threads, fibers, coroutines, etc.).
google-         Checks related to Google coding conventions.
llvm-           Checks related to the LLVM coding conventions.
modernize-      Checks that advocate usage of modern (currently modern means C++11) language constructs.
performance-    Checks that target performance-related issues.
portability-    Checks that target portability-related issues that don't relate to any particular coding style.
readability-    Checks that target readability-related issues that don't relate to any particular coding style.
```

Clang-tidy: How to use

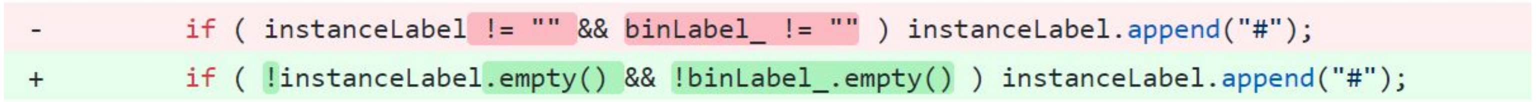
- For single file
  - clang-tidy file.cpp -checks=-*,clang-analyzer-* -- -Imy_project/include -DMY_DEFINES
- For large project, it is better to use a “compile commands database”
  - cmake version 3.5 and above already support it
    - use **-DCMAKE_EXPORT_COMPILE_COMMANDS=ON** to generate **compile_commands.json**
- Checks can be enabled either via command line “-checks=-*,clang-analyzer-*” or use **.clang-tidy** file
  - **clang-tidy** searches the closest parent directory for this file
- **clang-tidy-diff.py** can be used to apply checks on changed code only

clang-tidy: How to use ..

- ❖ Code should not have any compilation errors
- ❖ It can apply the fixes inplace (**-fix** command-line option)
- ❖ It can run in parallel but be careful with fixes for header files
  - ➢ multiple source files include same header
  - ➢ multiple clang-tidy processes can apply the same fixes resulted in invalid code
  - ➢ run-clang-tidy.py can be used to avoid this
    - ■ First runs clang-tidy to export the fixes in to yaml files
    - ■ Apply the suggested fixes
