



Coverity 2020.12 Command Reference

Reference for Coverity Analysis, Coverity Platform, and Coverity Desktop.

Copyright 2020 Synopsys, Inc. All rights reserved worldwide.

Table of Contents

1. Coverity Analysis Commands	1
cov-analyze	2
cov-blame	53
cov-build	58
cov-capture	89
cov-collect-models	97
cov-commit-defects	99
cov-configure	113
cov-copy-overrun-triage (Deprecated)	132
cov-count-lines	134
cov-emit	137
cov-emit-cs	151
cov-emit-go	155
cov-emit-java	156
cov-emit-swift	169
cov-emit-vb	171
cov-export-cva	176
cov-extract-scm	178
cov-find-function	185
cov-format-errors	188
cov-generate-hostid	193
cov-help	194
cov-import-msvsca	195
cov-import-results	199
cov-import-scm	202
cov-install-updates	206
cov-link	213
cov-make-library	217
cov-manage-emit	223
cov-preprocess	265
cov-record-source	268
cov-run-desktop	270
cov-run-fortran	296
cov-security-da	312
cov-test-configuration	314
cov-translate	318
cov-upgrade-static-analysis	327
cov-wizard	329
2. Coverity Analysis Ant Tasks	330
covanalyzeandcommit	331
covbuild	338
3. Test Advisor Commands	340
cov-emit-server	341
cov-emit-server-control	343
cov-manage-history	345
cov-patch-bulleye	349

4. Dynamic Analysis Commands	350
cov-start-da-broker	351
cov-stop-da-broker	354
5. Dynamic Analysis Ant Tasks	356
cov-dynamic-analyze-java	357
cov-dynamic-analyze-junit	360
cov-start-da-broker	362
cov-stop-da-broker	365
6. Coverity Connect Commands	367
cov-admin-db	368
cov-archive	376
cov-get-certs	380
cov-im-ctl	381
cov-import-cert	382
cov-manage-im	383
cov-start-im	414
cov-stop-im	415
cov-support	416
7. CVSS Report	418
cov-generate-cvss-report	419
8. Coverity Integrity Report	422
cov-generate-integrity-report	423
9. MISRA Report	425
cov-misra-report	426
cov-generate-misra-report	427
10. OWASP Web Top 10	429
cov-generate-owasp2017-report	430
11. Mobile OWASP Top 10	431
cov-generate-mobile-owasp-report	432
12. PCI DSS	433
cov-generate-pci-dss-report	434
13. Security Report	435
cov-security-report	436
cov-generate-security-report	437
14. Coverity CERT Report	439
cov-cert-report	440
cov-generate-cert-report	441
A. Accepted date/time formats	443
B. Coverity Glossary	444
C. Coverity Legal Notice	456
C.1. Legal Notice	456

Coverity Analysis Commands

Name

cov-analyze Analyze an intermediate directory for quality and security defects.

Synopsis

```
cov-analyze --dir <intermediate_directory> [OPTIONS]
```

Description

The `cov-analyze` command runs checkers on captured code in an intermediate directory and stores analysis results in that directory, which is specified with `--dir`. This command typically follows `cov-build` and precedes `cov-commit-defects` invocations on the same intermediate directory. Though `cov-analyze` does not report defects in Java and .NET bytecode, nor in some forms of source code not written by a person, the command does run an analysis of them for the benefit of finding interprocedural defects in editable source code.

A log file (`analysis-log.txt`) with information about the checkers used in the analysis, including notices of crashes, is located in the following directory: `<intermediate_directory>/output`

Note

If you get a fatal `No license found` error when you attempt to run this command, you need to make sure that `license.dat` was copied correctly to `<install_dir>/bin`.

On some Windows platforms, you might need to use administrative privileges when you copy the Coverity Analysis license to `<install_dir>/bin`. Due to file virtualization in some versions of Windows, it might look like `license.dat` is in `<install_dir>/bin` when it is not.

Typically, you can set the administrative permission through an option in the right-click menu of the executable for the command interpreter (for example, `Cmd.exe` or `Cygwin`) or Windows Explorer.

Options

`--aggressiveness-level <level>`

Enables a set of checker flags and `cov-analyze` options that cause Coverity Analysis to make more aggressive assumptions during analysis. Higher levels report more defects, and the analysis time increases. Values for `level` are `low`, `medium`, or `high`. Default is `low`.

Starting in version 7.0, this option applies to all programming languages that undergo analysis with `cov-analyze`. If a checker option applies to multiple languages, the aggressiveness level tuning will apply to that option for all supported languages. Changes to checker options that do not apply to a given language have no effect or related warnings.

The aggregate false positive rate for all checkers that are not parse warnings checkers is approximately 50% higher for `medium`, and 70% higher for `high`. Different aggressiveness levels do not change the rate of false positives that parse warning checkers report. A higher aggressiveness level for parse warning checkers enables more warnings for less severe defects.

The value `low` uses the default for all checkers and options. For a list of checker option defaults, see "Checker Enablement and Option Defaults by Language" in the *Coverity Checker Reference*.

The value `medium` uses the settings at the `low` level with the overrides shown in the following table:

Table 1. Increasing aggressiveness from ‘low’ to ‘medium’

Checker	Option	Low → Medium	Languages
BAD_ALLOC_STRLEN	<code>report_plus_any</code>	<code>false</code> → <code>true</code>	C, C++, CUDA, Objective C, Objective C++
BAD_EQ	<code>stat_threshold</code>	80 → 70	C#, Visual Basic
CALL_SUPER	<code>report_empty_override</code>	<code>false</code> → <code>true</code>	C#, Visual Basic
	<code>threshold</code>	0.65 → 0.55	C#, Java, Visual Basic
CHECKED_RETURN	<code>error_on_use</code>	<code>false</code> → <code>true</code>	C, C++, CUDA, Go, Java, Objective-C, Objective-C++
	<code>stat_threshold</code>	80 → 55	C, C++, CUDA, Go, Java, Objective-C, Objective-C++
CONSTANT_EXPRESSION_RESULT	<code>report_and_with_parents</code>	<code>false</code> → <code>true</code>	C, C++, CUDA, Go, Java, JavaScript, Objective-C, Objective C++, Ruby, TypeScript
	<code>report_constant_log_constants</code>	<code>false</code> → <code>true</code>	C, C++, CUDA, Go, Java, Objective-C, Objective C++, PHP, Ruby, Scala, Swift
FORWARD_NULL	<code>aggressive_null_source</code>	<code>false</code> → <code>true</code>	C, C++, CUDA, Objective-C, Objective C++, C#, Go, Java, JavaScript, PHP, Python 2, Python 3, Ruby, Scala, Swift, TypeScript, Visual Basic
	<code>deref_zero_errors</code>	<code>false</code> → <code>true</code>	C, C++, CUDA, Objective-C, Objective C++, Go, JavaScript, TypeScript
	<code>track_macro_nulls</code>	<code>false</code> → <code>true</code>	C, C++, CUDA, Go, JavaScript, Objective-C, Objective-C++, TypeScript
INFINITE_LOOP	<code>allow_asm</code>	<code>false</code> → <code>true</code>	C, C++, CUDA

Checker	Option	Low → Medium	Languages
	allow_pointer_derefs	false → true	C, C++, CUDA, C#, Java, Objective-C, Objective-C++
INTEGER_OVERFLOW	enable_const_overflow	false → true	C, C++, CUDA, Objective-C, Objective-C++
MISSING_RESTORE	report_restore_not_initialized_by_modify	false → true	C, C++, C#, CUDA, Java, Objective-C, Objective-C++
	report_unrelated_return	false → true	C, C++, C#, CUDA, Java, Objective-C, Objective-C++
MIXED_ENUMS	report_anonymous_enums	false → true	C, C++, CUDA, Objective-C, Objective-C++
NO_EFFECT	self_assign_to_locals	false → true	C, C++, CUDA, JavaScript, Objective-C, Objective-C++, PHP, Ruby, Scala, Typescript
	unsigned_enums	false → true	C, C++, CUDA, Objective-C, Objective-C++
NULL_RETURNS	allow_unimpl	false → true	C, C++, C#, CUDA, Go, Java, JavaScript, Objective-C, Objective-C++, TypeScript
	stat_bias	3 → 10	C#, Java
		1 → 10	JavaScript, TypeScript
		0 → 10	C, C++, CUDA, Objective-C, Objective-C++
	stat_min_checked	1 → 0	JavaScript, TypeScript
	stat_threshold	80 → 50	C, C++, C#, CUDA, Go, Java, JavaScript, Objective-C, Objective-C++, TypeScript
	suppress_under_relatives	true → false	C#, Java, Objective-C++
OVERFLOW_BEFORE_WRAP	WIDEN_macros	false → true	C, C++, CUDA, Objective-C, Objective-C++

Checker	Option	Low → Medium	Languages
	check_nonlocals	false → true	C, C++, CUDA, C#, Java, Objective-C, Objective-C++, Scala
	relaxed_operator_commutativity	false → true	C, C++, CUDA, C#, Java, Objective-C, Objective-C++, Scala
	report_intervening_calls	false → true	C, C++, CUDA, C#, Java, Objective-C, Objective-C++, Scala
OVERRUN	report_underrun	false → true	C, C++, CUDA, Objective-C, Objective-C++
RESOURCE_LEAK	allow_cast_to_int	false → true	C, C++, CUDA, Objective-C, Objective-C++
	allow_main	false → true	C, C++ , CUDA, Objective-C, Objective-C++
	allow_overwrite_memory	false → true	C, C++, CUDA, Objective-C, Objective-C++
	allow_unimpl	false → true	C, C++, CUDA, Objective-C, Objective-C++
	track_fields	false → true	C, C++ , CUDA, Objective-C, Objective-C++
SIZEOF_MISMATCH	strict_memcpy	false → true	C, C++, CUDA, Objective-C, Objective-C++
TAINTED_SCALAR	tainting_byteswaps	false → true	C, C++, CUDA, Objective-C, Objective-C++
UNINIT	check_arguments	false → true	C, C++, CUDA, Objective-C, Objective-C++
	check_mayreads	false → true	C, C++, CUDA, Objective-C, Objective-C++

Checker	Option	Low → Medium	Languages
	enable_write_context	false → true	C, C++, CUDA, Objective-C, Objective-C++
UNINIT_CTOR	report_scalar_arrays	false → true	C++, CUDA, Objective-C++
UNREACHABLE	report_unreachable_false_increment	false → true	C#, Java, JavaScript, PHP, TypeScript, Visual Basic
UNUSED_VALUE	report_dominating_arrays	false → true	C, C++, C#, CUDA, Go, Java, Objective-C, Objective-C++
	report_unused_finalizers	false → true	C, C++, C#, CUDA, Go, Java, Objective-C, Objective-C++
	report_unused_initializers	false → true	C, C++, C#, CUDA, Go, Java, Objective-C, Objective-C++
USELESS_CALL	include_current_objects	false → true	C, C++, C#, CUDA, Java, Objective-C, Objective-C++
	include_macro_calls	false → true	C, C++ , CUDA, Objective-C, Objective-C++
	include_macro_calls_in	false → true	C, C++ , CUDA, Objective-C, Objective-C++

The value high uses all the medium level settings, with the overrides shown in the following table:

Table 2. Increasing aggressiveness from ‘medium’ to ‘high’

Checker	Option	Medium → High	Languages
BAD_EQ	stat_bias	0.25 → 0.5	C#
CONSTANT_EXPRESSION_RESULT	report_subit_and_with_macros	false → true	C, C++ , CUDA, Objective-C, Objective-C++
	report_constant_logs_in_macros	false → true	C, C++ , CUDA, Objective-C, Objective-C++
	report_unnecessary_ops	false → true	C, C++, C#, CUDA, Go, Java, JavaScript, Objective-C, Objective-

Checker	Option	Medium → High	Languages
			C++, PHP, Python 2, Python 3, Scala, Swift, TypeScript
FORMAT_STRING_INJECTION	plain	false → true	C, C++, CUDA, Objective-C, Objective-C++
FORWARD_NULL	aggressive_derefs	false → true	C, C++, C#, CUDA, Go, Java, JavaScript, Objective-C, Objective-C++, PHP, Python 2, Python 3, Ruby, Scala, Swift, TypeScript, VisualBasic
	as_conversion	false → true	C#
INFINITE_LOOP	report_bound_type_mismatch	false → true	C, C++, C#, CUDA, Java, Objective-C, Objective-C++
	suppress_in_macro	true → false	C, C++, CUDA, Objective-C, Objective-C++
INTEGER_OVERFLOW	enable_all_overflow	false → true	C, C++, CUDA, Objective-C, Objective-C++
	enable_deref_sink	false → true	C, C++, CUDA, Objective-C, Objective-C++
LOCK	track_globals	false → true	C, C++, CUDA, Objective-C, Objective-C++
MIXED_ENUMS	report_disjoint_enums	false → true	C, C++, CUDA, Objective-C, Objective-C++
NESTING_INDENT_MISMATCH	report_bad_indentation	false → true	C, C++, C#, CUDA, Java, JavaScript, Objective-C, Objective-C++, PHP, Scala, TypeScript
NO_EFFECT	self_assign_in_macro	false → true	C, C++, CUDA, Objective-C, Objective-C++
NULL_RETURNS	stat_threshold	50 → 0	C, C++, C#, CUDA, Go, Java, JavaScript,

Checker	Option	Medium → High	Languages
			Objective-C, Objective-C++, TypeScript
OVERFLOW_BEFORE_WIDEN	check_bitwise_operands	false → true	C, C++, C#, CUDA, Java, Objective-C, Objective-C++, Scala
	check_types	(?:unsigned)?long long . *64. * → . *	C, C++, CUDA, Objective-C, Objective-C++
	ignore_types	s?size_t off_t time_t __off64_t ulong . *32. * → ^\$	C, C++ , CUDA, Objective-C, Objective-C++
	general_operator_conversion	false → true	C, C++, C#, CUDA, Java, Objective-C, Objective-C++, Scala
	relaxed_operator_conversion	true → false	C, C++, C#, CUDA, Java, Objective-C, Objective-C++, Scala
OVERRUN	check_nonsymbolic_dereference	false → true	C, C++, CUDA, Objective-C, Objective-C++
REGEX_CONVERSION	report_character_hygiene	false → true	Java
RESOURCE_LEAK	allow_address_taken	false → true	C, C++, CUDA, Objective-C, Objective-C++
	allow_constructor	false → true	C++ , CUDA, Objective-C++
	allow_template	false → true	C++, CUDA, Objective-C++
	allow_virtual	false → true	C++, CUDA, Objective-C++
UNCAUGHT_EXCEPT	report_all_fun	false → true	C++, CUDA, Objective-C++
UNINIT	allow_unimpl	false → true	C, C++, CUDA, Objective-C, Objective-C++
	assume_loop_always_executes	true → false	C, C++, CUDA, Objective-C, Objective-C++

Checker	Option	Medium → High	Languages
	check_malloc_wrapper	false → true	C, C++, CUDA, Objective-C, Objective-C++
UNINIT_CTOR	allow_unimpl	false → true	C++, CUDA, Objective-C++
UNREACHABLE	report_unreachable	false → true	C, C++, CUDA, Objective-C++

--all

Enables almost all checkers that are disabled by default (exceptions are noted below). Using this option is equivalent to using all of the following options:

- `--concurrency`
- `--enable-parse-warnings`
- `--enable PARSE_ERROR`
- `--enable STACK_USE`
- `--security`

To find out whether a checker can be enabled with this option, see the `--list-checkers` option.

Exceptions

The following checkers are disabled by default, and the `--all` option *does not* turn them on:

- Android security checkers (which are enabled with the `--android-security` option).
- DC.STRING_BUFFER
- ENUM_AS_BOOLEAN
- HARDCODED_CREDENTIALS
- HFA
- LOCK_INVERSION (for C#)
- MISRA_CAST
- ODR_VIOLATION for C++
- ORM_LOST_UPDATE
- Rule checkers (which are enabled with the `--rule` option).

- SECURE_CODING (Deprecated)
- SIZECHECK (Deprecated)
- UNENCRYPTED_SENSITIVE_DATA
- USER_POINTER
- WEAK_GUARD
- WEAK_PASSWORD_HASH
- Web application security checkers (such as XSS) are not affected by this option. To enable them, see `--webapp-security`.
- XML_INJECTION

Default checkers are enabled by default and are therefore unaffected by this option.

For information about enabling individual checkers, see the `--enable` option.

`--allow-unmerged-emits`

By default, the analysis fails if an intermediate directory contains emits of builds from multiple hosts. Specify this option to disable error checking and permit the analysis to continue in these cases.

If you use `cov-manage-emit add-other-hosts` to associate all emit repositories in the current intermediate directory with the current host, `--allow-unmerged-emits` is not needed to continue the analysis.

`--analyze-node-modules`

By default, `cov-analyze` does not analyze code in the `node_modules/` directories for JavaScript or TypeScript projects. This option enables analysis of the translation units in the `node_modules/` directories.

Even when you use the `--tu` or the `--tu-pattern` option, you must specify `--analyze-node-modules` in order to analyze translation units in `node_modules/` directories.

`--append`

Append defects from the last analysis run to the defects from this run.

By default, each analysis run includes individual checker-result files, the analysis summary file, and a metrics file. The `--append` option adds analysis results to the individual checker-result and summary files, but leaves the metrics file unchanged. So when you use the `--append` option, the metrics file will reflect the initial analysis without incremental analysis results for subsequent analyses that use the `--append` option.

The `--append` option is intended for appending issues found by custom Extend SDK checkers (see *Coverity Extend SDK 2020.12 Checker Development Guide* [↗](#)) and for importing issues by the `cov-import-results` and `cov-import-msvsca` commands. This option does not allow multiple `cov-`

`analyze` commands with standard checkers to write results into the same intermediate directory. To analyze a mixed-language code base, use a single `cov-analyze` invocation.

When this option is used with the `--output-tag` option, `--append` applies to the output location that is specified through `--output-tag`.

`--brakeman-aggressiveness-level <low|medium|high>`

Tune the aggressiveness of Brakeman Pro to only report defects that are above a certain confidence level. A higher setting reports more defects and increases the likelihood that any given defect is a false positive. Accepted values for this option are `low`, `medium`, or `high`. Default is `high`.

`--checker-option <checker_name>:<option>[:<option_value>], -co`

`<checker_name>:<option>[:<option_value>]`

Passes a checker option. Checker options and their default values are documented in the *Coverity 2020.12 Checker Reference* [↗](#).

Example:

```
INFINITE_LOOP:report_no_escape:true
```

Starting in version 7.0, when you specify the value of a checker option for a checker that supports the analysis of multiple languages, the value that you specify will apply to all languages to which that checker option applies. For example, if you set the `stat_threshold` to `NULL_RETURNS`, and you run an analysis on C/C++, C#, and Visual Basic code bases, the value you set for that option will apply to both languages. If you do not set the value, the checker will use the default values for those options, which in a very limited number of cases can vary by language.

Some checker options are language-specific, such as `FORWARD_NULL:dynamic_cast`. This option is only available for (and can only apply to) C/C++ even though the `FORWARD_NULL` checker supports multiple languages.

`--code-identity-file <file>`

The name of a code identity file, when required by the license. This file contains information about which files to include and/or exclude from line counts and analysis, as well as a signature of the licensed code base. The inclusions and exclusions are based on settings to `--search`, `--search-extensions`, and `--third-party-regex` in `cov-count-lines`. The code identity file should match the signature indicated by the optional string parameter `cbi_hash` in the license file.

When required by the license, `cov-analyze` searches for a matching file with a `cbi` extension in the following locations:

- The directory from which `cov-analyze` is invoked.
- The `bin/` subdirectory of the Coverity Analysis installation directory.
- The file specified by `--code-identity-file` (if provided).

Note that for licenses requiring a code identity file, `--strip-path` must be provided. See also the `--code-identity_file` option to `cov-count-lines`.

--code-version-date <date>

For Test Advisor and Desktop Analysis, use this option to specify the date of the source code. This date is used for impact analysis. If possible, it is recommended to use the SCM checkout date of the code being analyzed.

If this option is not specified, the products will use the latest invocation timestamp of `cov-build` as stored in the intermediate directory. If that is not available, then the current date and time are used instead.

This option is required when analyzing historical versions of your source code.

See Appendix A, *Accepted date/time formats* for proper formatting of the `<date>` argument.

 **Note**

So that Test Advisor can correctly interpret function history used for impact analysis, please ensure that EITHER:

1. All machines used to do analysis have the same timezone setting.
2. A timezone is specified for all dates in the policy file and the `--code-version-date` argument. For example, use `2010-01-01 00:00-08:00` instead of `2010-01-01`.

--codexm <checker-file>*

Specifies the CodeXM (`.cxm`) file or files to use in the analysis.

Example:

```
cov-analyze --dir mycxm --codexm myChecker.cxm --codexm myOtherChecker.cxm
```

CodeXM is a specialized language used to write customized checkers that run using the Coverity engine.

--codexm-print-debug

Enables the CodeXM `debug()` function.

When enabled, the `debug()` function prints values or messages to the system console.

If this option is not present when `cov-analyze` is invoked, calls to `debug()` are treated as no-ops.

--coding-standard-config <path/to/codingstandard_configuration_file>

[C/C++ MISRA option, required for a MISRA analysis.] Provides the path to a configuration file for a coding standard to run as part of the analysis. You can provide the option multiple times, with different configuration files to use multiple coding standards in an analysis run. Note that you cannot specify two configurations for the same standard in a single run.

 **Note**

Any analysis involving `--coding-standard-config` requires the information generated during `cov-build` when including the `--emit-complementary-info` option. See `--emit-complementary-info` for more information.

You can find sample configuration files in `<install_dir>/config/coding-standards/<name_of_standard>`. We recommend that you create a custom configuration based on these samples.

Coding standard analysis normally runs along with regular analysis. To analyze only for a single coding standard, use the `--disable-default` option along with `--coding-standard-config`

A configuration file can specify one of the following standards. These examples enable all supported rules for their respective standards.

- MISRA C 2004

Content for a configuration file for all supported 2004 standards (and no deviations):

```
{  version : "2.0",
  standard : "misrac2004",
  title: "your_title_here",
  deviations : []
}
```

- MISRA C++ 2008

Content for a configuration file for all supported 2008 standards (and no deviations):

```
{  version : "2.0",
  standard : "misrac++2008",
  title: "your_title_here",
  deviations : []
}
```

- MISRA C 2012

Content for a configuration file for all 2012 standards (and no deviations):

```
{  version : "2.0",
  standard : "misrac2012",
  title: "your_title_here",
  deviations : []
}
```

- AUTOSAR

The configuration files are all in the following directory:

```
<install-dir>/config/coding-standards/autosarcpp14/autosarcpp14-all.config
```

- CERT-C/CPP

The product is shipped with predefined configuration files (e.g. `cert-c-all.config`, `cert-c-L1-L3.config`, `cert-c-L2-L3.config`, `cert-c-L3-only.config`, `cert-c-L1-L2.config`, `cert-c-L1-only.config`, and `cert-c-L2-only.config`) under `<install>/config/coding-standards/cert*/`.

The levels are documented in the CERT-C/Cpp specifications:

- If you want to target only a specific level or permutation of levels, you can point to the predefined configuration file that matches, or you can define your own configuration file with your own custom deviations.
- If you want the entire set of standards rules that we support, you can use the `cert-c-all.config` file.

Content for a configuration file for all rules in CERT-C standard (and no deviations):

```
{
  version : "2.0",
  standard : "cert-c",
  title: "your_title_here",
  deviations : []
}
```

The HIS Metrics checker can also be used to measure MISRA coding standards. This is an optional checker setting that is enabled by adding the HIS Metrics settings to your MISRA configuration file. Once the checker is enabled, the defects that are found by the HIS Metrics checker are included in the emit process.

You can run the HIS Metrics checker by adding the following HIS Metrics settings to your MISRA configuration file:

```
{
  HIS_Metrics : {
    raw_metrics_filename : "raw-metrics.txt", // file in the output folder to
    which to dump the raw metrics for each function
    html_report_filename : "HIS_report.html", // The HTML report file
    policies : [ // Optional list of policies for each HIS metric
      {
        name : "COMF", // Could be one COMF, PATH, GOTO, CCM,
        CALLING, CALLS, PARAM, STMT, LEVEL, RETURN, VOCF, CYCLE
        compliant_range : {low : 0.4, high : 1,} // High and low values for
        the compliant range for this metric. Both are optional.
      },
      // More entries for each policy.
    ]
  }
}
```

Here's what the config looks like after adding the HIS Metrics settings to the MISRA configuration file. The `raw-metrics.txt` file is the file in the output folder to which the raw metrics for each function are dumped. The `HIS_report.html` file is the HTML report file. The `compliant_range` values set high and low values for the compliant range for this metric. Both are optional.

```
{
  "version": "2.0",
  "standard": "misrac2012",
  "title": "MISRA C-2012 All Rules",
```

```
"deviations": [],
HIS_Metrics : {
  raw_metrics_filename : "raw-metrics.txt",
  html_report_filename : "HIS_report.html",
  policies : [
    {
      name : "COMF",
      compliant_range : {low : 0.0, high : 1,}
    },
    // Could be one COMF, PATH, GOTO, CCM, CALLING, CALLS, PARAM, STMT,
    LEVEL, RETURN, VOCF, CYCLE
  ]
}
}
```

To commit the HIS Metrics checker results, run the `cov-analyze --coding-standard-config` command prompt. Then, type the `cov-commit-defects` command to commit all of the found HIS Metrics checker defects. The coding standard violations are then reported to Coverity Connect as an HIS Metric Violation.

If one or more HIS Metrics are unspecified in the configuration file, then the missing metric will typically default to the list of HIS Metrics values. Each metric has its own range of compliant values. (Default values for HIS Metrics are published and can be found online.) Note that there are two exceptions, where it is recommended that you modify the upper or lower bounds.

- The Coverity HIS Metrics checker for `COMF` defaults to a range between 0.2 and 100 (while the standard default range is between 0.2 and 1).
- The Coverity HIS Metrics checker for `CALLING` defaults to a range between 1 and 5 (while the standard default range is between 0 and 5).

The sample configuration files in `<install_dir>/config/coding-standards<name_of_standard>` specify a configuration and any deviations, along with the rules covered by the configuration. No violations will be reported for the rules specified in the `deviations` field.

Configuration example with deviations:

```
{
  version : "2.0",
  standard : "c2004",
  title: "C-2004 example with some deviations",
  deviations : [
    // Deviations for this example are Rules 5.6, 6.1, and 20.1.
    { deviation: "Rule 5.6", reason: "Currently disabled in the analysis
configuration." },
    { deviation: "Rule 6.1", reason: "Currently disabled in the analysis
configuration." },
    { deviation: "Rule 20.1", reason: "Currently disabled in the analysis
configuration." }
  ]
}
```

```
]
}
```

Note that for MISRA, the deviations are reported in the MISRA report, so they should explain why the deviation is claimed (either explaining in-line the measures being taken to mitigate risk or citing a separate document that does so) as per MISRA documentation. To keep a record of the claimed deviations, you might choose to store your configuration file in your source revision control repository.

The filenames of the sample configuration files identify what category of rules are run. The categories of rules vary between standards, but for example in MISRA_C2004, the configuration file `misrac2004-required-only.config` will run only the rules in the category required. In CERT, `cert-c-L1-L2.config` will run the rules in the categories L1 and L2.

For your custom configuration file, you can create and edit a copy of one of the samples instead of editing the file that Coverity provides. Using the copy will prevent the loss of your configuration upon upgrade and avoid the potential for other undesired behavior. Coverity also recommends adding the copy to your source stream to ensure that the history of changes to that file are tracked.

For MISRA rules and directives, see "MISRA Rules and Directives" [🔗](#) in the Coverity 2020.12 Checker Reference.

For the MISRA analysis workflow, see "Running coding standard analyses" [🔗](#) in Coverity Analysis 2020.12 User and Administrator Guide.

`--command <checker_pathname>`

[Extend SDK analysis option] Uses a Extend SDK checker at the specified path name.

`--concurrency`

[C/C++ analysis option] Enables C/C++ concurrency checkers that are disabled by default.

For a list of concurrency checkers that you can enable with this option, see `list-checkers`.

`--cpp`

[C/C++ analysis option] Filters by C/C++ translation units on which this command operates or reports. The command will fail with an informative error message if none of the translation units in the `emit` subdirectory match any of the specified language options in the intermediate directory.

`--cs`

[C# analysis option] Filters by C# translation units on which this command operates or reports. The command will fail with an informative error message if none of the translation units in the `emit` subdirectory match any of the specified language options in the intermediate directory.

`--cxx`

This option is deprecated and no longer has any effect. The corresponding checkers `BAD_OVERRIDE`, `CTOR_DTOR_LEAK`, `DELETE_ARRAY`, `INVALIDATE_ITERATOR`, `PASS_BY_VALUE`, `UNCAUGHT_EXCEPT`, `UNINIT_CTOR`, `WRAPPER_ESCAPE`, and, on Windows, `COM.BAD_FREE` and `COM.BSTR.CONV`, are now enabled by default. Checkers that can only find defects in C++ code automatically do not run on C code. Note that `PASS_BY_VALUE` can find defects in C code. If you were using `--disable-default --cxx`, replace it with individual `--enable` options.

`--cxx-container-type-regex <regex>`

Allows you to specify C++ container types for all checkers that look for them. The analysis will consider C++ classes whose name matches the specified `<regex>` to be container classes.

`--dc-config <file.json>`

Identifies a JSON file for one or more `DC.CUSTOM_*` (custom Don't Call) checkers that you intend to run in the analysis (see `DC.CUSTOM_CHECKER` in the *Coverity 2020.12 Checker Reference* [🔗](#))

Note that use of this option enables all the `DC.CUSTOM_*` checkers that are configured in the JSON file. You can disable them individually with `--disable <checker-name>`. The `--disable-default` option will disable all of them.

 **Note**

CodeXM is a language specifically designed for writing new checkers. If you have not already invested in custom DC checker configuration, we recommend you use CodeXM rather than the JSON configuration. See the section “Writing Your Own Don't Call Checker” in the manual *Learning to Write CodeXM Checkers*.

`--derived-model-file <derived_file.xmlldb>`

[Deprecated as of version 7.7.0] This option will be removed and replaced in a future release. Use `--model-file` instead.

`--dir <intermediate_directory>`

Path name to an intermediate directory that is used to store the results of the build and analysis.

`--disable <checker_name>, -n <checker>`

Disables a checker. This option can be specified multiple times. See also `--list-checkers` and `--disable-default`.

To find out whether a checker is enabled or disabled by default, see the `--list-checkers` option.

You can also use this option to disable individual SpotBugs bug patterns. To specify a pattern, you need to add an FB prefix to it. For example:

```
--disable FB.DM_EXIT
```

To disable SpotBugs bug patterns, you can also use `--disable-fb` or `--fb-exclude`.

`--disable-android-security`

Disables the Android application security checkers. Note that these checkers are disabled by default.

See also, `--android-security`.

`--disable-brakeman`

Disables Brakeman Pro checkers. Use the `--enable-brakeman` option to re-enable these checkers.

`--disable-default`

Disables default checkers. This option is useful if you want to disable all default checkers and then enable only a few with the `--enable` option.

For a list of checkers that are disabled through this option, see the `--enable` option documentation for the `cov-analyze` command.

`--disable-fnptr`

[C/C++ analysis option] Disables analysis of calls to function pointers for defects. See also `--enable-fnptr`.

`--disable-jshint`

[JavaScript option] Disables JSHint analysis. The JSHint analysis is disabled by default.

Do not specify both `--enable-jshint` and `--disable-jshint` on the same command line.

`--disable-parse-warnings`

[C/C++/Swift analysis option] Disables all parse warnings, and override other arguments that might have enabled them, such as `--all` or `--enable-parse-warnings`. The order of command-line options is irrelevant; the `--disable-parse-warnings` option takes precedence.

`--disable-openapi`

Disables Spectral OpenAPI checkers (OPENAPI.*). The checkers are enabled by default. See the description of OPENAPI.* checkers in the *Checker Reference Guide*.

`--enable <checker_name>, -en <checker>`

Enables a checker that is not otherwise enabled by default. The checker name is case insensitive. This option will enable a checker for all languages supported by the checker. Note that default enablement of a given checker can vary by language.

Checkers are enabled by name, so related checkers such as `MISSING_LOCK` and `GUARDED_BY_VIOLATION` are enabled independently. The checker name is case insensitive. You can specify this option multiple times. See also `--list-checkers` and `--disable-default`.

Unlike the `--disable` option, this option does not work with SpotBugs bug patterns.

`--enable-audit-mode`

Enables audit-mode analysis, which is intended to expose more potential security vulnerabilities by considering additional data sources that could be used in an exploit.

Using this option usually reports more defects that are less likely to represent true vulnerabilities. Audit mode analysis will take noticeably longer to complete: It analyzes all functions that are present in the source, not just those that are present in the call tree. This level of testing can be useful for auditors and for any users who want to see the maximum number of defects.

The `--enable-audit-mode` option is the equivalent of enabling the following two options:

- `--enable-audit-checkers`

Enables additional audit-mode checkers that normally are off by default: for example, `SQL_NOT_CONSTANT` and `INSECURE_COOKIE`.

For a list of all Audit Mode checkers, refer to the “Checker Enablement and Option Defaults by Language” chapter in the HTML version of the *Coverity Checker Reference*.

- `--enable-audit-dataflow`

This option has the following effects:

- It sets the `--webapp-security-aggressiveness-level` to `high`.
- It sets `--distrust-all`.
- For tainted dataflow security checkers, it introduces additional audit-mode sources of untrusted (tainted) data to model potential attacks. Such sources include all function parameters, and the return value from external functions (those that are not visible in the source code or bytecode, and for which no model exists).

`--enable-brakeman`

Enables Brakeman Pro checkers (default). Use `--disable-brakeman` to disable these checkers.

`--enable-callgraph-metrics`

Creates `<intermediate_directory>/output/callgraph-metrics.txt` and `<intermediate_directory>/output/checked-return.csv`.

The `callgraph-metrics.txt` file has information about which functions are analyzed. The file lists whether a function is implemented, which means it is analyzed, or whether a function is unimplemented, which means that it is not analyzed. A model is used if it is available. It also shows the number of callers for each function.

The `checked-return.csv` file stores information on the percentage of times that each the return value of each function is checked. This information can help you understand situations where the statistical checkers report different defects in local builds than they do in full builds.

For details, see *Coverity Analysis 2020.12 User and Administrator Guide*. [🔗](#)

Starting in version 7.0, applies to all programming languages.

`--enable-constraint-fpp`

Enables additional filtering of potential defects by using an additional false-path pruner (FPP). This option can increase the analysis time up to 20% (normally much less), but decrease the number of false positives that occur along infeasible paths. Because this FPP uses a different method for pruning false positives, it is possible that a very small number of true positives are pruned as well.

Note that use of this option requires an *additional* 200MB of memory per worker.

Starting in version 7.0, this option applies to C/C++, C#, Visual Basic, and Java.

`--enable-default`

Enables all default checkers. This option takes precedence over the `--disable-default` option or the `--test-advisor` option. That is, if this option is specified then all default checkers are enabled regardless of the use of those options. However, individual checkers can still be disabled using the `--disable` option.

`--enable-exceptions`

Enables exceptional control flow analysis for C++. If specified, this option will report the following type of resource leak as a defect:

```
        bool maybe;
void test1() {
    int *x = new int;
    if (maybe) {
        throw 0; // x is leaked
    }
    delete x;
}
```

By default, the analysis ignores the exception type `std::bad_alloc` because some applications might not be designed to handle out-of-memory scenarios. If you specify `--enable-exceptions --handle-badalloc`, the analysis will report the following example as a defect. The example leaks memory if `new char` throws a `std::bad_alloc` exception:

```
void test() {
    int *x = new int;
    char *y = new char; // Leaks 'x' if this throws std::bad_alloc
    delete y;
    delete x;
}
```

Note that defects are not limited to leaks. For example, the `FORWARD_NULL` checker finds the following defect:

```
int *global;

void foo() {
    int *y = 0;
    try {
        // if std::bad_alloc is thrown, y remains null
        global = new int;
        y = global;
    } catch (...) {
        // empty
    }
    *y = 1; //FORWARD_NULL defect.
}
```

This option is disabled by default for C++ but enabled by default for Java, Visual Basic, and C#.

See also, `--handle-badalloc`.

`--enable-fnptr`

[C/C++ analysis option] Enables analysis of calls to function pointers for defects. By default, calls through function pointers are not used by the analysis engine for interprocedural analysis. When specified, this option allows up to 100 function resolutions for any function pointer. If that limit is exceeded, the analysis engine reverts to the default behavior.

When using this option, the analysis time typically increases by approximately 20%. However, the false positive rate might increase. See also `--disable-fnptr`.

--enable-jshint

[JavaScript option] Enables the JSHint analysis of captured JavaScript source code *except for* minified source files (see JSHINT.* in the *Coverity 2020.12 Checker Reference* [🔗](#) for details). Note that running a JSHint analysis requires an additional pass over all analyzed JavaScript code, which can significantly increase overall analysis time. The JSHint analysis is disabled by default.

JSHint reports a large number of defects, unless a custom configuration is used to trim down the results or the code is highly polished.

If you want to run the JSHint analysis with a custom `.jshintrc` configuration file, use `--use-jshintrc`. Otherwise, the analysis will use the Coverity default configuration file in `jshint/config.json` and ignore any `.jshintrc` files in your source tree.

Do not specify both `--disable-jshint` and `--enable-jshint` on the same command line.

--enable-parse-warnings

[C/C++/Swift analysis option] Enables parse warnings, recovery warnings, and semantic warnings that are produced by the `cov-build` command so that they appear as defects in Coverity Connect. See also `--parse-warnings-config`.

This option is set automatically if the `--aggressiveness-level` option is set to `medium` (or to `high`).

--enable-single-virtual

Enables single, virtual-call resolution. By default, a C++ analysis treats all virtual functions as unimplemented, whereas full virtual call resolution is enabled by default for Java, Visual Basic, and C# analyses. When this option is enabled, interprocedural analysis across virtual calls takes place when the analysis engine finds only one implementation of a virtual function. When the analysis engine finds more than one implementation, it assumes that the virtual function is unimplemented. Do not specify this option if you specify the `--enable-virtual` option.

A C++ analysis can take longer than the default analysis because the analysis engine looks at implementations of virtual functions, which can result in more defect reports. Though this using this option might expedite Java, Visual Basic, and C# analyses, it also significantly affects results for interprocedural checkers.

Specify this option, or `--enable-virtual`, to enable interprocedural analysis of Apple Block invocations in C and C++ code.

Starting in version 7.0, applies to all programming languages.

--enable-openapi

Enables Spectral OpenAPI checkers (OPENAPI.*). The checkers are enabled by default, but this can be used in conjunction with the `--disable-default` option to selectively enable them. See the description of OPENAPI.* checkers in the *Checker Reference Guide*.

--enable-test-metrics

This option creates test metrics data files (in the intermediate directory) that can be subsequently committed to Coverity Connect for storage. This option can be specified with other `cov-analyze`

options and it does not require `cov-analyze` to be invoked with `--test-advisor` option. This option turns on all the necessary passes of the analysis that are required to compute test metrics.

Test metrics contain a mapping from the tests available in the emit directory to the functions that each test covers. This data is used during Test Prioritization to calculate properties of the tests (for example, the `covered_function_count` property) which can be used as part of the test score.

The `-enable-test-metrics-` option requires `--strip-path` option.

This option works with C/C++, C#, Visual Basic, and Java.

`--enable-virtual`

Enables full, virtual-call resolution. By default, a C++ analysis treats all virtual functions as unimplemented, whereas full virtual call resolution is enabled by default for Java and C# analyses. When specified, this option allows up to 100 function resolutions for any virtual method. If that limit is exceeded, the analysis engine reverts to the default behavior.

Do not use this option if you specify the `--enable-single-virtual` option. The analysis can take significantly longer than the default or when the `--enable-single-virtual` option is enabled because the analysis engine looks at all implementations of virtual functions, which can result in more defect reports.

Specify this option, or `--enable-single-virtual`, to enable interprocedural analysis of Apple Block invocations in C and C++ code.

Note

To make the analysis resolve to a model of a virtual or pure virtual function without using `--enable-virtual`, see "Analyzing models of virtual functions" in the Coverity 2020.12 Checker Reference [✚](#).

`--export-summaries <true|false>`

Collects function summary data for the analysis. The collected data provides interprocedural analysis information for Desktop Analysis users, and must be committed to any stream that is used by Desktop Analysis.

This option is `true` by default.

`--field-offset-escape`

[C++ analysis option] A pointer escapes the analysis if it is written to memory, passed to `free()`, or passed to a function definition that is inaccessible to `cov-analyze`. Once the pointer escapes the analysis, the storage to which it points will never be treated as a leak or uninitialized.

This option eliminates certain false positives in C++ by making the analysis treat `&v->field` as an alias for `v` because some programs exploit the fact that `(&v->field) - offsetof(typeof(v), field) == v` to free `v` given `&v->field`.

By default, this heuristic applies to only to C code (but not C++). This option enables this heuristic for C++, as well.

See also, `--no-field-offset-escape`.

`--force`

Turns off incremental analysis. This setting forces a full re-analysis of the source, even if the source file or other source files on which it depends have not changed since it was previously analyzed.

`--fnptr-models`

[C/C++ analysis option] Enables function pointer models if the analysis fails to analyze certain function pointers calls. You can enable analysis of calls to function pointers, without requiring explicit models, using the `--enable-fnptr` option. For more information and examples, see "Modeling function pointers" in the Coverity 2020.12 Checker Reference [↗](#).

`--handle-badalloc`

[C/C++ analysis option] Causes the analysis as a whole to handle exceptions of type `std::bad_alloc`, both for exceptional control flow and for `UNCAUGHT_EXCEPT`. By default, such exceptions are otherwise ignored even when you use `--enable-exceptions`.

For an example, see `--enable-exceptions`.

`--hfa`

[C-only analysis option] Reports unnecessary header file includes. For more information, see "HFA" in the Coverity 2020.12 Checker Reference [↗](#).

The `--all` option does not enable this checker.

`--hibernate-config`

Specifies a directory that contains Hibernate mapping XML files, if applicable. Pertains to the `HIBERNATE_BAD_HASHCODE` checker (see the *Coverity 2020.12 Checker Reference* [↗](#) for details).

`--ignore-deviated-findings`

Set this option to prevent reporting defects that are deviated with annotations.

Any defects or false positives annotated using the `#pragma Coverity compliance` directive will be suppressed and will not be reported by Coverity Connect. *All* recorded deviations in the current project version are then written to a CSV file. For more information see the *Coverity Checker Reference*.

`--inherit-taint-from-unions`

Enable taint to flow downwards from a C/C++ union to its component fields. This is required to check code that writes to a union using `memcpy(&u, &tainted, n)` and later reads using `u.field`.

Affects security checkers `TAINTED_SCALAR` [↗](#) and `INTEGER_OVERFLOW` [↗](#).

`--java`

[Java analysis option] Filters by Java translation units on which this command operates or reports. The command will fail with an informative error message if none of the translation units in the `emit` subdirectory match any of the specified language options in the intermediate directory.

`--jobs <number-of-workers> | auto | max<number-of-workers>, -j <number-of-workers> | auto | max<number-of-workers>`

Allows you to control the number of analysis workers that run in parallel, subject to any limits specified by your license. Starting in version 7.6.0, the need to use this option should be rare because the default typically sets the appropriate number of workers to use for your hardware, license, and analysis task. Note that the default for this option varies by license.

- Default for a non-Flexnet license (`license.dat`): `--jobs auto`
- Default for a Flexnet license: `--jobs max8`

In general, the analysis runs faster with more threads, but the scalability of that speed increase depends on the kind of analysis, the code language(s), and other properties of the code base. In general, analysis of C code parallelizes best, followed by C++, followed by C#, Visual Basic, and Java quality analysis, followed by Web application security analysis, which is largely not parallelized.

This option must specify one of the following values:

- `<number-of-workers>`: Specifies number of analysis workers to run in parallel.

Example: `--jobs 8`

The specified number of workers is not allowed to exceed suggested limits for your hardware unless you also use the `--override-worker-limit` option.

- `auto`: Automatically determines the number of workers to use. Hardware detection through `--jobs auto` attempts to optimize for the case where the analysis has full or nearly-full use of the machine's computational resources (memory and CPU). If that is not the case, you should consider setting `-j` explicitly, for example, where the analysis occupies one of several "executors" on a continuous integration server.

Example: `--jobs auto`

If `--jobs auto` is set, the analysis will determine the number of workers to run based on the minimum of the following:

- The largest number of workers that keeps the recommended minimum physical memory requirements below the actual physical memory of the machine.
- The number of logical CPUs, or virtual cores, on the machine. This is the number of threads or processes that the operating system can schedule simultaneously on the hardware.
- Six (6) times the number of "physical" CPU cores, when known.
- 48: Coverity has not found performance improvements from using more than 48 workers, and using more might reach limits on open file descriptors, and so on.
- The number permitted by the license or available for Flexnet checkout.

Detection of memory and logical CPUs should work on all analysis platforms, but detection might fail or produce incorrect results in some virtualization environments.

This value is not compatible with the `--override-worker-limit` option.

- `max<number-of-workers>`: Limits the number of analysis workers that can run in parallel based on the maximum you set and the amount of memory and number of cores that are available.

Example: `--jobs max8`

This value is not compatible with the `--override-worker-limit` option.

See [Running a Parallel Analysis](#) and [Running a Parallel Analysis with Coverity Analysis for Java](#) for guidance.

For backward compatibility, the `--j <number-of-workers>` syntax is still supported in this release.

`--jvm-max-mem`

Sets the value of the heap size for the JVMs used by the Java Web application security dynamic analyzer and the SpotBugs analysis. The option specifies the size (an integer) in megabytes. The option sets the option `--fb-max-mem`. An explicit `--fb-max-mem` setting overrides the heap size set by `--jvm-max-mem` for a particular JVM.

`--list-checkers`

Displays a list of checkers that are available in the current release. Each entry indicates whether the checker is enabled by default, and if not, how you can enable it. Some require the use of `--enable`, while others can be enabled with other options (for example, `--concurrency` or `--security`), as well. For detailed information about checkers, see the *Coverity 2020.12 Checker Reference*.

`--max-loop <num>`

Limits the maximum number of times that loops are traversed. The default limit is 32, which should be encountered rarely. -1 means unlimited.

`--max-mem <value>`

Sets the maximum amount of memory, in megabytes, that a single analysis worker process will use for the core analysis. The total memory required is approximately the product of the `--max-mem` and `-j` options. The default value is 512.

The worker will use some additional memory for miscellaneous purposes, and even more memory if you use `--enable-constraint-fpp` or enable `INTEGER_OVERFLOW`.

The analysis will reject a setting that is too large for the available physical memory and number of workers.

On 32-bit Windows systems, do not set `<value>` to more than 512 megabytes.

out-of-memory error

An out-of-memory error indicates that the analysis is trying to use more memory than is available to the system. If an out-of-memory error occurs, use the `--max-mem` option to decrease the amount of memory that the analysis is allowed to use.

Note that JVM max-mem options work in the opposite way. So it is necessary to *increase* (not decrease) the max-mem for out-of-memory errors related to the JVM. In this case, it is possible that the system will run out of memory in the JVM.

`--misra-config <path/to/misra_configuration_file>`

**Note**

This option has been deprecated as of 2018.01 and will be removed in a future release. Use the `--coding-standard-config` option instead.

[C/C++ MISRA option, required for a MISRA analysis.] Provides the path to a configuration file for a MISRA analysis of C or C++ code according to one of the MISRA standards. If you want analysis results for multiple MISRA standards, you must run separate analyses for each standard.

MISRA analysis can run together with non-MISRA analysis. To run a MISRA-only analysis, you can use the `--disable-default` option with `--misra-config`.

We recommend that you create a custom configuration file that is based on one of the samples found in `<install_dir>/config/MISRA`. Your configuration file should list the rules and deviations that matter to your organization.

A configuration file can specify only one of the following standards:

- MISRA C 2004

Content for a configuration file for all supported 2004 standards (and no deviations):

```
{  version : "2.0",
  standard : "c2004",
  title: "your_title_here",
  deviations : []
}
```

- MISRA C++ 2008

Content for a configuration file for all supported 2008 standards (and no deviations):

```
{  version : "2.0",
  standard : "cpp2008",
  title: "your_title_here",
  deviations : []
}
```

- MISRA C 2012

Content for a configuration file for all 2012 standards (and no deviations):

```
{  version : "2.0",
  standard : "c2012",
  title: "your_title_here",
}
```

```
    deviations : []  
  }
```

The sample configuration files in `<install_dir>/config/MISRA` specify a configuration and any deviations, along with the rules covered by the configuration. No violations will be reported for the rules specified in the `deviations` field.

Configuration example with deviations:

```
{  
  version : "2.0",  
  standard : "c2004",  
  title: "C-2004 example with some deviations",  
  deviations : [  
    // Deviations for this example are Rules 5.6, 6.1, and 20.1.  
    { deviation: "Rule 5.6", reason: "Currently disabled in the analysis  
configuration." },  
    { deviation: "Rule 6.1", reason: "Currently disabled in the analysis  
configuration." },  
    { deviation: "Rule 20.1", reason: "Currently disabled in the analysis  
configuration." }  
  ]  
}
```

Note that the deviations are reported in the MISRA report, so they should explain why the deviation is claimed (either explaining in-line the measures being taken to mitigate risk or citing a separate document that does so) as per MISRA documentation. To keep a record of the claimed deviations, you might choose to store your configuration file in your source revision control repository.

The filenames of the sample configuration files (for example, `MISRA_c2012_7.config`) identify the standard (`MISRA_c2004`, `MISRA_cpp2008`, or `MISRA_c2012`). Filenames ending in `_7.config` cover all supported rules for a given standard. The lower the trailing number in the filename (for example, the `_1` in `MISRA_c2012_1.config`), the fewer the rules covered by the analysis and the more deviations listed in the file. The numbering system is a legacy support feature.

For your custom configuration file, you can create and edit a copy of one of the samples instead of editing the file that Coverity provides. Using the copy will prevent the loss of your configuration upon upgrade and avoid the potential for other undesired behavior. Coverity also recommends adding the copy to your source stream to ensure that the history of changes to that file are tracked.

For MISRA rules and directives, see "MISRA Rules and Directives" [🔗](#) in the Coverity 2020.12 Checker Reference.

For the MISRA analysis workflow, see "Running MISRA analyses" [🔗](#) in Coverity Analysis 2020.12 User and Administrator Guide.

`--model-file <file>.xmldb`

Uses the specified file to override any function models that are automatically derived from the implementation. It can determine whether a specified file is for user or derived models. Note that

if the default file at `<install_dir_sa>/config/user_models.xml` exists, it is used even without specifying this option. This option can be specified multiple times.

Note that you can only use this option on the output of `cov-collect-models` or `cov-make-library`. For more information, see "Model search order" in the Coverity 2020.12 Checker Reference [☞](#).

--no-field-offset-escape

[C/C++ analysis option] Disables a heuristic that can cause `RESOURCE_LEAK` and `UNINIT` to produce false negatives when tracking aliases of pointers.

A pointer escapes the analysis if it is written to memory, passed to `free()`, or passed to a function whose definition is inaccessible to `cov-analyze`. Once the pointer escapes analysis, the storage to which it points will never be considered leaked or uninitialized.

To eliminate false positives in C code (but not C++ code), the analysis considers `&v->field` to be an alias for `v` because some programs exploit the fact that `(&v->field) - offsetof(typeof(v), field) == v` to free `v` given `&v->field`.

If a program does not use this idiom, this heuristic might lead to false negatives. For example, if you call `myfunction(&v->field)` when this heuristic is enabled, the analysis assumes that `v` escapes, so the analysis will not catch a `RESOURCE_LEAK` or `UNINIT` on `v`. This option disables the application of that heuristic.

This option is set automatically if the `--aggressiveness-level` option is set to `medium` (or to `high`).

--no-java

Disables Java analysis. By default, the `cov-analyze` command otherwise analyses any Java code it finds in the intermediate directory.

--no-log

Disables logging.

--one-tu-per-psf <true|false>

When set to `true`, analyzes exactly one translation unit (TU) found for a given primary source file name. If there is more than one TU for a primary source file, the analysis will pick a single TU using an algorithm that is intended to ensure consistency between analysis runs. However, if the build command lines change, the analysis may make different choices and the results may vary, even though the code appears to be unchanged. A `false` value enables analysis of all TUs, regardless of primary source file duplication. The default value is `true`.

--output-tag <name>

Specifies a non-default location within the intermediate directory for the results of one or more analyses. The name can be anything you choose, using characters allowed in file names. When specified *without* the `--append` option, prior results found in that location are replaced. When specified *with* `--append`, new results are added to the result set.

--override-worker-limit

Allows you to specify a value to `-j` that is greater than the recommended value. This option can be useful when the license allows more workers than the number of cores in the machine.

- `--parse-warnings-config <filename>`
[C/C++/Swift analysis option] Specifies the name for the configuration file, which allows you to change the parse warnings that pass through a warning filter. For a sample, see `config/parse_warnings.conf.sample`. See also `--enable-parse-warnings`.
- `--path-log-threshold <number>`
If a function has more than `<number>` paths, this count is output to the log file.
- `--paths <number>`
Sets the upper limit on the number of paths to traverse for each function. Default is 5000.
- `--preview`
This option has been deprecated. For backwards compatibility, this option enables the same checkers that it used to.
- `--print-paths`
Prints the number of paths explored for each analyzed function.
- `--report-in-minified-js`
[JavaScript application option] Enables the JavaScript checkers for minified source files. It ensures that Coverity Analysis scans the minified JavaScript source files and reports any defects that are found.
- `--resolve-calls-to-all-delegates <true|false>`
When `true` (the default is `false`), it allows resolving more calls to C# delegates, which may report more defects, notably `LOCK_INVERSION` defects. It may cause more false positives to be reported.
- `--rule`
[C/C++ analysis option] Enables rule checkers.
- For a list of rule checkers, you can use `list-checkers`.
- `@@<response_file>`
Specify a response file that contains a list of additional command line arguments, such as a list of files for analysis. Each line in the file is treated as one argument, regardless of spaces, quotes, etc. The file is read using the platform default character encoding. Using a response file is recommended when the list of input XML files is long or automatically generated.
- Optionally, you can choose a different encoding, by specifying it after the first "@". For example:
- ```
cov-analyze [OPTIONS] @UTF-16@my_response_file.txt
```
- You must use a supported Coverity encoding, listed under the `cov-build --encoding` option.
- `--security`  
[C/C++ and Objective-C/C++ analysis option] Enables C/C++ and Objective-C/C++ security-related checkers.
- For a list of the security checkers to which this option applies, you can use `--list-checkers`.

`--security-file <license file>, -sf <license file>`

Path to a valid Coverity Analysis license file. If not specified, this path is given by the `<security_file>` tag in the Coverity configuration or by `license.dat` (located in the Coverity Analysis `<install_dir>/bin` directory). A valid license file is required to run the analysis.

`--strip-path <path>, -s <path>`

Strips the prefix of a file name path in error messages and references to your source files. If you specify the `--strip-path` option multiple times, you strip all of the prefixes from the file names, in the order in which you specify the `--strip-path` argument values.

This option is also available with `cov-commit-defects` and `cov-import-results`.

The leading portion of the path is omitted if it matches a value specified by this option. For example, if the actual full pathname of a file is `/foo/bar/baz.c`, and `--strip-path /foo` is specified, then the name attribute for the file becomes `/bar/baz.c`.

 **Important!**

Coverity recommends using this option for a number of reasons:

Failure to use this option can result in poor Coverity Connect performance, triage issues related to component maps, an unnecessary increase the size of the Coverity Connect database, and even incorrect LOC counts.

This option shortens paths that Coverity Connect displays. It also allows your deployment to be more portable if you need to move it to a new machine in the future.

In addition, using this option during the analysis, rather than when committing the analysis results to Coverity Connect, can enhance end-to-end performance of the path stripping process itself.

The `--strip-path` option is mandatory when running `cov-analyze --test-advisor`. This changes how file-level violations are merged with existing violations in Coverity Connect. Violations generated with Test Advisor 2020.12 will not be merged with violations generated with previous versions of Test Advisor, and existing triage of previously-generated Test Advisor violations will not be associated with the new violations.

Linux example:

```
> cov-analyze --dir myDir --strip-path=`pwd`
```

Windows example:

```
> cov-analyze --dir myDir --strip-path=%cd%
```

In the less common case, the option should specify the root of your build tree.

`--suppress-vulnerabilities-in-dead-code`

Java only. Suppresses security defects from tainted dataflow checkers when they require the execution of dead code.

`--ticker-mode <mode>`

Sets the mode of the progress bar ticker display to:

`none`

No progress bar at all.

`no-spin`

Print stars only, without the spinning bar.

`spin`

(default) Stars with a spinning bar at the end. Each analyzed function corresponds to one step of spin.

`--tu <translation_unit_id(s)>, -tu <translation_unit_id(s)>`

Limits the scope of `cov-analyze` to a set of translation units (TUs), named by their numeric ID attribute(s). A translation unit approximately maps to the output from a single run of a compiler.

This option requires a comma-separated list of id(s), and `--tu` can be specified multiple times. The union of all these identifier sets is the set of TUs to operate on subsequently, for operations that work on TUs.

Even when using the `--tu` or `--tu-pattern` options, you must specify the `--analyze-node-modules` option in order to analyze translation units in `node_modules`.

It is an error if any of the specified IDs do not correspond to any existing translation unit. To get the IDs for translation units, use the `cov-manage-emit list` sub-command.

You can use the `--tu` and `--tu-pattern` options together.

`--tu-pattern <translation_unit_pattern>, -tp <translation_unit_pattern>`

Limits the scope of `cov-analyze` to a set of translation units specified with a translation unit pattern. The `--tu-pattern` option can be specified multiple times. Matching TU sets are unioned together across all patterns.

Both `--tu` and `--tu-pattern` can be specified on a single command line. The final set of TUs operated upon includes a given TU if it matches any specified translation unit pattern or its ID is listed explicitly as an argument to `--tu`.

Even when using the `--tu` or `--tu-pattern` options, you must specify the `--analyze-node-modules` option in order to analyze translation units in `node_modules`.

It is an error if at least one `--tu-pattern` argument is specified but no translation unit matches any of the specified patterns.

You can get useful information on TUs by using the `cov-manage-emit list` sub-command.

See Translation unit matching for more information.

`--use-jshintrc <path/to/your/.jshintrc>`

[JavaScript analysis] Identifies a custom configuration to use for the JSHint analysis in place of the default Coverity configuration file (see <http://jshint.com/docs/>  for information about `.jshintrc` file configuration).

Note that hierarchical configuration files, based on the source code directory hierarchy, are not supported, due to limitations in capture and in how `cov-analyze` invokes JSHint.

When using `--use-jshintrc`, you must also pass `--enable-jshint`.

`--user-model-file <user_file.xmlldb>`

[Deprecated as of version 7.7.0] This option will be removed and replaced in a future release. Use `--model-file` instead.

`--wait-for-license`

Indicates that if a license cannot be obtained from the license server, `cov-analyze` must wait until a license becomes available. After a license becomes available, `cov-analyze` acquires it and proceeds with the analysis. This option is ignored if `cov-analyze` does not use a floating-node license.

## Extend SDK options

`--dtd <directory>`

[Deprecated Extend SDK analysis option] Use the `--prevent-root` option instead.

`--prevent-root`

[Extend SDK analysis option] When running a Extend SDK checker, specify the location of the Coverity Analysis installation directory:

```
--prevent-root /<install_dir_sa>
```

See *Coverity Extend SDK 2020.12 Checker Development Guide* [↗](#) for more information.

## Java SpotBugs options

`--disable-fb`

[SpotBugs analysis option for Java] Disables SpotBugs analysis.

An error will occur if you combine `--enable-fb` with `--disable-fb`.

`--enable-fb`

[SpotBugs analysis option for Java] Enables SpotBugs (version 2.0.1) analysis (requires a Coverity license). You can use this option to enable SpotBugs analysis while disabling all other default checkers through the `--disable-default` option. If you attempt to run an analysis when using this option, but you do not have a Coverity license, Coverity Analysis will disable the checkers and print an error message.

SpotBugs [↗](#) is an open source program for finding bugs (defects) in Java code. It provides a large group of SpotBugs bug patterns that can detect a wide variety of defects.

A SpotBugs analysis complements the analysis with Coverity Analysis for Java checkers. By default, `cov-analyze` detects those medium-priority and high-priority SpotBugs defects that do not generate too many unimportant results (for details, see `<install_dir_sa>/spotbugs-ext/config/include.coverity-default.xml`). The default analysis also excludes results that overlap with

non-deprecated Coverity Analysis for Java checkers. However, you can include or exclude other bugs from the analysis. To refine the SpotBugs results, see `--fb-include`, `--fb-exclude`, and `--disable`.

You can use `cov-analyze` to run a SpotBugs analysis on builds in your intermediate directory, which are generated by `cov-build` or by `cov-emit-java`. However, Coverity Analysis for Java does not integrate with SpotBugs running outside of the `cov-analyze` command. The SpotBugs analysis should add no more than about 20% to the running time of `cov-analyze`, assuming that most Coverity checkers are enabled.

Coverity Analysis uses the FB prefix to distinguish defects that match SpotBugs bug patterns from defects found by Coverity checkers. For example, Coverity Analysis for Java uses `FB.DM_EXIT` for the SpotBugs `DM_EXIT` bug pattern. The console prints a summary of the results, which looks something like the following:

```
SpotBugs Checkers: 74 errors
FB.DE_MIGHT_IGNORE 1
FB.DM_EXIT 5
FB.DP_CREATE_CLASSLOADER_INSIDE_DO_PRIVILEGED 2
FB.EI_EXPOSE_REP 14
...
```

Keep in mind that the defect summary identifies the number of defect occurrences, which is likely to be somewhat larger than the number of CIDs in Coverity Connect.

When you run the analysis, the console output looks something like the following:

```
[STATUS] Running SpotBugs:
Scanning archives (2 / 2)
2 analysis passes to perform
Pass 1: Analyzing classes (2862 / 2862) - 100% complete
Pass 2: Analyzing classes (1793 / 1793) - 100% complete
Done with analysis
```

An error will occur if you combine `--disable-fb` with `--enable-fb`.

After running the analysis, you continue to run `cov-commit-defects` to commit the analysis results to Coverity Connect. Because the SpotBugs bugs are Static Java defects, Coverity Connect displays them in a Static Java stream along with the results of the Coverity Analysis for Java checkers.

Coverity Connect lists defects found by SpotBugs according to the bug categories [🔗](#) that come from SpotBugs, such as "SpotBugs: Correctness."

#### `--fb-dont-exclude-overlap`

[SpotBugs analysis option for Java. *Not recommended for production use*] Disables the default exclusion of certain SpotBugs checkers. By default, `cov-analyze` prevents SpotBugs analysis from reporting defects that are best found using Coverity checkers. In other words, Coverity checkers are better at finding some portion of the defects SpotBugs can report because the defects reported by the Coverity checkers generally have lower false positive (FP) rates and consist of more true positives. This "overlap" with SpotBugs is excluded by default when `cov-analyze` invokes SpotBugs analysis.

After using this option to disable the default exclusions, you can then customize the exclusions with `--fb-exclude` option and a filter file. You can base your file filter on `exclude.overlap.xml` filter file.

`--fb-exclude concurrency | <filter_file_name>`

[SpotBugs analysis option for Java] Specifies a set of SpotBugs bug patterns to exclude when you run an analysis. By default, the analysis excludes SpotBugs results that overlap with non-deprecated Coverity Analysis for Java checkers because the latter return more true positive defects, fewer false positive defects, or both in cases where such overlap occurs. You can use `--fb-exclude` to exclude additional defects from the analysis results.

 **Note**

If you exclude a defect, it will not appear in the results even if the defect has been included through `--fb-include`.

This option accepts one or more of the following values:

- `concurrency`

Exclude concurrency-related SpotBugs results. For details, see `<install_dir>/spotbugs-ext/config/exclude.concurrency.xml`.

- `<filter_file_name>`

Uses a SpotBugs filter file to specify the SpotBugs bug patterns or bug categories that you want to exclude. For example, to produce results equivalent to `--disable FB.DE_MIGHT_DROP --disable FB.DE_MIGHT_IGNORE`, you can specify the following in your filter file:

```
<SpotBugsFilter>
 <Match>
 <Bug pattern="DE_MIGHT_DROP,DE_MIGHT_IGNORE" />
 </Match>
</SpotBugsFilter>
```

To create a filter file, see the SpotBugs documentation at <http://spotbugs.readthedocs.io/>.

Note that you can use filter files not only to exclude or include individual bug patterns, but also to exclude or include categories of SpotBugs bugs. For example, to exclude medium-priority and low-priority bugs in the SpotBugs `MT_CORRECTNESS` (Multi-threaded correctness) category, you can use the following:

```
<SpotBugsFilter>
 <Match>
 <Bug category="MT_CORRECTNESS" />
 <Or>
 <Priority value="2" />
 <Priority value="3" />
 </Or>
 </Match>
```

```
</SpotBugsFilter>
```

You can specify filter files using relative or absolute paths. If you are using Cygwin, specify Windows native paths, not Cygwin paths.

You can specify this option multiple times on the command line. The order of the exclusions does not matter.

You can also use the `--fb-include` option. If a defect is excluded, it will not appear in the results even if the defect has been included through `--fb-include`.

`--fb-include high-priority | medium-priority | low-priority | <filter_file_name>`

[SpotBugs analysis option for Java] Specifies a set of SpotBugs bug patterns to run on your code. You can use this option to find defects based on SpotBugs priority categories, or you can use a filter file for this purpose.

This option accepts only a single value. For example:

- `--fb-include low-priority`

Returns the most SpotBugs results.

- `--fb-include medium-priority`

Returns only the defects that SpotBugs identifies as medium-priority or high-priority bugs.

Note that `cov-analyze` detects a significant subset of these defects by default (for details, see `<install_dir_sa>/spotbugs-ext/config/include.coverity-default.xml`).

- `--fb-include high-priority`

Returns only the defects that SpotBugs identifies as high-priority bugs.

- `--fb-include [<my_inclusions.xml>]`

Uses a SpotBugs filter file to specify the SpotBugs bug patterns or bug categories that you want to include. For guidance, see the `--fb-exclude` option for filter files.

If the option is not specified, the default file `<install_dir>/spotbugs-ext/config/include.coverity-default.xml` is used.

If `<install_dir>/spotbugs-ext/config/include.<arg>.xml` exists as a file, it is used as the filter file and it overrides the default file settings.

You can use this option at most once on the command line.

SpotBugs rates bugs based on their priority (high, medium, or low). These priorities are similar to the Coverity defect impacts described in each Coverity Integrity Report. In SpotBugs, high-priority defects are most likely to be actionable, while low-priority defects are least likely to require action. Note that Coverity Integrity Report rates bugs in the following SpotBugs categories as medium-impact defects: Correctness, Multithreaded correctness, and Security. All other categories of bugs are rated as low-impact defects.

You can also use the `--fb-exclude` or `--disable` option. If a defect is excluded or disabled, it will not appear in the results even if the defect has been included through `--fb-include`.

**--fb-max-mem**

[SpotBugs analysis option for Java] Sets the JVM heap size of the VM that is running SpotBugs. This option is similar to `--max-mem` in that it takes an integral value of megabytes, but differs in that the action to take if SpotBugs execution runs out of memory is to provide a larger value. If the option is not specified, the default value is 1024.

## Kotlin Detekt options

**--disable-detekt**

Disables Detekt analysis, which is otherwise enabled by default.

An error will occur if you combine `--enable-detekt` with `--disable-detekt`

**--enable-detekt**

Enables Detekt analysis (version 1.0.1) of captured Kotlin source code (see the `DETEKT.*` checker in the *Coverity 2020.03 Checker Reference Guide* for details). The Detekt analysis is enabled by default.

Coverity Analysis provides a default configuration that can be found at `<install_dir>/config/detekt/coverity-default-detekt-config.yml`. However, you can apply your own custom configuration instead by using the `--detekt-config-file *.yml` option, where `*.yml` specifies your configuration (see <https://arturbosch.github.io/detekt/configurations.html> for information about `*.yml` file configuration). If you do not use the option, the analysis will run the default configuration file and ignore any `*.yml` files in your source tree.

An error will occur if you combine `--enable-detekt` with `--disable-detekt`

**--detekt-config-file**

Identifies a custom configuration to use for the Detekt analysis in place of that defined in the default Coverity configuration file (see <https://arturbosch.github.io/detekt/configurations.html> for information about `*.yml` file configuration).

**--enable-formatting-ruleset**

Enables the Detekt formatting rule set. For more information about the Detekt rule sets, see <https://arturbosch.github.io/detekt/index.html>.

## Web and mobile application security options

**--add-password-regex <regular\_expression>**

[Web and mobile application security option] Treats field and method parameter names that match the specified regular expression as a password source. You can specify this option multiple times. Note that if you use the `--add-password-regex` and `--replace-password-regex`, the default regular expression will be replaced, then extended.

This option affects analysis by the `WEAK_PASSWORD_HASH` and `SENSITIVE_DATA_LEAK` checkers. See also, `--replace-password-regex`.

`--allow-jsp-include-param-blacklist`

[Java web application security option] Treats any servlet request parameters that are set through a `<jsp:include>` tag as untainted. This option reduces false positives when a servlet request parameter that is used from an included JSP file never contains tainted data but increases the risk of false negatives in cases where the parameter can be tainted.

This setting changes the default behavior of the XSS checker, a web and mobile applications security checker.

`--android-security`

Enables the checkers used for Android application security analysis. Use these checkers only if you need them because the security analysis adds non-trivial time and memory requirements to the overall analysis.

See "Running mobile application security analysis" in *Coverity Analysis 2020.12 User and Administrator Guide*. [↗](#) for information about using this option.

See also, `--disable-android-security`.

`--directive-file <JSON_file>`

[Security option] Takes a path to a JSON file with a number of user configuration directives, including Web and Android application security directives.

To create this file, see the appendix, "Security Configuration File Reference," in *Coverity 2020.12 Checker Reference* [↗](#)).

Use this option instead of `--webapp-security-config`.

`--disable-webapp-security`

[Web application security option] This option has been deprecated. Disables the Web application security checkers. Note that these checkers are disabled by default.

See `--webapp-security`.

`--disable-webapp-security-preview`

[Web application security option] This option has been deprecated. If you use this option, a warning will be displayed, but no other action will be taken.

`--distrust-all`

[Security option] This option is equivalent to setting all the `--distrust-*` options. It applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 2020.12 Checker Reference* [↗](#).

This option cannot be used with `--trust-all`.

`--distrust-mobile-other-app`

[Mobile application security option] Specifies the default behavior of the analysis, which is to treat data as tainted when it is received from any mobile application that does not require a permission to communicate with the current application component.

This option cannot be used with `--trust-mobile-other-app`.

**--distrust-mobile-other-privileged-app**

[Mobile application security option] Treats data as though it is tainted when it is received from any mobile application that requires a permission to communicate with the current application component. Such data is otherwise trusted by default.

This option cannot be used with `--trust-mobile-other-privileged-app`.

**--distrust-mobile-same-app**

[Mobile application security option] Treats data received from the same mobile application as though it is tainted. Such data is otherwise trusted by default.

This option cannot be used with `--trust-mobile-same-app`.

**--distrust-mobile-user-input**

[Mobile application security option] Specifies the default behavior of the analysis, which is to treat data obtained from user input as though it is tainted.

This option cannot be used with `--trust-mobile-user-input`.

**--distrust-command-line**

[Web application security option] Treats command line arguments as though they are tainted. Such data is otherwise trusted by default. For details, see the *Coverity 2020.12 Checker Reference* [🔗](#).

See also, `--trust-command-line`.

**--distrust-console**

[Web application security option] Treats data obtained from a console (for example, reading from `System.in`) as though it is tainted. Such data is otherwise trusted by default. This option applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 2020.12 Checker Reference* [🔗](#).

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following example produces an issue report:

```
public class ConsoleInj {
 public void testInjection(Statement stmt) throws Exception {
 BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
 String query = reader.readLine();
 stmt.executeQuery(query);
 }
}
```

This option cannot be used with `--trust-console`.

**--distrust-cookie**

[Web application security option] Specifies the default behavior of the analysis, which is to distrust data from HTTP cookies and treat it as though it is tainted. This option applies to all the checkers of type Security (Tainted dataflow checker). See the *Coverity 2020.12 Checker Reference* [🔗](#) for details.

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following example produces an issue report:

```
class SqlInjFromCookie extends HttpServlet {
 Statement sql_stmt;
 public void doPost(HttpServletRequest req, HttpServletResponse
resp) {
 try {
 sql_stmt.executeQuery(req.getCookies()[0].getValue());
 } catch(Exception e) {
 // ...
 }
 }
}
```

This option cannot be used with `--trust-cookie`

#### `--distrust-database`

Treats data obtained from a database (for example, SQL query results and Hibernate objects) as though it is tainted. Such data is otherwise trusted by default. This option applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 2020.12 Checker Reference* [↗](#).

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following example produces an issue report:

```
public class DatabaseInj {
 public void testInjection(int columnIndex, Statement stmt) throws
Exception {
 ResultSet rs = stmt.executeQuery("SELECT * FROM *");
 String query = "SELECT * FROM " + rs.getString(columnIndex);
 stmt.executeQuery(query);
 }
}
```

This option cannot be used with `--trust-database`.

#### `--distrust-environment`

[Web application security option] Treats data that the checker identifies as environment variables as though it is tainted. Such data is otherwise trusted by default. This option applies to all the checkers in the group Security (Tainted data checker). For details, see the *Coverity 2020.12 Checker Reference* [↗](#).

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following example produces an issue report:

```
public class EnvironmentInj {
 public void testInjection(Statement stmt, String getVar) throws Exception
 {
 String envVar = System.getenv(getVar);
 String query = "SELECT * FROM " + envVar;
 stmt.executeQuery(query);
 }
}
```

This option cannot be used with `--trust-environment`.

#### `--distrust-filesystem`

[Web application security option] Treats data obtained from a file system as though it is tainted. Such data is otherwise trusted by default. This option applies to all the checkers of type Security (Tainted dataflow checker). For details, see the *Coverity 2020.12 Checker Reference* [↗](#).

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

`--trust-filesystem`.

The following example produces an issue report:

```
public class FilesysInj {
 public void testRead(FileInputStream fis) throws Exception {
 byte[] b = new byte[50];
 fis.read(b);
 stmt.executeQuery("SELECT * FROM " + new String(b));
 }
}
```

This option cannot be used with `--trust-filesystem`.

#### `--distrust-http`

[Web application security option] Specifies the default behavior of the analysis, which is to treat Web input (for example, GET and POST parameters) as though it is tainted. This option applies to all the checkers of type Security (Tainted dataflow checker). For details, see the *Coverity 2020.12 Checker Reference* [↗](#).

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following Java example produces an issue report:

```
class ServletInj extends HttpServlet {
 Statement sql_stmt;
 public void doPost(HttpServletRequest req, HttpServletResponse resp)
 {
```

```
 try {
 sql_stmt.executeQuery(req.getParameter("x"));
 } catch(Exception e) {
 // ...
 }
 }
}
```

This option cannot be used with `--trust-http`.

#### `--distrust-http-header`

[Web application security option] Specifies the default behavior of the analysis, which is to distrust data from HTTP headers as though it is tainted. This option applies to all the checkers of type *Security (Tainted dataflow checker)*. For details, see the *Coverity 2020.12 Checker Reference* [🔗](#).

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following example produces an issue report:

```
class HttpHeadersInj extends HttpServlet {
 Statement sql_stmt;
 public void doPost(HttpServletRequest req, HttpServletResponse resp)
 {
 try {
 sql_stmt.executeQuery(req.getHeader("user-agent"));
 } catch(Exception e) {
 // ...
 }
 }
}
```

This option cannot be used with `--trust-http-header`.

#### `--distrust-js-client-cookie`

Treats data from `document.cookie` as though it is tainted. The default is to trust this data.

This option cannot be used with `--trust-js-client-cookie`.

#### `--distrust-js-client-external`

Treats response data from the response to `XMLHttpRequest` and similar requests as though it is tainted. This is the default behavior for this option. See also, `--distrust-js-client-http-header`.

This option cannot be used with `--trust-js-client-external`.

#### `--distrust-js-client-html-element`

Treats data from user input on HTML elements such as `textarea` and `input` elements as though it is tainted. The default is to trust this data.

This option cannot be used with `--trust-js-client-html-element`.

**--distrust-js-client-http-referer**

Treats data from the `referer` HTTP header (from `document.referrer`) as though it is tainted. This is the default behavior.

This option cannot be used with `--trust-js-client-http-referer`.

**--distrust-js-client-http-header**

Treats data as tainted when it is from the HTTP response header of the response to `XMLHttpRequest` or to a similar request. This data is trusted by default. See also, `--distrust-js-client-external`.

This option cannot be used with `--trust-js-client-http-header`.

**--distrust-js-client-other-origin**

Treats data as tainted when it is from content in another `frame` or from another `origin`, for example, from `window.name`. This is the default behavior.

This option cannot be used with `--trust-js-client-other-origin`.

**--distrust-js-client-url-query-or-fragment**

Treats data as tainted when it is from the `query` or `fragment` part of the URL, for example, `location.hash` or `location.query`. This is the default behavior.

This option cannot be used with `--trust-js-client-url-query-or-fragment`.

**--distrust-network**

[Web application security option] Specifies the default behavior of the analysis, which is to treat data obtained from a network connection (for example, a TCP socket or HTTP connection) as though it is tainted. This option applies to all the checkers of type `Security` (Tainted dataflow checker). For details, see the *Coverity 2020.12 Checker Reference* [🔗](#).

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following example produces an issue report:

```
class NetworkInj {
 public void func(Socket s, Statement stmt) throws SQLException,
IOException {
 InputStream is = s.getInputStream();
 InputStreamReader isr = new InputStreamReader(is);
 BufferedReader br = new BufferedReader(isr);
 String query = br.readLine();

 query = "SELECT * FROM " + query;
 stmt.executeQuery(query);
 }
}
```

This option cannot be used with `--trust-network`.

**--distrust-rpc**

[Web application security option] Specifies the default behavior of the analysis, which is to distrust data obtained from a Remote Procedure Call (RPC) as though it is tainted. This option applies to all the checkers in the group Security (Tainted data checker). For details, see the *Coverity 2020.12 Checker Reference* [🔗](#).

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following example, which uses an Enterprise Java Bean (EJB), produces an issue report:

```
@Remote(RemoteInterface.class)
public class TestEJB implements RemoteInterface {
 Statement stmt;
 public void testWrite(String taint) {
 ResultSet rs = stmt.executeQuery("SELECT * FROM *");
 String query = "SELECT * FROM " + rs.getString(columnIndex);
 stmt.executeQuery(query);
 }
}
```

This option cannot be used with `--trust-rpc`.

**--distrust-servlet**

[Deprecated Web application security option] This option has been deprecated as of version 7.7.0 and will be removed from a future release. Use `--distrust-http`, instead.

This option cannot be used with `--trust-servlet`.

**--distrust-system-properties**

[Web application security option] Treats system properties (those obtained from `System.getProperty()`) as though they are tainted. Such properties are otherwise trusted by default. This option applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 2020.12 Checker Reference* [🔗](#).

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following example produces an issue report:

```
public class SystemPropertiesInj {
 public void testInjection(Statement stmt, String p) throws Exception {
 stmt.executeQuery("SELECT * FROM " + System.getProperty(p));
 }
}
```

This option cannot be used with `--trust-system-properties`.

**--framework-analyzer-timeout**

[Web application security option] Increase the timeout (specified in minutes) for the framework analyzer. The default value is 60 (60 minutes).

Use this option if the framework analyzer takes too long and is killed. Note that the need to use this option suggests that the hardware in use is most likely overloaded and not powerful enough for the analysis.

`--not-tainted-field <fully_qualified_field_name>`

[Web application security option] The value `<fully_qualified_field_name>` is a Perl regular expression describing a fully qualified field name. Any matching fields will be asserted to be untainted. Additional defects may be reported by the `TAINT_ASSERT` checker, but reported issues involving unsafe uses of the value will be suppressed in the Web application security checkers.

The option can be specified multiple times on a single command line.

See [Adding Assertions that Fields are Tainted or Not Tainted](#) for details.

`--replace-password-regex <regular_expression>`

[Web application security option] Replaces the default regular expression that the checker uses to infer passwords. You can specify this option only once. Note that if you use the `--add-password-regex` and `--replace-password-regex`, the default regular expression will be replaced, then extended.

This option affects analysis by the `WEAK_PASSWORD_HASH` and `SENSITIVE_DATA_LEAK` checkers. See also, `--add-password-regex`.

`--report-null-field-address`

When you specify this option, the analysis considers `&p->field` as dereferencing `"p"`. Specifying this option would cause the "check NULL dereferencing" checkers (for example `FORWARD_NULL`, `NULL_RETURNS`, etc) to report more defects. While it is undefined behavior to form `p->field` when `p` is null, in practice `&p->field` just adds a constant to the value of `"p"` without performing a dereference. Some code relies on this behavior to delay the null check on `"p"`, so by default, this is not reported as a defect. See also `--field-offset-escape`

`--skip-android-app-sanity-check`

[Android application security option] Suppresses the warning message that normally appears if Android application security checkers are enabled with the `--android-security` option, but no Android application was captured.

The check, which this option overrides, is designed to catch the case where someone intended to run Android application security checkers but forgot to capture the Android application using `cov-build` filesystem capture, for example, with the `--fs-capture-search` or `--fs-capture-list` options to `cov-build`.

`--skip-webapp-sanity-check`

[Java-only Web application security option] Suppresses the warning message that normally appears if any Web application security checkers are enabled but `cov-emit-java --webapp-archive` was not used to emit the Web application (web-app) archive or directory.

The check, which this option overrides, is designed to catch the case where someone intended to run Web application security checkers but forgot to emit the WAR file. It is technically possible, but highly unlikely, for Java classes to contain an entire Web application (without any JSPs or framework configuration), in which case there would be no need for a WAR file.

For additional details, see `--webapp-security` and `--skip-war-sanity-check`.

`--tainted-field <fully_qualified_field_name>`

[Web application security option] Takes a Perl-style regular expression that describes a fully qualified field name. Any matching fields will be asserted to be tainted. Additional defects may be reported by the Web application security checkers, if any of the specified fields are used in an unsafe manner. The option can be specified multiple times on a single command line. As an example, passing the command line option `--tainted-field com.coverity.examples.Table.*` will assert that the fields `title` and `values` are tainted in the following code.

```
Package com.coverity.examples;

class Table {
 String title;
 String value;
 int id;
}
```

See [Adding Assertions that Fields are Tainted or Not Tainted](#) for more information.

`--trust-all`

[Security option] This option is equivalent to providing all the `--trust-*` options. This option applies to all the checkers in the group Security (Tainted data checker). For details, see the *Coverity 2020.12 Checker Reference*.

This option cannot be used with `--distrust-all`.

`--trust-mobile-other-app`

[Mobile application security option] Trusts data received from any mobile application when it does not require a permission to communicate with the current application component. Such data is otherwise distrusted by default.

This option cannot be used with `--distrust-mobile-other-app`.

`--trust-mobile-other-privileged-app`

[Mobile application security option] Specifies the default behavior of the analysis, which is to trust data received from any mobile application that requires a permission to communicate with the current application component.

This option cannot be used with `--distrust-mobile-other-privileged-app`.

`--trust-mobile-same-app`

[Mobile application security option] Specifies the default behavior of the analysis, which is to trust data received from the same mobile application.

This option cannot be used with `--distrust-mobile-same-app`.

`--trust-mobile-user-input`

[Mobile application security option] The analysis treats data obtained from user input as though it is not tainted. Such data is otherwise distrusted by default.

This option cannot be used with `--distrust-mobile-user-input`.

`--trust-command-line`

[Web application security option] Specifies the default behavior of the analysis, which is to treat command line arguments as though they are not tainted. For details, see the *Coverity 2020.12 Checker Reference* [↗](#).

See also, `--distrust-command-line`.

`--trust-console`

[Web application security option] Specifies the default behavior of the analysis, which is to treat data obtained from a console (for example, reading from `System.in`) as though it is not tainted. This option applies to all the checkers in the group Security (Tainted data checker). For details, see the *Coverity 2020.12 Checker Reference* [↗](#).

This option cannot be used with `--distrust-console`.

`--trust-cookie`

[Web application security option] Treats data that is obtained from an HTTP cookie as though it is not tainted. Such data is otherwise distrusted by default. This option applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 2020.12 Checker Reference* [↗](#).

This option cannot be used with `--distrust-cookie`

`--trust-database`

[Web application security option] Specifies the default behavior of the analysis, which is to treat data obtained from a database (for example, SQL query results and Hibernate objects) as though it is not tainted. This option applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 2020.12 Checker Reference* [↗](#).

This option cannot be used with `--distrust-database`.

`--trust-environment`

[Web application security option] Specifies the default behavior of the analysis, which is to treat data from environment variables as though it is not tainted. This option applies to all the checkers in the group Security (Tainted data checker). For details, see the *Coverity 2020.12 Checker Reference* [↗](#).

Note that the analysis trusts data from environment variables by default.

This option cannot be used with `--distrust-environment`.

`--trust-filesystem`

[Web application security option] Specifies the default behavior of the analysis, which is to treat data obtained from a file system as though it is not tainted. This option applies to all the checkers in the group Security (Tainted data checker). For details, see the *Coverity 2020.12 Checker Reference* [↗](#).

Note that the analysis trusts data from filesystem sources by default.

This option cannot be used with `--distrust-filesystem`.

**--trust-http**

[Web application security option] Treats Web input (for example, `GET` and `POST` parameters) as though it is not tainted. Web input is otherwise treated as tainted by default. This option applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 2020.12 Checker Reference* [🔗](#).

This option cannot be used with `--distrust-http`.

**--trust-http-header**

[Web application security option] Treats data that is obtained from an HTTP header as though it is not tainted. Such data is otherwise distrusted by default. This option applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 2020.12 Checker Reference* [🔗](#).

This option cannot be used with `--distrust-http-header`.

**--trust-js-client-cookie**

Trusts data from `document.cookie`.

This option cannot be used with `--distrust-js-client-cookie`. This is the default behavior.

**--trust-js-client-external**

Trusts response data from the response to `XMLHttpRequest` and similar requests. The default is to distrust this data. See also, `--trust-js-client-http-header`.

This option cannot be used with `--distrust-js-client-external`.

**--trust-js-client-html-element**

Trusts data from user input on HTML elements such as `textarea` and `input` elements. This is the default behavior.

This option cannot be used with `--distrust-js-client-html-element`.

**--trust-js-client-http-referer**

Trusts data from the `referer` HTTP header (from `document.referrer`). The default is to distrust this data.

This option cannot be used with `--distrust-js-client-http-referer`.

**--trust-js-client-http-header**

Trusts data from the HTTP response header of the response to `XMLHttpRequest` and similar requests. This is the default behavior. See also, `--trust-js-client-external`

This option cannot be used with `--distrust-js-client-http-header`.

**--trust-js-client-other-origin**

Trusts data from content in another `frame` or from another `origin`, for example, from `window.name`. The default is to distrust this data.

This option cannot be used with `--distrust-js-client-other-origin`.

**--trust-js-client-url-query-or-fragment**

Trusts data from the query or fragment part of the URL, for example, `location.hash` or `location.query`. The default is to distrust this data.

This option cannot be used with `--distrust-js-client-url-query-or-fragment`.

**--trust-network**

[Web application security option] Treats data obtained from a network connection (for example, a TCP socket or HTTP connection) as though it is not tainted. Such data is otherwise distrusted by default. This option applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 2020.12 Checker Reference* [🔗](#).

This option cannot be used with `--distrust-network`.

**--trust-rpc**

[Web application security option] Treats data obtained from a Remote Procedure Call (RPC) as though it is not tainted. Such data is otherwise distrusted by default. This option applies to all the checkers in the group Security (Tainted data checker). For details, see the *Coverity 2020.12 Checker Reference* [🔗](#).

This option cannot be used with `--distrust-rpc`.

**--trust-servlet**

[Web application security option] This option has been deprecated as of version 7.7.0 and will be removed from a future release. Use with `--trust-http`, instead.

This option cannot be used with `--distrust-servlet`.

**--trust-system-properties**

[Web application security option] Specifies the default behavior of the analysis, which is to treat data obtained from system properties (for example, `System.getProperty()`) as though it is not tainted. This option applies to all the checkers in the group Security (Tainted data checker). For details, see the *Coverity 2020.12 Checker Reference* [🔗](#).

This option cannot be used with `--distrust-system-properties`.

**--webapp-security**

[Web application security option] Enables the checkers that are used for Web application security analysis.

Java Prerequisite: Prior use of the `--webapp-archive` option to `cov-emit-java` for the WAR file. See *Running a Security Analysis on a Java Web Application* [🔗](#) for details.

.NET Recommendation: Coverity highly recommends ensuring that `cov-build` captured your Web application template and configuration files. See *Running a Security Analysis on a Java Web Application* [🔗](#) for details.

 **Note**

Use these checkers only if you need them because the security analysis adds non-trivial time and memory requirements to the overall analysis.

`--webapp-security-aggressiveness-level <low|medium|high>`

[Web application security option] Tunes the aggressiveness of assumptions that the analysis makes to find potential security vulnerabilities (security defects). Higher levels report more defects, but the analysis time increases and memory usage is likely to increase. Higher levels also increase the likelihood that any given defect is a false positive. Values for level are low, medium, or high. Default is low.

This option can assist security auditors who need to see more defects than developers might need to see.

When analyzing code that uses unsupported Web application frameworks, medium or high aggressiveness levels can be more useful than the default.

`--webapp-security-config <JSON_file>`

[Java Web application security option] Alias for `--directive-file`.

`--webapp-security-preview`

[Web application security option] Deprecated. Does the same thing as `--webapp-security`.

See also `--webapp-security`.

## Test Advisor options

`--compute-test-priority`

[Test Advisor option] For Test Advisor test prioritization, this option creates a prioritized list of test (outputs to the location specified in `--test-priority-output`) using the specified test priority policy (specified in `--test-priority-policy`).

For usage information, see *Test Advisor 2020.12 User and Administrator Guide*. [↗](#)

This option works with C/C++, C#, Visual Basic, and Java.

`--disable-test-metrics`

Disables Test Advisor test prioritization metric data. If specified, this option takes precedence over `--enable-test-metrics`.

`--test-advisor`

Enables Test Advisor analysis. The `--strip-path` option is required for Test Advisor.

`--test-advisor-policy <policy_file>`

Specifies that the file is read as the policy for the Test Advisor run. The file must be a JSON file. For more information about construction policy files, see "Creating Test Advisor policies" in the *Test Advisor 2020.12 User and Administrator Guide*.

`--test-advisor-eval-output <eval_output_dir>`

Causes the filter evaluation output to be written to the specified directory when Test Advisor runs. The specified directory is created if it does not already exist. All source files that are analyzed by Test Advisor are copied to this directory, and lines within each file that are excluded from analysis by the test policy are annotated with the filter that caused the exclusion.

`--test-advisor-eval-rule <rule_number>`

Causes the filter evaluation to be written only for the specified rule of the policy. Rules are numbered starting from 1 in order of their appearance in the policy file. This option has no effect unless `--test-advisor-eval-output` option is also used.

`--test-advisor-verbose`

Causes the output of verbose messages during the execution Test Advisor.

`--test-priority-eval-output <eval_output_file>`

For Test Advisor test prioritization, this option specifies the path and name for a file to which details about the analysis performed by test prioritization will be written. This option can be used to help debug the test prioritization policy.

The output file is in CSV format, where each line contains a record with the following information:

- A function, identified by its mangled name and filename
- A test, identified by its suite name and test name
- A rule from the policy file, identified by its name

Each record represents a function which passed the filters for the given rule, and a test which covers the function. If more than one test covers the function, that function shall have multiple records in the eval output, one for each test.

The `filtered_function_count` rule property for a given `(test, rule)` pair is thus the number of records which have that `(test, rule)` pair in the eval output. The functions listed in those records are the functions which contributed to the `filtered_function_count`. The eval output provides a way to validate that the `filtered_function_count` is correct. If a score in the test prioritization output is not as expected, the functions contributing to the rule's `filtered_function_count` can be examined to determine why the discrepancy exists.

The first line of the eval output contains a header which indicates the format of the subsequent lines, and does not contain a record itself.

Due to the volume of data involved, the output file size may be large. The `--tu-pattern` option of `cov-analyze` can be used to restrict analysis to specific translation units in order to limit the size of the output.

`--test-priority-output <output_file>`

For Test Advisor test prioritization, this option specifies the path and name for the file to which the test prioritization output will be written.

For information about the output format, see "Test scoring policy language" in the *Test Advisor 2020.12 User and Administrator Guide*.

`--test-priority-policy <policy_file>`

For Test Advisor test prioritization, this option specifies the path and name of the policy file which is used by test prioritization to determine how tests are scored. The file must be a JSON file.

For information about the policy file format, see "Test scoring policy language" in the *Test Advisor 2020.12 User and Administrator Guide*.