- ❖ Approx. takes same time as compilation

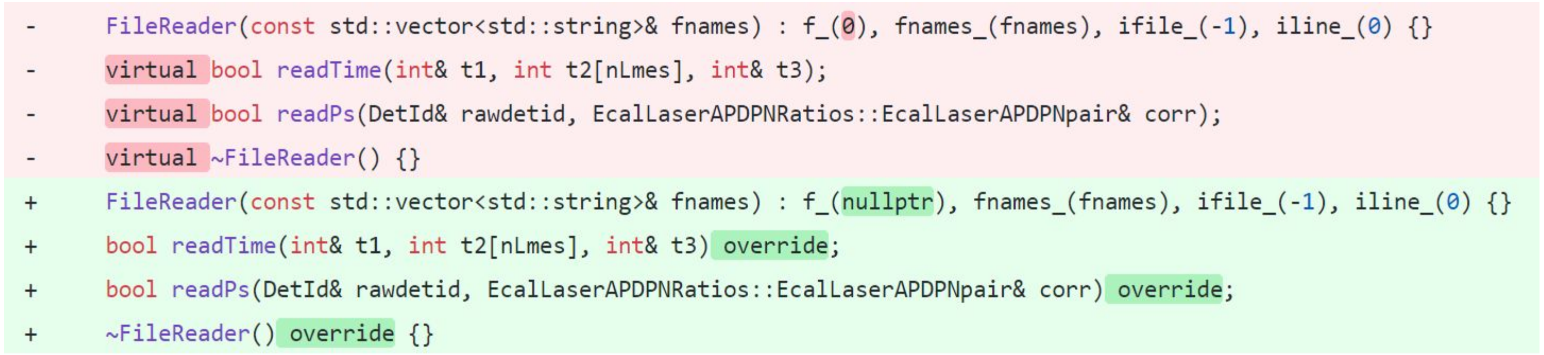
clang-tidy: Examples

readability-container-size-empty



```
-          if ( instanceLabel != "" && binLabel_ != "" ) instanceLabel.append("#");
+          if ( !instanceLabel.empty() && !binLabel_.empty() ) instanceLabel.append("#");
```

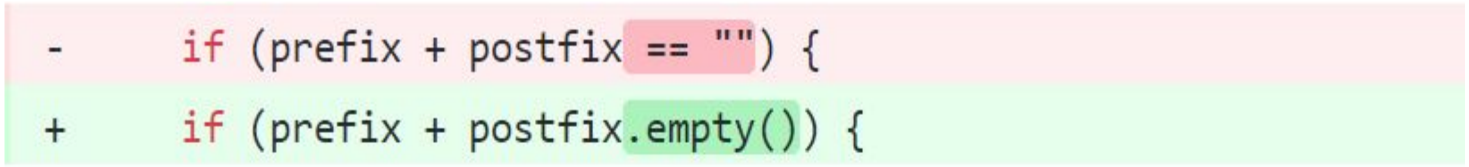
modernize-use-nullptr / modernize-use-override



```
-     FileReader(const std::vector<std::string>& fnames) : f_(0), fnames_(fnames), ifile_(-1), iline_(0) {}
-     virtual bool readTime(int& t1, int t2[nLmes], int& t3);
-     virtual bool readPs(DetId& rawdetid, EcalLaserAPDPNRatios::EcalLaserAPDPNpair& corr);
-     virtual ~FileReader() {}
+     FileReader(const std::vector<std::string>& fnames) : f_(nullptr), fnames_(fnames), ifile_(-1), iline_(0) {}
+     bool readTime(int& t1, int t2[nLmes], int& t3) override;
+     bool readPs(DetId& rawdetid, EcalLaserAPDPNRatios::EcalLaserAPDPNpair& corr) override;
+     ~FileReader() override {}
```

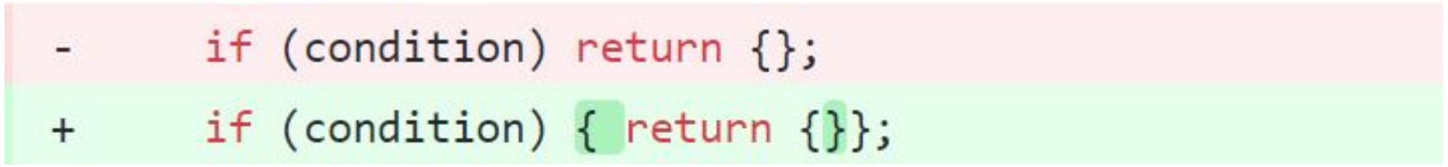
clang-tidy: examples

- ❖ Some checks can generate invalid code e.g.
  - ➢ **readability-container-size-empty**



```diff
-       if (prefix + postfix == "") {
+       if (prefix + postfix.empty()) {
```

  - ➢ **google-readability-braces-around-statements**



```diff
-        if (condition) return {};
+        if (condition) { return {}};
```

Use //NOLINT or //NOLINTNEXTLINE to ignore clang-tidy check

clang-format/clang-tidy
CMSSW

CMS Offline Software (CMSSW)

- CMSSW has large code base
  - 3.4M C++/C, 1.3M Python, 275K Fortran lines of code
    - 30K C++/C files
      - 16K source files
    - 2400+ shared libs/plugins , 850+ executables/tests
    - 450+ externals packages deps (Including 260+ python packages)
- Over 20 years of SW development and still very active
  - For last 2 years
    - 75+ unique contributors/month contributing to CMSSW code base
    - 750+ commits/month
- 12 Release cycles (5.3, 7.1, ... 12.0)

CMSSW: Code reviews

Pull Requests/week	# of PRs	# of lines +/-	# files
Created	75	975K/225K	3.5K
Merged	60	75K/53K	690

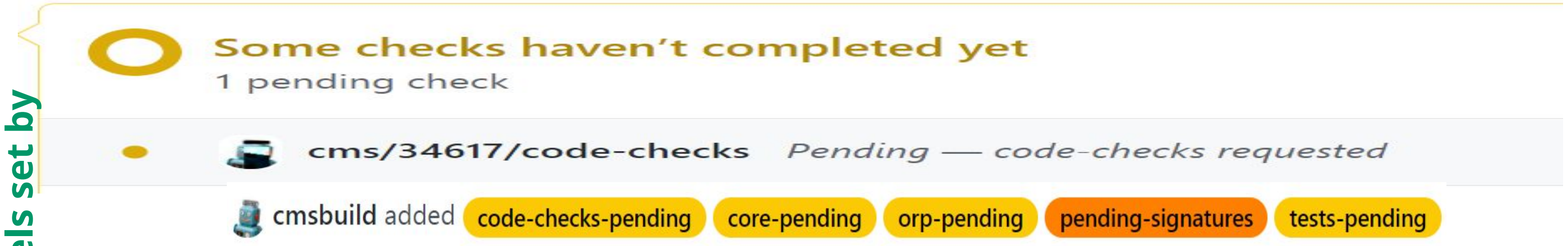
- CMSSW code reviewers review large number of PRs every day
  - Some automated checks are needed to filter-out the bad PRs
    - Majority of changes are bogus and code reviewers should not waste time on those
      - Mostly due to PR open for a wrong git branch
    - Code which does not compile
- Ease the code reviewer life by automatically enforcing code rules and styles

CMSSW code checks: clang-format/tidy

- Since 2017, CMS Offline software CI-bot has been using clang-format/tidy to enforce the CMS coding and style rules
  - Really helped us improving code quality and integration process
  - Saved a lot of code review time
- CMS CI-bot automatically runs code-checks for all pull requests opened/updated for development release cycle
  - Passing code-checks is prerequisite for code review
    - failing this check will not allow to start the build/tests process

CMSSW code-checks: CI-bot

Commit statuses/labels set by CI-bot

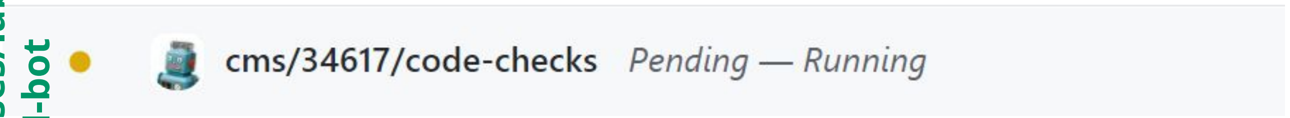


Some checks haven't completed yet

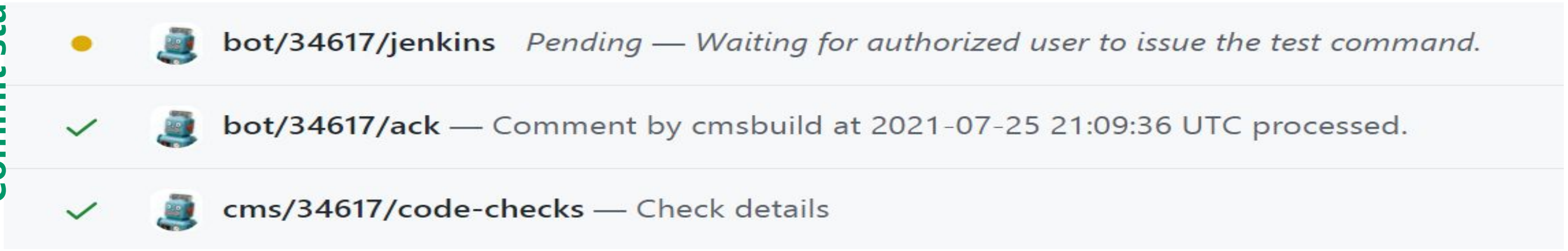
1 pending check

● cms/34617/code-checks *Pending — code-checks requested*