## Shared options

- `--config <coverity_config.xml> , -c <coverity_config.xml>`  
Uses the specified configuration file instead of the default configuration file located at `<install_dir_sa>/config/coverity_config.xml`.
- `--debug, -g`  
Turn on basic debugging output.
- `--ident`  
Displays the version of Coverity Analysis and build number.
- `--info`  
Displays certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.
- `--redirect stdout|stderr,<filename> -rd stdout|stderr,<filename>`  
Redirect either stdout or stderr to `<filename>`.
- `--tmpdir <tmp>, -t <tmp>`  
Specifies the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.
- `--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>`  
Sets the detail level of command messages. Higher is more verbose (more messages). Defaults to 1. Use `--verbose 0` to disable progress bars.

## Exit codes

- 0: The analysis was successful. Results should be considered usable and are ready to be committed with `cov-commit-defects`.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

Although the console output can provide diagnostics and warnings that might help to improve the analysis configuration, or suggest reporting "recoverable errors" to Coverity support, this information is auxiliary to the exit code. Users and scripts should rely on the exit code when determining whether to proceed in consuming the analysis results.

## Examples

Analyze the intermediate directory at `/home/user/apache` using only the DEADCODE checker:

```
> cov-analyze --dir /home/user/apache --disable-default --enable DEADCODE
```

## **See Also**

[cov-build](#)

[cov-commit-defects](#)

[cov-make-library](#)

---

## Name

`cov-blame` Compute Coverity Connect automatic owner assignment based on SCM history.

## Synopsis

```
cov-blame
 --dir <intermediate_directory>
 --preview-report <filename>
 [--scm <scm_type>]
 [--scm-tool <scm_tool_path>]
 [--scm-project-root <scm_root_path>]
 [--scm-tool-arg <scm_tool_arg>]
 [--scm-command-arg <scm_command_arg>]
 [--no-triage-filters]
 [--owner-assignment-rules <RULE1>,<RULE2>...,<RULEn>]
 [--stop-after <limit>]
 [OPTIONS]
```

## Description

The `cov-blame` command computes the automatic ownership assignments based on SCM (source code management) history and owner assignment rules. The SCM history consists of when, where, and by whom the code was changed and the ownership rules derive the owner of an issue based on the SCM history.

This command requires that source files remain in their usual locations in the checked-out source tree. If the files are copied to a new location after checkout, the SCM query will not work.

There are two main use cases for this command:

### 1. `cov-blame` is automatically called for owner assignment as part of the commit process.

In this case, `cov-commit-defects` automatically calls `cov-blame` based on owner assignment rules provided to the Coverity Connect UI. `cov-commit-defects` invokes `cov-blame` to compute the ownership assignments. If the rule assigned to the stream requires SCM data, `cov-blame` first attempts to retrieve SCM data for files related to defects from the intermediate directory. If SCM data has not been imported to the intermediate directory, the command can directly query the SCM for the relevant history data using the `--scm*` options defined in `cov-commit-defects`. The assigned owners (SCM users) are then written back into the intermediate directory. `cov-commit-defects` then picks up the ownership assignment files and Coverity Connect sets the owner accordingly in the triage pane.

### 2. `cov-blame` is manually invoked to test and compare ownership rules.

You can invoke `cov-blame` to produce a report of owner assignments for defects to help you accomplish the following:

- Verify, before you commit, that the automatic ownership is producing the proper/expected assignments. You can specify one or more owner assignment rules through the `--owner-assignment-rules` option for comparison.

- To compare automatic owner assignment to the owners that are already defined in Coverity Connect. In this way, you can see how successful an owner assignment rule would have been for historical defects that have already been manually assigned in Coverity Connect.

Before you run `cov-blame`, you must have an existing preview report or generate one using `cov-commit-defects --preview-report-v2`.

## Options

`--dir <int_dir>`

Pathname to an intermediate directory that is used to store the results of the build and analysis. This is required.

`--no-triage-filters`

Calculate owner assignment for all defects whose merge key is present in the preview report, regardless of the triage values. By default, owner assignment is only calculated for defects whose current triage values satisfy the following conditions:

- The Owner attribute is unset
- The Classification attribute is Unclassified, Pending, Bug, or Untested.

This option is useful, for example, to calculate owner assignment for comparison with owner assignments that have already been made manually in Coverity Connect to evaluate the accuracy of the owner assignment rules. In this case, you must use the `--no-triage-filters` option.

`--owner-assignment-rules <RULE1>,<RULE2>,...,<RULEn>`

Determines an owner by consulting history from the SCM system. If this option is specified, you must include at least one `<rule>`. Multiple `<rule>` options must be comma-separated. If this option is not specified, then all rules are applied. In the descriptions below, main event refers the event that Coverity Connect first highlights when the user clicks on the defect in the defects list.

The rules are:

- `file` - Queries the SCM for the person that most recently modified the file containing the main event, and that SCM user is the chosen owner.
- `line` - Queries the SCM for the person that most recently modified the particular line of code that has the main event.
- `function` - If there is a function associated with the main event, then `cov-blame` queries the SCM for the person who most recently modified that function. Otherwise, this rule acts the same as the file rule.
- `top_events` - Retrieves all of the lines of code that contain a non-interprocedural defect event in the issue, then returns the person that most recently modified any of those lines.
- `all_events` - Similar to the `top_events` rule, except that it also considers all interprocedural defect events.

- `all_functions` - Combines the `function` and `all_events` rules to query for the person who most recently modified the functions associated with all the defect events. If there are no functions at all, this rule behaves like the `all_files` rule.
- `all_files` - Combines the `file` rule with the `all_events` rule to query for the person who most recently modified the files containing all of the defect events.
- `default_component_owner` - Reports the issue's owner as the designated default owner for the component in Coverity Connect. The output of this rule, unlike all of the other rules, is a Coverity Connect user, and not an SCM user. This rule can yield no assignment if a component does not have a default owner.

`--preview-report <filename>`

Specifies the path and name of the preview report generated by `cov-commit-defects --preview-report-v2`. This option is required.

`--scm <scm_type>`

Specifies the name of the source control management system. For this option to function correctly, your source files must remain in their usual locations in the checked-out source tree. If the files are copied to a different location after checkout, the SCM query will not work.

Possible `scm_type` values:

- Accurev: `accurev`
- Azure DevOps Server (ADS): `ads`  
Windows only.
- ClearCase: `clearcase`
- CVS: `cvs`
- GIT: `git`
- Mercurial: `hg`
- Perforce: `perforce`
- Plastic: `plastic|plastic-distributed`.

Use `plastic` when working in a non- or partially-distributed Plastic configuration. Use `plastic-distributed` when working in a fully-distributed Plastic configuration.

- SVN: `svn`
- Team Foundation Server (TFS): `tfs`  
Windows only.

For usage information for the `--scm` option, see `cov-extract-scm`.

 **Note**

The following commands or setup utilities must be run before `cov-blame` in order to successfully communicate with the SCM server:

- `accurev`:

Login command

- `perforce`

The environment variable `P4PORT` should be set to the value expected by the `p4` tool.

- `tfs` or `ads`:

Windows credentials in Credential Manager to access the TFS or ADS server

`--scm-command-arg <scm_command_arg>`

This option has been deprecated. Instead of using `--scm-command-arg arg1`, use `--scm-param annotate_arg=arg1`. Specifies additional arguments that are passed to the command that retrieves the last modified dates. This option can be specified multiple times.

For usage information for the `--scm` option, see `cov-extract-scm`.

`--scm-param`

Specify extra arguments to be passed to the SCM tool in a context-aware manner. For usage information of the `--scm` option, see `cov-extract-scm`.

`--scm-project-root <scm_root_path>`

Specifies a path that represents the root of the source control repository. This option is only used when specifying `accurev` as the value to `--scm`. When this is used, all file paths that are used to gather information are interpreted as relative to this project-root path.

For usage information for the `--scm` option, see `cov-extract-scm`.

`--scm-tool <scm_tool_path>`

Specifies the path to an executable that interacts with the source control repository. If the executable name is given, it is assumed that it can be found in the `path` environment variable. If it is not provided, the command uses the default tool for the specified `--scm` system.

For usage information for the `--scm` option, see `cov-extract-scm`.

`--scm-tool-arg <scm_tool_arg>`

This option has been deprecated. Instead of using `--scm-tool-arg arg1`, use `--scm-param tool_arg=arg1`. Specifies additional arguments that are passed to the SCM tool, specified in the `--scm-tool` option, that gathers the last modified dates. The arguments are placed before the command and after the tool. This option can be specified multiple times.

For usage information for the `--scm` option, see `cov-extract-scm`.

`--stop-after <limit>`

Stops computing owner assignments after a certain number of defects, as specified in `<limit>`.

This allows you to quickly experiment with rules without waiting for a long time for each defect to be assigned an owner.

## Shared options

`--debug, -g`

Turn on basic debugging output.

`--ident`

Displays the version of Coverity Analysis and build number.

`--info`

Displays certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

`--tmpdir <tmp>, -t <tmp>`

Specifies the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

`--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>`

Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## See Also

`cov-commit-defects`

`cov-import-scm`

`cov-extract-scm`

---

## Name

`cov-build` Intercept all calls to the compiler invoked by the build system and capture source code from the file system.

## Synopsis

C, C++, C#, CUDA, Go, Java, JavaScript, Kotlin, Objective-C, Objective-C++, PHP, Python, Ruby, Scala, Swift, and Visual Basic:

```
cov-build (--dir <intermediate_directory> | --emit-server <emit_servername:port> | --da-broker <broker_servername:port>) [--capture-ignore <program.extension>] [--c-coverage | --cs-coverage | --java-coverage] [--fs-capture-list <file>] [--fs-capture-search <directory>] [--test-capture] [OPTIONS] BUILD_COMMAND | --no-command
```

## Description

The `cov-build` command is the primary tool to capture and emit source code. It performs build capture, where source code is emitted by intercepting all calls to the compiler invoked by the build system. It also performs filesystem capture, where source code is emitted directly from the file system. (For more information about the build capture processes, see [The build](#).)



### Note

Parallel builds for ASP.NET 4 and earlier applications cannot be virtualized directly with the `cov-build` command.



### Caution

When capturing compilations using the fast Scala compiler, you must run the `fsc -shutdown` command first.

In general, the `cov-build` command name and option can prefix the original build command. However, if the `cov-build` command depends on features of the command shell that usually invokes it, such as certain shell variables or non-alphanumeric arguments, you can invoke it using a wrapper script. This preserves the original behavior because the `cov-build` command is again invoked directly by the shell type (on which it depends).

For example, if the normal invocation of a Windows build is:

```
> build.bat Release"C:\Release Build Path\"
```

use `cov-build` as follows:

```
> cov-build --dir <intermediate_directory> <wrapper.bat>
```

where `<wrapper.bat>` is an executable command script that contains the original and unmodified build command.

On Windows, specify both the filename and extension for the build command when using `cov-build`. For example:

```
> cov-build --dir <intermediate_directory> custombuild.cmd
```

Because `cov-build` uses the native Windows API to launch the build command, the appropriate interpreter must be specified with any script that is not directly executable by the operating system. For example, if the normal invocation of a build within Msys or Cygwin is:

```
> build.sh
```

prefix `cov-build` with the name of the shell:

```
> cov-build --dir <intermediate_directory> sh build.sh
```

Similarly, if a Windows command file does not have Read and Execute permissions, or if you want `cov-build` to report the command file's `%ERRORLEVEL%`, explicitly invoke it as a `cmd.exe` command string. For example, to run the Java builder `ant.bat`:

```
> cov-build --dir <intermediate_directory> cmd /c "ant && exit/b"
```

 **Note**

If you run `cov-build` more than once, only the last build's metrics are saved.

 **Note**

If you change the set of compilation options used by your build process, delete the `<intermediate_directory>` and capture a full build from scratch. Otherwise, the translation units captured using the old options will remain in the emit. The new translation units will not replace the old translation units due to the changed compilation options.

As a final step, this command invokes `cov-security-da`, which runs a dynamic analysis in order to perform a security assessment.

 **Note**

Coverity Security Dynamic Analysis for C# and Visual Basic requires a Windows 64-bit or Linux 64-bit system that supports .NET Core 3.1.

## C, C++, C#, and Visual Basic build capture

For C, C++, C#, and Visual Basic source code, after the compiler calls have been intercepted, `cov-build` captures the compiler command line and other information and invokes `cov-translate` to translate the native command line into one appropriate for `cov-emit`, `cov-emit-cs`, or `cov-emit-vb`, which in turn parses and emits the code. Before running `cov-build`, you need to configure your compiler (such as `gcc` or `msvc`) by using the `cov-configure` command.

For C# and Visual Basic only, if an ASP.NET 4 and earlier Web application is detected, `cov-build` will attempt to run `Aspnet_compiler.exe` on the Web application. The output of `Aspnet_compiler.exe` is required by the C# security checkers.

 **Note**

C# and Visual Basic build capture is supported on Windows and Linux platforms. C# Unity Projects using the Roslyn Compiler are supported on MacOSX.

The `cov-build` command creates a log file called `build-log.txt` in the intermediate directory. This log file shows each command that is intercepted, including compiler invocations. For each compiler invocation, the call to `cov-translate` and `cov-emit` are shown, along with any parsing errors and any other compilation errors.

The `cov-build` command intercepts compiler invocations for single-threaded builds and parallel builds on a single machine. Distributed builds, which use remote procedure calls or some other protocol to invoke builds or compilations on several machines, cannot be virtualized directly with `cov-build`. Contact Coverity support for assistance.

On UNIX, `cov-translate` is invoked before each native compiler invocation. On Windows, `cov-translate` is invoked after each native compiler invocation.

The `cov-build` command expects the configured C and C++ compilers to be those used by the build. The analysis might be skewed if different compiler versions are used. So, consider values that the build scripts and makefiles might define for `$PATH`, `$CC`, and so on. If compiler pathnames are unknown, configure a template.

See the section called “Build and filesystem capture examples”.

## Go build capture

Go source files are emitted by performing a native build and intercepting calls to the Go command line tool (that is, `go <command> [arguments]`) when the command is `test` or `build`.

### Note

If the Go command line tool is invoked by way of some other Go application, the invocation of the Go command line tool is not captured.

## Java, Kotlin, and Scala build capture

The `cov-build` command works somewhat differently for the Java, Kotlin, and Scala compilers (than for the C, C++, and C# compilers). In addition to gathering and compiling source files, the `cov-build` command for Java, Kotlin, and Scala also collect compiled files and any JAR files or class files in the classpath.

For Java, Kotlin, and Scala, the command also runs the compilers in debug mode so that Coverity Analysis can analyze the compiled code. This automatic behavior is equivalent to running the `javac -g` command prompt or using the `debug="true"` property setting in an Ant compile task.

### Note

You must use supported compilers when using `cov-build` with Java, Kotlin, and Scala. For details, see the *Coverity 2020.12 Installation and Deployment Guide*.

You can use the `--config` option to `cov-configure` and `cov-build` to establish and maintain separate configuration directories for each language.

The `cov-build` command expects the build to use the configured Java, Kotlin, and Scala compilers. Compile commands with different pathnames will not be analyzed because `cov-build` cannot identify

the compiler version. So, consider which compilers the build scripts and tools might invoke. For example, `ant` can refer to `$JAVA_HOME` or a default pathname (not `$PATH`) to find the `java` command. When using `ant` on the Mac OS, you need to set the `$JAVA_HOME`. Otherwise, the Mac OS will select something other than what `cov-configure` will set. If compiler pathnames are unknown, then a template must be configured.

 **Note**

When running 64-bit Coverity Analysis tools against a 32-bit Java SDK, `cov-build` may fail to capture compilations. Use `--instrument` to work around the issue.

See the section called “Build and filesystem capture examples”.

## Java filesystem capture

Once Java file system capture is enabled, the `cov-build` command may be run to capture Java source files into the intermediate directory. To capture these Java source files, use the `cov-build` command along with one or more of the following options: `--no-command`, `--fs-capture-search` and `--fs-capture-list`.

```
> cov-build --dir <intermediate_directory> --no-command --fs-capture-search <source-directory>
```

The `cov-build` command line options above recursively search all files in the specified directory for Java source files (and JSP or Java Android files). It then emits them into the intermediate directory specified using the `--dir` option.

Additional notes:

The `--no-command` option is mandatory when filesystem capture is performed without build capture.

To improve build capture and analysis fidelity, you can look up any missing types in the build log and specify the required library files. Use the `--fs-library-path` option to add any required classes or JAR files to the classpath. These additions will be used for the compilation of Java source files and for the compilation of JSP files by the Jasper engine. All JAR files present in the captured directory are automatically included in the classpath.

Use the `--fs-capture-search-exclude-regex` option to make the `cov-build` command ignore any files that match a specific expression. Use this option to exclude any test code in the source directory from analysis.

For more information on Java filesystem capture, see the “Filesystem capture (for Java)” section in the *Coverity Analysis 2020.12 User and Administrator Guide* [↗](#).

## Filesystem capture for interpreted languages

Filesystem capture emits files directly from the filesystem without needing to see a compilation. The command uses `--fs-capture-search` and/or `--fs-capture-list` to generate a list of files for

JavaScript, JavaServer Pages (JSPs), Python, PHP, or Ruby files that the command captures into the intermediate directory.

See the section called “Build and filesystem capture examples”. For more detailed advice, see “Filesystem capture” in *Coverity Analysis 2020.12 User and Administrator Guide* [↗](#).

## Swift build capture

Swift source files are emitted using build capture by performing a native build and intercepting calls to the Swift compiler. (This workflow is similar to the build capture process for C/C++, C#, Objective-C, and Objective C++).

Swift build capture is only supported on macOS.

You must use one of the `cov-build` filesystem capture options, in order to point to the project root directory, or to emit any important configuration files. (Buildless capture is not supported for Swift.) See the example of running a “combined” capture in the section called “Build and filesystem capture examples”.

## Build and filesystem capture examples

- For compiled languages (build capture), including Java build capture:

```
> cov-build --dir <intermediate_directory> <BUILD_COMMAND>
```

- For scripts and interpreted code (filesystem capture), and Java filesystem capture:

```
> cov-build --dir <intermediate_directory> --no-command \
 --fs-capture-search <path/to/source/code>
```

If you are only performing a filesystem capture and not also performing a build capture, you need to pass the `--no-command` command. For more details, see *Coverity Analysis 2020.12 User and Administrator Guide* [↗](#).

- For a combined source code capture (build and filesystem capture):

```
> cov-build --dir <intermediate_directory> \
 --fs-capture-search <path/to/source/code> <BUILD_COMMAND>
```

Note that the build command must be specified last.

## Options

### `--add-arg <arg>`

Specifies an `--add-arg <arg>` option to the `cov-translate` invocations that are launched by `cov-build`. See the description of this option in the `cov-translate` command documentation.

### `--append-log`

Append a log of the current `cov-build` run to an existing build log file, instead of performing the default behavior, which is to overwrite the log file. For details about the file, see the section called “C, C++, C#, and Visual Basic build capture”.

**--auto-diff**

Compatible with C and C++ builds only. Have `cov-translate` attempt to diff preprocessed files when a compilation fails. See the description of `--auto-diff` in `cov-help cov-translate`.

**--build-description <string>**

[Test Advisor option] Provides an optional description of the build. `<string>` represents the description. The description is used in the build ID. Some examples might include using a tag:

```
cov-build ... --test-capture --build-description linuxbuild-rev1
```

**--build-id <build-id>**

[Test Advisor option] Specifies the build ID to use for the test capture run. You can specify only one of the following options: `--build-id-file`, `--build-id-dir`, or `--build-id`.

The build ID is a string that uniquely identifies a build. Every intermediate directory is associated with a build ID. The build ID string has the following format:

```
<user.specified.string>-<our.unique.md5>
```

The MD5 portion is an MD5 hash which identifies the build. Users can specify an additional string to prepend to this hash, which helps identify the build in a human-readable fashion. This additional string can be specified during the invocation of `cov-build`. For example:

```
cov-build --dir idir --c-coverage gcov --build-description \
"linux-build-tag-1234" make build
```

would identify the build with a build ID of the form:

```
linux-build-tag-1234-<md5-of-this-covbuildinvocation>
```

Generally, the user-specified portion of the build ID should contain information specific to the build that the user is interested in. Examples:

- Build platform - If you build the same codebase on different platforms, this tag can be used to identify the platform used for a particular build.
- Revision control tag - If you build different revisions of a codebase, this tag can be used to identify the tag associated with a particular build. A date may also be used here.

If you want to run `cov-build` multiple times on a single intermediate directory (for example if multiple commands need to be run to generate a build) and you do not want a new build ID generated for each invocation, use `--no-generate-build-id`.

The build ID for a given intermediate directory can be determined using `cov-manage-emit`, for example:

```
cov-manage-emit --dir idir query-build-id
```

**--build-id-dir <dir>**

[Test Advisor option] Identifies a directory that contains `build-id.txt`, a file that specifies a build ID [p. 63]. The `--build-id-dir` option is exactly like `--build-id-file <dir>/build-id.txt`. It is useful when the intermediate directory with the appropriate build ID is available locally, but the `--dir` option is not used.

You can specify only one of the following options: `--build-id-file`, `--build-id-dir`, or `--build-id`.

`--build-id-file <filename>`

[Test Advisor option] Specifies a filename (including the file path to the file) that contains the build ID [p. 63] to use for this run. You can specify only one of the following options: `--build-id-file`, `--build-id-dir`, or `--build-id`.

`--build-id-output <filename>`

[Test Advisor option] Writes the build ID that is generated for this run to the specified filename. This provides an easy way to transfer the build ID to different machines when remotely collecting coverage. For example:

```
cov-build --dir idir --c-coverage gcov --build-description \
"mybuild" --build-id-output write-build-id-here.txt make
```

`--bullseye-dir <dir>`

[Test Advisor option] Compatible with C and C++ builds only. Specifies the location of the Bullseye tools for use with `cov-build`. This option should point to the location of BullseyeCoverage installation directory. For example, `/opt/bullseye`.

This option must not include the `/bin` portion of the directory.

`--bullseye-lib-dir <dir>`

[Test Advisor option] Used in conjunction with `--c-coverage bullseye-small` to specify the directory where the small runtime was built. This has no effect with `--c-coverage bullseye`, and will produce a warning.

`--c-coverage <tool>`

[Test Advisor option] Compatible with C and C++ builds only. Enables C and C++ coverage collection using the specified coverage tool. `<tool>` represents the coverage tool. The options are `gcov`, `bullseye`, `bullseye-small`, and `function`.

Using the `bullseye` or `bullseye-small` options require a valid BullseyeCoverage installation.

- The following example demonstrates command line usage for `gcov`:

```
cov-build --dir idir --c-coverage gcov make build
```

- The following example demonstrates command line usage with Bullseye. With this option, you must also specify the `--bullseye-dir` option:

```
cov-build --dir t1 --c-coverage bullseye --bullseye-dir /opt/bullseye make build
```

- The following example demonstrates using a Bullseye small runtime with Test Advisor with `bullseye-small`. With this option, you must also specify the `--bullseye-dir` and `--bullseye-lib-dir` options:

```
cov-build --dir idir --c-coverage bullseye-small \
--bullseye-dir /path/to/bullseye \
--bullseye-lib-dir <output-dir>/lib make build
```

 **Note**

You must build the Bullseye small runtime before you can use it with `cov-build`. On Windows, the Bullseye small runtime only supports MSVC-based builds (for example, Visual Studio cl). or more information, see *Test Advisor 2020.12 User and Administrator Guide*. [↗](#)

You can use `c-coverage` along with `java-coverage` and/or `cs-coverage`, for example:

```
cov-build --dir idir --c-coverage gcov --java-coverage cobertura make build
```

- The following example demonstrates command line usage for Function Coverage Instrumentation:

```
cov-build --dir t1 --c-coverage function make build
```

**--capture**

Run the specified build command, and capture the actions and translation units in the `<intermediate_directory>` that is created by the `--initialize` option.

Alternatively, a build system can directly invoke `cov-translate`, possibly in several concurrent processes.

For C and C++ analysis, you can run concurrent, distributed builds across multiple machines with the `--capture` option if the `<intermediate_directory>` is located on an NFS partition. Distributed builds are only supported on Linux and Solaris systems.

**--capture-ignore <program-name-with-extension>**

On Windows 2000 only, use this option to specify the base name (including the extension) of a program invoked by the native build that does not exit during the build, such as a service or a daemon. Otherwise, `cov-build` will hang at the end the native build waiting for these programs to terminate. `cov-build` is already aware of NTVDM.EXE and MSPDBSVR.EXE. The program name is case insensitive.

**--chase-symlinks**

Compatible with C and C++ builds only. Follow symbolic links when determining filenames to report.

**--chcmdline-type <type>**

Do not use this option. It has been deprecated. Also, this option has no effect.

**--coverage-instrumentation-error-threshold <percentage>**

[Test Advisor option] The `--coverage-instrumentation-error-threshold` option sets a lower bound for measuring success in instrumenting source code for function coverage. The default threshold is 95 percent. If the instrumentation success rate falls below this threshold, instrumentation is considered unsuccessful, is reported as an error, and `cov-build` returns a non-zero exit code. This option can only be used in combination with the `--c-coverage function` option.

The following example demonstrates the `--coverage-instrumentation-error-threshold` option with a threshold of 60%:

```
cov-build --dir t1 --c-coverage function \
 --coverage-instrumentation-error-threshold 60 make build
```

**--coverity-response-file=<response\_file>**

Specify a "response file" that contains a list of additional command line arguments, such as a list of input files. Each line in the file is treated as one argument, regardless of spaces, quotes, etc. The file is read using the platform default character encoding. The response file cannot contain the build command, either in full or in part.

**--cs-coverage <tool>**

[Test Advisor option] Compatible with C# builds only. Enables C# coverage collection using the specified coverage tool, represented by <tool>. Currently, the only coverage tool option is `opencover`.

Running this command line option will cause Test Advisor to attempt to register the profiler DLL in order to collect coverage. In some situations, this might fail. To work around this, you will need to manually register the profiler DLL and then use the `--cs-no-register-profiler` on the command line to prevent the command from trying to register the profiler DLLs again. See `--cs-no-register-profiler` for instructions.

If you are running from a Cygwin shell you might see an error similar to the following:

```
Unhandled Exception: System.ArgumentException: Item has already been added.
Key in dictionary: 'TMP' Key being added: 'tmp'
```

To avoid this, `unset` the Cygwin versions of the environment variables, for example:

```
% unset tmp
% unset temp
```

You can use `cs-coverage` along with `java-coverage` and `c-coverage`.

**--cs-opencover-arg**

Used in conjunction with `--cs-coverage opencover` to specify arguments to pass to OpenCover. For example:

```
cov-build --dir idir --cs-coverage opencover --cs-opencover-arg "-threshold:1" make
build
```

An invalid argument will cause OpenCover to fail. For more information about OpenCover arguments, see the OpenCover documentation.

**--cs-filter <filters>**

[Test Advisor option] Used in conjunction with `--cs-coverage opencover` to specify a list of filters to OpenCover. Specifying a filter allows you to control what assemblies are instrumented for coverage collection. For example:

```
+[*]* -[Test]*
```

This filter would instrument everything except for those modules that start with 'Test'. For more information on filters, see the OpenCover documentation [↗](#).

**--cs-no-register-profiler**

[Test Advisor option] Instructs Test Advisor to NOT attempt to automatically register the profiler DLL. If you use this argument, you must manually register the profiler DLLs. To do so, use `regsvr32`:

```
% regsvr32 <install_dir>/bin/x86/OpenCover.Profiler.dll
% regsvr32 <install_dir>/bin/x64/OpenCover.Profiler.dll
```

This option should be used when you want to run multiple `cov-build` with the `--cs-coverage opencover` option.

You can change the `regsvr32` command to only register per-user, for example:

```
% regsvr32 /n /i:user <install_dir>/bin/x86/OpenCover.Profiler.dll
% regsvr32 /n /i:user <install_dir>/bin/x64/OpenCover.Profiler.dll
```

If you are not on a 64-bit platform, you do not need to register the x64 DLL.

`--cs-test-config <properties-xml-file>`

[Test Advisor option] Used in conjunction with `--cs-coverage` to provide the location of a user-created C# test properties file (`<properties-xml-file>`). This file configures per-test separation for C# coverage. If both `--cs-test` and `--cs-test-config` are specified, the value given to `--cs-test-config` takes precedence.

`--cs-test <test-framework-name>`

[Test Advisor option] Used in conjunction with `--cs-coverage` to select a pre-configured C# test properties file (`<test-framework-name>`). This file configures per-test separation for C# coverage. Valid values are:

- `nunit` - for NUnit tests.
- `xunit` - for XUnit tests.
- `mstest` - for MSTest tests.

Test Advisor provides three configuration files that specify test boundary configuration for the popular test execution frameworks: MSTest, NUnit and XUnit. The properties files are located in the following directory (but note that in the option parameter, you only need to specify the name, not the entire path):

```
<install_dir_ca>/config/
```

`--cygpath <path>`

Specify the path to the directory, which contains the bin directory of the Cygwin installation, if it is not in the `PATH` environment variable.

`--cygwin`

Compatible with C and C++ builds only. On Windows, indicates that the build is done with Cygwin. This option allows Cygwin-style paths to be used in the native build command. However, you must use Windows-style paths for all Coverity Analysis commands.

`--da-broker <broker_servername:port>`

[Dynamic Analysis option] Specifies the Dynamic Analysis broker server to receive Dynamic Analysis results for this run. You can specify a hostname or an IP address. The port is optional (the default is 4422). Currently, only IPv4 addresses are supported. This option is only used in conjunction

with `--java-da` and cannot be combined with `--dir`, but using `--log-dir` is recommended for diagnostic purposes.

See also, `--java-da-opts`.

`--default-instrumentation-mode (include|exclude), -im (include|exclude)`

[Test Advisor option] When used with "`--c-coverage function`", an optional way to tell whether files not referenced by `--include-instrumentation-path` or `--exclude-instrumentation-path` should be instrumented or not. If none of these three options are used, the default behavior is to instrument all source files, including system header files. If `--include-instrumentation-path` is used without `--default-instrumentation-mode`, then the default behavior changes to exclude all source files from instrumentation unless explicitly included.

See also: `--include-instrumentation-path`, `--exclude-instrumentation-path`.

For more information, see *Test Advisor 2020.12 User and Administrator Guide* [↗](#)

`--defer-decomp`

Only records the decompilations of byte code during the build. It does not attempt to decompile and emit the byte code. Later, `cov-build` can be rerun with `--replay-decomp` to decompile and emit the byte code.

See also, `--replay-decomp`.

`--delete-stale-tus`

Automatically deletes translation units that are created from source files that were renamed or removed. This capability is off by default. Use this command when you perform an incremental build when you have deleted/renamed source files.

`--desktop`

The `--desktop` option can be used when running `cov-build` in preparation for Desktop Analysis. It behaves similarly to `--record-only` for C/C++ builds, disables bytecode decompilation in Java and C# builds, and does a full build for other languages.

Please note that this option is supported for backward compatibility. The preferred method for capturing a build for Desktop Analysis is the `cov-run-desktop --build` option.

`--dir <intermediate_directory>`

Pathname to an intermediate directory that is used to store the results of the build and analysis.

Exactly one of the options `--emit-server`, `--da-broker`, or `--dir` must be specified. Coverity recommends that you use `--log-dir` when you are not using `--dir`.

`--disable-aspnetcompiler`

[C# and Visual Basic builds only] Disables the automatic invocation of `Aspnet_compiler.exe` for any ASP.NET 4 and earlier Web applications that are detected in the build. The output of `Aspnet_compiler.exe` is required by the C# and Visual Basic security checkers.

Use this option if you are manually running `Aspnet_compiler.exe` as part of your native build or as part of your Coverity Analysis workflow. For further information, see "Capturing an ASP.NET Web application" [↗](#) in *Coverity Analysis 2020.12 User and Administrator Guide*.

**--disable-compute-coverability**

[Test Advisor option] Skips computing coverability when you run a build or capture. This option is useful if the test will be run in a separate step. Using this option can speed up the build process.

**--disable-cs-parse-error-recovery**

Disables extra attempts to recover from problems parsing C# or Visual Basic source. Though this recovery process takes some extra time, it greatly reduces the impact of parsing problems in most cases. It works by attempting to emit subsets of the input files that would not otherwise reach the emit database.

Typically, this option is needed only if the recovery process takes too long and becomes unmanageable. Note that unlike for Java, this mode is enabled by default for C# and Visual Basic because it uses a more efficient algorithm.

**--disable-gcov-arg-injection**

[Test Advisor option] This option is compatible with C and C++ builds only, and is used in conjunction with `--c-coverage gcov`. It prevents `cov-build` command from automatically injecting the `gcov` command line arguments into the native build. This is useful if you want to use your build system's own `gcov` targets instead of having this command run it automatically. For example:

```
cov-build --dir idir --c-coverage gcov --disable-gcov-arg-injection make gcov
```

**--disable-java-per-class-error-recovery**

Disables per-class error recovery, which is the default behavior of `cov-build`. Enabling per-class error recovery increases the percentage of source files that can be emitted by attempting to use the class files of the source files that cause parse errors. This default behavior could potentially increase the time to emit files, but it is usually faster than `--enable-java-per-file-error-recovery` because it does not try to emit each file one at a time.

Per-class error recovery is unlikely to correct `cov-emit-java` crashes. It also requires the presence of output class files. To address such issues, see `--enable-java-parse-error-recovery`.

**--disable-local-emit-server**

Prevents the command from starting the emit server (as part of the automated startup process). This option is not for general use and should only be used if startup problems occur. This option only takes effect when `--emit-server` is specified.

**--disable-ms-pch**

When capturing a build that uses Microsoft compilers and precompiled headers, the Coverity compiler emulates the native precompiled header rules to improve build performance. This behavior is enabled by default and can be disabled by using the `--disable-ms-pch` option with `cov-build`.

**--emit-cmd-line-id**

Deprecated. This option is deprecated as of the 4.4 release.

**--emit-complementary-info**

Enables emitting of complementary information for compliance checkers such as MISRA checkers. Selecting this option results in a slower build capture but a faster analysis, and it should be applied when using compliance checkers. The default value is `--no-emit-complementary-info`

 **Note**

Enabling the `--emit-complementary-info` option prior to running an analysis is likely to turn up additional defects.

Any analysis involving `--coding-standard-config` requires the information generated during `cov-build` when including the `--emit-complementary-info` option. The `cov-build` command will take longer, so this option should only be used when `cov-analyze` is used with `--coding-standard-config`.

If `cov-build` did not include the `--emit-complementary-info` option and `cov-analyze` does include `--coding-standard-config`, `cov-analyze` automatically re-runs every `cov-emit` command (for the Translation Units to be analyzed). This excludes the native build and the `cov-translate` overhead, but it will add significant overhead to `cov-analyze`. Note that analysis will fail if the emit database does not include source; that is re-emit is not possible.

**--emit-parse-errors**

Deprecated. Compatible with C and C++ builds only. Uses the `--enable_PARSE_ERROR` option in `cov-analyze` instead. Specifies that compile failures should be made visible in Coverity Connect, appearing as defects.

**--emit-server <emit\_servername:port>**

[Test Advisor option] Specifies the emit server to send coverage to for this run. You can specify a hostname or an IP address. The port is optional (the default is 15772). Currently, only IPv4 addresses are supported.

This option is only used in conjunction with `--c-coverage gcov`.

**--enable-java-parse-error-recovery**

Enables the error recovery algorithm that produces the highest emit percentage in most cases. Currently, this option enables per-class error recovery with automatic fallback to per-file recovery in case of non-recoverable errors such as `cov-emit-java` crashes, but its behavior might change in future. Note that this algorithm is also enabled by default if no error recovery option is specified on the command line.

Explicitly enabling an error recovery algorithm on the command line will automatically disable any incompatible error recovery algorithms.

**--enable-java-per-class-error-recovery**

[Deprecated as of 7.6.0] The `cov-build` command now performs per-class error recovery by default, so it is no longer necessary to use this option.

For information about per-class error recovery and about how to disable it, see `--disable-java-per-class-error-recovery`. See also, `--enable-java-per-file-error-recovery`.

**--enable-java-per-file-error-recovery**

Use this option if your native Java compiler is able to compile your source code successfully, but `cov-emit-java` crashes. This option might also help when `cov-emit-java` has parse errors and there are no class files available for per-class error recovery.

`--enable-pch`

Compatible with C and C++ builds only. Tells `cov-emit` to use pre-compiled header files when the Microsoft Visual Studio compiler uses them. This is not strictly necessary for a Microsoft Visual Studio project that uses pre-compiled headers, but should improve performance. This option is only supported on 64-bit platforms.

`--encoding <enc>`

Compatible with C, C++, and JavaScript builds only. Specifies the encoding of source files. Use this option when the source code contains non-ASCII characters so that Coverity Connect can display the code correctly. The default value is US-ASCII. Valid values are the ICU-supported encoding names:

US-ASCII

UTF-8

UTF-16

UTF-16BE

UTF-16 Big-Endian

UTF-16LE

UTF-16 Little-Endian

UTF-32

UTF-32BE

UTF-32 Big-Endian

UTF-32LE

UTF-32 Little-Endian

ISO-8859-1

Western European (Latin-1)

ISO-8859-2

Central European

ISO-8859-3

Maltese, Esperanto

ISO-8859-4

North European

ISO-8859-5

Cyrillic

ISO-8859-6

Arabic

ISO-8859-7

Greek

ISO-8859-8  
Hebrew

ISO-8859-9  
Turkish

ISO-8859-10  
Nordic

ISO-8859-13  
Baltic Rim

ISO-8859-15  
Latin-9

Shift\_JIS  
Japanese

EUC-JP  
Japanese

ISO-2022-JP  
Japanese

GB2312  
Chinese (EUC-CN)

ISO-2022-CN  
Simplified Chinese

Big5  
Traditional Chinese

EUC-TW  
Taiwanese

EUC-KR  
Korean

ISO-2022-KR  
Korean

KOI8-R  
Russian

windows-1251  
Windows Cyrillic

windows-1252  
Windows Latin-1

windows-1256  
Windows Arabic

 **Note**

If your code is in SHIFT-JIS or EUC-JP, you must specify the `--output_object_encoding SHIFT-JIS` or `--output_object_encoding EUC-JP` option (respectively) for `cov-emit` in order to avoid receiving `STRING_OVERFLOW` false positives.

`--exclude-instrumentation-path <path-or-file>, -ep <path-or-file>`

[Test Advisor option] When used with "`--c-coverage function`", specifies a source file or path that will not be instrumented for function coverage. If a path is given, then all source files in subdirectories are also excluded. May be used more than once to specify multiple files or paths.

This option has higher precedence if it overlaps with files specified by `--include-instrumentation-path`.

See also: `--include-instrumentation-path`, `--default-instrumentation-mode`.

For more information, see *Test Advisor 2020.12 User and Administrator Guide* [🔗](#)

`--finalize, -fin`

Compatible with C and C++ builds only. Combines the build log and metrics from all the host machines that ran `cov-build` with the same `<intermediate_directory>`, and indicate any additional steps that are needed to prepare for a C and C++ analysis.

Do not specify a build command when using the `--finalize` option.

`--force`

Specifying this options causes the Coverity compiler to attempt all source files, including files that have already been emitted and whose timestamps have not changed. This is equivalent to `--force` in the respective compiler, for example `cov-emit`.

`--fs-capture-just-print-matches`

[Filesystem capture option] This option is for debugging purposes only.

Suppresses the emission of files matched with filesystem capture, and outputs a list of files that would have been emitted. This should generally be used with `--no-command`.

`--fs-capture-list <file>`

[Filesystem capture option] Specifies a file that contains a list of files to use as analysis inputs. Files that are recognized as useful for analysis, such as those matching patterns established in calls to `cov-configure`, will be emitted into the intermediate directory. For example, `cov-configure --python` marks `*.py` as useful for analysis, and `cov-configure --javascript` marks `*.js`, `*.html`, and `*.htm` as useful.

This option is a lower-level supplement to or an alternative to `--fs-capture-search`.

A recommended way of using this option is to provide a list of revision-controlled files as the potential analysis inputs. Example for a git repository:

```
git ls-files > scm_files.lst
cov-build --fs-capture-list scm_files.lst <other options>
```

Note that this option can be specified more than once to include multiple lists of analysis inputs.

See also, `--fs-capture-search` and `--no-command`.

#### `--fs-capture-search <directory>`

[Filesystem capture option] Recursively searches all files in the specified directory for analysis inputs. Files that are recognized as useful for analysis, such as those matching patterns established in calls to `cov-configure`, will be emitted into the intermediate directory. For example, `cov-configure --python` marks `*.py` as useful for analysis, and `cov-configure --javascript` marks `*.js`, `*.html`, and `*.htm` as useful. However, this option ignores any file matching `--fs-capture-search-exclude-regex` and certain files and directories when the specified directory is in the path:

- Symlinks are ignored unless the specified directory is itself a symlink or is located under a symlink.
- Files and directories starting with a period (`.`), including their contents, are ignored unless the specified directory itself starts with a period or is located under such a directory.
- Directories named `SCCS`, including their contents, are ignored unless the specified directory itself is named `SCCS` or is located under such a directory.
- Coverity intermediate directories and their contents are ignored.

Note that this option can be specified more than once to search multiple directories for analysis inputs.

See also, `--fs-capture-list`, `--fs-capture-search-exclude-regex`, and `--no-command`.

#### `--fs-capture-search-exclude-regex <regex>`

[Filesystem capture option] This option only applies when `--fs-capture-search` is used. Otherwise, it results in an error.

While `--fs-capture-search` searches its specified directory for analysis inputs, any files or subdirectories that match the specified `regex` will be excluded from the search, and thus will not be included in the analysis (unless they are independently selected by the `--fs-capture-list` option). Excluded directories will not have their contents searched for further matches, unless a subdirectory is specifically passed to `--fs-capture-search`.

The regular expression is matched against the filename using case insensitive matching.

We recommend using `--fs-capture-search-exclude-regex "[/\\]node_modules[/\\]"` when capturing Node.js applications that contain third-party modules installed by npm within the application directory. This exclusion is applied by default when using the `cov-configure --javascript` shortcut.

#### **Note**

Regular expressions are searched in full absolute paths, in host format. Thus, matching forward slash without also matching backslash, or vice-versa, will make the pattern platform-specific. For example, to exclude JavaScript files starting with 'mock', use the following (assuming bash-style single quoting):

```
--fs-capture-search-exclude-regex "[/\\]mock.*[.]js$"
```

Without the leading "[/\\"], a file called 'hammock.js' would be excluded. Similarly, to exclude all contents of directories named 'test' with parent directory named 'src', use the following:

```
--fs-capture-search-exclude-regex "[/\\"src[/\\"test$"
```

**--fs-library-path <path/to/lib>**

[Java and JSP filesystem capture option] For Java filesystem capture, the `--fs-library-path` option is added to the class and source paths of the Java compiler. Specifying this option is the equivalent to passing the `--classpath <path/to/lib>` argument and `--sourcepath <path/to/lib>` as command line options to the Java compiler.

For example, the following command adds classes, some.jar, and extrasrc to the Java compiler's class and source paths (in this order):

```
cov-build --config config.xml \
--fs-library-path path/to/classes \
--fs-library-path path/to/some.jar \
--fs-library-path path/to/extrasrc \
--fs-capture-search path/to/source/code
```

[See the `cov-configure` variant for `--fs-library-path` if you want to extend the class and source paths only once and do not need to set different `--fs-library-path` values for different `cov-build` invocations.]

[JavaScript, PHP, and Python filesystem capture option] Specifies third-party library locations for JavaScript Node.js `require` modules, ECMAScript 6 module imports, JavaScript HTML `script src=` includes, HANA XSC libraries imported with `$.import`, PHP `include/include_once/require/require_once` and Python imports. By default, these inclusions and imports are resolved relative to the source file doing the inclusion/import (according to language specific rules). The `cov-build` command also attempts to resolve them relative to directories passed to the `--fs-library-path` option.

For example, the following command adds `lib1` and `lib2` as additional library paths (searched in this order):

```
cov-build --config config.xml \
--fs-library-path lib1 \
--fs-library-path lib2 \
--fs-capture-search path/to/source/code
```

[See the `cov-configure` variant for `--fs-library-path` if you want to set the paths to the libraries only once, and if you do not need to set different `--fs-library-path` values for different `cov-build` invocations.]

The search for the library file is permissive: If the search does not find the library at a relative path specified by this option, a second search for the filename alone (excluding the specified path) will run.

**--include-instrumentation-path <path-or-file>, -ip <path-or-file>**

[Test Advisor option] When used with "`--c-coverage function`", specifies a source file or path that will be instrumented for function coverage. If a path is given, then all source files in

subdirectories are also included. May be used more than once to specify multiple files or paths. May be overridden by `--exclude-instrumentation-path`.

When not used, the default behavior is to instrument all source files, including system header files, unless explicitly overridden by `--exclude-instrumentation-path`. However, when this option is used, the default behavior changes to exclude all source files from instrumentation unless explicitly included.

See also: `--exclude-instrumentation-path`, `--default-instrumentation-mode`.

For more information, see *Test Advisor 2020.12 User and Administrator Guide* [↗](#)

#### `--initialize, -init`

Compatible with C and C++ builds only. Creates the specified `<intermediate_directory>` that a set of subsequent builds will use. You can only use this option once, and without a build command, before a parallel build.

#### `--instrument`

Compatible with Java, C, C++, C#, and Visual Basic builds on Windows only, uses the instrumentation mode instead of the debugger. For certain builds, this configuration can significantly improve build times. In particular, parallel builds will benefit most from `--instrument`.

#### Known issues and workarounds:

- If Visual Studio (2010 or newer) is running `Tracker.exe`, `cov-build` will skip running `Tracker.exe` by default. (The rest of the build will remain unaffected.) This behavior can be disabled with the `--no-disable-tracker` option.

Alternatively, you can set the environment variable `COVERITY_TRACKER_WHITELIST` to specify those `Tracker.exe` binaries that should **not** be disabled. This environment variable is a semi-colon delimited list. For example:

```
COVERITY_TRACKER_WHITELIST="C:\path1\Tracker.exe;C:\path2\Tracker.exe"
```

If the environment variable is set as in the example above, the `cov-build` command will not disable the Tracker when it is run from `C:\path1\Tracker.exe` or `C:\path2\Tracker.exe`.

- If `Tracker.exe` is permitted to run, you may run into a few known issues, which are outlined below.
  - The template compiler configuration can cause link failures in the build. To work around this issue, you can take either of the following actions:
    - Generate a non-template compiler configuration.
    - Disable file tracking in your build. If you use `msbuild` to build, you can disable the tracker by adding `/p:TrackFileAccess=false` to your command line. If you use `devenv` to build, you need to add the configuration value to your solution/project files.
- If Visual Studio (2010 or newer) is running `Tracker.exe`:

- The `cov-build` command issues a warning if it detects `Tracker.exe`.
- The capture DLL will still be loaded for persistent processes, even after `cov-build` exits. One such example of a process like this is `mspdbsvr.exe`, which is a special case that is automatically ignored. However, if you find another binary that persists, you can ignore it by adding `--capture-ignore foo.exe` to the `cov-build` command line. It is important to note, however, that you can only ignore the process if it does not start any compilations.
- The `--instrument` argument is incompatible with the `__COMPAT_LAYER` environment variable. If your environment sets this variable, you must unset it to use `--instrument`.

**--java-coverage <tool>**

[Test Advisor option] Enables Java coverage collection for Test Advisor using the specified tool. <tool> represents the coverage tool; either `cobertura` or `jacoco`.

You can use `c-coverage` along with `java-coverage` and/or `cs-coverage`, for example:

```
cov-build --dir idir --c-coverage gcov --java-coverage cobertura make build
```

**--java-da**

[Dynamic Analysis option] Enables the injection of a Dynamic Analysis agent into a command used for exercising your program or running tests on it. The `--java-da` option is only compatible with builds of Java code. This option disables build capture and coverage capture, so is incompatible with options related to those tasks. It currently requires the `--da-broker` option to reference a running Dynamic Analysis broker and a build id to be specified with one of `--build-id`, `--build-id-dir`, or `--build-id-file`.

 **Note**

We recommend that you use this option with the `--test-capture` option because it is not necessary to capture the build.

This option is not currently compatible with `--dir`, but using `--log-dir` is recommended for diagnostic purposes.

See also, the `--java-da-opts` option to this command.

**--java-da-opts <da\_agent\_option\_string>**

[Dynamic Analysis option] Adds extra options to the Java agents that are started (see Dynamic Analysis Agent command-line options and Ant task attributes [🔗](#) in the *Dynamic Analysis 2020.12 Administration Tutorial*).

See also, the `--java-da` and `--da-broker` options to this command.

Example:

```
--java-da-opts detect-resource-leaks=false,exclude-instrumentation=com.foo.mock
```

**--java-instrument-classes <list.csv>**

[Test Advisor option] Specifies a file that contains a list of classes to be present on the execution target to execute Java tests on a non-build host. <list.csv> is in CSV file format and has two columns:

- Column 1: The name of the class.
- Column 2: The full path to the source file for the class.

To construct the contents for this file, use the `list-compiled-classes` subcommand to `cov-manage-emit`.

`--java-test <config-name>`

[Test Advisor option] Used in conjunction with `--java-coverage` to select a pre-configured Java test properties file (`<test-framework-name>`). This file configures per-test separation for Java coverage. Valid values are:

- `junit` - for JUnit3 tests.
- `junit4` - for JUnit tests.

Test Advisor provides two configuration files that specify test boundary configuration for the popular test execution frameworks: `JUnit3` and `JUnit4`. The properties files are located in the following directory (but note that in the option parameter, you only need to specify the name, not the entire path):

```
<install_dir_ca>>/config/
```

`--java-test-config <path-to-config-file>`

[Test Advisor option] Used in conjunction with `--java-coverage` to provide the location of a user-created Java test properties file for the Java capture agent. This file configures per-test separation for Java coverage. If both `--java-test` and `--java-test-config` are specified, the value given to `--java-test-config` takes precedence.

`--js-template-da`

Deprecated as of 2019.06. The Javascript template `dynamic-analysis` will now be run automatically as part of the build process if applicable.

`--leave-raw-coverage`

[Test Advisor option] Used in conjunction with `--c-coverage gcov` or `--java-coverage cobertura`. When this option is specified, this build command will not automatically add the collected coverage data to the emit, but will instead leave it inside the intermediate directory to be used later.

`--log-dir <dir>`

When `--dir` is not used, this option specifies a directory, such as an intermediate directory, in which to store the build log or capture log file. By default, when neither `--dir` nor `--log-dir` is used, these logs go to a temporary directory that is erased.

`--log-server`

Compatible with C and C++ builds on Windows only, this argument allows `cov-build` to produce a consistent build log when using `--parallel-emit`. All output in the build log can be attributed to specific executions of each Coverity program. This is the default.

This argument has no effect to and cannot be used in combination with `--instrument`. You will receive an error message.

`--merge-raw-coverage <raw-coverage-directory>`

[Test Advisor option] Specifies a directory containing raw coverage data that should be added to the emit. When this is specified, a build command is not allowed.

`<raw-coverage-directory>` is the path to a coverage directory that was previously created with `--leave-raw-coverage`.

You can merge raw coverage from multiple directories in a single invocation by specifying each directory to a separate `--merge-raw-coverage` argument. Each directory to merge may contain any combination of coverage tools that are supported by the OS on which `cov-build --merge-raw-coverage` is being run.

The following scenario shows the basic usage of `--leave-raw-coverage --merge-raw-coverage`:

1. Perform your build.

```
cov-build --dir idir --c-coverage gcov make build
```

2. Run the tests, but leave the raw coverage, and copy it to a new location.

```
cov-build --test-capture --dir idir --c-coverage gcov --leave-raw-coverage make test
```

```
scp -r remote:idir /some/tmp/dir/idir-raw
```

3. Sometime later, merge in the raw coverage.

```
cov-build --test-capture --dir idir --merge-raw-coverage /some/tmp/dir/idir-raw
```

`--merge-raw-coverage-file <tool>:<filename>`

[Test Advisor option] Merges raw coverage data in the given file that was generated by the JaCoCO coverage tool. It is also possible to specify test meta properties with the following options:

- Test status with `--teststatus`.
- Test start date/time with `--teststart`.
- Test duration with `--testduration`.
- Suite name with `--suiteName`.
- Test name with `--testname`.

`--minimal-classpath-emit`

Limits the group of emitted JAR files to those needed for compilation of the Java files. The default behavior without this option is to emit all the JAR files in the classpath regardless of whether they are referenced by a Java file in the compilation. This option can improve performance of Java builds with large numbers of unused JAR files on the classpath at the risk of not capturing all the dependencies of those JAR files. For example if `A.java` references `A.jar`, which has dependencies on `B.jar`, this option will prevent `B.jar` from getting emitted even if `B.jar` is on the classpath.

`--msbuild-shutdown-maxnodes <N>`

For Visual Studio 2010 and newer (only available on Windows platforms): Specifies the maximum number of nodes that `cov-build` should attempt to shut down. Typically, this value is equal to the number of nodes that your Visual Studio project is configured to use. Use this option only if the default behavior is undesirably slow.

`--name <name>`

Tags a build with a name. This name can then be used for translation unit pattern matching through the `cov-manage-emit build_name` argument.

`--no-banner`

Suppresses the `cov-build` application name and version banner from the console output.

`--no-caa-info`

Compatible with C and C++ builds only. Do not collect the information required for Coverity Architecture Analysis in the intermediate directory.

`--no-command`

Specify this option if there is no command for build or test coverage capture. For example, in JavaScript-only analysis, you specify this option with `--fs-capture-search` or `--fs-capture-list`. However, if you are using both build capture and filesystem capture, a single invocation of `cov-build` is recommended, without this option.

`--no-disable-tracker`

This option allows the user to force `cov-build --instrument` to run `Tracker.exe`. By default, if Visual Studio (2010 or newer) is running `Tracker.exe`, then `cov-build --instrument` will skip running the tracker executable. The `--no-disable-tracker` option allows the user to bypass this skip.

 **Note**

This option may cause build issues. See the *Known Issues and Workarounds* section under the `--instrument` option.

`--no-emit-complementary-info`

Disables emitting of complementary information for compliance checkers such as MISRA checkers.

`--no-error-recovery`

Disables source-level error recovery in the parser. This typically should only be used if error recovery is causing problems and you have been instructed to use this option by Coverity support.

`--no-generate-build-id`

Prevents a new build ID from being generated for every `cov-build` invocation, in the case where you wish to run `cov-build` multiple times on a single intermediate directory.

`--no-log-server`

Compatible with C and C++ builds on Windows only, this argument forces `cov-build` to revert to its original behavior without the log server, with respect to the build log. This is only intended for use when issues arise using `--log-server`.

**--no-msbuild-shutdown**

For Visual Studio 2010 and newer (only available on Windows platforms): Disables shutdown of resident msbuilds that are created by the Microsoft Build Engine, `msbuild`. Use this option only if you know that you will not have any msbuild processes running, or if you kill the resident msbuilds through some other method.

**--no-network-coverage**

[Test Advisor option] Makes the command use the file system instead of the emit server. This option only takes effect when `--c-coverage gcov` is specified.

**--no-parallel-translate**

Compatible with C and C++ builds only. Disables `cov-translate` parallelization. This will prevent `cov-translate` from running in parallel regardless of the degree of parallelization requested, either directly to `cov-build`, `cov-translate`, through configuration files, or native command line translation.

This can also be added as a `cov-emit` argument in a configuration file (it is not actually passed to `cov-emit`). For example:

```
<prepend_arg>--no-parallel-translate</prepend_arg>
```

**--no-preprocess-next**

Compatible with C and C++ builds only. Disables the `--preprocess-next` option.

**--no-refilter, -nrf**

Compatible with C and C++ builds only. When combined with `--replay`, calls `cov-emit` directly with the previously translated command line arguments, instead of calling `cov-translate` again (which is the default).

This option does not work with MSVC PCH.

**--no-security-da**

Disables the dynamic analysis—that is, the execution of `cov-security-da`—that is typically run at the end of the build. The results of the dynamic analysis are used for a security analysis.

**--optimize-pch-space**

This option can be used to reduce disk space consumption by Coverity precompiled headers (PCH). This is only effective when running `cov-build --replay` or `--replay-from-emit`, with PCH enabled (`--enable-pch`).

 **Note**

Note that this option may slow the build process down, and in some rare cases, may have no effect on disk space consumption.

**--parallel-emit**

This C, C++, and Windows only argument will allow `cov-emit` processes to run in parallel. For certain builds, this argument can significantly improve build times. This argument is enabled by default (see `--serial-emit`).

This argument has no effect and cannot be used in combination with `--instrument`. You will receive an error message.

`--parallel-translate=<number_of_processes>`

Compatible with C and C++ builds only. Instructs `cov-translate` to run `cov-emit` in parallel when multiple files are seen on a single native compiler invocation. This is similar to the Microsoft Visual C and C++ `/MP` switch. Specify the `<number_of_processes>` to be greater than zero to explicitly set the number of processes to spawn in parallel, or zero to auto-detect based on the number of CPUs. When specified directly to either `cov-build` or `cov-translate`, this option will override any settings set in configuration files or translated through the native command line.

This can also be added as a `cov-emit` argument in a configuration file (it is not actually passed to `cov-emit`). For example:

```
<prepend_arg>--parallel-translate=4</prepend_arg>
```

`--parse-error-threshold <percentage>`

The percentage of translation units that must successfully compile for the `cov-build` command to not generate a warning. If less than this percentage compiles, the `cov-build` command will give a warning when the build completes. The default value is 95.

 **Note**

When used in conjunction with `--return-emit-failures`, `cov-build` will return error code 8, in addition to generating the warning, if less than the specified percentage compiled.

`--preprocess-first`

Compatible with C and C++ builds only. Uses the native compiler to preprocess source files and then invokes `cov-emit` to compile the output of the native processor. By default, `cov-emit` (which is invoked by `cov-build`) otherwise tries to preprocess and parse each source file.

Using this option can address some cases in which hard-to-diagnose causes for macro predefinitions are different, or for header files that cannot be found by `cov-emit`. Usually, `cov-configure` attempts to intelligently guess the native compiler's predefined macros and built-in include directories, but sometimes `cov-configure` guesses incorrectly. Using the `--preprocess-first` option circumvents the problem, but at the cost of losing macro information during analysis. Using `--preprocess-first` does not always work because it requires rewriting the native compiler command line, which the native compiler may or may not like.

See also, `--preprocess-next`.

`--preprocess-next`

Compatible with C and C++ builds only. Attempts to use `cov-emit` to preprocess source files. If that attempt fails, or if `cov-emit` encounters a parse error, this option preprocesses the files with the native preprocessor, and invokes `cov-emit` to compile the output of the native processor. This offers the benefit of using the higher-fidelity `cov-emit` preprocessor, while also providing a fallback in case of errors.

This option can be disabled with `--no-preprocess-next` (the latter has precedence over the former). See `--preprocess-first` for information about the effects of using the native preprocessor.

**--python-version**

Specify the major version of the Python language being used: '2' or '3'. By default, a Python script is compiled assuming version 2 of the Python language, and if a syntax error occurs, an attempt is made to recompile the script as Python 3. This option overrides a value, if specified, for the `--version` option to the `cov-configure --python` command syntax.

**--record-only, -ro**

Compatible with C and C++ builds only. Only record the compiles done during the build, do not attempt to parse and emit the code. Later, `cov-build` can be rerun with `--replay` to actually parse and emit the code.

 **Note**

Note that you must not relocate your intermediate directory (specified with `--dir`) between the `--record-only` and `--replay` steps. If you need to move your intermediate directory to a new location or separate machine, use `--record-with-source` and `--replay-from-emit`.

**--record-with-source, -rws**

Compatible with C, C++, and Java builds only. Compile translation units far enough to pull in all the `#include` files needed by the compilation, and store these in the emit. Later, `cov-build` can be rerun with `--replay-from-emit` to actually parse and emit the code. See the entry for `--replay-from-emit` for more information and examples.

This argument has no effect to and cannot be used in combination with either `--preprocess-first` or `--preprocess-next`. You will receive an error message.

**--replay, -rp**

Compatible with C and C++ builds only. Replay the parse and emit steps that were previously recorded for a build in the given intermediate directory. If you specify this option, do not specify a build command. This option can be used to quickly update a previously emitted build if the source files have changed.

 **Note**

Note that you must not relocate your intermediate directory (specified with `--dir`) between the `--record-only` and `--replay` steps. If you need to move your intermediate directory to a new location or separate machine, use `--record-with-source` and `--replay-from-emit`.

**--replay-decomp**

Decompile translation units from byte code source contained within the emit directory. Replaying from the emit will have the same results, regardless of changes to the files in the filesystem (including deletion).

See also, `--defer-decomp`.

**--replay-failures, -rpf**

Compatible with C and C++ builds only. Only attempt to replay the emit for files that had parsing or other compilation failures.

**--replay-from-emit, -rpf**

Compatible with C, C++, and Java builds only. Recompile translation units from source contained within the emit directory. Replaying from the emit will have the same results, regardless of changes to the files in the filesystem (including deletion).

This can be used when translation units were added with normal `cov-build` processes (although this will have no real effect unless `--force` has been passed), or with translation units added with `--record-with-source`.

The advantage of using `--record-with-source` and `--replay-from-emit` is that temporary files (such as created by `#import`) are captured in the emit, and so projects that use `#import` can be replayed, which they cannot with `--replay`. In addition, it is possible to transport the intermediate directory to a different computer/platform and replay it there.

For example, you can record a build on Windows and transfer the intermediate directory to Linux and replay it there. (You will have to use `cov-manage-emit reset-host-name` to change the host.)

This argument has no effect to and cannot be used in combination with either `--preprocess-first` or `--preprocess-next`. You will receive an error message.

**--replay-processes <count>, -j <count>**

Compatible with C and C++ builds only. When performing `--replay`, spawn up to `<count>` `cov-emit` processes in parallel (on a single machine).

This option accepts the number of processes, or `auto` which sets the number of replay processes to the number of logical processors in the machine (`-j 0` is also accepted and is the same as `auto`).

**--return-emit-failures**

The `cov-build` command returns with an error code if an emit failure occurs. The return value is a combination (binary OR) of the following flags:

- 1: The build returned an error code.
- 2: The build terminated with an uncaught signal (for example, segmentation fault).
- 4: No files were emitted.
- 8: Some files failed to compile. By default, this error code is returned if fewer than 95% of the compilation units compiled successfully. You can change this percentage by using the `--parse-error-threshold` option.
- 16: Command line error, such as an unrecognized option.

 **Note**

The `cov-build` command always returns an error code if the native build fails.

**--serial-emit**

This C, C++, C#, and Visual Basic Windows-only argument forces `cov-emit` processes to run in serial. This option is disabled by default (see `--parallel-emit`).

This option has no effect and cannot be used in combination with `--instrument`. Attempts to use the two options together will result in an error message.

`--suite-name <suite-name>`

Test suite name to attribute coverage to when using `--merge-raw-coverage-file`.

`--system-encoding <enc>`

Compatible with C and C++ builds only. Specifies the encoding to use when interpreting command line arguments and file names. If not specified, a default system encoding is determined based on host OS configuration.

See `--encoding` for a list of accepted encoding names.

This option has no effect when used in conjunction with one of the `--replay` options.

`--telemetry-network-error-exit-code <exit code>`

[Test Advisor option] Specifies the exit code to use when the process aborts due to it exhausting all retries. This option can be customized to allow for the test harness to appropriately react to these failures.

The default exit code is 19.

For more information, see "Error recovery" in the *Test Advisor 2020.12 User and Administrator Guide*. [↗](#)

`--telemetry-network-error-log <filename>`

When a network error is encountered, by default an error will be written to `STDERR` which can interfere with some test processes. This argument allows these errors to be redirected to a user-specified file.

For more information, see "Error recovery" in the *Test Advisor 2020.12 User and Administrator Guide*. [↗](#)

`--telemetry-network-error-max-wait <# of seconds>`

[Test Advisor option] Specifies the maximum time to wait when recovering from a network related error. The behavior when encountering an error is to sleep and then retry. It will sleep for 1, 5, 30, 60 seconds, respectively, until the maximum wait time has been reached.

The default maximum wait time is 300 seconds.

For more information, see "Error recovery" in the *Test Advisor 2020.12 User and Administrator Guide*. [↗](#)

`--test-capture`

[Test Advisor option] Use to gather test coverage. Additional Test Advisor options are provided for further customization.

The `cov-build` command gathers source code and test coverage whether or not you use the `--test-capture` option; this option is recommended if your tests run separately from your build. This option modifies the behavior of `cov-build` in the following ways:

- No warning is given if no source files are emitted.
- Log data is written to `<intermediate_directory>/capture-log.txt`.
- `--no-generate-build-id` is implied.
- Does not save `build-cwd.txt` for use by `cov-analyze --strip-path`.
- Does not attempt to automatically compile any ASP.NET Web applications.

**Note**

We recommend using the `--test-capture` option when using `--java-da` to do Dynamic Analysis.

`--test-capture-run-tag <string>`

[Test Advisor option] Specifies a custom tag to allow for easy selection of this test capture run. For example:

```
linux-build
```

`--test-capture-run-timestamp <timestamp>`

[Test Advisor option] Specifies the timestamp to use for the test capture run for this invocation. If it is not specified, the current time will be used, which is the typical use case. This option is only provided as a way for multiple test runs to use the same test capture run.

See Appendix A, *Accepted date/time formats* for proper formatting of the `<timestamp>` argument.

`--test-capture-status-file <file containing status>`

[Test Advisor option] Specifies a filename that, at the end of the test run, will contain the status that should be used for this test capture run. The file should contain one of:

- `success`
- `failure`
- `unknown`

Anything other than the above will be interpreted as `unknown`.

`--testduration <test duration in milliseconds>`

[Test Advisor option] Test duration for the test when using `--merge-raw-coverage-file`.

`--testname <testname>`

[Test Advisor option] Test name to attribute coverage to when using `--merge-raw-coverage-file`.

`--teststart <test_start_date/time>`

[Test Advisor option] Test start date/time when using `--merge-raw-coverage-file`. Must be in the format: `yyyy-MM-dd HH:mm:ss`.

`--teststatus [pass | fail | unknown]`  
[Test Advisor option] Test status when using `--merge-raw-coverage-file`.

`--treat-as-64bit <exe-name>`  
[Deprecated as of version 8.7] This option is deprecated and will be removed from a future release. The underlying issue it addressed has been fixed, so the option is no longer needed and no longer has any effect.

## Shared options

`--config <coverity_config.xml> , -C <coverity_config.xml>`  
Uses the specified configuration file instead of the default configuration file located at `<install_dir_sa>/config/coverity_config.xml`.

`--debug, -g`  
Turn on basic debugging output.

`--debug-flags <flag> [, <flag>, ...]`  
Controls the amount of debugging output produced during a build. These flags can be combined on the command line using a comma as a delimiter.

Valid flags are `build`, `capture`, `translate`, `translate-phases`. For example, `--debug-flags build, translate`.

`--ident`  
Displays the version of Coverity Analysis and build number.

`--redirect stdout|stderr,<filename>, -rd stdout|stderr,<filename>`  
Redirect either stdout or stderr to `<filename>..`

`--tmpdir <tmp>, -t <tmp>`  
Specifies the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

`--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>`  
Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

## Exit codes

By default, `cov-build` returns the exit code from the native build. For example, if native build command returns 57, `cov-build` will return 57. However, in the case of invalid arguments, `cov-build` can return 16, which can also occur if the native build returns 16.

If you pass `--return-emit-failures` to `cov-build`, the return codes change to match the specified emit failure responses.

## See Also

`cov-configure`

cov-emit

cov-translate

cov-security-da

---

## Name

cov-capture Capture source files for analysis from the file system or from an SCM repository, without a build.

## Synopsis

The `cov-capture` command has several modes of operation, depending on the use of some key options. The following syntax diagrams illustrate each mode, which are then discussed in detail in the "Description" section.

### Project Checkout Mode

```
cov-capture --project-dir <project_dir> --dir <idir>
[--language <language>]...
[--exclude-project-file <file>]...
```

### SCM Mode

```
cov-capture --scm-url <scm_url> --dir <idir>
[--exclude-project-file <file>]...
[--language <language>]...
[--scm-type <scm_type>]
[--scm-checkout-dir <scm_dir>]
[--scm-branch <scm_branch>]
[--scm-revision <scm_revision>]
```

### Source Files Mode

```
cov-capture --source-dir <source_dir>|--source-list <source_list>...
--dir <idir>
[--case-sensitive <type>]...
[--delete-stale-tus]
[--file-glob <lang>=<glob>]...
[--file-regex <lang>=<glob>]...
[--exclude-regex <lang>=<regex>]...
[--exclude-glob <lang>=<glob>]...
[--just-print-matches]
[--language <language>]...
[--library-dir <lang>=<lib_dir>]...
[--library-file <lang>=<lib_file>]...
[--max-batch-size <size>]
[--no-friend-language <lang>]...
[--parse-error-threshold <percent>]
[--python-version <version>]
[--return-emit-failures]
```

### Config Files Mode

```
cov-capture --config-dir <config_dir>|--config-list <config_list>...
--dir <idir>
[--delete-stale-tus]
[--just-print-matches]
[--max-batch-size <size>]
```

## Description

Captures files for analysis from the file system or from an SCM repository, without a build. The `cov-capture` command operates in one of four modes: “Project”, “Project with SCM Checkout”, “Source Files”, or “Config Files”.

As a final step, this command invokes `cov-security-da`, which runs a dynamic analysis in order to perform a security assessment.

### Note

Coverity Security Dynamic Analysis for C# and Visual Basic requires requires a Windows 64-bit or Linux 64-bit system that supports .NET Core 3.1.

## Project mode (activated by `--project-dir`)

Searches for known supported project files and parses these project files to obtain information about how to capture the source files contained in the project directory.

Supported project files:

- JavaScript
  - npm, yarn: `package.json`
  - bower: `bower.json`
- Java
  - Maven: `pom.xml`
  - Gradle: `build.gradle`
- .NET Core
  - C#: solution and project files
- All other languages
  - No project files are used.

Use this mode when the project directory with source files already exists on disk and contains supported project files.

## Project with SCM Checkout mode (activated by `--scm-url`)

Automatically checks out the source code for the project to a new directory prior to entering Project mode, treating the new directory as the project directory.

`cov-capture` supports the Git SCM system, version 1.7.5 and greater, including Git sub-modules.

Use this mode when the project directory with source files does not exist on disk but can be checked out from a supported SCM system.

### Source Files mode (activated by `--source-dir` or `--source-list`)

Searches for known source files where the language is determined by filename extension.

Use this mode when your build tool is not supported, or if you need greater control over what gets captured.

### Config Files mode (activated by `--config-dir` or `--config-list`)

Searches for known configuration files.

Primarily used for languages such as Swift, which require their configuration files to be captured along with code. In this case, `cov-capture` in Config Files mode is typically invoked after a build capture.

### Supported languages and specifying a language

A number of the options described in the following section require you to specify a particular language. The `--file-glob` option is one of these. To specify a custom filename extension for Java source, for example, you would enter the following:

```
--file-glob java="*.java"
```

The table that follows shows the languages supported by `cov-capture`, and the identifiers used to specify them.

**Table 3. Supported Languages**

Language	Language Identifier
C#	cs
Java	java
JavaScript	javascript
TypeScript	typescript
PHP	php
Python	python
Ruby	ruby

### Options

`--case-sensitive <type>`

By default, the options `--exclude-regex`, `--exclude-glob`, `--file-regex`, and `--file-glob` treat their regular expression or glob pattern in a case-insensitive way. This option tells the other options to treat their regex or glob in a case-sensitive way.

The valid values for `<type>` are `regex` or `glob`.

`--config-dir <config_dir>`

Specifies the directory in which to look for configuration files. The directory must already exist.

`--config-list <config_list>`

Specifies the list of config files that should be captured. This file must already exist.

`--coverity-response-file=<response_file_name>`

Specifies a *response file* that contains a list of additional command-line arguments; for example, a list of source directories.

Each line in the file is treated as a single argument. The option ignores quotes and spaces or other white space. `cov-capture` reads the file using the default character encoding of the platform on which Coverity Analysis is being run; that is, of Linux, macOS, or Windows.

`--delete-stale-tus`

Automatically deletes translation units that are created from source files that were renamed or removed. This capability is off by default. Use this command when you perform an incremental build after deleting or renaming source files.

`--dir <idir>`

This required value specifies the intermediate directory to emit to.

`--exclude-glob <lang>=<glob>`

Specifies a glob pattern, `<glob>`, used to exclude source files of the specified language from being captured. The glob pattern is matched against the entire file path, relative to the source directory where the file was found.

For example, given the following directory structure:

```
/path/to/cwd
/path/to/cwd/src
/path/to/cwd/src/catchMe.java
```

... and the following basic command invocation:

```
% cov-capture --source-dir src
```

... then adding the option `--exclude-glob java="*catchMe.java"` would exclude the file `catchMe.java` from the capture, but adding the option `--exclude-glob java="src/*catchMe.java"` would *not* exclude it.

`--exclude-project-file <file>`

Indicates that the project file should be excluded from capture. If this path is absolute, it is used as-is; otherwise, it is understood as relative to the project directory.

`--exclude-regex <lang>=<regex>`

Specifies a regular expression that is used to exclude source files of the specified language from being captured.

The regular expression is matched against the entire file path, relative to the source directory where the file was found.

For example, given the following directory structure:

```
/path/to/cwd
/path/to/cwd/src
/path/to/cwd/src/catchMe.java
```

... and the following basic command invocation:

```
% cov-capture --source-dir src
```

... then adding the option `--exclude-regex java=".*catchMe.java$"` would exclude the file `catchMe.java` from the capture, but adding the option `--exclude-regex java=".*src/catchMe.java"` would *not* exclude it.

`--file-glob <lang>=<glob>`

Specifies a glob pattern used for matching the names of files belonging to the specified language.

You can include this option multiple times for a single invocation of `cov-capture`

`--file-regex <lang>=<regex>`

Specifies a regular expression used for matching the names of files belonging to the specified language.

You can include this option multiple times for a single invocation of `cov-capture`

`--just-print-matches`

Tells `cov-capture` to simply print out the names of the files it would have captured. When this option is present, the command does not actually capture those files.

`--language <language>`

Explicitly enables a language for capture. Only those languages specified by this option will be enabled. If this option is absent, the tool captures all supported languages. See “Supported languages and specifying a language”, above.

`--library-dir <lang>=<lib_dir>`

Specifies a library directory in which to look for library dependencies for the specified language. The specified directory will be searched recursively.

`--no-friend-language <lang>`

By default, `cov-capture` captures files for both the specified language and related languages. For example, `--language java` also captures JSP and configuration files. This option prevents such “friend” language files from being captured.

The value of `<lang>` can be one of the following: `android`, `config-files`, `html`, `jsp`, `jsx`, or `vue`.

**--no-security-da**

Disables the dynamic analysis—that is, the execution of `cov-security-da`—that is typically run at the end of the capture. The results of the dynamic analysis are used for a security analysis.

**--library-file <lang>=<lib\_file>**

Specifies an additional library file for the specified language.

**--parse-error-threshold <percentage>**

The percentage of translation units that must successfully compile. If this percentage is not achieved, `cov-capture` generates a warning when it completes. The default value is 95.

This option has no effect unless `--return-emit-failures` is provided as well.

**--project-dir <project\_dir>**

Specifies the directory in which to look for project files. These project files will be used to obtain the list of source files to capture, along with their dependencies. The directory must already exist.

**--python-version <versionNumber>**

Specifies the major version of the Python language being used: either 2 or 3.

If this option is not present, `cov-capture` first attempts to compile a Python script using version 2 of the Python language. If a syntax error occurs, `cov-capture` attempts to compile the script as Python 3.

**--return-emit-failures**

Causes the `cov-capture` command to return with an error code if an emit failure occurs. The return value is a combination (a bitwise OR) of the following flags:

**Table 4. Codes for Emit Failures**

Code	Meaning
4	No files were emitted.
8	Some files failed to compile.  By default, this error code is returned if fewer than 95% of the compilation units compiled. You can change this threshold percentage by specifying <code>--parse-error-threshold</code> .

**--scm-branch <scm-branch>**

Specifies the SCM branch to check out. Do not use with `--scm-revision`. Examples for `git`:

- `master`
- `next`

**--scm-checkout-dir <scm\_dir>**

Specifies the directory where the source code will be checked out. If not specified, this defaults to the name of the repository in the current working directory. The directory must *not* exist before invoking `cov-capture`.

`--scm-revision <scm-revision>`

Specifies the SCM revision to check out. Must not be used with `--scm-branch`. Examples for `git`:

- tag: v1.9, STABLE
- commit: abcd094354

`--scm-type <type>`

Specifies the type of SCM to use when checking out the code. If this option is absent, `cov-capture` will detect the type of SCM from the URL. Supported SCM systems: `git`

`--scm-url <scm_url>`

Specifies a URL for a source repository that contains the code to capture. The format of the URL is specific to each SCM system and must be accepted by the SCM system in use. Supported SCM systems: `git`

Examples of `git` URLs:

- `/srv/git/project.git`
- `git://example.com/group/project.git`
- `https://github.com/example/project.git`
- `git@github.com:example/project.git`
- `ssh://git@github.com:example/project.git`

`--source-dir <source_dir>`

Specifies the directory in which to look for source files. The directory must already exist.

`--source-list <source_list>`

Specifies the list of source files that should be captured. This file must already exist.

## Shared options

`--debug, -g`

Turn on basic debugging output.

`--ident`

Displays the version of Coverity Analysis and the build number.

`--redirect stdout|stderr,<file_name>, -rd stdout|stderr,<file_name>`

Redirects either the `stdout` or the `stderr` stream to the specified file.

## Exit codes

- 0: The command successfully completed the requested task.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.

- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic or debugging output, such as a stack trace.

### **See Also**

cov-build

cov-security-da

---

## Name

`cov-collect-models` Gather all C/C++ function models from an analyzed intermediate directory into a single model file.

## Synopsis

```
cov-collect-models [--input <file> | -if <file>] [--output <file.xmlldb> | -of
<file.xmlldb>] [--make-dc-config] [--text]
```

## Description

The `cov-collect-models` command gathers all of the function models from a C/C++ intermediate directory previously analyzed with `cov-analyze` and collects them into a single output model file. This model file can be subsequently passed to `cov-analyze` with the `--model-file` option.

The primary purpose of `cov-collect-models` is to allow interprocedural information from a full analysis run to be used when analyzing only a small portion of the code base. This usually results in finding some interprocedural errors even when only a small portion of the code base is analyzed, and it also usually helps lower the false positive rate.



### Note

To use the derived model file on a Windows SMB network shared drive (for example, when running Coverity Desktop local analyses that use derived models), it is necessary to generate the file on a physical disk, then copy it to the shared drive for read-only access by other processes.

## Options

`--dir <intermediate_directory>`

Path name to an intermediate directory that is used to store the results of the build and analysis.

`--input <file.xmlldb>, -if <file.xmlldb>`

Instead of reading models from an intermediate directory, use this input file. This option can be used more than once. If you specify multiple input files, the models in them are merged together and placed into the output file. Models from input files that are specified first have precedence over those in files that are specified later.

`--make-dc-config`

[C/C++ only] For a description, see the `--make-dc-config` option to `cov-make-library`.

`--output-file <file.xmlldb>, -of <file.xmlldb>`

The path name for the file to store the collected models.

By default, using this option will append to the output file if the output file already exists. For example, the following command appends a new `model-11.xmlldb` file to the `all-models` collection

```
$ cov-collect-models --input model-1.xmlldb --output-file all-models.xmlldb
```

`--output-tag <name>`

Use this option if you used it when generating analysis results. See the `--output-tag` option to `cov-analyze`.

**--text**

Output the models as text. This format is far less efficient than the standard `.xmldb` format, but is easier to debug.

**Shared options**

**--debug, -g**

Turn on basic debugging output.

**--ident**

Displays the version of Coverity Analysis and build number.

**--info**

Displays certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

**--tmpdir <tmp>, -t <tmp>**

Specifies the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

**Exit codes**

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

**See Also**

`cov-analyze`

`cov-make-library`

---

## Name

cov-commit-defects Commit Coverity Analysis defect reports to a Coverity Connect database.

## Synopsis

cov-commit-defects

```
--dataport <port_number> | --port <port_number> | --https-port <port_number>
--dir <intermediate_directory>
--host <host_server_name>
--stream <stream_name>
[--authenticate-ssl]
[--auth-key-file <keyfile>]
[--cva]
[--description "description"]
[--encryption <requirement_level>]
[--extra-output <path>]
[--output-tag <name>]
[--on-new-cert <trust | distrust>]
[--password <password>]
[--preview-report <filename> | --preview-report-v2 <filename>]
[--scm <scm_type>]
[--scm-tool <scm_tool_path>]
[--scm-project-root <scm_root_path>]
[--scm-tool-arg <scm_tool_arg>]
[--scm-command-arg <scm_command_arg>]
[--snapshot-id-file <filename>]
[--strip-path <path>]
[--target <platform>]
[--ticker-mode <mode>]
[--url <path>]
[--user <user_name>]
[--version <version>]
[OPTIONS]
```

## Description

The `cov-commit-defects` command reads analysis output and source data stored in an intermediate directory and writes the data to a Coverity Connect instance in a stream that you specify. The data are written as a unit; this unit is a snapshot.

The command passes the data to the Coverity Connect server either through the server's HTTPS port or its Commit port, depending on the command-line options you choose. You should use the HTTPS port because the Commit port is deprecated and will be removed in a future release.

To use the HTTPS port, both Coverity Connect and Coverity Analysis must be release 2020.12 or later and you must use one of the following command options:

- The `--url` option with a scheme of `https`, for example:

```
--url https://my_domain.com:8443
```

- The `--https-port` option. (This option is not recommended because it is deprecated and will be removed in a future release.)

See the `--url` option section for more information about sending data to the HTTPS port.

The Commit port is used if any of the following are true:

- Coverity Connect is older than release 2020.12.
- Coverity Analysis is older than release 2020.12.
- You use the `--dataport` command option. (This option is deprecated and will be removed in a future release.)
- You use the `--url` command option with a `commit` scheme, for example:

```
--url commit://my_domain.com:9999
```

(This scheme is deprecated and will be removed in a future release.)

 **Note**

Although you can use the HTTP port instead of the HTTPS port, HTTP is not secure and is therefore suitable only for demonstration purposes. For information on how to use the HTTP port, see the `--url` option section.

Commits conducted over the HTTPS port are always secure.

Commits conducted over the Commit port are secure by default but can be conducted without security if the Coverity administrator configures the server to not enforce security.

After you perform a commit, you can view the defects in Coverity Connect alongside the source code that generated them. The issues in the intermediate directory are discovered through the `cov-analyze` command.

 **Note**

It is possible to use the `cov-build` command to capture builds for many different languages to the same intermediate directory. The target stream's "Language" configuration setting must match the source code language in the intermediate directory. The recommended "Any" setting accepts everything, including mixed-language intermediate directories.

After a successful commit, `cov-commit-defects` checks for any new Coverity Analysis updates. If there are updates, a message appears with the number of updates that you can download. Use the `cov-install-updates` command to manage and install the updates.

 **Note**

SCM-related command line options (`--scm*`) are used to collect SCM (source code management) data solely for the purposes of automatic ownership assignment (see `cov-blame` and *Coverity Platform 2020.12 User and Administrator Guide* [↗](#)). To display SCM data in the Coverity Connect source browser, use `cov-import-scm` prior to running `cov-analyze`.

This command requires that source files remain in their usual locations in the checked-out source tree. If the files are copied to a new location after checkout, the SCM query will not work.

## Options

### `--authenticate-ssl`

Reject self-signed certificates when doing the SSL/TLS handshake. You cannot use this option together with the `--on-new-cert distrust` option.

### `--auth-key-file <keyfile>`

Specify the location of a previously created authentication key file, used for connecting to the Coverity Connect server.

### `--certs <filename>`

In addition to CA certificates obtained from other trust stores, use the CA certificates in the given `filename`. This file is in PEM format.

### `--cid-assignment-timeout <timeout-seconds>`

When committing with `--preview-report` or `--preview-report-v2` to an instance of Coverity Connect in a clustered environment, assignments of CIDs to issues can delay the completion of the preview report. This option specifies the number of seconds to wait for this phase of preview report processing to complete. If assigning CIDs takes longer than `<timeout-seconds>`, Coverity Connect will leave some of the CIDs unassigned. They will have null values in the preview report. The default CID assignment timeout is 60 seconds.

### `--comparison-snapshot-id <snapshot-id>`

This option is used in conjunction with the `--preview-report-v2` option to specify the snapshot with which the preview report will compare the commit's defect instances. A boolean flag, called `presentInComparisonSnapshot`, is included in the preview report indicating whether each of this commit's defect occurrences is present in the given snapshot. The default value is the most recent snapshot ID in the specified stream.

### `--cva`

Generate architectural information for a C/C++ snapshot. After using this option, Coverity Connect users can download this snapshot's `.cva` file. This option is not valid for any other languages.

For information about downloading the `.cva` file, see the Architecture Analysis appendix in the *Coverity Platform 2020.12 User and Administrator Guide* [↗](#).



### Note

This option has been deprecated. Use the `cov-export-cva` command instead.

### `--dataport <port_number>`

You should use the `--url` option instead of this option. This option is deprecated and will be removed in a future release.

Used with the `--host` option to specify the Commit port on Coverity Connect. You can use only one of `--port`, `--dataport`, or `--https-port` to specify the Commit port.

`--description <description>`

Specify a description for the committed snapshot.

`--dir <intermediate_directory>`

Path name to an intermediate directory that is used to store the results of the build and analysis.

`--encryption <requirement_level>`

This option is deprecated and will be removed in a future release. You should use the `--url` option and send analysis data to the HTTPS port instead.

`cov-commit-defects` uses this option to communicate with Coverity Connect to determine if the dataport connection will be encrypted. By default, the value for `--encryption <requirement_level>` is "preferred".

The available values for `<requirement_level>` are:

`required`

The commit will proceed only if the server requires or prefers encryption. The connection will be encrypted.

`preferred`

The connection will be encrypted if the server requires or prefers encryption. Otherwise, the connection will be unencrypted

`none`

The commit will proceed only if the server prefers encryption or has an encryption setting of none (meaning it requires no encryption). The connection will be unencrypted.

`--extra-output <path>, -xo <path>`

[Deprecated] This option is deprecated as of version 5.5 and subject to removal or change in a future release.

Specify additional output directories from parallel analysis snapshots. Use this option for each output directory, in addition to the default `<intermediate_directory>/output` directory, that you want to commit into a single snapshot ID in the Coverity Connect.

`--host <server_hostname>`

You should use the `--url` option instead of this option. This option is deprecated and will be removed in a future release.

Specify the server hostname to which to send the results. The server must have a running instance of Coverity Connect.

If unspecified, the default is the `host` element from the XML configuration file.

 **Note**

- If you're running `cov-commit-defects` on a Linux OS, or using `--ssl`, you must enter the full host and domain name for the `--host` parameter:

```
--host <server_hostname.domain.com>
```

- The `--host` switch, while still supported, now produces a deprecation warning that it may be removed in a later release. The `--url` syntax is the preferred replacement.

`--https-port <port_number>`

You should use the `--url` option instead of this option. This option is deprecated and will be removed in a future release.

If both Coverity Connect and Coverity Analysis are at release 2020.12 or later, this option directs commit data to the Coverity Connect server's HTTPS port.

If either Coverity Connect or Coverity Analysis are older than release 2020.12, this option first retrieves the Coverity Connect server's Commit port number from the server's HTTPS port and then directs commit data to the Commit port.

This option requires the `--host` option.

`--misra-only`

[Deprecated in 8.0] Using this option will result in an error.

`--noxrefs`

Tells `cov-commit-defects` to skip the phase of transferring xrefs (cross-reference data) to Coverity Connect. It is useful for debugging and doing commits where the user doesn't mind that, when viewed in the Coverity Connect, their code lacks cross-reference information. A user might prefer that if they were sensitive to the amount of time taken by commit to execute.

`--on-new-cert <trust | distrust>`

Indicates whether to trust (with `trust-first-time`) self-signed certificates, presented by the server, that the application has not seen before. Default is `distrust`. For information on the new SSL certificate management functionality, please see *Coverity Platform 2020.12 User and Administrator Guide* [🔗](#)

`--output-tag <name>`

Use this option if you used it when generating analysis results. See the `--output-tag` option to `cov-analyze`.

`--password <password>`, `-pa <password>`

You should use the `--url` option instead of this option. This option is deprecated and will be removed in a future release.

Specify the password for either the current user name, or the user specified with the `--user` option. For security reasons, the password transmitted to the Coverity Connect is encrypted. If unspecified, the default is (in order of precedence):

1. The password from the `--url` option.
2. The `password` element from the XML configuration file.
3. The environment variable `COVERITY_PASSPHRASE`.

4. The password in the file pointed to by the environment variable `COVERITY_PASSPHRASE_FILE`.

 **Note**

The passphrase can be stored in a file without any other text, such as a newline character.

 **Warning**

On multi-user systems, such as Linux, users can see the full command line of all commands that all users execute. For example, if a user uses the `ps -Awf` command, identifying information such as usernames, process identities, dates and times, and full command lines display.

This attribute supports the commit process.

**--port <port\_number>**

You should use the `--url` option instead of this option. This option is deprecated and will be removed in a future release.

If both Coverity Connect and Coverity Analysis are at release 2020.12 or later, this option directs commit data to the Coverity Connect server's HTTP port.

If either Coverity Connect or Coverity Analysis are older than release 2020.12, this option first retrieves the Coverity Connect server's Commit port number from the server's HTTP port and then directs commit data to the Commit port.

This option requires the `--host` option.

**--ssl**

Specifies that SSL is to be used for both HTTPS port and dataport connections. For the negotiation with the server on whether to use SSL on the dataport, this is the equivalent of `--encryption required`.

**--preview-report <filename>**

Instead of sending files, cross-references, and other assets to the server, this option sends only the defect occurrences. The server returns a commit preview report, which is written in JSON format, to `<filename>`.

The commit preview report uses the structure defined in the following table. Note that the order of items contained within objects or arrays is arbitrary.

There is a second version of the preview report, used by Coverity Desktop Analysis, which contains additional details for each CID. See `--preview-report-v2 <filename>` for more information.

**Table 5. Report v1 element syntax**

Report element	Comments
<pre>report &lt;- {   "header" : header,</pre>	

Report element	Comments
<pre>"analysisInfo" : analysisInfo, "issueInfo" : issueInfo, }</pre>	
<pre>header &lt;- {   "format" : "commit preview report",   "version" : 1, }</pre>	
<pre>analysisInfo &lt;- {   "command" : string,   "reportTimestamp" : string,   "user" : string, }</pre>	ReportTimestamp has format yyyy-mm-ddThh:mm:ss.mmmZ. The T separates the date from the time. The Z indicates that the timestamp is in UTC.
<pre>issueInfo &lt;- [   {     "cid" : number or null,     "mergeKey" : string,     "occurrences" : occurrences,     "triage" : triage,   }, ... ]</pre>	Each distinct issue has a unique identifier, the mergeKey. The CID may sometimes be null in clustered installations.
<pre>occurrences &lt;- [   {     "checker" : string,     "file" : string,     "function" : string,     "extra" : string,     "subcategory" : string,     "mainEvenLineNumber" : integer,     "mainEventDescription" : string,   }, ... ]</pre>	Each issueInfo has one or more occurrences, all with the same mergeKey.
<pre>triage &lt;- {   "classification" : string,   "action" : string,   "fixTarget" : string,   "severity" : string,   "owner" : string, }</pre> (one additional item for each custom attribute. Value is <i>string</i> or null.)	

**--preview-report-v2 <filename>**

Similar to `--preview-report`, this option sends only defect occurrences to the server, which then returns a commit preview report, written in JSON format, to `<filename>`. Version two of the preview report contains all of the information present in version one, with several additional fields.

The commit preview report (v2) uses the structure defined in the following table. Note that the order of items contained within objects or arrays is arbitrary.

Table 6. Report v2 element syntax

Report element	Comments
<pre>report &lt;- {   "header" : header,   "analysisInfo" : analysisInfo,   "issueInfo" : issueInfo, }</pre>	
<pre>header &lt;- {   "format" : "commit preview report",   "version" : 2, }</pre>	
<pre>analysisInfo &lt;- {   "command" : string,   "reportTimestamp" : string,   "user" : string,   "comparisonSnapshotId" : string,   "ownerAssignmentRule" : string   "ownerLdapServerName" : string, }</pre>	<p>ReportTimestamp has format yyyy-mm-ddThh:mm:ss.mmmZ. The T separates the date from the time. The Z indicates that the timestamp is in UTC.</p> <p>The comparisonSnapshotId is the snapshot identifier given by the --comparison-snapshot-id command line parameter. If not specified by --comparison-snapshot-id, this will be the ID of the most recent snapshot.</p>
<pre>issueInfo &lt;- [   {     "cid" : number or null,     "mergeKey" : string,     "occurrences" : occurrences,     "triage" : triage,     "customTriage" : customTriage,      "presentInComparisonSnapshot" : boolean,     "firstDetectedDateTime" : string,     "ownerLdapServerName" : string,    }, ... ]</pre>	<p>Each distinct issue has a unique identifier, the mergeKey. The CID may sometimes be null in clustered installations.</p> <p>The presentInComparisonSnapshot flag is true if this issue occurs in the comparison snapshot identified by the comparisonSnapshotId (listed in the analysisInfo element).</p>
<pre>occurrences &lt;- [   {     "checker" : string,     "file" : string,     "function" : string,     "extra" : string,     "subcategory" : string,     "mainEvenLineNumber" : integer,     "mainEventDescription" : string,     "componentName" : string,     "componentDefaultOwner" : string,     "componentDefaultOwner " string    }, ... ]</pre>	<p>Each issueInfo has one or more occurrences, all with the same mergeKey.</p>

Report element	Comments
]	
<pre> trriage &lt;- {   "classification" : string,   "action" : string,   "fixTarget" : string,   "severity" : string,   "owner" : string,   "legacy" : string,   "externalReference" : string, } </pre>	
<pre> customTriage &lt;- {   "quotedString" : string } </pre>	The custom triage attributes, if any, are listed here.