cmsbuild added code-checks-pending core-pending orp-pending pending-signatures tests-pending



● cms/34617/code-checks *Pending — Running*

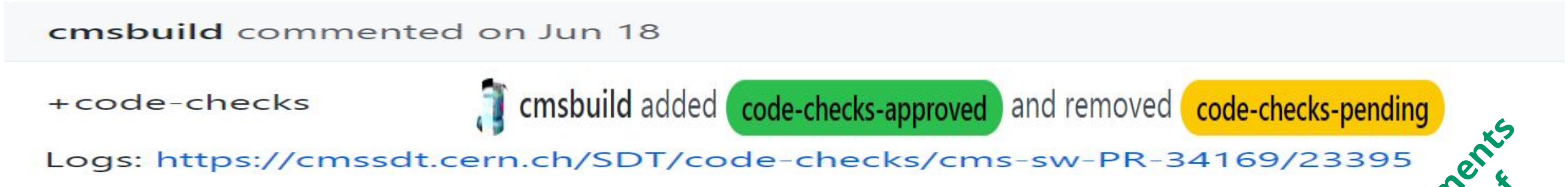


● **bot/34617/jenkins** *Pending — Waiting for authorized user to issue the test command.*

✓ **bot/34617/ack** — Comment by cmsbuild at 2021-07-25 21:09:36 UTC processed.

✓ **cms/34617/code-checks** — Check details

CMSSW Code-checks: CI-bot results

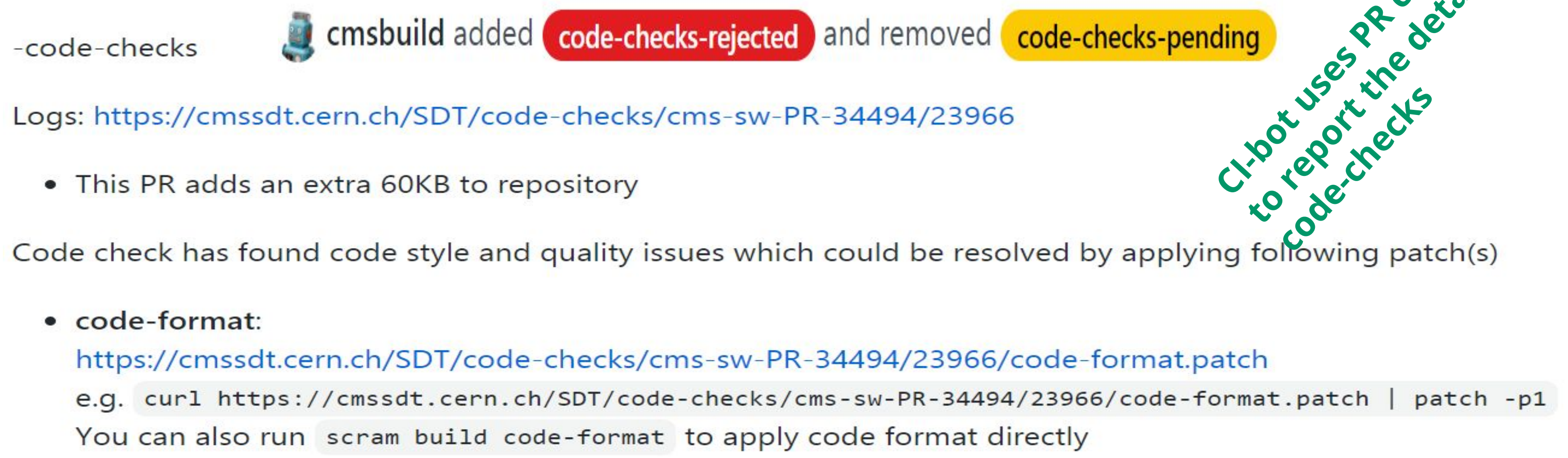


cmsbuild commented on Jun 18

+code-checks

cmsbuild added code-checks-approved and removed code-checks-pending

Logs: https://cmssdt.cern.ch/SDT/code-checks/cms-sw-PR-34169/23395



-code-checks

cmsbuild added code-checks-rejected and removed code-checks-pending

Logs: https://cmssdt.cern.ch/SDT/code-checks/cms-sw-PR-34494/23966

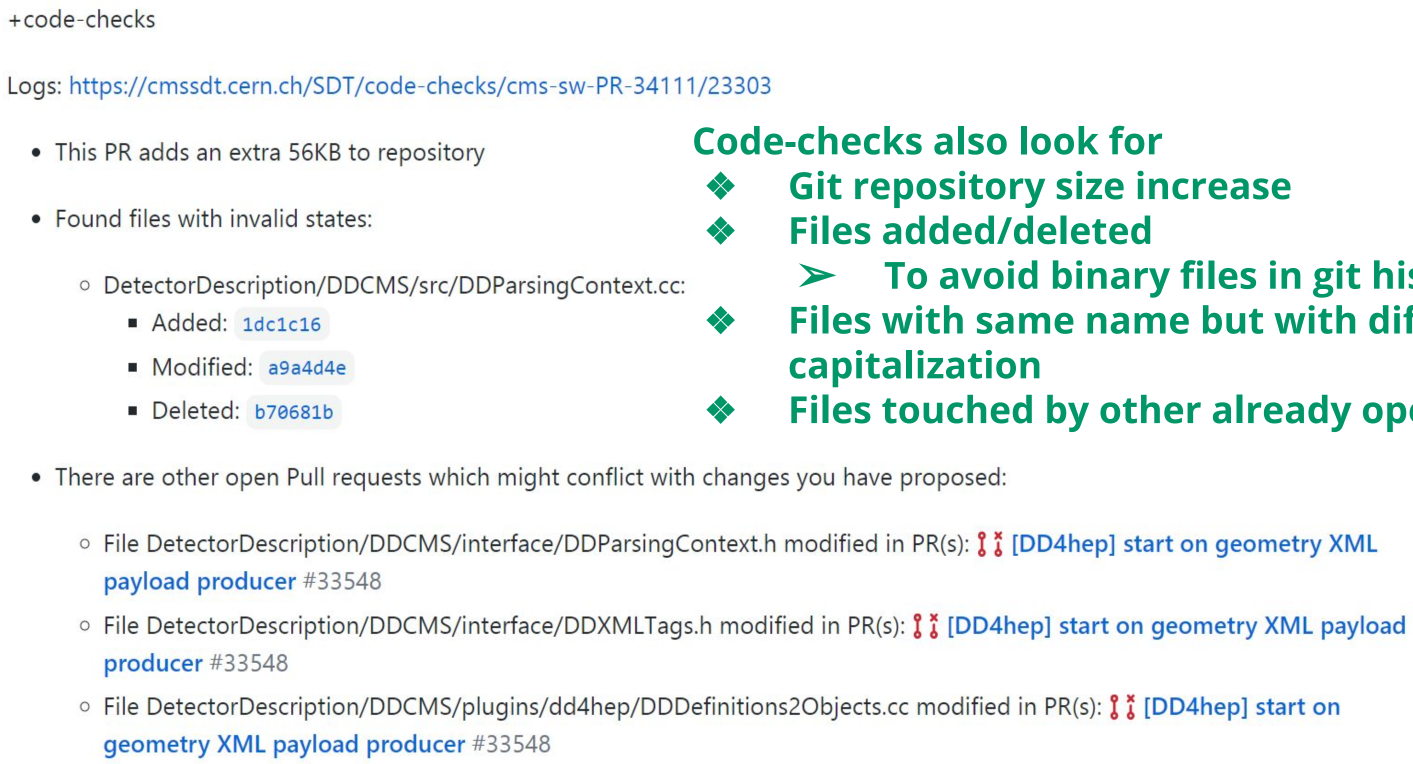
- This PR adds an extra 60KB to repository

Code check has found code style and quality issues which could be resolved by applying following patch(s)

- **code-format**:
  https://cmssdt.cern.ch/SDT/code-checks/cms-sw-PR-34494/23966/code-format.patch
  e.g. `curl https://cmssdt.cern.ch/SDT/code-checks/cms-sw-PR-34494/23966/code-format.patch | patch -p1`
  You can also run `scram build code-format` to apply code format directly

CI-bot uses PR comments to report the details of code-checks

CMSSW Code-checks: CI-bot results



+code-checks

Logs: https://cmssdt.cern.ch/SDT/code-checks/cms-sw-PR-34111/23303

- This PR adds an extra 56KB to repository
- Found files with invalid states:
  - DetectorDescription/DDCMS/src/DDParsingContext.cc:
    - Added: `1dc1c16`
    - Modified: `a9a4d4e`
    - Deleted: `b70681b`
- There are other open Pull requests which might conflict with changes you have proposed:
  - File DetectorDescription/DDCMS/interface/DDParsingContext.h modified in PR(s): [DD4hep] start on geometry XML payload producer #33548
  - File DetectorDescription/DDCMS/interface/DDXMLTags.h modified in PR(s): [DD4hep] start on geometry XML payload producer #33548
  - File DetectorDescription/DDCMS/plugins/dd4hep/DDDefinitions2Objects.cc modified in PR(s): [DD4hep] start on geometry XML payload producer #33548

Code-checks also look for

- ❖ **Git repository size increase**
- ❖ **Files added/deleted**
  - ➢ **To avoid binary files in git history**
- ❖ **Files with same name but with different capitalization**
- ❖ **Files touched by other already opened PRs**

CMSSW Code-checks: What we have done

- Selected the clang-tidy checks and clang-format style to enable
  - Started a campaign to run clang-tidy and format for full CMSSW
    - Done via an automated Jenkins job
    - PRs with max 200 files/PR
      - Separate commits for clang-tidy and format fixes
    - Skipped files touched by already opened PRs
      - To avoid possible merge conflicts
- Enabled CI code-checks for all newly opened or updated PRs
  - CI code-checks runs only on files touched by PR
  - clang-tidy/format runs for full file contents instead of changes only
  - Clang-tidy does not run for newly added headers which are not included in any source file

CMSSW Build Rules: PR code check

- CMSSW uses SCRAM as a build system (MAKE based rules)
  - To run clang-tidy: **scram build -j $(nproc) code-checks**
    - Generates compile commands DB
    - Run clang-tidy for files touched by PR and export the fixes
    - Process the exported fixes and remove changes for files not touched by PR
      - Changes for included headers
    - Apply the fixes
  - To run clang-format: **scram build -j $(nproc) code-format**
  - To run on all checked out files: **code-[checks|format]-all**

ROOT: clang-tidy/format integration

- As ROOT uses CMAKE so it should not be hard to integrate these tool
  - It might take more time to setup CI to run these tool properly
- clang-format is straight forward. Create valid **.clang-format**
  - You can run **clang-format** directly on your source files
  - CMAKE rules can help running in it parallel
- clang-tidy should also be easy enough to setup. All you need is to
  - Create a valid **.clang-tidy** file
  - generate compile commands DB, process it and remove any files for which you do not want to run clang-tidy and run **run-clang-tidy.py**

```
cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=ON ..
#cleanup compile_commands.json if needed
run-clang-tidy.py -header-filter='.*' -fix
```

Things to remember for automatic CI jobs

- User can execute arbitrary code if build rules and code exist in same repository
  - Using execute_process()/command() CMAKE commands
  - In CMSSW we do not have this issue as BuildRules are in different repository.
- clang-format can run for all PR as long one does not use the build system
- clang-tidy can also run automatically with some workarounds
  - Use **compile_commands.json** from release area
  - For newly added sources, use the compile command of other files in same directory
- Do not run clang-tidy on full code base
  - Only run it on changed files
  - Revert clang-tidy fixes for header files which are not touched by PR