--product <product\_name>  
 Deprecated. See --stream.

--scm <scm\_type>  
 Specifies the name of the source control management system. For this option to function correctly, your source files must remain in their usual locations in the checked-out source tree. If the files are copied to a different location after checkout, the SCM query will not work.

Possible scm\_type values:

- Accurev: `accurev`
- Azure DevOps Server (ADS): `ads`  
 Windows only.
- ClearCase: `clearcase`
- CVS: `cvs`
- GIT: `git`
- Mercurial: `hg`
- Perforce: `perforce`
- Plastic: `plastic|plastic-distributed`.

Use `plastic` when working in a non- or partially-distributed Plastic configuration. Use `plastic-distributed` when working in a fully-distributed Plastic configuration.

- SVN: `svn`
- Team Foundation Server (TFS): `tfs`  
 Windows only.

For usage information for the `--scm` option, see `cov-extract-scm`.

 **Note**

The following commands or setup utilities must be run before `cov-commit-defects` in order to successfully communicate with the SCM server:

- `accurev`:

Login command

- `perforce`

The environment variable `P4PORT` should be set to the value expected by the p4 tool.

- `tfs` or `ads`:

Windows credentials in Credential Manager to access the TFS or ADS server

`--scm-command-arg <scm_command_arg>`

This option has been deprecated. Instead of using `--scm-command-arg arg1`, use `--scm-param annotate_arg=arg1`. Specifies additional arguments that are passed to the command that retrieves the last modified dates. This option can be specified multiple times.

For usage information for the `--scm` option, see `cov-extract-scm`.

`---scm-param`

Specify extra arguments to be passed to the SCM tool in a context-aware manner. For usage information of the `--scm` option, see `cov-extract-scm`.

`--scm-project-root <scm_root_path>`

Specifies a path that represents the root of the source control repository. This option is only used when specifying `accurev` as the value to `--scm`. When this is used, all file paths that are used to gather information are interpreted as relative to this project-root path.

For usage information for the `--scm` option, see `cov-extract-scm`.

`--snapshot-id-file <filename>`

If the commit succeeds, write the snapshot ID for this commit to the specified file, and make this file writable.

`--scm-tool <scm_tool_path>`

Specifies the path to an executable that interacts with the source control repository. If the executable name is given, it is assumed that it can be found in the `path` environment variable. If it is not provided, the command uses the default tool for the specified `--scm` system.

For usage information for the `--scm` option, see `cov-extract-scm`.

`--scm-tool-arg <scm_tool_arg>`

This option has been deprecated. Instead of using `--scm-tool-arg arg1`, use `--scm-param tool_arg=arg1`. Specifies additional arguments that are passed to the SCM tool, specified in the

`--scm-tool` option, that gathers the last modified dates. The arguments are placed before the command and after the tool. This option can be specified multiple times.

For usage information for the `--scm` option, see `cov-extract-scm`.

`--security-file <license file>, -sf <license file>`

Path to a Coverity Analysis license file. If not specified, this path is given by the `security_file` element in the XML configuration file, or `license.dat` in the same directory as `<install_dir_sa>/bin`.

`--stream <stream_name>`

Specifies a stream name to which to commit these defects.

If the stream option is not specified, the `stream` element from the XML configuration file is used.

If the stream is associated with a specific language and you attempt to commit results from other languages to that stream, the commit will fail. However, in Coverity Connect, it is possible to associate a stream with multiple languages even if the stream was previously associated with a single programming language.

`--strip-path <path>, -s <path>`

Strips the prefix of a file name path in error messages and references to your source files. If you specify the `--strip-path` option multiple times, you strip all of the prefixes from the file names, in the order in which you specify the `--strip-path` argument values.

Note that instead of using this option when committing issues to Coverity Connect through `cov-commit-defects`, you can enhance end-to-end performance by using this option with `cov-analyze` when analyzing code, or with `cov-import-results` when importing third party issues.

`--target <target_name>`

Target platform for this project (for example, `i386`).

`--ticker-mode <mode>`

Set the mode of the progress bar ticker. The available modes are:

`none`

No progress bar is displayed.

`no-spin`

Only the print stars are displayed; the spinning bar is not.

`spin`

This is the default mode. Stars with a spinning bar at the end are displayed. Each file, function, or defect committed corresponds to steps of spin.

`--url <path>`

Use this option to specify the information needed to connect to a Coverity Connect server. You should use this option instead of the `--dataport`, `--host`, `--https-port`, `--port`, and `--user` options (these options are deprecated and will be removed in a future release).

- The `--url` switch now allows a username and password to be supplied, as an alternative to `--user` and `--password`. The syntax used to supply those credentials in the URL is `https://[<USERNAME>[:<PASSWORD>]@]<HOSTNAME>[:<PORT>][/<CONTEXT_ROOT>]`, where the brackets show which parts are optional.
- The parallel construct also exists for `http://` and `commit://`.
- The `--host` switch, while still supported, now produces a deprecation warning that it may be removed in a later release. The `--url` syntax is the preferred replacement.

The value you specify for this option can have one of two forms: one used with HTTPS or HTTP, or one used with the commit scheme. Examples are provided in the following table:

Scheme	Meaning	Example
https or http	Use HTTPS or HTTP to connect to the Coverity Connect HTTPS or HTTP port. HTTPS is the preferred scheme.  For <code>http</code> , the default port is 80; for <code>https</code> , the default port is 443.	<code>https://example.com/coverity</code> <code>https://cimpop:8008</code> <code>http://cim.example.com:8080</code>
commit	Connect to the data port specified by the URL. This scheme is deprecated and will be removed in a future release.	<code>commit://cim.example.com:9999</code> <code>commit://cim.example.com</code>

Refer to the following table as an aid in updating existing command lines that use the `--host`, `--port`, `--https-port`, and `--dataport` options:

Existing command form	New command form
<code>cov-commit-defects --host &lt;hostname&gt; --port &lt;http-port&gt; --encryption &lt;level&gt; ...</code>	<code>cov-commit-defects --url https://&lt;hostname&gt;:&lt;https-port&gt; ...</code>
<code>cov-commit-defects --url http://&lt;hostname&gt;:&lt;http-port&gt; --encryption &lt;level&gt; ...</code>	For example:
<code>cov-commit-defects --url commit://&lt;hostname&gt;:&lt;commit-port&gt; --encryption &lt;level&gt; ...</code>	<code>cov-commit-defects --url https://admin:1256@coverity_server1 --stream xalan --dir xalan_int_dir</code>

#### `--user <user_name>`

You should use the `--url` option instead of this option. This option is deprecated and will be removed in a future release.

Specifies the user name that is shown in Coverity Connect as having committed this snapshot. If unspecified, the default is:

1. The username specified by the `--url` option, if any.
2. The `user` element from the XML configuration file.

3. The environment variable `COV_USER`.
4. The environment variable `USER`.
5. The name of the operating system user invoking the command (where supported).
6. The UID of the operating system user invoking the command (where supported).
7. `admin`.

`--version <version>`  
This snapshot's project version.

### Shared options

`--config <coverity_config.xml> , -C <coverity_config.xml>`  
Uses the specified configuration file instead of the default configuration file located at `<install_dir_sa>/config/coverity_config.xml`.

`--debug, -g`  
Turn on basic debugging output.

`--ident`  
Displays the version of Coverity Analysis and build number.

`--info`  
Displays certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

`--tmpdir <tmp>, -t <tmp>`  
Specifies the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

`--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>`  
Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

### Exit codes

This command returns the following exit codes:

- 0: Success.
- 3: Non-fatal error.
- Other: Internal error.

Any errors during a commit are recorded in Coverity Connect in the `<install_dir_cc>/logs/cim.log` file. Errors and warnings are also recorded to the `<intermediate-dir>/output/commit-error-log.txt` file.

## Examples

Commit data to host `coverity_server1` for the `xalan` stream:

```
> cov-commit-defects --url https://admin:1256@coverity_server1:8443 --stream xalan --dir xalan_int_dir
```

Commit data to host `coverity_server1` for the `xalan` stream, using an XML configuration file for all settings except the intermediate directory:

```
> cov-commit-defects --dir xalan_int_dir --config test_cim_commit.xml
```

The `test_cim_commit.xml` XML configuration file contents are shown next:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE coverity SYSTEM "coverity_config.dtd">
<coverity>
 <config>
 <cim>
 <host>coverity_server1</host>
 <client_security>
 <user>admin</user>
 <password>1256</password>
 </client_security>
 <commit>
 <port>9090</port>
 <source-stream>xalanSource</source-stream>
 </commit>
 </cim>
 </config>
</coverity>
```

The `port` element in this example refers to the commit port (equivalent to the `--dataport` option).

---

## Name

cov-configure Create a configuration for a native compiler or scripting language, and generate a `coverity_config.xml` file.

## Synopsis

CUDA (nvcc):

```
cov-configure [--config <cov_config_file>] --cuda
```

Clang:

```
cov-configure [--config <cov_config_file>] --clang
```

Go:

```
cov-configure [--config <cov_config_file>] --go
```

GNU C/C++ compiler (gcc/g++):

```
cov-configure [--config <cov_config_file>] --gcc
```

Java filesystem capture:

```
cov-configure [--config <cov_config_file>] --javafs [--no-capture-config-files]
[--no-android] [--no-jsp]
```

JavaScript:

```
cov-configure [--config <cov_config_file>] --javascript [--no-html] [--no-jsx]
[--no-typescript] [--no-vue] [--no-capture-config-files] [--fs-library-path path1
--fs-library-path path2 ...]
```

Kotlin:

```
cov-configure [--config <cov_config_file>] --kotlin [--no-capture-config-files]
```

Microsoft C/C++ compiler (cl):

```
cov-configure [--config <cov_config_file>] --msvc
```

Microsoft C# compiler (csc):

```
cov-configure [--config <cov_config_file>] --cs
```

### Note

If your system uses .NET to compile C#, `cov-configure` correctly sets up that environment.

Microsoft Visual Basic compiler (vbc):

```
cov-configure [--config <cov_config_file>] --vb
```

**Note**

If your system uses .NET to compile Visual Basic, `cov-configure` correctly sets up that environment.

Oracle Java compiler (javac):

```
cov-configure [--config <cov_config_file>] --java [--no-capture-config-files] [--no-android] [--no-jsp]
```

PHP:

```
cov-configure [--config <cov_config_file>] --php [--no-capture-config-files]
```

Python:

```
cov-configure [--config <cov_config_file>] --python [--no-capture-config-files] [--version 2|3]
```

Ruby:

```
cov-configure [--config <cov_config_file>] --ruby
```

Scala:

```
cov-configure [--config <cov_config_file>] --scala [--no-capture-config-files]
```

Swift:

```
cov-configure [--config <cov_config_file>]
(--swift [--no-capture-config-files] | --swiftc [--no-capture-config-files])
```

TypeScript:

```
cov-configure [--config <cov_config_file>] --typescript [--no-html] [--no-jsx]
[--no-javascript] [--no-vue] [--no-capture-config-files] [--fs-library-path path1 [--fs-library-path path2 ...]]
```

Other compiler:

```
cov-configure [--config <cov_config_file>] [--template]
--compiler <name> --comptype <type> [--version <comp_version>]
[--cygpath <path>] [--cygwin] [--force]
[--xml-option=[tag][@<language>]]
```

Filesystem capture with custom file pattern:

```
cov-configure [--config <cov_config_file>] --comptype <type>
(--file-glob <glob>|--file-regex <regex>)
[--xml-option=[tag][@<language>]]
```

**Description**

The `cov-configure` command creates a configuration for a compiler (or compiler family) and/or a scripting language, such as JavaScript. Choices in this configuration file impact filesystem capture.

The `--config` option specifies the name of the configuration file. By default, if no other configuration file or directory is specified, the configuration is created at `<install_dir>/config/coverity_config.xml`. Each invocation of the `cov-configure` command adds a given compiler's configuration in its own subdirectory under the directory that contains the output configuration file. Within each compiler's configuration subdirectory the `coverity_config.xml` contains an include directive for that compiler-specific configuration.

 **Note**

On some Windows platforms, you might need to use Windows administrative privileges when you run `cov-configure`.

Typically, you can set the administrative permission through an option in the right-click menu of the executable for the command interpreter (for example, `cmd.exe` or Cygwin) or Windows Explorer.

### Friend compilers (Android, HTML, JSP, JSX, TypeScript, Vue.js SFC, and configuration files)

The configuration templates for certain languages automatically configure "friend compilers", which tell `cov-build` to capture additional files for related applications in the configured language. For example, `cov-configure --javascript` automatically configures the capture of HTML files with the file-include pattern `*.(htm|html)`. Likewise, `cov-configure --java` automatically configures the capture of Android configuration files (with a file-include pattern of `*.xml`) and JSP configuration files (with a file-include pattern of `*.(jsp|jspx)`). See "Table 3. Default Files for Coverity Analysis Compilers" for additional file patterns configured by `cov-configure --javascript`.

Java, JavaScript, and many other language configurations include the capture of miscellaneous files used for understanding the application and framework configuration, and in `TEXT.CUSTOM_CHECKER` checkers support. For more information about the `TEXT.CUSTOM_CHECKER` checker, see the [TEXT.CUSTOM\\_CHECKER](#) .

### Filesystem capture configurations

A filesystem capture configuration (for languages like Java, JavaScript, PHP, Python, or Ruby) can be generated using template files that are provided. For example, the following command uses the default template to generate a configuration for Python:

```
cov-configure --python --config my-python-config.xml
```

The generated configuration specifies a set of file-include and `-exclude` patterns. These file-include patterns define which files will be captured (or excluded) when the code is captured for analysis. The configuration also associates the include patterns with their corresponding Coverity analysis compiler. The `cov-build` command then uses the configuration file to determine the set of files it will emit during filesystem capture.

The include and exclude patterns are matched with case-insensitive matching.

The sample command above, using the default Python template, generates a configuration with a file-include pattern of `*.py`, and associates these files with the Python compiler. When you invoke `cov-`

`build --fs-capture-search <search directory>`, `cov-build` recursively searches the specified directory for files that match the `*.py` filename pattern and emits those files so they will be analyzed.

The following table shows these default file-include patterns and their associated compilers for JavaScript. For other languages (Java, Python, PHP, Ruby) you can refer to the default configuration templates.

**Table 7. Default Files for JavaScript Coverity Analysis Compilers**

Language	File Types
<code>--javascript</code>	<code>*.js, *.xsjs, *.xsjlib, *.map</code> → JavaScript compiler
	<code>*.html, *.htm</code> → HTML compiler (unless <code>--no-html</code> )
	<code>*.vue</code> → Vue.js Single File Component compiler (unless <code>--no-vue</code> )
	<code>*.ts, *.tsx, tsconfig.json</code> → Typescript compiler (unless <code>--no-typescript</code> )
	<code>*.jsx</code> → JSX compiler (unless <code>--no-jsx</code> )
<code>--typescript</code>	Alias for <code>--javascript</code> , with support for a <code>--no-javascript</code> option to suppress capture of <code>*.js, *.xsjs, *.xsjlib, and *.map</code> files.

You can further customize the default capture configurations by using the `--file-glob` or `--file-regex` options to extend the list of file-include patterns.

To add a new filename extension for HTML files—for example, `.ihtm`—you could execute the following command lines:

```
cov-configure --javascript -config my-js-config.xml
cov-configure --comptype html --file-glob "*.ihtm" --config my-fs-config.xml
```

You can also create a custom configuration from scratch, without using or altering the default templates. To do so, use `--file-glob` or `--file-regex` in conjunction with `-comptype`. For example, you might go through the following steps:

1. `cov-configure --comptype php --file-glob "*.php(5|7)" -c new-php-config.xml`
2. `cov-configure --comptype php --file-glob "*.phtml" -c new-php-config.xml`

For more details, see the descriptions of the `--file-regex`, `--file-glob`, and `--comptype` options.

## Language-specific Configurations

The following sections explain build-capture and filesystem-capture configurations for each supported language.

## C/C++ compiler and build capture configuration

In general, for C/C++ `cov-configure` tries to make an intelligent guess as to the native compiler's built-in macro definitions and system include directories. For some compilers, such as `gcc`, there are command line arguments that reveal this information, and `cov-configure` invokes the native compiler to discover this information. For some compilers, there is no standard way of getting this information, so `cov-configure` tries several methods to gather this information. However, these methods are not perfect and sometimes a configuration is generated that is incomplete or incorrect, with the result that some obscure parsing error occurs during the parsing of some source file or header file. Some manual configuration could be necessary. See the compiler information in the *Coverity Analysis 2020.12 User and Administrator Guide*.

Because the `cov-configure` command invokes the native compiler to determine its built-in macro definitions and the system include directories, you must run it in an environment that is identical to the one in which your native compiler runs. Otherwise, the emulation will be inaccurate.

## CUDA build capture configuration

Use the `cov-configure --cuda` command to configure build capture for CUDA.

## C# build capture configuration

Use the template configuration, `cov-configure --cs`, unless you have a clear understanding of the alternative C# configuration option, as described in the paragraph that follows.

If you do not use the template configuration, you must specify the location of the C# compiler, `csc`, to the `--compiler` option. If you do not use the template configuration, the `cov-build` command will capture only those C# compile commands that have the same absolute pathname as a compiler that was identified by a preceding `cov-configure` command. Any symbolic links in the pathnames of the configured and executed compile commands will be replaced by real directory names before comparison. For example, on a Unix system, the `inode` of the compile commands must match. Because the pathnames to the C# compile command can vary, you need to configure all of the pathnames in use, or to define a custom template configuration to handle all of them.

## Clang build capture configuration

Use the configuration, `cov-configure --clang`.

## Go build capture configuration

Use the `cov-configure --go` command to configure build capture for Go source code.

## Java build capture configuration

Use the template configuration, `cov-configure --java`, unless you have a clear understanding of the alternative Java configuration option.

If you do not use the template configuration, you must specify the location of the Java compiler, `javac`, to the `--compiler` option. This specification configures both the compiler and the virtual machine

(`java`, `java.exe`, or `javaw.exe`). If you do not use the template configuration, the `cov-build` command will capture only those Java compile commands that have the same absolute pathname as a compiler that was identified by a preceding `cov-configure` command. Any symbolic links in the pathnames of the configured and executed compile commands will be replaced by real directory names before comparison. For example, in a Unix system, the `inode` of the compile commands must match. Because the pathnames to the Java compile command can vary, you need to configure all of the pathnames in use or to define a template configuration to handle all of them.

The Java template configuration also enables filesystem capture for the following file types:

- You can limit which specific configuration files are captured by using the `--fs-capture-list` option with the `cov-build` command.
- JavaServer Pages (JSPs) for files with the `.jsp` and `.jspx` extensions. The filesystem capture of JSPs can be disabled by using the `--no-jsp` option with the `cov-configure` command.
- Java Android files that are needed by Coverity Analysis, including the manifest (`AndroidManifest.xml`) and the layout resource files. The filesystem capture of Java Android files can be disabled by using the `--no-android` option along with the `cov-configure` command.

For information about other compilers, see the section called “Friend compilers (Android, HTML, JSP, JSX, TypeScript, Vue.js SFC, and configuration files)”.

### Java filesystem capture configuration

Use the template configuration, `cov-configure --javafs` command syntax to enable Java filesystem capture.

The generated configuration matches Java source files (such as `*.java`). When using the `cov-build --fs-capture-search` command, Coverity Analysis will recursively search the specified `<search_directory>` for matching Java source files. You can customize the match pattern by using the `--file-glob` option and `--file-regex` options along with the specified pattern.

The Java filesystem template configuration also enables filesystem capture for the following file types:

- You can limit which specific configuration files are captured by using the `--fs-capture-list` option with the `cov-build` command.
- JavaServer Pages (JSPs) for files with the `.jsp` and `.jspx` extensions. The filesystem capture of JSPs can be disabled by using the `--no-jsp` option with the `cov-configure` command.
- Java Android files that are needed by Coverity Analysis, including the manifest (`AndroidManifest.xml`) and the layout resource files. The filesystem capture of Java Android files can be disabled by using the `--no-android` option along with the `cov-configure` command.



#### Note

It is an error to enable Java filesystem capture and also Java build capture (using the `--java` option). Any pre-existing Java build configuration must be deleted before filesystem capture is configured. For further details, see Section 1.4.2.3. “Filesystem capture (for Java)” in the *Coverity*

*Analysis 2020.12 User and Administrator Guide* [↗](#). See also the section called “Friend compilers (Android, HTML, JSP, JSX, TypeScript, Vue.js SFC, and configuration files)”.

### JavaScript-related filesystem capture configuration

Use the template configuration `cov-configure --javascript` command syntax to enable JavaScript filesystem capture.

By default, `cov-configure --javascript` configures filesystem capture to search for files ending in `*.js`, `*.jsx`, `*.htm`, `*.html`, `*.map`, `*.ts`, `tsconfig.json`, `*.tsx`, `*.vue`, `*.xsjs`, and `*.xsjslib` and to emit their code for later analysis.

The optional `--no-html` option excludes `*.html` and `*.htm` files from the emit. Similarly, `--no-jsx` excludes `*.jsx` files; `--no-typescript` excludes `*.ts`, `*.tsx`, and `tsconfig.json` files; and `--no-vue` excludes `*.vue` files.

You can limit which specific configuration files are captured by using the `--fs-capture-list` option with the `cov-build` command.

See also, the section called “Friend compilers (Android, HTML, JSP, JSX, TypeScript, Vue.js SFC, and configuration files)”.

### Kotlin build capture configuration

Use the `cov-configure --kotlin` command to configure build capture (to capture Kotlin source files from your build) and filesystem capture (to capture configuration files). The Kotlin configuration also enables filesystem capture of configuration files by default. You can limit which specific configuration files are captured by using the `--fs-capture-list` option with the `cov-build` command.

### PHP filesystem capture configuration

Use the template configuration `cov-configure --php` syntax to enable PHP filesystem capture.

By default, `cov-configure --php` configures filesystem capture for files with the `*.php`, `*.phtml`, `*.php3`, `*.php5`, and `*.php7` filename extensions.

You can limit which specific configuration files are captured by using the `--fs-capture-list` option with the `cov-build` command.

### Python filesystem capture configuration

Use the template configuration `cov-configure --python` syntax to enable Python filesystem capture.

By default, the `cov-configure --python` configures filesystem capture for files with the `*.py` filename extension.

#### ⚠ Caution

This command does *not* detect Python scripts that don't have this filename extension.

You can limit which specific configuration files are captured by using the `--fs-capture-list` option with the `cov-build` command.

## Ruby filesystem capture configuration

Use the `cov-configure --ruby` command to configure filesystem capture for Ruby source code.

By default, the `cov-configure -ruby` configures filesystem capture for files with the `*.rb` filename extension.

### ⚠ Caution

This command does *not* detect Ruby scripts that don't have this file name extension.

## Scala build capture configuration

Use the `cov-configure --scala` command to configure build capture (to capture Scala source files from your build) and filesystem capture (to capture configuration files). The Scala configuration also enables filesystem capture of configuration files by default.

You can limit which specific configuration files are captured by using the `--fs-capture-list` option with the `cov-build` command.

## Swift build configuration

If you use the legacy build system in Xcode, use `cov-configure --swiftc` to configure build capture so that `cov-build` will capture Swift source files. If you use the new build system, use `cov-configure --swift` to configure build capture so that `cov-build` will capture Swift source files. By default, Xcode uses the new build system; in Xcode, you can check the current preference by choosing File > Project/Workspace Settings > Build System.

Both the `--swiftc` and `--swift` options configure filesystem capture to capture configuration files such as `.plist` files for iOS apps. (For Swift projects, filesystem capture does not capture source files.)

## Visual Basic build capture configuration

Use the template configuration, `cov-configure --vb`, unless you have a clear understanding of the alternative Visual Basic configuration option, as described in the paragraph that follows.

If you do not use the template configuration, you must specify the location of the Visual Basic compiler, `vb`, to the `--compiler` option. If you do not use the template configuration, the `cov-build` command will capture only those Visual Basic compile commands that have the same absolute pathname as a compiler that was identified by a preceding `cov-configure` command. Any symbolic links in the pathnames of the configured and executed compile commands will be replaced by real directory names before comparison. For example, on a Unix system, the `inode` of the compile commands must match. Because the pathnames to the Visual Basic compile command can vary, you need to configure all of the pathnames in use, or to define a custom template configuration to handle all of them.

## Options

--

Indicate the end of `cov-configure` options. Following this option, you can specify additional compiler options. For example, GNU compiler installations that use a non-standard path to the `cpp0` preprocessor require the additional GNU `-B` option to specify its path:

---

## cov-configure

---

```
> cov-configure --compiler gcc -- -B/home/coverity/gcc-cpp0-location/bin
```

If your build explicitly uses the GNU compiler on the command line with either the `-m32` or `-64` option, also supply the option to the `cov-configure` command. For example:

```
> cov-configure --compiler gcc -- -m32
```

### `--compiler <name>, -co <name>`

Specify the compiler to configure. If the compiler `<name>` is not in the `PATH`, specify the full pathname to the compiler. To specify additional compiler options, use `--` followed by the options, for example:

```
> cov-configure --comptype gcc --compiler C:\Mingw\bin\gcc.exe -- -D__STDC__
```

### `--comptype <type>, -p <type>`

Specify the type of compiler to configure. In many cases, `cov-configure` guesses the compiler type based on the `--compiler` argument, but if the name of your compiler is non-standard, specify `--comptype`.

As a general rule, never configure a compiler as a C++ compiler. Do so only in the case of a particular problem that you are trying to work around. If you configure a compiler as a C compiler, `cov-configure` will automatically take care of the C++ case.

To see a full list of supported compiler types, run the `cov-configure --list-compiler-types` option.

For information about configuring compilers, see the *Coverity Analysis 2020.12 User and Administrator Guide* [🔗](#).

### `--coverity-response-file=<response_file>`

Specify a "response file" that contains a list of additional command line arguments, such as a list of input files. Each line in the file is treated as one argument, regardless of spaces, quotes, etc. The file is read using the platform default character encoding.

### `--delete-compiler-config`

This option accepts a compiler configuration by an absolute path or a path relative to the top-level configuration file. Only configurations specified in the top-level configuration will be deleted, otherwise this command has no effect.

**Example:** Top-level configuration file, `conf/config.xml` contains three configurations:

1. `template-gcc-config-0`
2. `template-msvc-config-0`
3. `template-javac-config-0`

The following example will remove the configuration `template-gcc-config-0`, leaving the remaining two configurations untouched.

```
cov-configure --delete-compiler-config template-gcc-config-0 -c conf/conf.xml
```

**--file-glob <pattern>**

[Filesystem capture only] Used in conjunction with `--comptype` to specify a glob pattern to match for source files of the specified type. This option allows you to customize the predefined file-include patterns (including also those for friend compilers) that are used when you specify one of the language-specific options (such as `--java` or `--javascript`).

For example, the following command creates a configuration for JavaScript that captures only files with a `.js` extension:

```
> cov-configure --comptype javascript --file-glob '*.js'
```

Note that the glob pattern will only match on filenames, not on directories or path information.

The glob expression is matched against the filename using case insensitive matching.

Do not use the `--file-regex` with this option.

You will receive a warning if you specify a pattern that is identical to one that in a previously generated configuration file for an interpreted language or friend compiler.

**--file-regex <pattern>**

[Filesystem capture only] Used in conjunction with `--comptype` to specify a regex pattern to match for source files of the specified type. This option allows you to customize the predefined file-include patterns (including also those for friend compilers) that are used when you specify one of the language-specific options (such as `--java` or `--javascript`).

For example, the following command creates a configuration for JavaScript that captures only files with a `.js` extension:

```
> cov-configure --comptype javascript --file-regex '^.*\.js$'
```

Note that the regular expression will only match on filenames, not on directories or path information.

The regular expression is matched against the filename using case insensitive matching.

Do not use the `--file-glob` with this option.

You will receive a warning if you specify a pattern that is identical to one that in a previously generated configuration file for an interpreted language or friend compiler.

**--force**

Generate the configuration even if the compiler specified does not behave as expected for a compiler of the specified type.

**--fs-library-path <path/to/the/library>**

[Java filesystem capture option] For Java filesystem capture, the `--fs-library-path` option is added to the class and source paths of the Java compiler. Specifying this option is the equivalent to passing the `--classpath <path/to/lib>` argument and `--sourcepath <path/to/lib>` as command line options to the Java compiler.

For example, the following command adds classes, `some.jar`, and `extrasrc` to the Java compiler's class and source paths (in this order):

```
cov-configure --config config.xml \
--javafs \
--fs-library-path path/to/classes \
--fs-library-path path/to/some.jar \
--fs-library-path path/to/extrasrc
```

[If you need to modify the class and source paths for different `cov-build` invocations, use the `cov-build` variant for `--fs-library-path` instead.] [You can also avoid modifying the paths with each build by using the `cov-configure` variant. This extends them only once in your configuration or installation.]

[JavaScript, PHP, and Python filesystem capture only] Specifies third-party library locations for JavaScript Node.js `require` modules, ECMAScript 6 module imports, JavaScript HTML `script src=` includes, and HANA XSC libraries imported with `$.import`. PHP `include/include_once/require/require_once` and Python imports. By default, the `cov-build` command resolves these inclusions and imports relative to the source file doing the inclusion/import (according to language specific rules). The `cov-build` command also attempts to resolve them relative to directories passed to the `--fs-library-path` option.

Passing directories to the `cov-configure --fs-library-path` option stores them in the configuration (for use with any `cov-build` command that uses that configuration). If you need to specify different libraries for different `cov-build` invocations, use the `cov-build` variant of `--fs-library-path` instead.

For example, the following command adds `lib3` and `lib4` as additional library paths (searched in that order):

```
cov-configure --config config.xml \
--javascript \
--fs-library-path lib3 \
--fs-library-path lib4
```

The search for the library file is permissive: If the search does not find the library at a relative path specified by this option, a second search for the filename alone (excluding the specified path) will run.

#### --javascript

[Filesystem capture only] Configures filesystem capture for JavaScript source code. Also associates files ending in `.htm`, `.html`, `.js`, `.jsx`, `.ts`, `.tsx`, and `.vue` with a configuration for JavaScript so they can be saved in the intermediate directory.

This configuration automatically excludes files that match the following regular expressions:

- `//node_modules//`
- `//jquery[^//]*[.]js$`
- `//[^//]*-vsdoc[.]js$`

See also, `--fs-library-path`, `--no-html`, `--no-jsp`, `--no-jsx`, `--no-typescript`, and `--no-vue`.

#### --list-compiler-types <output>, -lsc <output>

Lists the supported compiler types described in the `--comptype` option.

`--list-compiler-types` , `-lsct`

Generates a list of the supported compiler types. Usage:

```
cov-configure --list-compiler-types
```

`--list-configured-compilers <output>`, `-lscv <output>`

Lists the configured compilers defined in your `<install_dir_ca>/config/coverity_config.xml` file. The `output` option defines which output format you want the compiler configuration information displayed. It must be one of:

- `csv`
- `json`
- `text`

Each format displays the following categories for each configured compiler:

- Configuration name
- Configuration path ("`json`" format only)
- Compiler type
- Compiler
- Template configuration
- Enabled options/required arguments

Template configurations have "Config Args" while instantiated configurations have "Required Args". "Config Args" are used to probe the compiler along with any "Required Args" when instantiating a template configuration.

If the value for any field cannot be determined (for example, if an option is not defined in the configuration file), "null" is printed in that field instead.

The "`json`" format displays the configuration name AND the full path to the configuration in the "Config Name" and "Config Path" elements. The following example shows the "`json`" format:

```
{
 "Config Name" : "template-gcc-config-0",
 "Config Path" : "C:\\\\cygwin\\\\tmp\\\\template-gcc-config-0",
 "Compiler Type" : "gcc",
 "Compiler" : "gcc",
 "Is Template?" : "yes",
 "Config Args" : "-DBAR"
},
```

The "`text`" and "`csv`" formats show only the directory and configuration names (not the full path).

The following example shows the "text" and "csv" formats (they are identical):

```

Config Name, CompType, Compiler, Template?, Config/Required Args
-----, -----, -----, -----, -----
template-gcc-config-0,gcc,gcc,yes,null
template-gcc-config-1,gcc,g++,yes,null
template-javac-config-0,javac,javac,yes,null
template-java-config-0,java,java,yes,null
template-apt-config-0,apt,apt,yes,null

```

Note that in real usage, `$prevent$` and `$REAL_CC$` are actual paths.

#### `--list-required-arguments, -lsra`

Outputs a list of all the potential required arguments for a given compiler. Pass the `--compiler` or `--comptype` arguments to specify the compiler type on which you want `list-required-arguments` to operate.

#### `--no-android`

[Filesystem capture option] Disables the filesystem capture of Java Android files. The default behavior for the Java template configuration is to enable the filesystem capture of Java Android files that are needed by the analysis, including the manifest (`AndroidManifest.xml`) and the layout resource files.

This option is valid only when either the `--java` option or `--javafs` option is also specified.

#### `--no-capture-config-files`

This option disables the filesystem capture of miscellaneous configuration files. By default, when the Java, JavaScript, PHP, Python, Scala, or Swift templates are configured for filesystem capture, they will also capture any smaller files that aren't media file types. The `--no-capture-config-files` option overrides this default behavior. Typical files that are captured this way include XML files, `.plist` files, framework configuration files, and other kinds of textual configuration files. Capturing these files aids Coverity Analysis in understanding application and framework configuration. It also enables various checkers (including user-defined `TEXT.CUSTOM_CHECKER` checkers) to run on them and to report any potential defects. You can limit which specific configuration files are captured using the `--fs-capture-list` option with the `cov-build` command.

#### **Note**

We do not recommend using this option for anything outside of troubleshooting scenarios, or unless advanced tuning is required for your deployment.

#### `--no-header-scan`

Disables performing a header scan for macro candidates during probing of a compiler.

#### `--no-html`

[Filesystem capture option] Disables the filesystem capture of HTML files. The default behavior for the JavaScript template configuration is to enable the filesystem capture and HTML compilation of files with the `*.htm` and `*.html` filename extensions.

This option is valid only when the `--javascript` or `--typescript` option is also specified.

**--no-javascript**

[Filesystem capture option] Disables the filesystem capture of JavaScript files. The default behavior for the TypeScript template configuration is to enable the filesystem capture and JavaScript compilation of files with the `*.js`, `*.xsjs`, `*.xsjlib`, and `*.map` filename extensions.

This option is valid only when the `--typescript` option is also specified.

**--no-jsp**

[Filesystem capture option] Disables the filesystem capture of JavaServer Pages (JSPs). The default behavior for the Java template configuration is to enable the filesystem capture and JSP compilation of files with the `.jsp` and `.jspx` filename extensions.

This option is only valid when either the `--java` option or `--javafs` option is also specified.

**--no-jsx**

[Filesystem capture option] Disables the filesystem capture of JSX files. The default behavior for the JavaScript template configuration is to enable the filesystem capture and JSX compilation of files with the `*.jsx` filename extension.

This option is valid only when the `--javascript` or `--typescript` option is also specified.

**--no-typescript**

[Filesystem capture option] Disables the filesystem capture of TypeScript files. The default behavior for the JavaScript template configuration is to enable the filesystem capture and TypeScript compilation of files with the `*.ts`, `*.tsx`, and `tsconfig.json` filename extensions.

This option is valid only when the `--javascript` option is also specified.

**--no-vue**

[Filesystem capture option] Disables the filesystem capture of Vue.js Single File Component files. The default behavior for the JavaScript template configuration is to enable the filesystem capture and compilation of files with the `*.vue` filename extension.

This option is valid only when the `--javascript` or `--typescript` option is also specified.

**--php**

[Filesystem capture option] Configures filesystem capture for PHP source code. For supported versions of the PHP language, see "Language Support" in *Coverity Analysis 2020.12 User and Administrator Guide* [🔗](#), and for the complete PHP analysis workflow, see "Getting started with Coverity analyses" in the same guide.

**--python**

[Filesystem capture option] Configures filesystem capture for Python source code. For supported versions of the Python language, see "Language Support" in *Coverity Analysis User and Administrator Guide*.

**--ruby**

[Filesystem capture option] Configures filesystem capture for Ruby source code. For supported versions of the Ruby language, see "Language Support" in *Coverity Analysis User and Administrator Guide*.

`--set-instrument-var <[platform=platform_name,] var_name=var_value>`

Sets or overrides an instrumentation variable in the `coverity_config.xml` file. On Windows only, the optional "platform=platform\_name" allows platform-specific arguments to be specified for building a combination of 32-bit and 64-bit binaries within one project. If the platform is not specified, the `platform_name` defaults to "all", and the value is shared across all platforms.

The platforms supported on Windows are "x86" for 32-bit binaries, and "x64" for 64-bit binaries.

### Example 1:

The user wishes to deploy the libraries to a location outside of the default Coverity directory. The libraries will be colocated, since they can be distinguished by name. When not designated, the platform name defaults to "all" and will be used for both platforms:

```
cov-configure --config config/coverity_config.xml ^
 --set-instrument-var cov_lib_deployment_path=c:\common-lib-path ^
 --msvc
```

The above command will generate the following xml code in the appropriate section of the compiler configuration file:

```
<instrument_variable>
 <var_name>cov_lib_deployment_path</var_name>
 <cond_platform_value>
 <platform_name>all</platform_name>
 <var_value>c:\common-lib-path</var_value>
 </cond_platform_value>
</instrument_variable>
```

### Example 2:

The user wishes to locate the 32-bit and 64-bit versions of the Instrumentation Runtime Library in different locations on the deployment test machine:

```
cov-configure --config config/coverity_config.xml ^
 --set-instrument-var platform=x86,cov_lib_deployment_path=c:\x86-lib-path ^
 --set-instrument-var platform=x64,cov_lib_deployment_path=c:\x64-lib-path ^
 --msvc
```

The above command will generate the following xml code in the appropriate section of the compiler configuration file:

```
<instrument_variable>
 <var_name>cov_lib_deployment_path</var_name>
 <cond_platform_value>
 <platform_name>x86</platform_name>
 <var_value>c:\x86-lib-path</var_value>
 </cond_platform_value>
</instrument_variable>
<instrument_variable>
 <var_name>cov_lib_deployment_path</var_name>
 <cond_platform_value>
```

```
<platform_name>x64</platform_name>
 <var_value>c:\x64-lib-path</var_value>
</cond_platform_value>
</instrument_variable>
```

Instrument variables that are supported by default are described in the following table:

**Table 8. Instrument variables**

Name	Default Value	Comments
cov_lib_name	ci-runtime-platform	The name of the Source Code Instrumentation Library, where "platform" designates the target platform for the library. By default, either x86 or x64.
cov_lib_path	PREVENT\lib	The location of the runtime's import libraries. By default these are in the "lib" subdirectory of the Coverity installation.
cov_lib_deployment_path	PREVENT\lib	The location of the runtime's DLL libraries. By default, these are in the "lib" subdirectory of the Coverity installation, but a different location can be specified when the program runs on a machine without a Coverity installation.
cov_lib_header_path	PREVENT\sdk\runtime\ta-runtime	Location of the header files needed for source code instrumentation.
cov_lib_extra	PREVENT\sdk\runtime\ta-runtime	No default value on Windows. References the thread libraries on Linux. Can pass additional arguments to the compiler and linker.

#### --template, -tm

Provides a template configuration for building with a related set of compilers. The necessary compiler configurations are generated with the required arguments as needed during the build process. For example, if a g++ command that specified -m64 was encountered, a g++ configuration would be generated specifying the -m64 argument.

If you specify this flag, the argument to --compiler is a name of the compiler without a path. Do not use the --version option with this option.

#### Note

For the following compilers, you can generate a template configuration without using the --template option:

- For Clang:

```
> cov-configure --clang
```

- For GNU GCC and G++ (gcc and g++):

```
> cov-configure --gcc
```

- For Java (java, javac, javaw, and apt):

```
> cov-configure --java
```

- For Microsoft C/C++ (cl.exe):

```
> cov-configure --msvc
```

- For Microsoft C# (csc.exe):

```
> cov-configure --cs
```

- For Microsoft Visual Basic (vbc.exe):

```
> cov-configure -vb
```

**--template-dir <directory\_path>, -td <directory\_path>**

[Compiler Integration Toolkit (CIT) option] Specifies a template directory for custom Compiler Integration Toolkit (CIT) templates that override templates found in the default location. This option makes it possible to use custom templates without the need to modify anything in the default template directory. Multiple `--template-dir` options are allowed with directories specified in order of decreasing priority.

**--version <version>, -v <version>**

For C/C++, specify the compiler version. In many cases, cov-configure will guess the compiler version for you. For Microsoft Visual C/C++, the version is of the form "1310".

For Java, values match valid source levels to the javac compiler: '1.4', '1.5', '5', '1.6', '6', '1.7', '7'.

For Python, specify the major version of the Python language being used: '2' or '3'. By default, a Python script is compiled assuming version 2 of the Python language, and if a syntax error occurs, an attempt is made to recompile the script as Python 3.

**--xml-option <option>**

Adds user-specified XML to `coverity_config.xml`. This option is useful for adding items to the file without using an editor.

### Usage

```
--xml-option=[tag][@<language>]:value
```

- [tag] is the basic XML tag to be added, for example, `add_arg`.
- [@language] specifies that the switch is only to be added for compiler variants with the given language. Valid values are: C, C++, Java, Scala, CS, ObjC, ObjC++, or NC (where NC stands for "Not Compiled" and applies to JavaScript, PHP, Python, and Ruby). If this specifier is omitted, then the tag will be added for all compiler types being configured.
- *value* is the value contained within the tag. This can be any value, including arbitrary XML.

Simple example:

```
--xml-option=append_arg:-Ihello
```

Simple example in C config only:

```
--xml-option=append_arg@C:-Ihello
```

Arbitrary XML: `--xml-option` allows any number of XML elements. See example below, quoted as required for Windows:

```
--xml-option="<append_arg>-Ihello</append_arg><append_arg>--ppp_translator</append_arg>"
```

Arbitrary XML in C++:

```
--xml-option=@C++:"<append_arg>-Ihello</append_arg>"
```

## C/C++ options

`--cygpath <path>`

Specify the path to the directory, which contains the bin directory of the Cygwin installation, if it is not in the PATH environment variable.

`--cygwin`

On Windows, indicates that Cygwin is necessary for a GCC compiler. The `cov-configure` command can detect if Cygwin is necessary without this, but you can use this option to force Cygwin if needed.

## Shared options

`--config <coverity_config.xml> , -c <coverity_config.xml>`

Uses the specified configuration file instead of the default configuration file located at `<install_dir_sa>/config/coverity_config.xml`.

`--debug, -g`

Turn on basic debugging output.

`--ident`

Displays the version of Coverity Analysis and build number.

`--info`

Displays certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

`--redirect stdout|stderr,<filename> -rd stdout|stderr,<filename>`

Redirect either stdout or stderr to `<filename>`.

`--tmpdir <tmp>, -t <tmp>`

Specifies the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>

Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

---

## Name

cov-copy-overrun-triage (Deprecated) Migrate triage data from OVERRUN\_STATIC and OVERRUN\_DYNAMIC defects to OVERRUN.

## Synopsis

```
cov-copy-overrun-triage --host <host> --port <port> --user <user> [--password <password>]
[--triageStoreFilter <glob>] [--dryrun]
```

## Description

This command is deprecated as of version 7.0. The `cov-copy-overrun-triage` command copies triage data in a Coverity Connect instance from OVERRUN\_STATIC and OVERRUN\_DYNAMIC defects to their corresponding OVERRUN defects. OVERRUN defects that already have triage data will not be modified.

All actions performed by this command are recorded in `cov-copy-overrun-triage.log`.

Note that Coverity Connect automatically performs the primary functionality of this command.

## Options

`--dryrun`

Do not make any modifications on the server. CIDs of interest are listed in `cov-copy-overrun-triage.log`.

`--host <host>`

The server hostname on which to copy triage data. The server must have a running instance of Coverity Connect.

`--password <password>`

The password for the user specified by the `--user` parameter. If unspecified, defaults to the value of the environment variable `COVERITY_PASSPHRASE`.

### Warning

On multi-user systems, such as Linux, users can see the full command line of all commands that all users execute. For example, if a user uses the `ps -Axf` command, identifying information such as usernames, process identities, dates and times, and full command lines are displayed.

`--port <port>`

The HTTP port on the Coverity Connect host.

`--triageStoreFilter <glob>`

Pattern specifying a set of triage stores. For each triage store that matches, triage data will be copied to and from software issues (defects) that are in the same stream.

`--user <user>`

The user name that is shown in Coverity Connect as having triaged new OVERRUN defects.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## Examples

Copy triage data on host `coverity_server1` for the Default Triage Store:

```
> cov-copy-overflow-triage \
 --host coverity_server1 \
 --port 8080 \
 --user admin \
 --password 1256 \
 --triageStoreFilter "Default Triage Store"
```

Copy triage data on host `coverity_server1` for all streams:

```
> COVERITY_PASSPHRASE=1256 \
 cov-copy-overflow-triage \
 --host coverity_server1 \
 --port 8080 \
 --user admin
```

---

## Name

cov-count-lines Perform a line count of source code.

## Synopsis

```
cov-count-lines [--code-identity-file <filename>] [--file <filename>] [--list <file-list>] [--search <path>] [--search-extensions <list of extensions>] [--third-party-regex <regex>]
```

## Description

Count the number of source code lines in the file(s) specified that are available for Coverity pricing.

Coverity analyzes the number of lines based on the code collected. This may include some third-party code, which is included by default in the line count because it is part of the full code that gets compiled. However, you can exclude third-party code and other files (specifically, test code and generated code) from the line count and the analysis with the `--third-party-regex` option.

When counting lines, Coverity strips away blank lines and comments, but does not strip away single braces or parentheses (comments and blank lines are not counted as lines). The command uses the name and contents of the file to identify its language and how it should be parsed.

## Options

`--code-identity-file <filename>`

Creates a filename for a code base identity, stored as a `.cbi` file. The content of the `.cbi` file specifies the source files which are to be included and excluded from the analysis. When this option is specified, the `cov-count-lines` command line console output will also end with a hash of this file.

For example:

```
$ cov-count-lines --search . --third-party-regex files_to_exclude \
 --third-party-regex dir --code-identity-file my-example.cbi
File: /path/to/source/example1.java; Analyzable lines: 8
File: /path/to/source/example2.cc; Analyzable lines: 12
File: /path/to/source/example3.cc; Analyzable lines: 14
File: /path/to/source/example4.h; Analyzable lines: 4
Total Analyzable Lines for Coverity Pricing: 38
Code Base Identity Hash:
910b8f9b8699597dc0334ac3dc5af6fc0b61c7a95aa38edb553d63062088c2b9
```

If you do not provide the full path to the `--search` option, a stack trace is produced as shown below

```
cov-count-lines --search .
filename-class.cpp:1866: assertion failed: getWithFilesystemStripped expects an
absolute path name
call stack backtrace:
cov-count-lines linux64 2020.03
0x448760
0x4488c1
```

```
0x44b84e
0x44b94c
0x465b88
0x4222ed
0x425614
0x455295
0x414745
libc.so.6 linux64 2020.03
0x21b97
```

Once the identity hash has been generated, you can use it to create a license file. The `.cbi` file that has been created and saved will then be used to run `cov-analyze`.

 **Note**

Coverity recommends that you store the code identity file in your source control management (SCM) system. This prevents anyone else from changing the content of the code base after a license has been issued. The code identity file should only be updated when a Coverity license needs to be re-issued.

See also, the `cov-analyze` version of `--code-identity-file`.

`--file <filename>`

Counts the lines in `<filename>`. You can use absolute or relative file/path names.

`--list <file-list>`

Specifies a text file (`<file-list>`), which contains one filename per line. The filenames can be absolute or relative, but relative filenames must be relative to the current working directory.

`--search <path>`

Searches for all source files in the specified path and its subdirectories, recursively. This option may be specified more than once. The absolute filenames found by `--search` are added to those specified with `--file` and `--list`.

See also, `--search-extensions`.

`--search-extensions <list of extensions>`

Overrides the classification of what source files should or should not be counted. By default, all files that are recognized as an analyzable source will be counted. This option takes a comma-separated list of case-sensitive filename extensions. When specified, only files with these extensions will be counted.

For example:

```
c , c++ , cc , cp , cpp , cs , cxx , h , h+
+ , hh , hpp , htm , html , hxx , java , js , jsp , m , mm , php , php3 , phtml , py , pyt , rb , rpy
```

`--third-party-regex <regex>`

Specifies a case-insensitive regular expression (regex) of absolute filenames to *exclude* from `--search` and to add (as exclusions) to the code identity file (see `--code-identity-file`). This option may be specified more than once. The regex uses Perl syntax and matches if it matches a substring.

Note that these regexes can be used to exclude any files that should not be counted or reported for defects. These may include third-party files, test code, and generated code.

## Shared options

- `--debug, -g`  
Turn on basic debugging output.
- `--ident`  
Displays the version of Coverity Analysis and build number.
- `--info`  
Displays certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.
- `--tmpdir <tmp>, -t <tmp>`  
Specifies the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.
- `--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>`  
Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects`(*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## Examples

Flag files with a line count greater than 1000:

```
> cov-count-lines --list l | awk '{if ($5 > 1000) print $0;}'
```

Report the line count for the Apache `regcomp.c` file:

```
> cov-count-lines --file /home/user/apache_1.3.33/src/regex/regcomp.c
```

---

## Name

cov-emit Parse and emit a C/C++ source file.

## Synopsis

```
cov-emit [OPTIONS] <file.c>
```

## Description

The `cov-emit` command parses a source file and outputs it into a directory (emit repository) that can later be analyzed with `cov-analyze`. The `cov-emit` command is typically called by `cov-translate`, which is in turn typically called by `cov-build` (`cov-emit` is a low-level command and is not normally called directly). The `cov-emit` command defines the `__COVERITY__` preprocessor symbol.

## Options

`--add_builtin_stdarg_macro`

Enables macros to be defined when a builtin include of `<stdarg.h>` is processed.

`--add_type_modifier=[<modifier>]`

Enables `cov-emit` to recognize and parse previously unknown type modifiers such as `__data16`, `__code32`, `__huge` etc.

Consider the following code:

```
void foo(int *x) {
 /* Do something */
}

void foo(int __data16 * x) {
 /* Do something else */
}
```

Utilizing the `--add_type_modifier=__data16` will tell `cov-emit` that `__data16` is a type modifier, and it will treat the two functions as distinct. Implicit conversions between types is done with reliance on the native compiler to enforce the conversion rules - that is, it is assumed that all conversions are valid.

### Note

You can specify one type modifier to be the *default*. The *default* setting is used when instantiating templates to mimic the native compiler's usage of default type modifiers as shown in the following example:

```
--add_type_modifier=__data8,__data16:default
```

`--allow-incompat-return-types`

Allows a prototype of a function to specify different return types than those in the actual function definition.

**--allow\_incompat\_throw**

Indicates to `cov-emit` that it should not report an error if there are multiple prototypes of the same function with incompatible C++ exception specifications.

**--angle\_include\_search\_first={default|user|system}**

Controls the order that directories are searched when trying to find an included file. This option applies to files added by `#include <...>`.

- `default` - Indicates that there is no change from previous behavior.
- `user` - Indicates that user include directories (added with `-I`) will be searched first.
- `system` - system include directories (added with `--sys_include`) will be searched first.

The `--sys_include_first` option, which is now deprecated, is equivalent to `--angle_include_search_first=system`.

**--arg\_file <file>**

Read arguments from the response file `<file>`. This is typically done by `cov-translate` to avoid command-line length limitations.

The response file format is as follows:

```
<compiler_args>
 <cov_emit_cmd_args>
 <arg>[cov-emit arg]</arg>
 <arg>[another cov-emit arg]</arg>
 </cov_emit_cmd_args>
</compiler_args>
```

 **Note**

Note that spaces within the `<arg>` tags are interpreted literally. This means that `<arg>-DFOO=bar</arg>` will work, while `<arg> -DFOO=bar</arg>` will cause an error, as the argument is interpreted as a source file name.

**--c**

Compile standard C code (C89).

**--c++11**

Enable c++11 language features.

**--c++14**

Enable c++14 language features.

**--c++17**

Enable c++17 language features.

**--c++**

Compile standard C++ code. This is the default.

**--c99**

Enable C 99 extensions to the C programming language.

**--cache\_include\_search**

If you use large numbers of `#include` search directories with the `-I` option, specify this option to speed up compilation.

**--calling\_convention\_group**

Specifies the comma-delimited calling conventions that should be considered equivalent. For example, `--calling_convention_group default,stdcall,vectorcall` causes `stdcall` and `vectorcall` calling conventions to be treated the same as the default, or unspecified, calling convention in `cov-emit`. The list of valid calling conventions are: `default`, `cdecl`, `fastcall`, `stdcall`, `thiscall`, `vectorcall`, `fastcall`.

**--char\_bit\_size <integer>**

Used to specify the size (in bits) of the `char` type. If this option is not specified, the default is 8-bit chars.

**--cygwin**

This switch tells `cov-emit` to attempt to convert Unix-based (Cygwin) paths into their corresponding Windows (real) paths.

**-D <identifier>[=<value>]**

Add a macro definition of `<identifier>` with optional `<value>` .

**--dir**

Specifies the emit repository (an intermediate directory) into which the `cov-emit` command outputs its results.

**-E**

Only preprocess the source file.

**--emit\_complementary\_info**

Enables emitting of complementary information for compliance checkers such as MISRA checkers. Selecting this option results in a slower build capture but a faster analysis, and it should be applied when using compliance checkers. The default value is `--no_emit_complementary_info`

 **Note**

Enabling the `--emit_complementary_info` option prior to running an analysis is likely to turn up additional defects.

Any analysis involving `--coding-standard-config` requires the information generated during `cov-build` when including the `--emit-complementary-info` option. The `cov-build` command will take longer, so this option should only be used when `cov-analyze` is used with `--coding-standard-config`.

If `cov-build` did not include the `--emit-complementary-info` option and `cov-analyze` does include `--coding-standard-config`, `cov-analyze` automatically re-runs every `cov-emit` command (for the Translation Units to be analyzed). This excludes the native build and the `cov-translate` overhead, but it will add significant overhead to `cov-analyze`. Note that analysis will fail if the emit database does not include source; that is re-emit is not possible.

**--enable\_80bit\_float**

The following switches provide the ability to turn on/off 80-bit float intrinsic types. This overrides implicit enablement or disablement implied by other `cov-emit` switches.

- `--enable_80bit_float`
- `--no_enable_80bit_float`

**--enable\_128bit\_float**

The following switches provide the ability to turn on/off 128-bit float intrinsic types. This overrides implicit enablement or disablement implied by other `cov-emit` switches.

- `--enable_128bit_float`
- `--no_enable_128bit_float`

**--enable\_128bit\_int**

The following switches provide the ability to turn on/off 128-bit integer types, independent of `gnu_version`:

- `--enable_128bit_int`
- `--no_enable_128bit_int`
- `--enable_128bit_int_extensions`
- `--no_enable_128bit_int_extensions`

`--enable_128bit_int` implies `--enable_128bit_int_extensions`, however the same does not apply to the `no_*` variants.

`--enable_128bit_int` enables type `__int128`, while `--enable_128bit_int_extensions` enables types `__int128_t` and `__uint128_t`.

**--enable\_user\_sections**

Enables the user sections compiler extension allowing variable placement at specific addresses in memory. Compilers that support this extension include the IAR ARM compiler which uses the "@" operator for this purpose, and the CodeWarrior compiler which uses ":". Please consult your compiler manual for more information.

This option supersedes `cov-emit`'s deprecated `--allow_declare_at_address` option.

**--encoding <enc>**

Specifies the encoding of source files. Use this option when the source code contains non-ASCII characters so that Coverity Connect can display the code correctly. The default value is US-ASCII. Valid values are the ICU-supported encoding names:

US-ASCII

UTF-8

UTF-16

UTF-16BE  
UTF-16 Big-Endian

UTF-16LE  
UTF-16 Little-Endian

UTF-32

UTF-32BE  
UTF-32 Big-Endian

UTF-32LE  
UTF-32 Little-Endian

ISO-8859-1  
Western European (Latin-1)

ISO-8859-2  
Central European

ISO-8859-3  
Maltese, Esperanto

ISO-8859-4  
North European

ISO-8859-5  
Cyrillic

ISO-8859-6  
Arabic

ISO-8859-7  
Greek

ISO-8859-8  
Hebrew

ISO-8859-9  
Turkish

ISO-8859-10  
Nordic

ISO-8859-13  
Baltic Rim

ISO-8859-15  
Latin-9

Shift\_JIS  
Japanese

EUC-JP  
Japanese

 **Note**

EUC-JP is now a valid output object encoding. See `--output_object_encoding`.

ISO-2022-JP  
Japanese

GB2312  
Chinese (EUC-CN)

ISO-2022-CN  
Simplified Chinese

Big5  
Traditional Chinese

EUC-TW  
Taiwanese

EUC-KR  
Korean

ISO-2022-KR  
Korean

KOI8-R  
Russian

windows-1251  
Windows Cyrillic

windows-1252  
Windows Latin-1

windows-1256  
Windows Arabic

 **Note**

If your code is in SHIFT-JIS or EUC-JP, you must specify the `--output_object_encoding SHIFT-JIS` or `--output_object_encoding EUC-JP` option (respectively) in order to avoid receiving `STRING_OVERFLOW` false positives.

For more information, see `--output_object_encoding`.

`--encoding_selector <encoding-or-regularexpression>`

Treats all files with file names that match the given, case-sensitive regular expression as though they have the specified encoding. For a list of valid encodings, see the `--encoding` option to

this command. The regular expression syntax is a Perl regular expression, as described in <http://perldoc.perl.org/perlre.html> .

Encoding selectors also apply to files that are included in source files, not just to the files specified on the `cov-emit` command line. This behavior allows for a finer granularity in selecting encodings.

Note that encoding selectors have a higher priority than the `--encoding` option. If the `cov-emit` command line contains both `--encoding <encoding>` and `--encoding_selector <encoding>/<regular expression>`, and the regular expression is a match for the file that is currently getting opened, the encoding specified through the encoding selector will take precedence.

`--encoding_selector_icode <encoding>/<regular expression>`

Identical to `--encoding_selector`, except that the regular expression is case insensitive.

`--error_limit <number>`

For a file that fails to compile, specifies the number of errors that are output to `build-log.txt` before moving to the next file. Default is 5.

`--float_bit_patterns`

Enables parsing the ARM Development Studio C/C++ language extension that allows floating-point bit pattern literals, for example:

```
float f = 0f_00000000;
double d = 0d_0000000000000000;
```

`--force`

Disables incremental compilation by forcing `cov-emit` to compile and generate output for a file, even if a copy of that file has already been compiled and is present in the Intermediate Directory.

`--gcc`

Allow parsing of C code with `gcc` (GNU) extensions.

`--g++`

Allow parsing of C++ code with `g++` (GNU) extensions.

`--gnu_version=<version>`

Specifies the version of the GNU compiler to emulate. Only required when the code you are compiling exploits version-dependent features or bugs in the `gcc` or `g++` compilers. If the version of `gcc` you are using is 3.4.2, for example, then specify the version as 30402.

`--ignore_std`

Specifies that the namespace "std" should be ignored entirely. `g++` versions prior to 3.0 ignored the "std" namespace.

`--include_recursion_limit <value>`

Specifies the maximum number of times a source file is allowed to include itself (directly or indirectly) before the recursion is assumed to be infinite and compilation is terminated. The default limit is 10 levels of recursion.

`--incompat_proto`

Allows a prototype of a function to specify a different set of arguments than those in the actual function definition.

**--inline\_keyword**

Explicitly enables support for the ISO C99 `inline` keyword, overriding any language dialect or compiler emulation mode setting. By default, `inline` is treated as a keyword if the selected language dialect or compiler emulation mode requires it.

**-I <dir>**

Add a directory to search for `#include` files. Directories added with this switch are considered 'user' include directories.

The default behaviour is to search for headers in the order that both the `-I` and `--sys_include` were specified on the command line, regardless of `#include` type. This can be adjusted using `--angle_include_search_first` and `--quote_include_search_first`.

**--lazy\_hex\_pp\_number**

This option affects grammar in which a statement such as `'0x1e-1'` can be parsed as either a single pp-number (C++11 2.10 [lex.ppnumber]) or as a subtraction expression.

When using the Compiler Integration Toolkit (CIT), you will only need to use this option if the compiler correctly parses the example above, but `cov-emit` does not.

In order to determine if you need to use this option, you will receive an error message from `cov-emit` that looks like the following:

```
"test.cpp", line 3: error: extra text after expected end of number
int foo = (0xD8E-0xD64);
 ^
```

**--list\_macros**

Print all macros defined in the translation unit to standard out.

Note that this option prints *all* macros in the translation unit, while `--print_predefined_macros` prints only predefined macros.

**--lowercase\_header\_filenames**

In the `cov-emit` preprocessor, when the source refers to a header filename, turn it into all-lowercase before asking the operating system for the file. This can be useful when transitioning to a case-sensitive filesystem. Also translates backslash to slash, and removes a leading drive letter and (back)slash, as these are needed in the same situations.

**--macro\_stack\_pragmas**

Enables parsing of GNU macro stack manipulation pragmas (`#pragma push_macro` and `#pragma pop_macro`). This option is enabled by default in most cases, but may be automatically disabled in certain compatibility modes.

To manually disable this option, use `--no_macro_stack_pragmas`.

**--microsoft**

Allow parsing of Microsoft extensions.

**--ms\_asm**

When specified, enable support for parsing Microsoft-style inline assembly. By default, inline assembly is assumed to follow the format specified by the C standard, e.g.:

```
asm("int $3");
```

When enabled, inline assembly is parsed following Microsoft's style, e.g.:

```
asm int 3;
asm { int 3; };
```

When enabled, Microsoft-style inline assembly may be specified using the `asm`, `__asm` or `__asm` keywords interchangeably.

**--multiline\_string**

Allow the Coverity compiler to accept multi-line strings. Multi-line strings are supported by compilers such as gcc 3.4.

**--nested\_comments**

Allow the Coverity compiler to accept nested block comments. Nested comments are supported by compilers such as Renesas RX 2.03.

**--new\_array\_args**

Enables parsing for options to array element constructors invoked by the `new[ ]` variant of the C++ memory allocation operator.

**--no\_atomic\_commit**

This option is deprecated as of the 5.0 release.

**--no\_caa\_info**

Do not collect the information required for Architecture Analysis in the intermediate directory.

**--no\_emit\_complementary\_info**

Disables emitting of complementary information for compliance checkers such as MISRA checkers.

**--no\_enable\_user\_sections**

Disables the user sections compiler extension. To enable the use of user sections, see `--enable_user_sections`.

**--no\_exceptions**

Disable parsing for exception handling in C++.

**--no-headers**

This option is deprecated as of the 5.0 release.

**--no\_inline\_keyword**

Explicitly disables support for the ISO C99 `inline` keyword, overriding any language dialect or compiler emulation mode setting. By default, `inline` is treated as a keyword if the selected language dialect or compiler emulation mode requires it.

**--no\_macro\_stack\_pragmas**

Disables parsing of GNU macro stack manipulation pragmas (`#pragma push_macro` and `#pragma pop_macro`). This option is enabled by default in most cases, but may be automatically disabled in certain compatibility modes.

To enable parsing of these pragmas, use `--macro_stack_pragmas`.

**--no\_ms\_asm**

Disable parsing of Microsoft-style inline assembly. This disables the keywords `asm`, `_asm` and `__asm`.

**--no\_predefined\_feature\_test\_macros**

Do not predefine the testing macros described in `--predefined_feature_test_macros`.

This is the default behavior of `cov-emit`.

**--no\_predefined\_stdcl**

Do not predefine `__STDC__`.

**--no\_predefines**

Do not predefine any macros internally. All macro definitions must be in the source code or explicitly on the command line.

This is the default behavior for `cov-emit`. Use the `--predefines` option to predefine specific macros.

Note that `--no_predefines` has no effect on the following macros, which may still be predefined even if this option is specified:

- Certain C/C++ standard macros (e.g., `__FILE__`, `__LINE__`)
- Macros that begin with "`__COVERITY`"
- Macros that are controlled by another switch (e.g., `__STDC__` and `--no_predefined_stdcl`)

**--old\_g++**

For GNU versions prior to 3.x, specifies a more permissive version of `g++` compatibility.

**--output\_object\_encoding**

Specifies the output character encoding. This option accepts an encoding as a required argument. The accepted encodings are `Shift-JIS` and `EUC-JP`. For example:

```
--output_object_encoding EUC-JP
```

Using `--shiftjis_encode_obj` is effectively the same as specifying `--output_object_encoding Shift-JIS`.

If `--output_object_encoding` is not specified, then the object encoding is UTF-8.

**--pending\_instantiations <integer>**

Specifies the maximum number of instantiations of a given template that can be in progress at any given time. Use 0 (zero) to specify an unlimited number. You might need to increase this limit when using recursive templates.

**--ppp\_translator <translator>**

Add `--ppp_translator <translator>` to the `cov-emit` command line to translate files before they are preprocessed. Possible `<translator>` values are:

- `cmd:<command>`— Pipes file through `<command>` .

- `replace/<from>/<to>`— Replaces regular expression `<from>` with `<to>`. The regular expression syntax is a Perl regular expression, as described in <http://perldoc.perl.org/perlre.html>. The `'` character can be replaced with any other character; and this separator character can be quoted with a backslash `\`.

**--pp\_sizeof**

Allows the use of `sizeof()` in preprocessing directives. When compiling, the argument to `sizeof` can be anything permitted in an expression. However, when preprocessing, it is only possible to use built-in types like `int`. This means that preprocessing output might be different than what the compiler encounters during compilation. This feature is nonstandard and only supported by a few compilers.

**--pre\_preinclude <file.h>**

Specify header file that should be processed prior all other source and header files.

**--predefined\_feature\_test\_macros**

Compatible with C++ only. The WG21 working paper N3694 [provides](#) guidelines for predefined feature testing macros. When this option is specified, `cov-emit` will predefine the appropriate macros as suggested by these guidelines.

**--predefines**

When specified, `cov-emit` will predefine additional macros based on the current emulation mode.

For example:

```
> cov-emit --microsoft --predefines test.c
```

The above command will predefine `_MSC_VER`, while the following command predefines `__GNUC__`.

```
> cov-emit --gcc --predefines test.c
```

Note that `--predefines` has no effect on the following macros:

- Certain C/C++ standard macros (e.g., `__FILE__`, `__LINE__`)
- Macros that begin with `"__COVERITY"`
- Macros that are controlled by another switch (e.g., `__STDC__` and `--no_predefined_std`)

**--preinclude\_macros <file.h>**

Specify macros-only header file that should be processed immediately after the files specified with `--pre_preinclude` option (see above) and prior to all other source and header files.

**--preinclude <file.h>**

Specify header file that should be processed immediately after the files specified with `--pre_preinclude` and `--preinclude_macros` options (see above) and prior to all other source and header files.

**--print\_predefined\_macros**

Print all predefined macros (and their values) to stdout.

Note that this option prints only predefined macros, while `--list_macros` prints all macros in the translation unit.

`--ptrdiff_t_type <builtin-type>`

Specify the type of `ptrdiff_t`. This is stored in the `__COVERITY_PTR_DIFF_TYPE__` macro (as the type, not the character code). If unspecified, `__COVERITY_PTR_DIFF_TYPE__` is set to the same type as `ptrdiff_t` (for example, "signed int"). The character code for `<builtin-type>` is typically one of the signed types shown in `--size_t_type`. For example, `--ptrdiff_t_type i` sets the type of `ptrdiff_t` to "signed int".

`--quote_include_search_first={default|user|system}`

Controls the order that directories are searched when trying to find an included file. This option applies to files added by `#include "..."`.

- `default` - Indicates that there is no change from previous behavior.
- `user` - Indicates that user include directories (added with `-I`) will be searched first.
- `system` - system include directories (added with `--sys_include`) will be searched first.

`--short_enums`

Enables optimization of enumeration sizes. The size of each enumeration will be set based on the largest values present when this option is specified.

`--size_t_type <builtin-type>`

Specify type of `size_t`. This is stored in the `__COVERITY_SIZE_TYPE__` macro (as the type, not the character code). If unspecified, `__COVERITY_SIZE_TYPE__` is set to the same type as `size_t` (generally, the default is "unsigned int"). Use an unsigned integral type from the single-character codes for `<builtin-type>` as follows:

```
a # signed char
h # unsigned char
s # short
t # unsigned short
i # int
j # unsigned int
l # long
m # unsigned long
x # long long, __int64
y # unsigned long long, __int64
```

For example, `--size_t_type j` sets the type of `size_t` to "unsigned int".

`--source_chroot <chroot-path>`

This option ensures that `cov-emit` only searches for source files under the listed chroot path. Source files outside of the chroot path will not be found. Note that for the `--source_chroot` option to work properly, the current working directory must be a child of the chroot path.

`--sys_include <dir>`

Add a directory to search for `#include` files. Directories added with this switch are considered 'system' include directories.

The default behaviour is to search for headers in the order that both `-I` and `--sys_include` were specified on the command line, regardless of `#include` type. This can be adjusted using `--angle_include_search_first` and `--quote_include_search_first`.

`--system_encoding <enc>`

Specifies the encoding to use when interpreting command line arguments and file names. If not specified, a default system encoding is determined based on host OS configuration.

See `--encoding` for a list of accepted encoding names.

`--type_alignments <builtin-type>`

Specify type of `type_alignments`. The `<builtin-type>` string consists of the ABI chars shown in `--size_t_type`, plus the following:

```
f # float
d # double
e # long double, __float80
P # Coverity extension: pointer
```

and lengths. For example, `--type_alignments x8li4s2P4` sets `type_alignments` to long long 8, long & int 4, short 2, ptr 4.

`--type_sizes <builtin-type>`

Specify type of `type_sizes`. The `<builtin-type>` string consists of the ABI chars shown in `--size_t_type`, plus the following:

```
w # wchar_t
f # float
d # double
e # long double, __float80
n # __init128
o # unsigned __init128
g # __float128
P # Coverity extension: pointer
```

and lengths. For example, `--type_sizes w4x8li4s2P4`, sets `type_sizes` to "wide char 4 bytes, long long 8, long & int 4, short 2, ptr 4".

If unspecified, `cov-emit` uses the machine's native type sizes.

The C standard mandates that `sizeof(char) == 1` and `sizeof(<any other type>) == <multiple of sizeof(char)>`. Therefore, all type sizes should be specified as multiples of a char size (and char should always be size 1). To set the bit size of a char, see `--char_bit_size`.

For example, assume you have a compiler that has the following:

- 16-bit chars
- 16-bit shorts
- 32-bit ints

- 32-bit longs

The correct arguments for this compiler are:

```
cov-emit --char_bit_size 16 --type_sizes stlijlm2
```

 **Note**

Note that if this option specifies contradictory sizes for signed and unsigned versions of the same type, the last value specified will be used. For example, `--type_sizes i4j6` will set the length of "int" and "unsigned int" to 6, and the 4 will be ignored.

`--wchar_t_keyword`

Indicates that `cov-emit` should treat the type `wchar_t` as a keyword built into the language.

`--wchar_t_name <identifier>`

Uses the specified identifier for the `wchar_t` intrinsic type. This option does not imply `--wchar_t_keyword`.

`--wchar_t_type <builtin-type>`

Specifies the type `--wchar_t`, where `<builtin-type>` is one of the unsigned integral types shown in `--size_t_type`. For example, `--wchar_t_type j` sets the type of `wchar_t` to "unsigned int".

`-U <identifier>`

Undefine the macro `<identifier>`.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## See Also

`cov-build`

`cov-translate`

---

## Name

cov-emit-cs Parse C# source code and emit output to the intermediate directory.

## Synopsis

```
cov-emit-cs --dir <intermediate_directory> [OPTIONS] [sourcefiles]
```

## Description

This command parses source code, decompiles referenced assemblies, and saves CSC compiler output to the emit repository in the intermediate directory.

## Options

### --addmodule <file>

Identifies a module that is referenced by the compilation but not added to the emit repository. The location of the module can be absolute or relative. For rules on specifying the location, see `--reference`.

It is an error if the referenced module is not found. To change this behavior, see `--allow-missing-refs`.

You must specify this option separately for each module.

### --allow-missing-refs

Issues a warning if any referenced assemblies are missing. If you do not set this option, missing assemblies result in an error that stops the process. This error applies to explicitly referenced assemblies (see `--reference` and `--addmodule`) that are absolute or not found in the Common Language Runtime (CLR) system directory. The error also applies to `mscorlib.dll` (see `--nostdlib`). Note that missing `csc.rsp` assemblies (see `--noconfig`) always result in a warning.

### --codepage <codepage>

Identifies the numeric codepage corresponding to codepages that are supported by CSC with the `/codepage` option. Source file encodings are determined in the following manner:

If a byte order mark (BOM) is present in the source file, the command uses the BOM-related encoding. If a BOM is *not* present, encoding is determined in the following manner:

- The character encoding of the specified codepage is used. If a codepage is not specified, the command attempts to detect and use UTF-8. If neither of the preceding alternatives is possible, the command uses the system default codepage.

### --compiler-dir <directory>

Identifies the CLR system directory. It is an error to specify a directory that does not exist. The CLR system directory is used as a search path for referenced assemblies (see `--reference`) and to locate the `csc.rsp` file (see `--noconfig`). If `--compiler-dir` is not specified, the command defaults to `$SYSTEM_ROOT/Microsoft.NET/Framework/<version>`, where `<version>` is the latest supported framework version (for details, see the *Coverity 2020.12 Installation and Deployment Guide* [🔗](#)). It is an error if no suitable CLR system directory is found.

**--define <define>**

Corresponds to the CSC preprocessor directive and `/define` option. Note that it is necessary to specify a separate `--define` option for each directive.

**--dir <intermediate\_dir>**

Identifies the intermediate directory into which this command emits source files and referenced assemblies. An error occurs if the specified intermediate directory exists but is not valid, or if the directory does not exist and cannot be created.

This option is required.

**--disable-ref-assembly-replacement**

By default, the `cov-emit-cs` command attempts to replace each reference assembly it encounters with a version of the assembly that includes an implementation. Use this switch to disable this default behaviour. The recommended use of this switch is as an `xml-option` option to a `cov-configure` command.

Example of adding this switch to a C# configuration:

```
cov-configure --cs -c config/config.xml --xml-option="<append_arg>--disable-ref-assembly-replacement</append_arg>"
```

**--enable-cs-parse-error-recovery**

Makes `cov-emit-cs` fall back to error recovery mode when compilation errors are encountered during the processing of source files.

This option is disabled by default.

**--force**

By default, if all the specified source files exist in the emit repository with the same timestamp as the file on disk, the command skips the files and exits with a successful return code. Specifying the `--force` option makes the command process the source files, even if they exist unchanged in the emit repository.

**--langversion <C# language version>**

Specifies the C# Language version to use. Corresponds to the CSC option `/langversion`.

**--lib <directory>, -L <directory>**

Identifies a library directory to use when searching for referenced assemblies (see `--reference`). A warning (not an error) occurs if you specify a directory that does not exist. This option corresponds to the CSC `/lib` option.

You must specify this option separately for each library directory.

**--link <[alias=]filename> | -l <[alias=]filename>**

Supersedes `--use-link- semantics`. Effectively the same as `--reference`, but changes how the compiler treats certain COM interop types. Corresponds to the CSC option `/link`.

**--noconfig**

Ignores the `csc.rsp` file under the CLR system directory (see `--compiler-dir`). If this option is not set, the references `/r` or `/reference` within `csc.rsp` are added to the list of referenced

assemblies. Any `csc.rsp` references that are not absolute filenames are subject to the search directory rules (for details, see `--reference`). Corresponds to the CSC option `/noconfig`.

**--nostdlib**

Disables the default behavior of searching for `mscorlib.dll` in the CLR system directory and adding the file to the list of referenced assemblies. If that file is not found, the next search in this directory is for a `csc.exe.config` file that specifies a `requiredRuntime` version. If a version is found, the search continues to the corresponding directory (the parent directory of the CLR system directory).

This option corresponds to the CSC `/nostdlib` option.

**--no-friends**

Prevents the compilation from accessing internal types or members. This option works by disabling the processing of compiler output retrieved through the `--out` option. This behavior corresponds to CSC behavior in the case where `/out` is not specified and a default name is used.

**--no-banner**

Suppresses the `cov-emit-cs` application name and version banner.

**--out <file>**

Specifies the compiler output file, which is then used by the command to access internal types or members in referenced assemblies.

Subsequent calls to `cov-emit-cs` will not re-emit the output file if a referenced assembly is found.

It is necessary to specify `--out` or `--no-out` (or the alternative, `-n`), else an error will occur.

**--no-out, -n**

Indicates that there is no compiler output.

It is necessary to specify `--out` or `--no-out` (or the alternative, `-n`), else an error will occur.

**--ref-assembly-replacement-search-path <path>**

Adds a search path to assist `cov-emit-cs` in finding implementation versions of reference assemblies. You can specify this option multiple times. This option is useful for pointing `cov-emit-cs` at the correct .NET Core framework directory in cases where `cov-emit-cs` fails to find the correct version. The recommended use of this switch is as an `xml-option` option to a `cov-configure` command.

Example of adding this switch to a C# configuration:

```
cov-configure --cs -c config/config.xml --xml-option="<append_arg>--ref-assembly-replacement-search-path=D:\utilities\dotnet-core\dotnet-sdk-2.1.403-win-x64\shared\Microsoft.NETCore.App\2.1.5\</append_arg>"
```

**--reference <[alias=]filename> | -r <[alias=]filename>**

Identifies an assembly to provide for compilation and addition to the emit repository, unless this setting is overridden by other options. The location of the assembly can be absolute or relative. If relative, the command searches the following paths in the following order:

1. Current working directory.
2. CLR system directory.
3. Each directory specified by the `--lib` option, in the order specified.

By default, it is an error if a referenced file cannot be found on disk. See `--allow-missing-refs` for complete rules.

Note that `[alias=]` identifies an extern alias directive, just as it does for CSC.

Example with an alias:

```
--reference v1=my.dll
```

#### `--target`

Specifies the type of assembly that was created. Corresponds to the CSC option `/target`.

#### `--unsafe`

Allows analysis C# constructs that are declared with the `unsafe` modifier. This option corresponds to the `/unsafe` option to CSC.

#### `--use-link-semantic`s

Corresponds to the `/link` option to CSC, which changes how the compiler treats certain COM interop types.

Any unrecognized options result in an error, which causes an immediate exit with an appropriate error message.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects`(*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

---

## Name

`cov-emit-go` Parses one or more source files and packages and outputs it into an intermediate directory.

## Synopsis

```
cov-emit-go --dir <intermediate_directory> [OPTIONS] [sourcefiles] [packages]
```

## Description

The `cov-emit-go` command parses source files and packages and saves `go build` output to the emit repository in the intermediate directory.

For code bases that contain CGo dependencies (in other words, Go code that imports the pseudo-package "C"): Your environment must be configured to successfully compile such code using the native Go compiler before you execute `cov-emit-go` or `cov-build` on your code base. This requirement is necessitated because the `cov-emit-go` command, the Go compiler, and the CGo tool, must access additional tools to process such code (they execute a C compiler and generate bindings for the compiled C functions). These same tools execute underneath the `cov-build` command.

C code that is compiled as part of processing CGo dependencies will not be captured for analysis by either `cov-build` or `cov-emit-go`.

## Options

`--dir <intermediate_directory>`

Path name to an intermediate directory that is used to store the results of the build and analysis.

## Error codes

0

Success.

2

Most likely a user-generated error, such as code that will not compile.

4

Irrecoverable error. Please contact `software-integrity-support@synopsys.com` if you receive this error code.

---

## Name

`cov-emit-java` Parses one or more source files and outputs it into an intermediate directory.

## Synopsis

```
cov-emit-java [OPTIONS] [SOURCE FILES]
```

## Description

The `cov-emit-java` command parses Java source code and bytecode, and saves `javac` outputs. It stores these outputs to an emit repository for subsequent static analysis and outputs it into a directory (emit repository) that can later be analyzed with `cov-analyze`. The `cov-emit-java` command is typically called by `cov-build`.

You need to invoke this command when you are running Java Web application security analysis and in the rare case that you cannot compile your Java code with `cov-build`. For details about the latter case, see the discussion of the alternative build process in the Coverity Analysis.

When specifying multiple source files, you need to separate each source file by a space, for example:

```
src/pkg/SomeClass.java src/pkg/OtherClass.java
```

Note that you can specify the options to `cov-emit-java` in any order, but the list of source files must appear last.

## Error codes

0

Success. Also applies to errors from which it is possible to recover.

The `cov-emit-java` command is less strict than `javac` in that it recovers from some invalid or inconsistent inputs by discarding problematic code, printing a message, and continuing to function.

2

Most likely, a user-generated error, such as code that will not compile.

4

Irrecoverable error. Please contact [software-integrity-support@synopsys.com](mailto:software-integrity-support@synopsys.com) if you receive this error code.

## Options

`--add-modules <modules>`

[Java9+ only] Allows you to specify the name of one or more modules in a comma-delimited list.

These modules are added to the set of observable root modules. The default root modules may vary by JDK, and are defined by the module declaration in the `java.se` JDK module.

`--add-exports <module-name/package-name=target-module>`

[Java 9+ only] Allows you to specify that the `<package-name>` in `<module-name>` is exported explicitly to the `<target-module>`. The effect of this option is the same as declaring an `exports` clause in the module declaration for the `<module-name>`.

This option can be specified more than once if multiple add-exports options are required.

`--add-reads <module-name=target-modules>`  
[Java 9+ only] Allows you to specify additional read edges between `<module-name>` and one or more target module names in a comma-delimited list. Note that in the context of the Java module system, `reads` is synonymous with `requires`, so the effect of this option is the same as declaring `requires` clauses in the module declaration.

This option can be specified more than once if multiple add-reads options are required.

`--android-apk <Android_APK_file>`  
[Used for Coverity Extend SDK checkers that analyze Android applications] Identifies an Android APK file that is associated with a specified input file and read by custom Extend SDK checkers that are built for Android analysis.

Requirement: When using this option, you must also specify an `--input-file` option to this command. For information about custom Android checker development, see the section, "Reporting events and defects on input files" in *Coverity Extend SDK 2020.12 Checker Development Guide*.

`--auto <project_directory>`  
[Not supported for module-based code in Java 9+] Allows you to specify a directory where source (`*.java`), input Jar files (`*.jar`), and compiler outputs (`*.class`) can be found. If you set this option, `cov-emit-java` will recursively search for these items.

Note that `--auto <project_directory>` is functionally equivalent to the following:

```
--findsource <project_directory>
--findjars <project_directory>
--compiler-outputs <project_directory>
```

If necessary, you can specify `--findsource`, `--findjars`, or `--compiler-outputs` options along with the `--auto` option. This sort of specification might be useful if you have a simple project that requires specific Ant or JDK dependencies.

Example:

```
cov-emit-java --dir <intermediate_directory> --auto $PROJECT_ROOT --
findjars $ANT_HOME:$JAVA_HOME
```

`--bootclasspath <directories_or_jar_files>`  
Allows you to specify a list of directories or Jar files, which must be separated by a semi-colon (`;`) on the Windows platform, and by a colon (`:`) on other platforms. Classes specified through the `--bootclasspath` are emitted, but the bodies of their methods are not, because `cov-emit-java` expects to have models for them. Coverity Analysis for Java comes with models for the entire Java Runtime Environment (JRE) and Android SDK, which should address all cases.

Like `javac`, `cov-emit-java` searches these entries for bytecode with the referenced classes when attempting to resolve names in source files. The directories/jar files are searched in the order specified in `<directories_or_jar_files>`.

Generally, you do not need to specify this option because `cov-emit-java` selects the bootclasspath of the JRE that comes with Coverity Analysis for Java. However, if you are compiling

code against a non-standard JRE, one that is not API-compatible with the standard JRE, then you might need to use this option.

`--classpath <directories_or_jar_files>`

Allows you to specify a list of directories or Jar files, which must be separated by a semi-colon (;) on the Windows platform, and by a colon (:) on other platforms. Wildcard (\*) in classpath is supported. The wildcard will find all Jar files in the given directory (ie: `foo/*` finds all jars in `foo/`). Like `javac`, `cov-emit-java` searches these entries for bytecode with the referenced classes when attempting to resolve names in source files. The directories/jar files are searched in the order specified in `<directories_or_jar_files>`. Since Coverity Analysis analyzes these class files, it is better, when possible, to specify the implementations that are loaded at runtime rather than the stubs that are used for compilation.

 **Note**

Note that directories specified with `--classpath` will only be searched for Jar files if the wildcard is used. Otherwise, directories will be searched only for bytecode.

If a directory (`foo/`) may contain bytecode as *well* as Jar files, you should include both "`foo/`" AND "`foo/*`" in your arguments to `--classpath`.

The `cov-emit-java` command captures the same bytecode in Jar files referenced on the classpath that `java` or `javac` find (that is, bytecode where the package name matches the directory within the Jar file).

The `cov-emit-java` command respects the `Class-Path` entry in Jar manifests.

If the `--classpath` option is not specified, the current directory will be used as the default classpath.

The `cov-emit-java` command is not affected by the `CLASSPATH` environment variable.

 **Note**

When loading certain jar files, in particular the Scala runtime library, `cov-emit-java` can consume more than 10GB of memory. This can cause out-of-memory failures on low-memory systems, including 32-bit systems in general.

`--compiler-outputs <class_files_or_directories>`

Captures a list of class files that have debug symbols for subsequent analysis by SpotBugs. For proper display of defects detected by the SpotBugs analysis, such class files should have been compiled with all debugging information (for example, with `javac -g`).

The list contains class files separated by the classpath separator (colon or semi-colon). Any directories in the list will be recursively searched for class files. Compiler outputs should be specified on the same command line as the source code from which they were compiled. Subsequent invocations of `cov-emit-java` on the same source files will replace the compiler outputs from the previous invocation with those of the new invocation.

Jar files passed to `--compiler-outputs` will have their contained class files included. Directories passed to `--compiler-outputs` will not have their contained Jar files included.

 **Note**

Do not pass obfuscated bytecode to this option.

`--dir <intermediate_directory>`

Specifies the emit repository (an intermediate directory) into which the `cov-emit-java` command outputs its results.

This option is required.

The command fails with an error if either of the following are true:

- The specified directory exists but is not a valid intermediate directory.
- The intermediate directory does not exist and cannot be created.

`--enable-java-parse-error-recovery`

Enables the error recovery algorithm that produces the highest emit percentage in most cases. Currently, this option enables per-class error recovery with automatic fallback to per-file recovery in case of non-recoverable errors such as `cov-emit-java` crashes, but its behavior might change in future.

Explicitly enabling an error recovery algorithm on the command line will automatically disable any incompatible error recovery algorithms.

`--enable-java-per-class-error-recovery`

This option can help to increase the percentage of source files that can be emitted by attempting to use the class files of the source files that cause parse errors. It could potentially increase the time to emit files, but is usually faster than `--enable-java-per-file-error-recovery` because it does not try to emit each file one at a time.

Per-class error recovery is unlikely to correct `cov-emit-java` crashes. It also requires the presence of output class files. To address such issues, see `--enable-java-parse-error-recovery`.

This option cannot be used with `--enable-java-per-file-error-recovery`.

`--enable-java-per-file-error-recovery`

Use this option if your native Java compiler is able to compile your source code successfully, but `cov-emit-java` crashes. This option might also help when `cov-emit-java` has parse errors and there are no class files available for per-class error recovery.

`--encoding <character_encoding>`

Applies the specified character encoding to all source files processed by this invocation of `cov-emit-java`. Defaults to UTF-8. This default might not be the same one that `javac` uses.

Supports the following encodings (note that these differ from what `javac` supports):

US-ASCII

UTF-8

UTF-16

UTF-16BE

UTF-16 Big-Endian

UTF-16LE

UTF-16 Little-Endian

UTF-32

UTF-32BE

UTF-32 Big-Endian

UTF-32LE

UTF-32 Little-Endian

ISO-8859-1

Western European (Latin-1)

ISO-8859-2

Central European

ISO-8859-3

Maltese, Esperanto

ISO-8859-4

North European

ISO-8859-5

Cyrillic

ISO-8859-6

Arabic

ISO-8859-7

Greek

ISO-8859-8

Hebrew

ISO-8859-9

Turkish

ISO-8859-10

Nordic

ISO-8859-13

Baltic Rim

ISO-8859-15

Latin-9

Shift\_JIS  
Japanese

EUC-JP  
Japanese

 **Note**

EUC-JP is now a valid output object encoding. See `--output_object_encoding`.

ISO-2022-JP  
Japanese

GB2312  
Chinese (EUC-CN)

ISO-2022-CN  
Simplified Chinese

Big5  
Traditional Chinese

EUC-TW  
Taiwanese

EUC-KR  
Korean

ISO-2022-KR  
Korean

KOI8-R  
Russian

windows-1251  
Windows Cyrillic

windows-1252  
Windows Latin-1

windows-1256  
Windows Arabic

MacRoman  
The `cov-emit-java` treats MacRoman as Macintosh.

 **Note**

Some other unsupported encoding names might be supported if a known alias is supported. For example, the Java canonical `x-EUC-TW` is mapped to `EUC-TW`.

The `cov-emit-java` command attempts to tolerate encoding errors and logs a warning when it finds bytes that cannot be decoded.

`--findears <directory_list>`

[Java Web application option] Searches the specified directories recursively for EAR (Enterprise Archive) files and adds the ones that it finds to the emit in the intermediate directory. The directory list must be separated by a semi-colon (;) on the Windows platform, and by a colon (:) on other platforms.

In most cases, Coverity recommends that you use only one of these related options (`--findwars`, `--findwars-unpacked`, `--findears`, or `--findears-unpacked`). The option should match the final packaged `web-app` format. Otherwise, the search might find and emit unwanted temporary build artifacts.

 **Note**

The Web application archives should not contain obfuscated classes.

`--findears-unpacked <directory_list>`

[Java Web application option] Searches the specified directories recursively for unpacked `web-app` root directories and add the ones that it finds to the emit in the intermediate directory. The `web-app` root directories are identified by the presence of a `META-INF/application.xml` file. The directory list must be separated by a semi-colon (;) on the Windows platform, and by a colon (:) on other platforms.

In most cases, Coverity recommends that you use only one of these related options (`--findwars`, `--findwars-unpacked`, `--findears`, or `--findears-unpacked`). The option should match the final packaged `web-app` format. Otherwise, the search might find and emit unwanted temporary build artifacts.

 **Note**

The Web application directories should not contain obfuscated classes.

`--findjars <Jar_containing_directories>`

[Java Web application option] Allows you to specify a list of directories, which must be separated by a semi-colon (;) on the Windows platform, and by a colon (:) on other platforms. The `cov-emit-java` command searches these directories recursively for Jar files and adds the ones that it finds to the classpath. The directories are searched in the order specified in `<Jar_containing_directories>`.

Note that this option can result in an error if the number of Jar files exceeds the limit on the number of open files that is allowed by your operating system.

`--findsource <source_directories>`

Lists directories, which must be separated by a semi-colon (;) on the Windows platform, and by a colon (:) on other platforms. The `cov-emit-java` command searches these directories recursively for source files. It process the source files that it finds as if they were specified directly on the `cov-emit-java` command line.

**--findwars <directory\_list>**

[Java Web application option] Searches the specified directories recursively for WAR (Web application archive) files and adds the ones that it finds to the emit in the intermediate directory. The directory list must be separated by a semi-colon (;) on the Windows platform, and by a colon (:) on other platforms.

In most cases, Coverity recommends that you use only one of these related options (`--findwars`, `--findwars-unpacked`, `--findears`, or `--findears-unpacked`). The option should match the final packaged `web-app` format. Otherwise, the search might find and emit unwanted temporary build artifacts.

 **Note**

The Web application archives should not contain obfuscated classes.

See also `--webapp-archive` and `--findwars-unpacked`.

**--findwars-unpacked <directory\_list>**

[Java Web application option] Searches the specified directories recursively for unpacked `web-app` root directories and adds the ones that it finds to the emit. The `web-app` root directories are identified by the presence of a `WEB-INF/web.xml` file. The directory list must be separated by a semi-colon (;) on the Windows platform, and by a colon (:) on other platforms.

In most cases, Coverity recommends that you use only one of these related options (`--findwars`, `--findwars-unpacked`, `--findears`, or `--findears-unpacked`). The option should match the final packaged `web-app` format. Otherwise, the search might find and emit unwanted temporary build artifacts.

 **Note**

The Web application directories should not contain obfuscated classes.

See also `--webapp-archive` and `--findwars`.

**--force**

Disables incremental compilation by forcing `cov-emit-java` to compile and generate output for a file, even if a copy of that file has already been compiled and is present in the Intermediate Directory.

**--help**

Prints a usage message to the command console and exits.

**--ignore-sccs**

Ignores all directories named `SCCS`. This is useful for version control systems that store metadata with `.java`, `.jar`, and `.class` extensions in directories named `SCCS`.

**--input-file <resource\_file>**

[Used for Coverity Extend SDK checkers that analyze Android applications] Identifies a resource file, typically a `AndroidManifest.xml` file, that can be read by custom Extend SDK checkers built for Android analysis. This option can be specified multiple times on the command line.

Requirement: When using this option, you must also specify the `--android-apk` option. For information about custom Android checker development, see the section, "Reporting events and defects on input files" in *Coverity Extend SDK 2020.12 Checker Development Guide*.

`--javac-version`

Identifies which implementation's bugs to emulate. Oracle Javac is the standard, but there are places where Oracle Javac does not conform to the specification and Eclipse does. To accommodate this, Eclipse attempts to implement Oracle bugs and ties it to the `--javac-version` switch. If the `--source` option is explicitly defined, then the `--javac-version` option is set to the same value. If the `--javac-version` option is explicitly defined, then the `--source` option is set to the same value. If both options are defined, then they work with the values that they are explicitly set to. The default value is 1.8.

`--jvm-max-mem`

[Java Web application security option] Sets the value of the JVM that is used for invoking the Jasper engine for JSP compilation. The option accepts an integer that specifies a number of megabytes (MB). The default value is 1024.

`--kotlin-jvm`

Enables the compilation of Kotlin source code instead of Java source code. The `cov-emit-java` command can be used to capture bytecode written in either Java or Kotlin.

`--limit-modules <module-names>`

[Java 9+ only] Allows you to specify the name of one or more modules in a comma-delimited list. The observable modules will then be restricted to the transitive closure of those those specified in the `limit-modules` option, in addition to any modules specified by the `--add-modules` option.

`--lombok-jar`

Set this option to the location of the lombok jar file when running `cov-emit-java` on files that use Lombok.

`--minimal-classpath-emit`

Limits the group of emitted JAR files to those needed for compilation of the Java files. The default behavior without this option is to emit all the JAR files in the classpath regardless of whether they are referenced by a Java file in the compilation. This option can improve performance of Java builds with large numbers of unused JAR files on the classpath at the risk of not capturing all the dependencies of the those JAR files. For example if `A.java` references `A.jar`, which has dependencies on `B.jar`, this option will prevent `B.jar` from getting emitted even if `B.jar` is on the classpath.

`--module-path <directories_or_jar_or_jmod_files>`

[Java 9+ only] This option allows you to specify a list of directories, JAR files, or JMOD files, which must be separated by a semi-colon (;) on the Windows platform, and by a colon (:) on other platforms.

`--module-source-path <directory>`

[Java 9+ only] This option allows you to specify where to find source files for multiple modules.

`--no-compiler-outputs`

Indicates that the `--compiler-outputs` option is intentionally unspecified. Use of this option is not recommended because both the dynamic analysis for Java Web application security and the

SpotBugs analysis rely on a compiler output specification. Without emitting compiler outputs, you can expect to see false positive XSS reports and missing SpotBugs reports.

It is an error to run `cov-emit-java` without exactly one of the following options: `--no-compiler-outputs` or `--compiler-outputs`.

`@@<response_file>`

Specify a response file that contains a list of additional command line arguments, such as a list of input files. Each line in the file is treated as one argument, regardless of spaces, quotes, etc. The file is read using the platform default character encoding. Using a response file is recommended when the list of input XML files is long or automatically generated.

Optionally, you can choose a different encoding, by specifying it after the first "@". For example:

```
cov-emit-java [OPTIONS] @UTF-16@my_response_file.txt
```

You must use a supported Coverity encoding, listed under the `cov-build --encoding` option.

`--skip-emit-war-javascript-source`

[Java Web application option] Skip capture of JavaScript source code embedded in a Web application archive (.WAR file, .EAR file, or equivalent unpacked directory).

`--skip-war-sanity-check`

[Java Web application option] Suppresses a failure in the case that the emit process determines that expected contents of the Web application (web-app) archives are missing.

This option overrides the following sanity check on each WAR file or Web application directory on the command line:

- The check that each contains a `/WEB-INF` directory and `/WEB-INF/web.xml` file.

This option overrides the following sanity check on the set of all Web application archives or directories on the same command line:

- The check that the Web applications do not contain enough (>20%) of the classes captured during build capture or manual `cov-emit-java` invocations.

These checks are designed to catch cases where someone passes the wrong items to `--webapp-archive`. Turn off this check *only if* you are certain that you are passing the correct Web application files or directories to `cov-emit-java`, despite the warnings.

For additional details, see `--webapp-archive` and `--skip-webapp-sanity-check`.

`--scala`

Enables the compilation of Scala source code instead of Java source code. The `cov-emit-java` command can be used to capture bytecode written in either Java or Scala.

 **Note**

The Scala compiler will automatically add default libraries like `scala-library.jar` to the classpath, unlike the typical `cov-emit-java` behavior (where libraries do not need to be

added explicitly). The `cov-build` command will do this automatically and is the recommended approach for capturing Scala source code.

`--source <Java_version>`

Identifies which version of the Java language to emulate. For example, `--source=1.7` will allow `cov-emit-java` to handle binary literals and other features that appeared in Java 7. If `--source` is explicitly defined, then the `--javac-version` is implicitly set to the same value. If `--javac-version` is explicitly defined, then the `--source` option is implicitly set to the same value. If both are explicitly defined, then both have the value they are explicitly set to. The default value is 1.8.

`--sourcepath <source_directories>`

Lists directories, which must be separated by a semi-colon (;) on the Windows platform, and by a colon (:) on other platforms. Like `javac`, the `cov-emit-java` command searches these directories for source files that contain referenced classes. If no `--sourcepath` is provided, the `sourcepath` will default to the expanded classpath.

`--system <directory> | none`

[Java 9+ only] Allows you to specify the location of the JRE or JDK to pull system libraries from. This is the replacement for the `bootclasspath` in Java 9+. For more information on how system libraries are used during analysis, see the `--bootclasspath <directories_or_jar_files>` option.

`--use-fe`

The `--use-fe` option specifies either `edg` or `ecj` as a frontend. `edg` specifies the EDG-based frontend, and `ecj` specifies the Eclipse-based frontend.

For example, to configure capture to use the EDG frontend for java:

```
cov-configure --xml-option 'append_arg:--use-fe=edg' --java
```

Note that it is not recommended to use this command sequence unless encountering significant issues with the default frontend. The default frontend as of Napa (2018.01) is Eclipse.

`--webapp-archive <archive_file_or_dir>, --war <archive_file_or_dir>, --ear <archive_file_or_dir>`

[Java Web application option] The `--webapp-archive` and `--war` options store the contents of the specified Web Archive (WAR, `.war`) file, Enterprise Archive (EAR, `.ear`), or directory with the unpacked contents of either to the intermediate directory (emit repository) and prepares them for analysis. For these two options, the `cov-emit-java` command inspects the file or directory that is provided as argument and it guesses its type, based on the presence of `WEB-INF` (for WAR) or `META-INF` (for EAR), falling back to WAR by default. The `--ear` is similar, but it only interprets its argument as an EAR.

These options can be passed multiple times to store and analyze multiple archives.

 **Note**

You need to emit any JSP files so that the analysis can find and report defects in them, particularly XSS issues. The build capture does not emit JSP files (which are typically compiled at runtime).

The preferred method to emit JSPs is to use this option to capture the Web application archive(s) that contain them. The advantage of this approach is that the archives also include compilation dependencies and important configuration files.

Another method to emit JSPs is with filesystem capture. See the section called "Filesystem capture for interpreted languages" in the `cov-build` documentation. This method is appropriate if the JSPs are not packaged into a Web application archive file. This includes Spring Boot "Fat JARs" and other deployment systems that do not include JSP source for runtime compilation.

Because these two methods are complementary, care should be taken to avoid emitting redundant copies of the same JSP files. To exclude specific filesystem paths, see `cov-build --fs-capture-search-exclude-regex`. To disable the filesystem capture of JSPs, see `cov-configure --no-jsp`.

In addition to JSP files, JavaScript files embedded inside Web application archives are emitted.

Example:

```
cov-emit-java --dir my/intermediate/dir
--webapp-archive path/to/webapp.war
--webapp-archive path/to/webapp2.war
```

You can also specify a list of directories to search for WAR or EAR files (or unpacked directories) using one of the following options to this command: `--findwars`, `--findwars-unpacked`, `--findears`, or `--findears-unpacked`.

After using this option, you can run an analysis with the `--webapp-security` option to `cov-analyze`. See "Running a security analysis on a Java Web application" [↗](#).

See also `--findwars` and `--findwars-unpacked`.

`--verbose`

Outputs extra information about inputs. Valid values: 0, 1, 2, 3, 4. Defaults to 1.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects(*Coverity 2020.12 Command Reference*), cov-analyze, and cov-build.

## Examples

```
> cov-emit-java --findsource src --findjars lib:build-lib/ --dir
my/intermediate/dir --compiler-outputs build/classes/:build/junitclasses/
```

## See Also

cov-build

cov-analyze

---

## Name

`cov-emit-swift` Parses one or more source files and outputs it into an intermediate directory.

## Synopsis

```
cov-emit-swift --dir <intermediate_directory> [OPTIONS] [sourcefiles]
```

## Description

The `cov-emit-swift` command parses source code and saves `swiftc` compiler output to the emit repository in the intermediate directory.

## Options

- D <value>  
Mark a conditional compilation flag as true.
- F <value>  
Add directory to framework search path.
- I <value>  
Add directory to the import search path.
- import-objc-header <value>  
Name of the Objective-C header file to import.
- import-underlying-module  
Implicitly imports an Objective-C half of a module.
- module-name <value>  
Name of the module to build.
- no-module-emit-observing  
Disables the default `cov-emit-swift` behavior, where dependencies are required to be built from the source files in order to be imported correctly.
- sdk <sdk>  
Compile against <sdk>.
- target <value>  
Emit code for the given target.
- Xcc <value>  
Pass <arg> to the C/C++/Objective-C compiler.

## Error codes

- 0  
Success.

2

Most likely, a user-generated error, such as code that will not compile.

4

Irrecoverable error. Please contact [software-integrity-support@synopsys.com](mailto:software-integrity-support@synopsys.com) if you receive this error code.

---

## Name

cov-emit-vb Parse Visual Basic source code and emit output to the intermediate directory.

## Synopsis

```
cov-emit-vb --dir <intermediate_directory> [OPTIONS] [sourcefiles]
```

## Description

This command parses source code, decompiles referenced assemblies, and saves VBC compiler output to the emit repository in the intermediate directory.

## Options

### --addmodule <file>

Identifies a module that is referenced by the compilation but not added to the emit repository. The location of the module can be absolute or relative. For rules on specifying the location, see --reference.

It is an error if the referenced module is not found. To change this behavior, see --allow-missing-refs.

You must specify this option separately for each module.

### --allow-missing-refs

Issues a warning if any referenced assemblies are missing. If you do not set this option, missing assemblies result in an error that stops the process. This error applies to explicitly referenced assemblies (see --reference and --addmodule) that are absolute or not found in the Common Language Runtime (CLR) system directory. The error also applies to `microsoft.dll` (see --nostdlib). Note that missing `mscorlib.rsp` assemblies (see --noconfig) always result in a warning.

### --codepage <codepage>

Identifies the numeric codepage corresponding to codepages that are supported by VBC with the /codepage option. Source file encodings are determined in the following manner:

If a byte order mark (BOM) is present in the source file, the command uses the BOM-related encoding. If a BOM is *not* present, encoding is determined in the following manner:

- The character encoding of the specified codepage is used. If a codepage is not specified, the command attempts to detect and use UTF-8. If neither of the preceding alternatives is possible, the command uses the system default codepage.

### --compiler-dir <directory>

Identifies the CLR system directory. It is an error to specify a directory that does not exist. The CLR system directory is used as a search path for referenced assemblies (see --reference) and to locate the `mscorlib.rsp` file (see --noconfig). If --compiler-dir is not specified, the command defaults to `$SYSTEM_ROOT/Microsoft.NET/Framework/<version>`, where <version> is the latest supported framework version (for details, see the *Coverity 2020.12 Installation and Deployment Guide* [🔗](#)). It is an error if no suitable CLR system directory is found.

**--define <define>**

Corresponds to the VBC preprocessor directive and `/define` option. Note that it is necessary to specify a separate `--define` option for each directive.

**--dir <intermediate\_dir>**

Identifies the intermediate directory into which this command emits source files and referenced assemblies. An error occurs if the specified intermediate directory exists but is not valid, or if the directory does not exist and cannot be created.

This option is required.

**--disable-ref-assembly-replacement**

By default, the `cov-emit-vb` command attempts to replace each reference assembly it encounters with a version of the assembly that includes an implementation. Use this switch to disable this default behaviour. The recommended use of this switch is as an `xml-option` option to a `cov-configure` command.

Example of adding this switch to a Visual Basic configuration:

```
cov-configure --cs -c config/config.xml --xml-option="<append_arg>--disable-ref-assembly-replacement</append_arg>"
```

**--enable-cs-parse-error-recovery**

Makes `cov-emit-vb` fall back to error recovery mode when compilation errors are encountered during the processing of source files.

This option is disabled by default.

**--force**

By default, if all the specified source files exist in the emit repository with the same timestamp as the file on disk, the command skips the files and exits with a successful return code. Specifying the `--force` option makes the command process the source files, even if they exist unchanged in the emit repository.

**--imports <namespace name>**

Imports a namespace from an assembly. Corresponds to the VBC option `/imports`.

**--langversion <Visual Basic language version>**

Specifies the Visual Basic Language version to use. Corresponds to the VBC option `/langversion`.

**--lib <directory>, -L <directory>**

Identifies a library directory to use when searching for referenced assemblies (see `--reference`). A warning (not an error) occurs if you specify a directory that does not exist. This option corresponds to the VBC `/lib` option.

You must specify this option separately for each library directory.

**--link <[alias=]filename> | -l <[alias=]filename>**

Supersedes `--use-link-semantics`. Effectively the same as `--reference`, but changes how the compiler treats certain COM interop types. Corresponds to the VBC option `/link`.

**--no-banner**

Suppresses the `cov-emit-vb` application name and version banner.

**--noconfig**

Ignores the `vbc.rsp` file under the CLR system directory (see `--compiler-dir`). If this option is not set, the references `/r` or `/reference` within `vbc.rsp` are added to the list of referenced assemblies. Any `vbc.rsp` references that are not absolute filenames are subject to the search directory rules (for details, see `--reference`). Corresponds to the VBC option `/noconfig`.

**--no-friends**

Prevents the compilation from accessing internal types or members. This option works by disabling the processing of compiler output retrieved through the `--out` option. This behavior corresponds to VBC behavior in the case where `/out` is not specified and a default name is used.

**--no-out, -n**

Indicates that there is no compiler output.

It is necessary to specify `--out` or `--no-out` (or the alternative, `-n`), else an error will occur.

**--nostdlib**

Disables the default behavior of searching for `mscorlib.dll` in the CLR system directory and adding the file to the list of referenced assemblies. If that file is not found, the next search in this directory is for a `vbc.exe.config` file that specifies a `requiredRuntime` version. If a version is found, the search continues to the corresponding directory (the parent directory of the CLR system directory).

This option corresponds to the VBC `/nostdlib` option.

**--optioncompare**

Controls whether string comparisons should be binary or use locale-specific text semantics.

Corresponds to the VBC option `/optioncompare`.

**--optionexplicit**

Controls whether the compiler enforces explicit declaration of variables. Corresponds to the VBC option `/optionexplicit`.

**--optioninfer**

Controls whether the compiler allows use of local type inference in variable declarations.

Corresponds to the VBC option `/optioninfer`.

**--optionstrict**

Controls whether the compiler uses strict language semantics. Corresponds to the VBC option `/optionstrict`.

**--out <file>**

Specifies the compiler output file, which is then used by the command to access internal types or members in referenced assemblies.

Subsequent calls to `cov-emit-vb` will not re-emit the output file if a referenced assembly is found.

It is necessary to specify `--out` or `--no-out` (or the alternative, `-n`), else an error will occur.

**--ref-assembly-replacement-search-path <path>**

Adds a search path to assist `cov-emit-vb` in finding implementation versions of reference assemblies. You can specify this option multiple times. This option is useful for pointing `cov-emit-vb` at the correct .NET Core framework directory in cases where `cov-emit-vb` fails to find the correct version. The recommended use of this switch is as an `xml-option` option to a `cov-configure` command.

Example of adding this switch to a Visual Basic configuration:

```
cov-configure --vb -c config/config.xml --xml-option="<append_arg>--ref-assembly-replacement-search-path=D:\utilities\dotnet-core\dotnet-sdk-2.1.403-win-x64\shared\Microsoft.NETCore.App\2.1.5\</append_arg>"
```

**--reference <[alias=]filename> | -r <[alias=]filename>**

Identifies an assembly to provide for compilation and addition to the emit repository, unless this setting is overridden by other options. The location of the assembly can be absolute or relative. If relative, the command searches the following paths in the following order:

1. Current working directory.
2. CLR system directory.
3. Each directory specified by the `--lib` option, in the order specified.

By default, it is an error if a referenced file cannot be found on disk. See `--allow-missing-refs` for complete rules.

Note that `[alias=]` identifies an extern alias directive, just as it does for VBC.

Example with an alias:

```
--reference v1=my.dll
```

**--removeintchecks**

Disables integer overflow checking by the compiler. Corresponds to the VBC option `/removeintchecks`.

**--rootnamespace <namespace name>**

Specifies a namespace for all type declarations. Corresponds to the VBC option `/rootnamespace`.

**--sdkpath <directory>**

Specifies where to search for `mscorlib.dll` and `Microsoft.VisualBasic.dll`. Corresponds to the VBC option `/sdkpath`.

**--target**

Specifies the type of assembly that was created. Corresponds to the VBC option `/target`.

**--vbruntime -|+|\***

Specifies whether the compiler should compile with a reference to `Microsoft.VisualBasic.dll` (+), without a reference to it (-), or to embed it within the assembly (\*). Corresponds to the VBC option `/vbruntime[+|-|*]`.

--vbruntime-path <filename>

Specifies the exact file the compiler should use for `Microsoft.VisualBasic.dll`. Corresponds to the VBC option `/vbruntime:<filename>`.

Any unrecognized options result in an error, which causes an immediate exit with an appropriate error message.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

---

## Name

`cov-export-cva` Produce a Coverity CVA file to be used by Architecture Analysis.

## Synopsis

```
cov-export-cva --output-file <filename>
```

## Description

The `cov-export-cva` command produces a CVA file to be used by Architecture Analysis. This allows you to create and retrieve the CVA file from your intermediate directory instead of having to run a commit and then log in to Coverity Connect.

A `cov-analyze` command with at least one `--strip-path` option must be run at some point prior to running this command. After this command is executed, the filename you specified in `--output-file <filename>` is created.

The file is compressed in `gzip` format and is ready to be read by Architecture Analysis.

## Options

`--dir <intermediate_directory>`

Path name to an intermediate directory that is used to store the results of the build and analysis.

`--output-file <filename>`

The name of the file CVA file to be created.

`--output-tag <name>`

Use this option if you used it when generating analysis results. See the `--output-tag` option to `cov-analyze`.

## Shared options

`--debug, -g`

Turn on basic debugging output.

`--ident`

Displays the version of Coverity Analysis and build number.

`--info`

Displays certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

`--tmpdir <tmp>, -t <tmp>`

Specifies the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects`(*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

### **See Also**

`cov-analyze`

---

## Name

`cov-extract-scm` Extracts the change data for each line from files in the SCM system.

## Synopsis

```
cov-extract-scm --scm <scm_type> --output <output_file> --input <input_file>
[--scm-tool <tool_path>] [--scm-project-root <root_path>] [--scm-tool-arg
<tool_args>] [--scm-command-arg <command_arg>] [--log <log_path>] [--ms-delay
<int>] [--get-baseline-code-version][--get-modified-files][--OPTIONS]
```

## Description

Queries an SCM system for information of use to Coverity, Automatic Owner Assignment and Fast Desktop. `cov-extract-scm` operates in one of two modes: "Annotate" (the default mode) or "Code version".

### Annotate mode

Retrieves the change data for each line of a file from the SCM system. This is the default mode.

### Code version mode

Retrieves information regarding the code version the user has most recently checked out, updated from, or pulled. The information is either about the code version itself or the unpublished changes since the code version. (See `cov-run-desktop`.)

This mode is activated by the `--get-baseline-code-version` or `--get-modified-files` options.

## Using SCM argument tags

`cov-extract-scm` issues a call to the SCM system to get information about last modified dates for each line in a source file. On most systems, this command is either "blame" or "annotate". For example:

```
accurev annotate foo.c
```

In some cases, SCM systems have additional items that need to be specified in the command to allow Coverity to get the appropriate information from the system. The `--scm-param` option (as well as the deprecated `--scm-tool-arg` and `--scm-command-arg` options) allow for this functionality. Furthermore, the `--scm-tool` allows you to specify an SCM tool that is non-standard or provides command output in a format that is not easily parsed by Coverity. For example:

```
<tool> <tool-args> <command> <command-args> <coverity-mandated-flags> <target-file>
```

- `<tool>` is from the `--scm-tool` argument. If it is not specified, an appropriate default is used for the specified SCM type.
- `<tool-args>` and `<command-args>` are , respectively, lists of values associated with the `tool_arg` and `annotate_arg` keys passed to `--scm-param`.
- `<command>` is specific to each source control management system and can not be modified; typically a variation of blame or annotate. If the command is not specified, Coverity uses the appropriate command for the specified SCM type.

- <coverity-mandated-flags> are also specific to each source control management system and can not be modified.
- <target-file> is generated from the data passed in the --input option.

The <command> and <coverity-mandated-flags> syntax for each supported SCM is as follows:

- accurev:

```
<tool-args> annotate <command-args> -fudtv <file>
```

- clearcase:

```
<tool-args> annotate <command-args> -out - -nheader -f -fmt "revision %X\nauthor %u\nusername %Lu\nndate %d\n\t" <file>
```

- cvs:

```
log <file>
```

```
<tool-args> annotate -F <command-args> <file>
```

 **Note**

For CVS, <command-args> and <tool-args> are NOT passed to both commands, they are only applied to the annotate command.

- git:

```
<tool-args> blame <command-args> -p <file>
```

- hg:

```
1. <tool-args> log <command-args> -f --template \
 'author {author|person}\nusername {author|user}\nndate \
 {date|hgdate}\nchangeset_long {node}\nchangeset_short {node|short}\n---
 \n' <file>
```

```
2. <tool-args> annotate <command-args> -c <file>
```

Any tool-args/command-args are passed to both commands.

- perforce:

```
1. <tool-args> changes <command-args> -t -i <file>
```

```
2. <tool-args> annotate <command-args> -q -I <file>#have
```

Any tool-args/command-args are passed to both commands.

- plastic:

```
<tool-args> annotate <command-args> <file> --format={owner}|{date}|{rev}
```

- plastic-distributed:

```
<tool-args> annotate <command-args> <file> --format={owner}|{date}|{rev}
```

- svn:

```
<tool-args> blame <command-args> -xml <file>
```

- tfs and ads:

Coverity uses a custom application (`cov-internal-tfs`) to gather information from the Team Foundation Server (TFS) or Azure DevOps Server (ADS) tools. Because of this, the `<tool-args>` and `<command-args>` arguments are rejected, and the underlying commands to TFS/ADS can not be modified.

#### Note

The value for `<file>` is almost always the file name and not a file path given in `--input`. Coverity changes to the directory where the file resides and issues the command. The exception to this is with `accurev` using the `--scm-project-root` option. In this case, Coverity changes to the directory where the file resides but `<file>` is a filepath that is relative to the project root. This is because a valid deployment model for `accurev` is a detached workspace, in which file information must be called with a relative path to the root of where it was checked out.

Additionally, when TFS/ADS named items (such as directories, files, and branches) are renamed, some of the historical information (in particular changeset history) is potentially not retrievable for modifications that occurred before the rename. Because of this, `cov-import-scm/cov-extract-scm` is unable to properly associate SCM data on a per line basis.

When modifications are performed on a branch and then later merged to a second branch, running `cov-import-scm/cov-extract-scm` on the second branch will indicate that all modifications occurred at the date of the merge, rather than the date they actually occurred.

## A note on Git superprojects

Git superprojects are unsupported by `cov-extract-scm`'s code version mode (activated by the `--get-baseline-code-version` and `--get-modified-files` options).

You may be able to workaround this issue by creating a script to access Git using submodules, and specify that script with the `--scm-tool` option. This is an advanced usecase, and should only be attempted by experienced users.

## Options

`--error-threshold <percentage>`

Sets the percentage of successful extractions required for `cov-extract-scm` to exit with a success return code (0). If the extraction rate is below this threshold, `cov-extract-scm` will print a warning and exit with return code 8. The default percentage is 80.

`--get-baseline-code-version`  
(Code version mode only)

This switch indicates that instead of performing its usual function of querying the SCM for "annotate" information, the tool shall write to its `--output` file some information about the code version that the user has most recently checked out, updated from, or pulled.

The output shall be a JSON file using ASCII character encoding and platform-native line endings. It consists of a single JSON object with a single attribute called "date" using YYYY-MM-DDThh:mmZ syntax.

Example output file:

```
{
 date: "2013-12-18T15:34Z"
}
```

`--get-modified-files`  
(Code version mode only)

This switch also indicates to suppress normal processing and instead retrieve the set of files with unpublished local changes. These are the files with differences relative to the version of the code indicated by `--get-baseline-code-version`.

The output shall be written to the `--output` file as JSON using ASCII character encoding and "\u" escapes in strings as needed to represent non-ASCII characters. The output is a single JSON object with two attributes, "modified\_files" and "untracked\_files". The former are files that the SCM knows about and have been modified from their baseline version. The latter are files that are not checked in to the SCM and are also not excluded by SCM "ignore" filtering (like .gitignore). Each is an array of strings representing the file names. File names have their letter case preserved and use the platform native syntax for file names and directory separators. The file names shall be relative to the repository root, which is assumed to be the current directory unless a different root is specified as a command line option with `--scm-project-root`.

Example output file (using Windows separators):

```
{
 modified_files: [
 "utilities\\cov-format-errors\\cov-format-errors.cpp",
 "Makefile"
],
 untracked_files: [
 "analysis\\cov-run-desktop\\some-new-file.cpp",
 "analysis\\cov-run-desktop\\some-new-file.hpp",
 "name with \u1234 non-ASCII character"
]
}
```

`--input <input_file>`  
(Annotate mode only)

Specifies the path to a file that contains information about the files that gather last modified dates. The format of this file is the same as the output of the `list-scm-unknown` option of `cov-manage-emit`.

`--log <log_path>`

Specifies the path to a file to which output from the `--scm-tool` executable and other recoverable errors are written.

`--ms-delay <int>`

Specifies a delay in milliseconds between calls to the underlying SCM. This is useful for preventing a denial of service situation.

`--output <output_path>`

Specifies the path to a file to which the output data is written to. The format of this output (in Annotate mode) is used as input to the `add-scm-annotations` subcommand for `cov-manage-emit`. See `--get-baseline-code-version` and `--get-modified-files` for the format of this output in Code version mode.

`--scm <scm_type>`

Specifies the name of the source control management system. For this option to function correctly, your source files must remain in their usual locations in the checked-out source tree. If the files are copied to a different location after checkout, the SCM query will not work.

Possible `scm_type` values:

- Accurev: `accurev`
- Azure DevOps Server (ADS): `ads`  
Windows only.
- ClearCase: `clearcase`
- CVS: `cvs`
- GIT: `git`
- Mercurial: `hg`
- Perforce: `perforce`
- Plastic: `plastic|plastic-distributed`.

Use `plastic` when working in a non- or partially-distributed Plastic configuration. Use `plastic-distributed` when working in a fully-distributed Plastic configuration.

- SVN: `svn`
- Team Foundation Server (TFS): `tfs`

Windows only.

For usage information for the `--scm` option, see `cov-extract-scm`.

 **Note**

The following commands or setup utilities must be run before `cov-extract-scm` in order to successfully communicate with the SCM server:

- `accurev`:

Login command

- `perforce`:

The environment variable `P4PORT` should be set to the value expected by the `p4` tool.

- `tfs` or `ads`:

Windows credentials in Credential Manager to access the TFS server

`--scm-command-arg <command_arg>`  
(Annotate mode only)

This option has been deprecated. Instead of using `--scm-command-arg arg1`, use `--scm-param annotate_arg=arg1`. Specifies additional arguments that are passed to the command that retrieves the last modified dates. The arguments are placed after the command and before the target file. This option can be specified multiple times.

`--scm-param`

Specifies additional arguments that are passed to the SCM tool in a context-aware manner. The value passed to `--scm-param` must have the format `key=arg`; the key specifies what the arg is to be used for. For example, `--scm-param tool_arg=--foo` causes the argument `--foo` to be added to the `<tool-args>` list, and `--scm-param annotate_arg=--bar` causes the argument `--bar` to be added to the `<command-args>` list. Specific SCMs may accept other keys, if they require more information.

`--scm-project-root <root_path>`

Specifies a path that represents the root of the source control repository.

In Annotate mode, this option is only used when specifying `accurev` as the value to `--scm`. When this is used, all file paths that are used to gather information are interpreted as relative to this project-root path.

In Code version mode, this option allows `cov-extract-scm` to run from a directory other than the root of the source control repository. All filenames returned by `--get-modified-files` are relative to this path.

`--scm-tool <tool_path>`

Specifies the path to an executable that interacts with the source control repository. If the executable name is given, it is assumed that it can be found in the path environment variable. If not provided, the command uses the default tool for the specified `--scm` system.

`--scm-tool-arg <tool_args>`

This option has been deprecated. Instead of using `--scm-tool-arg arg1`, use `--scm-param tool_arg=arg1`. Specifies additional arguments that are passed to the SCM tool, specified in the `--scm-tool` option, that gathers the last modified dates. The arguments are placed before the command and after the tool. This option can be specified multiple times.

## Shared options

`--debug, -g`

Turn on basic debugging output.

`--ident`

Displays the version of Coverity Analysis and build number.

`--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>`

Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.
- 8: This exit code is specific to `cov-extract-scm`. It signifies either that the command attempted to extract a file that does not exist in the SCM, or that there may have been an unknown error.

For exceptions, see `cov-commit-defects`, `cov-analyze`, and `cov-build`.

## See Also

`cov-blame`

`cov-import-scm`

`cov-manage-emit`

`cov-run-desktop`

---

## Name

`cov-find-function` Find a mangled or internal function name when given part of the actual name.

## Synopsis

```
cov-find-function [OPTIONS] <name>
```

## Description

In the Extend SDK, it is relatively easy to match on a C function with a specific name by passing the name to the `Fun` pattern constructor. Otherwise, the mangled name must be used to disambiguate which overloaded function should be matched. The `cov-find-function` command looks for all of the mangled names that contain the given `<name>`.

For other languages, this command finds the internal representation used by the analysis.

This command makes it easier to find the name needed for matching on a specific function or method, even an overloaded one. The `cov-find-function` command can accept a regular expression to denote a set of functions to display.

## Options

`--cpp`

Filters the results by the C/C++ translation units on which this command operates or reports. The command will fail with an informative error message if none of the translation units in the `emit` subdirectory match any of the specified language options. See also, `--cs` and `--java`.

`--cs`

Filters the results by the C# programming language. The command will fail with an informative error message if none of the translation units in the `emit` subdirectory match any of the specified language options. See also, `--cpp` and `--java`.

`--dir <intermediate_directory>`

Path name to an intermediate directory that is used to store the results of the build and analysis.

`--exact, -x`

Assume that `<name>` is a full function prototype, not just a substring of the mangled name. Not commonly used.

`--include-builtins`

Look for given functions in the built-in model library.

`--java`

Filters the results by the Java programming language. The command will fail with an informative error message if none of the translation units in the `emit` subdirectory match any of the specified language options. See also, `--cs` and `--cpp`.

`--model-file <file.xmlldb>`

Look for the function in the given user model file. See also, the `--model-file` option to `cov-analyze`.

`--module <module>, -m <module>`

Requires `--show`. Pick the model for module `<module>` when showing a model for a function. Values can be *all*, *generic*, *security*, *concurrency*, *stack\_use*, *uninit*, and *ptr\_arith*. Defaults to *generic*.

`--output <directory>, -of <directory>`

Specify the directory in which the output of `--save` is stored. The default is the current directory. If the directory does not exist it is created.

`--save`

Save the model file in `<key>.<module>.models.xml`, a description of edges in `<key>.<module>.model_edges`, and a `.ps` file of the graph (as shown by `--show`) in `<key>.<module>.ps`. Each model is uniquely identified by an MD5 hex-key. `cov-find-function` uses this *key* to immediately find a model. If given a function name, it does a linear search of the model database. This search might take some time, but when it finds a function, it prints its model key for the specified module.

Requires the `dot` command from the Graphviz package.

`--show, -s`

Show the model for the function. Requires `dot` (from the Graphviz package) as well as `ggv` (GNOME's PS viewer).

`--subdir`

When used in conjunction with `--use-emit`, specifies a subdirectory in which to look for the given function. Might substantially speed execution.

`--use-emit, -ue`

Iterate over the emit directory to find functions, instead of looking at the cache database. Useful if the cache is not present (for example, it has been cleaned or the analysis has never been run) or corrupted.

`--user-model-file <file.xmlldb>`

[Deprecated] This option is deprecated as of version 7.7.0. Use `--model-file` instead.

## Shared options

`--debug, -g`

Turn on basic debugging output.

`--ident`

Displays the version of Coverity Analysis and build number.

`--info`

Displays certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

`--tmpdir <tmp>, -t <tmp>`

Specifies the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## Example

Look for a function named `get` and save the results in the directory `get_models`:

```
> /cov-find-function --dir /home/user/apache_intdir --info --save --output
get_models get
```

---

## Name

`cov-format-errors` Generate static HTML pages of defect reports.

## Synopsis

`cov-format-errors`

```
--dir <intermediate_dir>
[--emacs-style]
[--exclude-files <regex>]
[--file <file_substring>]
[--filesort]
[--function <function>]
[--functionsort]
[--html-output <directory>]
[--include-files <regex>]
[--json-output-v7 <filename>]
[--output-tag <name>]
[--security-file <file>]
[--title <text>]
[-x]
[-X | --noX]
```

## Description

The `cov-format-errors` command reads defects from an intermediate directory and creates static HTML pages in the specified directory.

**Deprecated behavior:** By default, this command writes HTML output into the `<intermediate_directory>/output/errors` directory, but this usage is deprecated. Instead, you should use the `--html-output` option to specify the HTML output directory.

To commit defects to Coverity Connect, use `cov-commit-defects` instead of `cov-format-errors`.

Note that `cov-format-errors` does not have access to any triage information stored in Coverity Connect. Therefore, the output of `cov-format-errors` will include defects even though they have been marked as *Intentional* or *False Positive* within Coverity Connect. In addition, the output of `cov-format-errors` is only accessible to users who have access to the local file system; it is not made available through a network service.

## Options

`--dir <intermediate_directory>`

Path name to an intermediate directory that is used to store the results of the build and analysis.

`--emacs-style`

Print a short version of the defect event with line numbers to `stdout`, formatted as gcc compiler warnings. This option is useful for integration into an editor such as Emacs.

**--exclude-files <regex>**

Do not output defects from files that are in paths that match the specified regular expression. You cannot use this option multiple times in the same command line. You can use it with the `--include-files` option. If you use both options together, `--exclude-files` takes precedence over `--include-files`. For example, defects from a given file are excluded from the output in the case that the regular expressions both include and exclude the file.

Example that excludes paths that contain both `/bar/` and `.c`:

```
--exclude-files '/bar/.*\.c$'
```

The example excludes the following file:

```
/foo/bar/test.c
```

The example does not exclude the following files:

```
/foo/test.c
```

```
/foo/bar/test.cc
```

 **Note**

The example above uses single quotes around the string value. However, your command interpreter might require double quotes (for example, `"/bar/.*\.c$"`).

**--file <filename>**

Only generate pages for errors in files containing `<filename>` as a substring of the full pathname.

**--filesort**

Sort rows by filename.

**--function <func>**

Only generate pages for errors in function `<func>`.

**--functionsort**

Sort rows by function name.

**--html-output <directory>**

Write the HTML results to the specified directory, and create this directory first if it does not exist.

You must either specify a directory previously written by `cov-format-errors`, or a directory that is empty, or that does not yet exist.

**--include-files <regex>**

Output defects only from files that are in paths that match the specified regular expression. You cannot use this option multiple times in the same command line. You can use it with the `--exclude-files` option. If you use both options together, `--exclude-files` takes precedence over `--include-files`. For example, defects from a given file are excluded from the output in the case that the regular expressions both include and exclude the file.

Example that includes `/foo/`:

```
--include-files '/foo/'
```

The example includes the following file:

```
/bar/foo/test.c
```

The example excludes the following file:

```
/bar/test.c
```

 **Note**

The example above uses single quotes around the string value. However, your command interpreter might require double quotes (for example, `"/foo/"`).

`--json-output-v7 <filename>`

Writes `cov-format-errors` output to the specified file in JSON format [🔗](#). You can include either an absolute path or a path relative to the location in which you execute the command. If you want the filename to end in `.json`, you must include it in the filename.

The `--json-output-v7` option is the recommended JSON output option because it contains the most complete information. The `json-output-v1` through `json-output-v6` options are supported for backward compatibility.

`--lang <language>`

Write event messages in the specified language. Currently, the supported values are `en` (for English), `ja` (for Japanese), and `zh-cn` (for Simplified Chinese). The default language is English (`en`).

`--misra-only`

[Deprecated in 8.0] Using this option will result in an error.

`--noX`

Do not build cross-reference information. Normally, cross-reference information is built if the `-x` option is specified.

`--output-tag <name>`

Use this option if you used it when generating analysis results. See the `--output-tag` option to `cov-analyze`.

`--security-file <license file>, -sf <license file>`

Path to a Coverity Analysis license file. If not specified, this path is given by the `security_file` element in the XML configuration file, or `license.dat` in the same directory as `<install_dir_sa>/bin`.

`--strip-path <directory>`

Strips the prefix of a file name path in error messages and references to your source files. If you specify the `--strip-path` option multiple times, you strip all of the prefixes from the file names, in the order in which you specify the `--strip-path` argument values.

The leading portion of the path is omitted if it matches a value specified by this option. For example, if the actual full pathname of a file is `/foo/bar/baz.c`, and `--strip-path /foo` is specified, then the name attribute for the file becomes `/bar/baz.c`.

`--title <title>`

Specify a title for the generated index pages.

`-X`

Run the `cov-internal-build-xrefs` command first. Without this option, the identifiers in the source code will not be hyperlinked. When this has been done once on an intermediate directory, it does not need to be done again until the intermediate data changes. `-x` automatically implies `-X` unless `--noX` is also specified.

`-x`

Use cross-reference information when building static pages. Without this flag, the identifiers in the source code will not be hyperlinked. This option needs to be specified every time you want the generated pages to have cross-reference information.

## Shared options

`--debug, -g`

Turn on basic debugging output.

`--ident`

Displays the version of Coverity Analysis and build number.

`--info`

Displays certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

`--tmpdir <tmp>, -t <tmp>`

Specifies the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

`--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>`

Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.

- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects`(*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## See Also

`cov-commit-defects`

---

## Name

cov-generate-hostid Return host id information for node-locked licensing.

## Synopsis

```
cov-generate-hostid [-of <filename>]
```

## Description

The `cov-generate-hostid` command outputs the host id information needed for node-lock licensing. Email this information to `software-integrity-license@synopsys.com`.

## Options

`--output-file <filename>, -of <filename>`

The file to create with this information. By default, the information is sent only to standard out.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## Example

Create the host id information in the named file:

```
> cov-generate-hostid -of /user/foo/cov_host.txt
```

---

## Name

cov-help Display help for a command.

## Synopsis

```
cov-help <command>
```

## Description

The `cov-help` command displays help for a specified command.

You can also read each command's help page by specifying the `--help` option on the command line. For example:

```
> cov-analyze --help
```

## Environment Variables

### \$PAGER

The program through which to pipe output. The `$PAGER` variable is ignored if the output is not a terminal. If `$PAGER` is not set, the `cov-help` command looks for the `less` or `more` commands in paths set by the `$PATH` environment variable.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

---

## Name

cov-import-msvsca Import Microsoft Visual Studio Code Analysis issues

## Synopsis

```
cov-import-msvsca <MSVSCA_xml_files> [--append] [--skip-unrecognized] [--no-threshold-check]
```

## Description

`cov-import-msvsca` allows you to import Microsoft Visual Studio Code Analysis (MSVSCA) issues on C# code into your Coverity Analysis intermediate directory by specifying XML files generated by Microsoft Visual Studio Code analysis (MSVSCA). The results can then be committed to and viewed in Coverity Connect. MSVSCA is integrated into many versions of Microsoft Visual Studio, including Professional, Premium, and Ultimate editions of Visual Studio 2012.

Also known as FxCop, Code Analysis for Managed Code reports issues in programs compiled for the .NET Framework, including languages other than C#. Although `cov-import-msvsca` can import most issues from non-C# managed code, `cov-import-msvsca` works best for C# code. `cov-import-msvsca` does not currently support importing results from Code Analysis for C/C++ or Code Analysis for Drivers. `cov-import-msvsca` requires a valid license for Coverity Analysis for C#.

Event descriptions imported by `cov-import-msvsca` are not localized in Coverity Connect. However, you can import localized results (including localized event descriptions). If you configure Visual Studio for a particular localization, the Microsoft Visual Studio Code analysis (MSVCA) results will use that localization. You can then import the localized results to Coverity Connect. Users will see the localized event descriptions in Coverity Connect regardless of their Coverity Connect language configuration.

The `cov-import-msvsca` command is only available on the Windows platform.

For Microsoft's complete list of Code Analysis warnings for Managed Code, see <http://msdn.microsoft.com/en-us/library/dd380629> .

## Importing MSVSCA results and viewing them in Coverity Connect

1. Run the MSVSCA analysis to generate one or more XML files of MSVSCA results.

When the Code Analysis is run from Visual Studio or as part of a build, an XML file with the results for each project is saved in the same directory as the compiler output. For example:

```
<path_to_compiler_output>.CodeAnalysisLog.xml
```

The `FxCopCmd.exe` program that is included with MSVSCA can also generate results in the expected XML format.

2. Run `cov-import-msvsca`, referencing the XML file(s) with MSVSCA results.

For example:

```
cov-import-msvsca --dir idir <xml_files> ...
```

The `cov-import-msvsca` command expects that the assemblies analyzed to generate the MSVSCA XML files were compiled with debug information, and that the assembly, its debug symbol (pdb) file, and original source code have not moved on the filesystem.

Imported MSVSCA results and Coverity Analysis results can be combined in the same intermediate directory using `--append`. All C# results in an intermediate directory will be committed to a single stream in Coverity Connect. If the intermediate directory only contains imported results, source files in the relevant project that do not contain defects might not be stored and committed. (It is possible to correct this with a supplementary `cov-import-results --append` with just source files.)

`cov-import-msvsca` tracks how many MSVSCA issues are in the XML input files and what proportion of those are dropped because it is unable to associate them with a source code line. This can occur if there is missing debug information and/or missing source files. If that proportion is greater than 10%, it is reported as a user error (see `--no-threshold-check` below) and no issues are imported.

Only results that are generated by the Code Analysis with Visual Studio 2012 are officially supported, but other versions are compatible. A warning is printed if an unsupported input file is specified, but the tool attempts to import anyway.

#### **Note**

When `cov-import-msvsca` runs on high-density files (files with more than 100 issues that also average more than 1 issue for every 10 lines of code), the console will print a warning that names all the files that exceed the threshold, and the import process will exclude all issues associated with the affected files from the intermediate directory. This change prevents the Coverity Connect source browser from becoming too crowded with issues.

To suppress this density check (allowing all issues to be imported) in version 7.0, define the environment variable `COVERITY_ALLOW_DENSE_ISSUES` (`COVERITY_ALLOW_DENSE_ISSUES=1`) when running the commands.

### 3. Commit the results to Coverity Connect.

Commit the results with `cov-commit-defects`, specifying the intermediate directory to which you imported the MSVSCA results.

When you (or another user) log into Coverity Connect the MSVSCA issues are similar to other Coverity Analysis issues, but the checker name has a prefix of `MS..`. The rest of the checker name uses the Code Analysis ID. For example, `MS.CA1303`.

## Options

`--dir <intermediate_directory>`

Path name to an intermediate directory that is used to store the results of the build and analysis.

`--append`

Specify that issues should be appended to the intermediate directory. Without `--append`, existing C# issues are deleted before storing the imported issues.

See also, `--output-tag`.

`--codepage <identifier>`

Specifies Microsoft code page source encoding. `<identifier>` is an integer represents the code page identifier, for example `--codepage 1201`. For the list of code page identifiers, see <http://msdn.microsoft.com/en-us/library/windows/desktop/dd317756%28v=vs.85%29.aspx> .

Source with a BOM will have the encoding auto-detected, even if `--codepage` or `--encoding` is specified. However, if you specify `--encoding`, `cov-import-msvsca` will log a warning recommending that you use `--codepage` instead. `--codepage` results in using the .NET mechanisms for decoding, which more closely mimic the Microsoft tools.

You cannot use `--codepage` and `--encoding` together.

`--encoding <encoding_name>`

Specify that source files are read using the named character encoding, such as UTF-8. The default is based on detection of byte-order marks and falling back on the operating environment default character encoding. For a list of encoding options, see `--encoding <enc>` from `cov-emit`.

`--no-threshold-check`

By default, if more than 10% of reported issues do not have file and line information because their referenced assemblies, their associated pdbs, or referenced source files (either from the MSVSCA XML file(s) or assembly pdb) are missing, then `cov-import-msvsca` will fail without importing any issues and print an informative error message that mentions `--no-threshold-check`. However, if `--no-threshold-check` is specified, these messages are just informative and are not treated as an error. The defects that depend on the missing information are omitted from the results and the remaining issues are imported normally.

`--output-tag <name>`

Specifies a non-default location within the intermediate directory for the results of one or more imports. The name can be anything you choose, using characters allowed in file names. When specified *without* the `--append` option, prior results found in this location are replaced. When specified *with* `--append`, new results are added to the result set.

`@@<response_file>`

Specify a response file that contains a list of additional command line arguments, such as a list of input files. Each line in the file is treated as one argument, regardless of spaces, quotes, etc. The file is read using the platform default character encoding. Using a response file is recommended when the list of input XML files is long or automatically generated.

Optionally, you can choose a different encoding, by specifying it after the first "@". For example:

```
cov-import-msvsca [OPTIONS] @UTF-16@my_response_file.txt
```

You must use a supported Coverity encoding, listed under the `cov-build --encoding` option.

`--skip-unrecognized`

By default, `cov-import-msvsca` fails if a specified input file is not in a recognized format or if the list of input files is empty. If `--skip-unrecognized` is specified, files in an unrecognized format are

simply skipped with a warning, and the list of files can be empty. Thus, the translation of any input files in the recognized format proceeds normally, even if there are none.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## See Also

`cov-commit-defects`

---

## Name

cov-import-results Import third-party issues into Coverity Connect

## Synopsis

```
cov-import-results dir [--append] <filename>
```

## Description

`cov-import-results` is the command tool for the Coverity Connect Third Party Integration Toolkit. It imports to the intermediate directory source code files of any type and any issues pertaining to those source files generated by third-party analysis tools.

This command imports third party issue information through a specified JSON import file (<filename> in the above syntax).

### Note

When `cov-import-results` runs on high-density files (files with more than 100 issues that also average more than 1 issue for every 10 lines of code), the console will print a warning that names all the files that exceed the threshold, and the import process will exclude all issues associated with the affected files from the intermediate directory. This change prevents the Coverity Connect source browser from becoming too crowded with issues.

To suppress this density check (allowing all issues to be imported) in version 7.0, define the environment variable `COVERITY_ALLOW_DENSE_ISSUES` (`COVERITY_ALLOW_DENSE_ISSUES=1`) when running the commands.

See [Using the Third Party Integration Toolkit](#) for information about using the Third Party Integration Toolkit and creating the JSON import file.

## Options

### `--append`

Append issues to any issues that exist in the intermediate directory. If `--append` is absent, all of the issues in the intermediate directory are deleted before importing and analysis summaries will not be captured.

If `--append` is present, they are not deleted.

See also, `--output-tag`.

### `--dir <intermediate_directory>`

Path name to an intermediate directory that is used to store the results of the build and analysis.

### `--<lang>`

The value for `lang` may be `cpp`, `cs`, `java`, `javascript`, `objc`, `php`, `python2`, `python3`, `ruby`, `scala`, `swift`, `text-files`, or `vb`.

This option sets the source language and analysis domain in the output `error.xml` file.

- For `cpp`, `cs`, and `java`, the corresponding domain is `STATIC_C`, `STATIC_CS`, `STATIC_JAVA`, or `DYNAMIC_JAVA`.
- For all other source languages, the domain is `OTHER`.

You can only import results for one source language per invocation of `cov-import-results`. However, you can use the `--append` option to add results attributed to other source languages before committing the results to Coverity Connect.

See also, `--output-tag`.

`--no-banner`

Hide the version of Coverity Analysis and build number.

`--output-tag <name>`

Specifies a non-default location within the intermediate directory for the results of one or more imports. The name can be anything you choose, using characters allowed in file names. When specified *without* the `--append` option, prior results found in this location are replaced. When specified *with* `--append`, new results are added to the result set.

`--strip-path <path>, -s <path>`

Strips the prefix of a file name path in error messages and references to your source files. If you specify the `--strip-path` option multiple times, you strip all of the prefixes from the file names, in the order in which you specify the `--strip-path` argument values.

This option is also available with `cov-commit-defects` and `cov-analyze`.

The leading portion of the path is omitted if it matches a value specified by this option. For example, if the actual full pathname of a file is `/foo/bar/baz.c`, and `--strip-path /foo` is specified, then the name attribute for the file becomes `/bar/baz.c`.

 **Note**

Coverity recommends using this option for a number of reasons:

You can enhance end-to-end performance of the path stripping process by using this option during the analysis of your source code, rather than when committing the analysis results to Coverity Connect.

It shortens paths that Coverity Connect displays. It also allows your deployment to be more portable if you need to move it to a new machine in the future.

## Shared options

`--debug, -g`

Turn on basic debugging output.

`--ident`

Displays the version of Coverity Analysis and build number.

**--info**

Displays certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

**--tmpdir <tmp>, -t <tmp>**

Specifies the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

---

## Name

`cov-import-scm` Collects change data for source files from the SCM.

## Synopsis

```
cov-import-scm --scm <scm_type> --dir <intermediate_directory> [--filename-regex <regex>] [--scm-tool <tool_path>] [--scm-project-root <root_path>] [--scm-tool-arg <tool_arg>] [--scm-command-arg <command_arg>] [--log <log_path>] [--ms-delay <int>] [--OPTIONS]
```

## Description

The `cov-import-scm` command simplifies the process of retrieving the SCM change data for source files and adding them to the emit directory. This command automates the following command line flow:

1. `cov-manage-emit list-scm-unknown`
2. `cov-extract-scm`
3. `cov-manage-emit add-scm-annotations`

## Options

`--dir <intermediate_directory>`

Specifies the intermediate directory that is used to store the emit repository.

`--error-threshold <percentage>`

Sets a threshold for the percentage of successful extractions (from `cov-extract-scm`) below which this import command (`cov-import-scm`) will display a warning, indicating the need to check for a potential problem. Note, however, that `cov-import-scm` will attempt to add all successful extractions to the emit. The default percentage is 80.

`--filename-regex <regex>`

Allows finer control over SCM information gathering. Information is gathered only for filenames that match the regex. Any files that do not match are skipped. This is beneficial when there are specific locations where code is known to exist under SCM control and other locations where it is not (such as system headers).

`--log <log_path>`

Specifies the path to a file to which executable and other recoverable errors encountered in `cov-extract-scm` are written.

`--ms-delay <int>`

Specifies a delay in milliseconds between calls to the underlying SCM. This is useful for preventing a denial of service situation.

`--scm <scm_type>`

Specifies the name of the source control management system. For this option to function correctly, your source files must remain in their usual locations in the checked-out source tree. If the files are copied to a different location after checkout, the SCM query will not work.

Possible `scm_type` values:

- Accurev: `accurev`
- Azure DevOps Server (ADS): `ads`  
Windows only.
- ClearCase: `clearcase`
- CVS: `cvs`
- GIT: `git`
- Mercurial: `hg`
- Perforce: `perforce`
- Plastic: `plastic|plastic-distributed`.

Use `plastic` when working in a non- or partially-distributed Plastic configuration. Use `plastic-distributed` when working in a fully-distributed Plastic configuration.

- SVN: `svn`
- Team Foundation Server (TFS): `tfs`

Windows only.

For usage information for the `--scm` option, see `cov-extract-scm`.

 **Note**

The following commands or setup utilities must be run before `cov-import-scm` in order to successfully communicate with the SCM server:

- `accurev`:

Login command

- `perforce`

The environment variable `P4PORT` should be set to the value expected by the p4 tool.

- `tfs` or `ads`:

Windows credentials in Credential Manager to access the TFS or ADS server

`--scm-command-arg <command_arg>`

This option has been deprecated. Instead of using `--scm-command-arg arg1`, use `--scm-param annotate_arg=arg1`. Specifies additional arguments that are passed to the command that gathers the last modified dates. The arguments are placed after the command and before the target file. For usage information, see `cov-extract-scm` [↗](#).

**--scm-param**

Specify extra arguments to be passed to the SCM tool in a context-aware manner. For usage information of the `--scm` option, see `cov-extract-scm`.

**--scm-project-root <root\_path>**

Specifies a path that represents the root of the source control repository. This option is only used when specifying `accurev` as the value to `--scm`. When this used, all file paths that are used to gather information are interpreted as relative to this project-root path. For usage information, see `cov-extract-scm` [↗](#).

**--scm-tool <tool\_path>**

Specifies the path to an executable that interacts with the source control repository. If the executable name is given it is assumed that it can be found in the path environment variable. If not provided uses the default tool for the specified `--scm` system. For usage information, see `cov-extract-scm` [↗](#).

**--scm-tool-arg <tool\_arg>**

This option has been deprecated. Instead of using `--scm-tool-arg arg1`, use `--scm-param tool_arg=arg1`. Specifies additional arguments that are passed to the SCM tool specified in the `--scm-tool` option that gathers the last modified dates. The arguments are placed before the command and after the tool. This option can be specified multiple times. For usage information, see `cov-extract-scm` [↗](#).

## Shared options

**--debug, -g**

Turn on basic debugging output.

**--ident**

Displays the version of Coverity Analysis and build number.

**--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>**

Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects`(*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

**See Also**

`cov-extract-scm`

`cov-manage-emit`

---

## Name

`cov-install-updates` Manage and install updates.

## Synopsis

```
cov-install-updates <SUB-COMMAND> <COMMAND_OPTIONS>
```

## Description

The `cov-install-updates` command manages the installation of incremental, minor, or major Coverity Analysis updates. Use it with its sub-commands to query and list the available updates, install the updates in order, and if required, roll back an undesired update. The `cov-install-updates` command requires one of the five sub-commands listed below. See each sub-command section to see the options that apply to that sub-command.

### Note

To list major version updates (upgrades), specify the `--show=upgrades` option.

To install a major version update, you must use the `--end-version` sub-command.

## Sub-Commands

### check

Communicates with the Coverity Connect server and checks if there are any Coverity Analysis updates available newer than the installed Coverity Analysis version, as it appears in the `VERSION.xml` file. If there are updates available, it displays on the console the number of updates available to download and install. If there are updates available, the exit code is 0. If there are no updates available, the exit code is 1.

This sub-command requires connection options sufficient to access update information from a connected CIM server. For details, see the Connection Options section.

This sub-command also accepts the following options as described in the Sub-Command Options section.

```
--installer-dir
```

```
--installation-dir
```

### install

Determines the upgrade path, downloads the available Coverity Analysis update files, creates a backup of the current Coverity Analysis installation, lists the updates and any warnings, and then installs each Coverity Analysis update in order. Some updates can impact your normal workflow: therefore, they contain a warning message that prints to the console. If any warnings are present, the `install` sub-command will wait for confirmation before proceeding with the installation. If not confirmed, the installation will abort. See the `--continue` sub-command option for more information about installation confirmation.

The `install` command selects from available updates to create an update path from the currently-installed version to the selected end-version. See the `--end-version` option for more details.

This sub-command requires connection options sufficient to access update information from a connected CIM server. For details, see the Connection Options section.

This sub-command also accepts the following options as described in the Sub-Command Options section.

```
--continue
--end-version
--force
--installer-dir
--installation-dir
```

The install sub-command returns with an exit code of 0 if updates were successfully installed. The exit code is 1 if the command completed successfully but no updates were installed.

#### list

Displays a list of the available Coverity Analysis updates with a brief description for each update.

This sub-command requires connection options sufficient to access update information from a connected CIM server. For details, see the Connection Options section.

This sub-command also accepts the following options as described in the Sub-Command Options section.

```
--installer-dir
--installation-dir
--show
```

#### rollback

Rolls back the Coverity Analysis installation to the state it was in before you last ran the `cov-install-updates install` command. Since it is possible for the `install` sub-command to install several updates within the same session, the effect of the `rollback` sub-command is to roll back all the updates.

#### **Note**

Occasionally, an update package may contain a post-installation script. When this occurs, the installer will discontinue installing packages in the selected sequence and transfer control to the script. The script performs special actions and then normally continues the installation by re-invoking `cov-install-updates`. When this occurs, rolling back an installation can only return to the installation state that existed following execution of the script.

This sub-command accepts the following options as described in the Sub-Command Options section.

```
--force
```

`--installer-dir`

`--installation-dir`

`version`

Displays the version number for the installed Coverity Analysis, as it appears in the `VERSION.xml` file.

This sub-command accepts the following options as described in the Sub-Command Options section.

`--installation-dir`

## Sub-Command Options

`--continue <answer>`

By default, the install sub-command waits for confirmation if there are any warnings that impact your current installation. You can use the `--continue` option to automate this confirmation. You must use the `--continue` option with great care, otherwise you could inadvertently impact your installation and workflow.

For use in scripts, the `--continue=yes` option can be used to provide confirmation, allowing the installation to proceed.

`--end-version <version>`

Specifies the last version, in the update path, to install. If there are newer available updates they will not be installed. The `--end-version` value is a string that represents a specific release version, for example, 2018.03-1. (You cannot specify a version older than the currently installed version.)

If you do not specify a specific `--end-version`, then the default value is the newest update that has the same base version (major release version) as the current installation.

### Important

To install an upgrade (that is to upgrade to a newer *major* release) you must specify the `--end-version` for the newer release.

`--force`

When used with the install sub-command, the installer applies the updates only to those files that are in their original install state (were not altered). It skips updates for any altered files. When used with the `rollback` sub-command, `--force` is a required option, which means the roll back cannot be undone.

`--installation-dir <path>`

Provides a path to an alternate installation to be updated. If omitted, the installation containing this `cov-install-updates` command is used.

`--installer-dir <path>`

Chooses a directory where the update list and downloaded installers are stored temporarily. If omitted, a temporary directory is used. The temporary directory is removed after successful completion of the `cov-install-updates` command.

`--show upgrades | raw`

When `updates` is specified, the `list` subcommand will list installer packages in an update path that includes one or more upgrades (to a newer major release version). When `raw` is specified, all available update packages will be listed — not just those that lie along a valid update path. Both options can be specified using multiple `--show` options.

 **Note**

An upgrade (as opposed to an update) installs the next major release, which usually requires a re-emit and may include changes that cause churn.

## Connection Options

The `check`, `list` and `install` sub-commands accept the options in this list. These sub-commands must connect to a Coverity Connect instance. The options in this list provide the information needed to establish that connection.

Minimally, the `--host`, `--port` (or `-https-port`) and `--auth-key-file` (or `--user` and `--password`) options are required.

`--auth-key-file <filename>`

This option specifies the location of an authentication key file that was previously created. It is used to connect to the Coverity Connect server. Authentication keys can be registered with a Coverity Connect instance and used for authentication in place of the `--user` and `--password` options. For information about Working with authentication keys, see *Coverity Platform 2020.12 User and Administrator Guide* [↗](#)

`--authenticate-ssl`

This is equivalent to `--on-new-cert distrust`.

`--certs <filename>`

In addition to CA certificates obtained from other trust stores, use the CA certificates in the given `<filename>`. For information on the new SSL certificate management functionality, see *Coverity Platform 2020.12 User and Administrator Guide* [↗](#)

`--connect-timeout <n>`

Sets the timeout for establishing connections to `<n>` seconds. If a connection to Coverity Connect cannot be established within this time, the transaction is aborted. This timeout cannot be disabled. The default value is 60 seconds.

`--host <coverityconnect_host> --port <coverityconnect_port>`

The hostname and port of the Coverity Connect instance to download Coverity Analysis updates from. `--port` is an optional property. If `--port` is not specified on the command line, the default is 8080 without `--ssl` and 8443 with `--ssl`. For commands that accept Coverity Connect options, the `--host` option is required.

`--https-port <coverityconnect_port>`

Using `--https-port <coverityconnect_port>` is equivalent to specifying `--port <coverityconnect_port> --ssl`.

`--max-retries <n>`

Sets the number of times to retry failed or aborted requests with Coverity Connect to `<n>`. Note that this does not include the initial attempt, so a setting of 1 results in at most 2 request attempts. A setting of 0 means to never retry failed requests. The default value is 1.

`--on-new-cert <trust | distrust>`

Indicates with `--ssl` whether to trust (with trust-first-time) self-signed certificates, presented by the server, that the application has not seen before.

`--response-timeout <n>`

Sets the response timeout to `<n>` seconds. For every request for data sent to Coverity Connect, if a response is not received within this time, the request is aborted. A setting of 0 disables this timeout. The default value is 300 seconds.

`--sleep-before-retry <n>`

Sets the time to sleep before retrying a failed or aborted request with Coverity Connect to `<n>` seconds. A setting of 0 disables this sleep. The default value is 1 second.

`--ssl`

Enables SSL encryption for communication with Coverity Connect.

`--user <username>, --password <password>`

The username and password used to log into the Coverity Connect instance. These will be encrypted if `--ssl` is used. These options are required if the `--auth-key-file` option is not present.

## Shared Options

`--debug, -g`

Turn on basic debugging output.

`--ident`

Displays the version of Coverity Analysis and build number.

`--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>`

Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects`(*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## Examples

The following examples illustrate the use of the `cov-install-updates` command.

This command checks the number of updates available:

```
C:\Users\thildeb>cov-install-updates check --host=emmett --port=14111 --user=userName
--password=aPassword

5 updates are available.
```

This command checks the content of available updates:

```
C:\Users\thildeb>cov-install-updates check --host=emmett --port=14111 --user=userName
--password=aPassword

5 updates are available.

Updates available for product: Coverity Static Analysis

**> 2018.03-1
User documentation errata and updates.

**> 2018.03-2
Add a script to simplify model extraction from JavaScript frameworks.

**> 2018.03-3
Expanded Japanese documentation.

**> 2018.03-4
Improvement and expansion of selected security checkers.

**> 2018.03-5
Selected new QA checkers are now enabled for the Swift language.

```

This command displays the version number:

```
C:\Users\thildeb>cov-install-updates version
2018.03
```

This example shows the installation of updates up to the specified version number:

```
C:\Users\thildeb>cov-install-updates install --host=emmett --port=14111 --
user=userName --password=aPassword
--end-version=2018.03-1
[STATUS] Downloading updates list
[INFORMATION] 1 update to install.
[STATUS] Downloading cov-analysis-win64-2018.03-1-update.zip
[STATUS] Unpacking cov-analysis-win64-2018.03-1-update.zip
```

```
[INFORMATION] Selected installers:
----- 2018.03 ==> 2018.03-1
[STATUS] Validating installers
[STATUS] Listing backup files from cov-analysis-win64-2018.03-1-update.zip
[STATUS] Installing cov-analysis-win64-2018.03-1-update.zip
[STATUS] Verifying current installation.
[STATUS] Done.
```

This example illustrates the use of the `rollback` subcommand:

```
C:\Users\thildeb>cov-install-updates rollback --force
[STATUS] Rolling back from C:\Program Files\Coverity\Coverity Static Analysis
\.coverity\rollback
[STATUS] Done.
```

## See Also

[cov-commit-defects](#)

---

## Name

cov-link Create an intermediate directory with resolved duplicate function calls for C/C++.

## Synopsis

```
cov-link {--collect | <link_file>...} [--compile-arg <arg> | --compile-arg-regex <regex> | --no-compile-arg <arg> | --no-compile-arg-regex <regex> | --source-file-regex <source_file_regex>]... {--output-dir <output_dir> | --output-file <output_file>} --dir <intermediate_dir>} [OPTIONS]
```

## Description

Sometimes the same file is compiled several times with different command-line options. To avoid errors in function-call resolution (especially in C code, which does not have name mangling), you can use the `cov-link` command to create a new intermediate directory that contains a subset of the files that are in your original intermediate directory. This new intermediate directory can then be analyzed without function-call resolution issues.

The input consists of an intermediate directory (with an emit repository), plus a set of translation units. The translation units are either collected dynamically from the emit repository with the `--collect` option, or they are specified inside one or more of the link files that were previously created with `cov-link`.

Typically, `cov-link` is first called with the `--collect` option to produce a link file. You can look at this file for clues on which filters to apply to generate an intermediate directory with just the subset of files that you are interested in. Once you have an idea of which filters you need to apply, you can call `cov-link` again with your filters to produce a new intermediate directory. This new intermediate directory can then be analyzed.

To use the `cov-link` command:

1. Run `cov-link` with the `--collect` and `--output-file` options. This operation collects linkage information on all files compiled in an emit directory.
2. Create one or more additional link files by filtering information using either an argument or a portion of the pathname that was used during command-line compilation. Compiled files are identified based on:
  - A portion of the pathname to the file when it was compiled. Use the `--source-file-regex` option to specify a Perl regular expression to use when looking at the pathname.
  - The options given on the command line when it was compiled. Use the `--compile-arg`, `--compile-arg-regex`, `--no-compile-arg`, and `--no-compile-arg-regex` options to group by command-line options.
3. Use the link files created in the previous steps, and the emit repository in the original intermediate directory, to create a new intermediate directory with an emit repository with resolved function calls.
4. Use `cov-analyze` on the new intermediate directory.

For more information, including detailed examples, see the *Coverity Analysis 2020.12 User and Administrator Guide*.

## Options

### Input options:

You must specify one of the following options. You must not use both together.

`--collect, -co`

Collects linkage information from all of the entries in an emit repository.

`<link_file>`

Specifies which source files are linked together. You can specify multiple link files.

### Filter options:

These options are not required but can be specified multiple times.

`--compile-arg <arg>, -a <arg>`

Specify an argument that was given when compiling the files on the command line.

`--compile-arg-regex <regex>, -r <regex>`

Specify an argument that was given when compiling the specified files, as a Perl regular expression.

`--no-compile-arg <arg>, -na <arg>`

Specify an argument that was NOT given when compiling the files on the command line.

`--no-compile-arg-regex <regex>, -nr <regex>`

Specify an argument that does NOT match any argument given when compiling the files on the command line, as a Perl regular expression.

`--source-file-regex <source_file_regex>, -s <source_file_regex>`

Specify a portion of the source file pathname that was used during compilation, as a Perl regular expression. You can use a forward slash (/) as a directory separator in this string, for example `/proj1/` matches if `proj1` is a directory that is in the pathname. Note that on Windows, the matching is case-insensitive, and (/) is used as the directory separator (not \). You can specify this option more than once (as in `-s <source_file_regex> -s <source_file_regex>`). If there are several `-s` options, the source file's name only needs to match one of the specified expressions.

### Output options:

You must specify one of the following options. You must not use both together.

`--output-dir <output_dir>, -odir <output-dir>`

Specifies an intermediate directory for the `cov-link` command to create. If you use this option, you must also use the `--dir` option to this command.

Note that the `--dir` option to the `cov-analyze` command will use the specified `<output_dir>` as its value.

`--output-file <output-file>, -of <output-file>`

Specifies the pathname to the link file that is created. If you use `--collect`, any existing file with this name is replaced. If you specify `--source-file-regex`, any existing file with this name is appended to.

If you use this option, you must also use the `--dir` option to this command.

## Shared options

`--dir <intermediate_directory>`

Path name to an intermediate directory that is used to store the results of the build and analysis.

`--ident`

Displays the version of Coverity Analysis and build number.

`--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>`

Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## Examples

Create a link file based on an existing emit repository:

```
> cov-link --dir . --collect -of /usr/foo/all.link
```

Create a link file based source files with `apache` in the pathname:

```
> cov-link --dir . -s /apache/ -of /usr/foo/link_reports/apache.link \
all.link
```

Create a link file based on the source files with `apache_1.3.33` in the pathname, include only the files that were compiled with the `DEBUG` macro defined on the command line, and then create an intermediate directory with an emit repository:

```
> cov-link --dir . -a -DDEBUG -s /apache_1.3.33/ \
-of /usr/foo/link_reports/apache1333_DEBUG.link all.link
> cov-link --dir . --output-dir emit_apache1333_DEBUG \
/usr/foo/link_reports/apache1333_DEBUG.link
```

Create a new emit repository based on the source files with `apache_1.3.33` in the pathname, and include only the files that were compiled with the `DEBUG` macro defined on the command line (same as previous example, but without creating the intermediary link file):

```
> cov-link --dir . --collect -a -DDEBUG -s /apache_1.3.33/ \
 --output-dir emit_apache1333_DEBUG
```

## See Also

`cov-analyze`

---

## Name

`cov-make-library` Create a user model file from C/C++, Objective-C, Objective-C++, C#, Go, Java, Swift, and Visual Basic source code.

## Synopsis

```
cov-make-library [-of <modelfile>] [--security] [--checker-option <option-
setting>] [--classpath <directories_or_jar_files>] [-co <compiler>] [--
compiler-opt <compiler_option>] [--concurrency] [--enable CHECKER] [--disable
CHECKER] [--disable-default] [--disable-webapp-security] [--framework] [--
make-dc-config] [--quality] [--reference <referenced_assembly>] [--security]
[--security-file <license file>, -sf<license file>] [--sdk] [--enable-cgo-
for-go-models] [--webapp-security] [--target] <sources>
```

## Description

The `cov-make-library` command creates user model files from source files. User model files contain information that overrides what `cov-analyze` can derive for itself. See the *Coverity 2020.12 Checker Reference* for more information about how to model source files. There are also examples in the `<install_dir>/library` directory.

### Note

The files in the `<install_dir>/library` directory should not be provided as arguments to `cov-make-library`. You should instead create your own new files for models. Using the existing files creates duplicate, identical user models and Coverity default models.

The file is appended if it already exists, and created if it does not exist. The search order used to determine where to create the model file is the filename specified by `-of` or a default value of `<install_dir>/config/user_models.xmlldb`.

The `cov-make-library` command works by calling `cov-emit` to parse and emit the source files, followed by `cov-analyze`, and then `cov-collect-models` to collect the analyzed models.

### Default behavior

For C/C++, Objective-C, Objective-C++, Go, and Swift, the default behavior of this command is to generate models for checkers that are enabled by default.

For Java, C#, and Visual Basic, the default behavior of this command is to generate a model for use by all checkers, quality and security. Some of the command line options allow you to limit the generation of models to those used by groups of checkers.

Source files are compiled as C, C++, Objective-C, Objective-C++, C#, Go, Java, Swift, or Visual Basic, depending on their file extension:

- Compiled as C code: `.c` extension
- Compiled as Objective-C code: `.m` extension
- Compiled as C++ code: `.cc`, `.CC`, `.cp`, `.cpp`, `.cxx`, `.c++`, and *with the exception of Windows*, `.C` extensions

- Compiled as Objective-C++ code: `.mm` extension
- Compiled as C# code: `.cs` extension
- Compiled as Go code: `.go` extension
- Compiled as Java code: `.java` extension
- Compiled as Swift code: `.swift` extension
- Compiled as Visual Basic code: `.vb` extension

## Options

### C options and related standards

The following options indicate that `c` files may be compiled with the corresponding standard. The default is `--c11`.

- `--c11`
- `--c90`
- `--c99`

### C++ options and related standards

The following options indicate that `c++` files may be compiled with the corresponding standard. The default is `--c++17`.

- `--c++11`
- `--c++14`
- `--c++17`
- `--c++98`

`--checker-option <checker_name>:<option>[:<option_value>]`

Passes the specified checker option to `cov-analyze` when invoking `cov-analyze` after the library has been built.

Example:

```
UNINIT:enable_deep_read_models:true
```

Checker options and their default values are documented in the *Coverity 2020.12 Checker Reference* [🔗](#).

### ⚠ Caution

The `cov-analyze` command allows `-co` as a shorthand form of this option, but `cov-make-library` already uses `-co` to designate the compiler, so don't confuse these option names.

`--classpath <directories_or_jar_files>`

[Java option] Lists directories or Jar files, which must be separated by a semi-colon (;) on the Windows platform, and by a colon (:) on other platforms. Like `javac`, `cov-make-library` searches these entries for bytecode with the referenced classes when attempting to resolve names in source files. See also the `--classpath` option to `cov-emit-java`. For `cov-make-library`, it is possible to use stubs.

`--compiler <compiler>, -co <compiler>`

[C/C++ option] Specifies a previously configured compiler (configured with `cov-configure --compiler`) that is used to determine how to compile the files (using `cov-translate`). This is useful if you need to include standard headers. For example:

```
> cov-configure --compiler ABC
> cov-make-library --compiler ABC foo.c
```

`--compiler-opt <opt>`

[C/C++ option] Specify an option to the compiler specified by `--compiler` (or `cov-emit` if no compiler was specified). For instance, you can specify an include directory with `--compiler-opt -I --compiler-opt include_dir`

`--concurrency`

[C/C++ option] Use this option if you write a custom model using concurrency primitives.

`--disable <CHECKER>`

[C/C++ option. Deprecated for C# and Java.] Disables the creation of function models used by the specified checker.

Note that this option disables a checker only if it is enabled by default *and not enabled in any other way*, for example, through a group enablement option such as `--all`.

`--disable-default`

[C/C++, C#, Java, Visual Basic option] Disables the creation of function models for *all* checkers. Use options such as `--concurrency`, `--security`, `--quality`, and `--webapp-security` to choose a set of checkers for which to generate models.

When using this option, you must also use an enablement option, such as `--quality` or `--security`. It is an error to use `--disable-default` without such an option.

`--disable-webapp-security`

[Java, C#, Visual Basic option] Disables the creation of models for Web application security checkers (for example, XSS and SQLI). You typically use this option when you only want to generate models for Java quality checkers. For a complete list of Web application security checkers, see the "Coverity Checkers" table in the *Coverity 2020.12 Checker Reference*.

See also, `--webapp-security` and `--quality`.

`--enable <CHECKER>`

[C/C++ option. Deprecated for C# and Java.] Enables the creation of function models used by the specified checkers. If you want the specified checkers to check the source that uses your custom models, you must enable those checkers with this option.

**--enable-cgo-for-go-models**

[Go option] Enables building Go models for code that contain C dependencies. These models are disabled by default.

C code that is compiled as part of processing CGo dependencies will not be captured for analysis by `cov-make-library`.

For code bases that contain CGo dependencies (in other words, Go code that imports the pseudo-package "C"): Your environment must be configured to successfully compile such code using the native Go compiler before you execute `cov-make-library` on your modeling code. This is required because the `cov-emit-go` command, the Go compiler, and the CGo tool, must access additional tools to process such code (they execute a C compiler and generate bindings for the compiled C functions).

For more information about CGo support, see "Table 8.4. Supported compilers: Coverity Analysis for Go" in the *Coverity 2020.12 Installation and Deployment Guide*.

**--framework**

[Swift only] Use this option to specify the directory containing third-party frameworks that are used in the model.

**--java**

[Deprecated] Deprecated in version 7.0 because the command automatically determines the language based on the Java file extension.

**--make-dc-config**

[C/C++ only] Upgrades models for the deprecated `SECURE_CODING` checker to `DC.CUSTOM_CHECKER` checker configurations. Specifically, the option searches for `__coverity_secure_coding_function__` models in the source code and generates a JSON configuration file for a custom checker called `DC.CUSTOM_CHECKER`. The configuration file specifies function names and information found in the custom models.

Example:

```
> cov-make-library -of config.json --make-dc-config my_models.c
```

To use the resulting configuration file in the analysis, you simply pass it through the `--dc-config` option.

Example:

```
> cov-analyze --dir <intermediate_dir> --dc-config config.json -en
DC.CUSTOM_CHECKER
```

The `--make-dc-config` option is also available to `cov-collect-models`.

**--output-file <modelfile>, -of <modelfile>**

Specify the name of the output model file. The default file name is `user_models.xmlldb`, at `<install_dir_sa>/config/`.

**--quality**

[C/C++, C#, Java, Visual Basic option] Generates models for quality checkers, including concurrency checkers. Use this option with `--disable-default` to generate models for only the quality checkers. For a list of quality checkers, see the "Coverity Checkers" table in the *Coverity 2020.12 Checker Reference*.

See also, `--disable-default`, `--disable-webapp-security`, and `--webapp-security`.

**--reference <referenced\_assembly>**

[C#, Visual Basic option] Specify a referenced assembly.

**--sdk**

[Swift only] Use this option to specify the iOS SDK location. It is useful when any aspects of the model refer to iOS APIs located in the SDK.

**--security**

[C/C++ option] Use this option if you write a custom model using security-related checkers such as `TAINTED_DATA`, `TAINTED_STRING`, `STRING_SIZE`, and `STRING_NULL`.

**--security-file <license file>, -sf <license file>**

Path to a valid Coverity Analysis license file. If not specified, this path is given by the `security_file` tag in the Coverity configuration, or `.security` in the same directory as `cov-analyze`. A valid license file is required to run the analysis.

**--target**

[Swift only] Use this option to specify the iOS target.

**--webapp-security**

[Web application security option] Generates models for Web application security checkers (for example, XSS and SQLI). Use this option with `--disable-default` to generate models for only the Web application security checkers. For a complete list of Web application security checkers, see the "Coverity Checkers" table in the *Coverity 2020.12 Checker Reference*.

See also, `--disable-default`, `--disable-webapp-security`, and `--quality`.

## Shared options

**--config <coverity\_config.xml>, -c <coverity\_config.xml>**

Uses the specified configuration file instead of the default configuration file located at `<install_dir_sa>/config/coverity_config.xml`.

**--debug, -g**

Turn on basic debugging output.

**--ident**

Displays the version of Coverity Analysis and build number.

**--info**

Displays certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

--tmpdir <tmp>, -t <tmp>

Specifies the temporary directory to use. On UNIX, the default is \$TMPDIR, or /tmp if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>

Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects(*Coverity 2020.12 Command Reference*), cov-analyze, and cov-build.

## See Also

cov-analyze

cov-emit

cov-emit-go

cov-collect-models

---

## Name

cov-manage-emit Manage an intermediate directory.

## Synopsis

```
cov-manage-emit <GENERAL OPTIONS> <COMMANDS> <COMMAND OPTIONS>
```

GENERAL OPTIONS:

```
[--cpp | --cs | --java]
```

```
{--dir <intermediate_directory>|--idir-library
<intermediate_directory_library>}
```

```
[--tu <tu_ids> | --tu-pattern <pattern>]
```

```
[--tus-per-psf <value>]
```

COMMANDS and COMMAND OPTIONS:

```
[add-other-hosts | check-integrity | delete-source | list-builds | repair |
reset-host-name]
```

```
[add <int_dir> | list | list-json | preprocess | link-file <out_file>]
```

```
[decompile-binary-tus-from-dir <decompile_options>]
```

```
extract-files --output-dir <dir> [--strip-path <path>]... {--regex <regex> |
<filename>...}
```

```
[{recompile | retranslate | retranslate-or-emit} <recompile_options>]
```

```
[find [OPTIONS]]
```

```
[add-coverage [OPTIONS] | compute-coverability { --verbose } | delete-
coverage | delete-test-coverage [OPTIONS] | [list-coverage-known | list-
coverage-unknown] --output <out_file> {--filename-regex <regex>} {--count} |
remove-coverability {--verbose} | list-tests {--suite-name <name>} {--count}]
```

```
[list-compiled-classes]
```

```
[add-to-library --dir <intermediate_directory>]
```

```
[remove-from-library --dir <directory_name>]
```

```
[export-json-build [OPTIONS]]
```

```
[import-json-build [OPTIONS]]
```

```
[list-json-schema-versions]
```

```
[Shared options]
```

## Description

The `cov-manage-emit` command is used to query and manipulate an emit repository. Each intermediate directory contains a single emit repository that contains data for languages emitted via `cov-build`, `cov-emit`, and other similar commands.

The `cov-manage-emit` command requires the `--dir` option, plus at least one sub-command. the `cov-manage-emit` command line typically follows this pattern:

```
cov-manage-emit <general_options> <sub-command> <sub-command_options>
```

The sub-commands can be used for various operations, including:

- Repairing database integrity (`repair`).
- Recompiling (`recompile`).
- Decompiling (`decompile`).
- Copying information from one intermediate directory into another (`add`).
- Aggregating the results of a distributed build into a single intermediate directory (`add-other-hosts`).
- Listing source files (`print-source-files`).
- Listing AST definitions (`find --print-definitions`).
- Adding test coverage data for Test Advisor (`add-coverage`).
- Inputing and outputting SCM data (`add-scm-annotations`, `dump-scm-annotations`).
- Starting, stopping, and querying the emit server.
- Export C/C++ coverage data suitable for use with Test Advisor QA Edition.

The `cov-manage-emit` options are grouped by basic, options that cannot be filtered by translation units, options that require translation unit filtering, options for listing emit database information, and options for recompiling.

The `cov-manage-emit` command returns the following success or failure values:

- 0 – Success
- 2 – Error
- 4+ – Internal error, contact [software-integrity-support@synopsys.com](mailto:software-integrity-support@synopsys.com)

## Options

### Basic options

Either the `--dir` option or `--idir-library` option is required. If you use a sub-command that uses translation units, you can filter this information with the `--tu` or `--tu-pattern` option, or both.

`--dir <intermediate_directory>`

Specifies an existing intermediate directory that was created with the `cov-build` command. While certain other sub-commands (for example, `add`) allow you to specify intermediate directories, the one specified with `--dir` is the directory modified by `cov-manage-emit`.

`--idir-library <intermediate_directory_library>`

Specifies the location of an intermediate directory library, which may contain multiple intermediate directories created with the `cov-build` command. Certain sub-commands, such as `start-server`, will optionally accept an intermediate directory library instead of a single intermediate directory.

`--cpp`

Filters by C/C++ translation units on which this command operates or reports. The command will fail with an informative error message if none of the translation units in the emit match any of the specified language options.

`--case-normalized-filename`

By default, `cov-manage-emit` displays case-preserved file names in the output. Specifying this option allows `cov-manage-emit` to display normalized file names (that is, names that are entirely lower case).

For example (assuming you are on Windows and have a file in your emit named `MyFile.c`):

```
cov-manage-emit.exe --dir intdir --case-normalized-filename list
```

The output will include the following:

```
c:/cygwin/space/int_dir/myfile.c
```

 **Note**

In previous releases, case-preserved file names were always printed for Java and C# (regardless of `--case-preserved-filename`). As of the 7.5.0 release, Java and C# file names will be case-preserved or case-normalized according to specification of this option (`--case-normalized-filename`), like C/C++ file names. As a result, it is impossible to get the old output of `cov-manage-emit` (which would case-normalize C/C++ but case-preserve Java/C#) in a multi-language scenario.

`--case-preserved-filename`

Allows `cov-manage-emit` to display case-preserved file names. This option is enabled by default, so you do not need to specify it with `cov-manage-emit list`. To switch to normalized file names, use `--case-normalized-filename`.

`--cs`

Filters by C# translation units on which this command operates or reports. The command will fail with an informative error message if none of the translation units in the `emit` subdirectory match any of the specified language options.

`--java`

Filters by Java translation units on which this command operates or reports. The command will fail with an informative error message if none of the translation units in the emit match any of the specified language options.

**--preprocess-native**

Invokes the native compiler to generate preprocessed output. The emit database is not modified by this operation. The preprocessed output file is stored in the `c/output/preprocessed` subdirectory of the intermediate directory. This option works only for C/C++ source code.

**--tmpdir <tmp> or -t <tmp>**

Specifies the temporary directory to use.

- On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist.
- On Windows, the default is to use the temporary directory specified by the operating system.

**--tu <translation\_unit\_id(s)>, -tu <translation\_unit\_id(s)>**

Identifies a set of translation units (TUs), named by their numeric ID attribute(s). A translation unit approximately maps to the output from a single run of a compiler. This option requires a comma-separated list of `id(s)`, and `--tu` can be specified multiple times. The union of all these identifier sets is the set of TUs to operate on subsequently, for operations that work on TUs. It is an error if any of the specified IDs do not correspond to any existing translation unit. To get the IDs for translation units, use the `list` sub-command.

You can use the `--tu` and `--tu-pattern` options together.

**--tu-pattern <translation\_unit\_pattern>, -tp <translation\_unit\_pattern>**

Identifies a set of translation units specified with a translation unit pattern. The `--tu-pattern` option can be specified multiple times. Matching TU sets are unioned together across all patterns.

Both `--tu` and `--tu-pattern` can be specified on a single command line. The final set of TUs operated upon includes a given TU if it matches any specified translation unit pattern or its ID is listed explicitly as an argument to `--tu`.

It is an error if at least one `--tu-pattern` argument is specified but no translation unit matches any of the specified patterns.

You can get useful information on TUs with the `list` sub-command.

For more information, see [Translation unit pattern matching](#).

**--tu-sort <sort\_spec>**

Specifies the sort order for TU output. The `<sort_spec>` accepts the values listed below. To sort on more than one attribute, you can use a non-empty, comma-separated list of values. Additionally, to specify ascending or descending sort order for any attribute, you can add `:a` or `:d` (respectively) directly after the attribute name. All attributes are ordered in ascending order by default.

The available sort attributes are:

- `emittime`: the time spent emitting TUs.

**--tus-per-psf <value>**

Indicates how the set of primary source files affects the set of selected TUs. The possible values are as follows:

- `all`: Select all TUs, possibly as specified by other TU filters. This is the default.
- `latest`: Select only the latest TU with a given primary source file according to the time of compilation. If there are multiple TUs with the same primary source file within a single build, a deterministic TU is chosen within that build, regardless of time of compilation, which allows determinism in the case of parallel builds. This corresponds to the default set of TUs that `cov-analyze` analyzes. That is, `cov-analyze` with `--one-tu-per-psf` corresponds to `--tus-per-psf=latest` without any other filtering options (see the `--one-tu-per-psf` option to `cov-analyze`).

With at least one `-tu` option and without a search pattern, the option has no effect. In this case, the system includes only the TUs specified with `-tu`.

- `non-latest`: Select any but the `latest` TU with a given primary source file. This is applied after search pattern filtering. The result is undefined if `-tu` is also used. For instance, to keep only one TU per primary source file, run the following command:

```
cov-manage-emit --tus-per-psf=non-latest delete
```

Examples:

To list all the TUs that `cov-analyze` will operate on:

```
> cov-manage-emit --dir <intermediate_directory> \
 --tus-per-psf=latest list
```

To delete TUs and leave only the ones that `cov-analyze` would operate on:

```
> cov-manage-emit --dir <intermediate_directory> \
 --tus-per-psf=non-latest delete
```

## Non-filtered sub-commands

These sub-commands cannot be filtered with translation unit options.

`add-coverage <command_options>`

Adds the coverage data contained in the specified file to the intermediate directory. Valid command line options:

- `--batch <filename>`

Parses the specified file as batch commands to run. If this command option appears on the command line, all other `add-coverage` command options on the command line are ignored.

### **add-coverage batch file format:**

To efficiently add data from many separate coverage files, `add-coverage` supports batch files. Each line in a batch file specifies a single batch command to be executed by `add-coverage`, and batch commands are executed in the order of appearance in the batch file. The grammar for batch commands is:

```
BatchCommand ::= 'run' | BatchOption
BatchOption ::= <option_name> ':' <length> ':' <value> <option_name>
```

A `BatchOption` batch command sets the specified option to the specified value for all subsequent run batch commands. This value overrides any previous setting of the specified option. Valid option names are any command line option available to the `add-coverage` command except for the `--batch` command line option. If the command line option accepts a value, the option is followed by a colon, then the string length of the value, then another colon, then the value itself. If the command line option does not accept a value, only the option name is specified in the corresponding `BatchOption`. When used in a `BatchOption` batch command, the leading `--` is omitted from the option name.

The run batch command causes a single coverage file to be read and its data added to the intermediate directory. The name of the coverage file to read from and other relevant settings are set by the `BatchOption` batch commands prior to the run batch command.

The following is an example batch file:

```
suitename:8:FooSuite
testname:7:FooTest
teststart:19:2012-03-19 07:12:11
verbose
gcda:9:test.gcda
run
gcda:10:test2.gcda
run
```

The above example batch file is equivalent to the following commands:

```
> cov-manage-emit --dir apache_2111 \
add-coverage --suiteName FooSuite --testName FooTest \
--teststart "2012-03-26" --verbose --gcda test.gcda

> cov-manage-emit --dir apache_2111 \
add-coverage --suiteName FooSuite --testName FooTest \
--teststart "2014-03-26" --verbose --gcda test2.gcda
```

#### Note

Note that any test-specific options in the batch file will be replaced with their default values if they precede `testname` in the batch file. To avoid this, make sure the `testsource`, `teststart`, `teststatus`, and `testduration` options are specified *after* the `testname` option.

- `--bb <filename>`

Reads the specified file as a file containing coverage data in `gcov bb` format. If this option is specified but the `--bbg` or `--da` option is not, the missing options will be inferred based on the filename specified in this option.

- `--bbg <filename>`

Reads the specified file as a file containing coverage data in `gcov bbg` format. If this option is specified but the `--bb` or `--da` option is not, the missing options will be inferred based on the filename specified in this option.

- `--bullseye-csv`

Specify the Bullseye CSV file from which to add coverage. This file can be generated from a Bullseye `.cov` file using the Bullseye `covbr` tool. This can be generated with the command:

```
covbr --no-banner --quiet --csv --file <file.cov> --output <output.csv>
```

- `--bullseye-verbose`

Enables verbose diagnostic messages in the Bullseye CSV parser.

- `--compilation-directory <dirname>`

Uses the specified directory as the compilation directory. This is used to determine absolute paths of files referenced in the coverage data.

- `--coverage-file=<filename>`

Where `<filename>` is the name of data captured using Function Coverage Instrumentation. See the Test Advisor 2020.12 User and Administrator Guide [🔗](#) for more information.

- The `--coverage-file` option cannot be used in combination with the `--batch` option.
- The `--testname`, `--suite`, `--testsource`, `--testsource-encoding`, `--teststatus`, `--teststart`, and `--testduration` options are ignored when used with the `--coverage-file` option.
- `--coverage-selection <coverage-selection>`

Describes what coverage is to be selected when merging coverage. For example:

```
--coverage-selection "latest from all"
```

see *Test Advisor 2020.12 User and Administrator Guide* [🔗](#).

- `--da <filename>`

Reads the specified file as a file containing coverage data in `gcov bbg` format. If this option is specified but the `--bb` or `--bbg` option is not, the missing options will be inferred based on the filename specified in this option.

- `--dry-run-list-tests`

Indicates that execution will not actually merge coverage, but instead lists the tests which would be included in the target intermediate directory. This can be used to validate that a given coverage-selection option performs as intended.

- `--from-dir <source-idir>`

This option indicates that `add-coverage` should use the specified intermediate directory as the source of coverage data to be merged. It is an error to use this option with `--from-idir-library`.

- `--from-idir-library <mdir>`

This option indicates that `add-coverage` should use the intermediate directory `library` in `<mdir>` as the source of coverage data to be merged. It is an error to use this option with `--from-dir`.

- `--gcov <filename>`

Reads the specified file as a file containing coverage data in `gcov` text format.

- `--gcov-version <version>`

Parses `gcno/gcda` files that are derived from the specified version of `gcc`. This can be used to override the version declared in the file header in case this is incorrect. Valid values for `<version>` are of the form `x.y`, where `x` is the `gcc` major version number and `y` is the `gcc` minor version number. The patch level is ignored. For example, `gcc-4.6.3` would be specified with a version of `4.6`.

- `--gcov-verbose`

Enables verbose diagnostic messages in the `gcov` binary parser.

- `--gcno <filename>`

Reads the specified file as a file containing coverage data in `gcov gcno` format. If this option is specified but the `--gcda` option is not, that option will be inferred based on the filename specified in this option.

- `--gcda <filename>`

Reads the specified file as a file containing coverage data in `gcov gcda` format. If this option is specified but the `--gcno` option is not, that option will be inferred based on the filename specified in this option.

- `--purecov-text <filename>`

Reads the specified file as a file containing coverage data in PureCoverage text format.

By default, PureCoverage produces binary coverage files, however the tool has options to create text files instead, for example:

For Linux:

```
purecov -export=results.txt foo.pcv
```

This command occurs after the instrumented binary execution as a post processing step.

For Windows:

```
purecov /SaveTextData=results.txt <executable>
```

This occurs during the execution of the instrumented binary.

For official instructions about how to produce a text file, see the PureCoverage documentation [↗](#).

The `compute-coverability` command line option must be run after the build and before the analysis. This can be placed on the command line before or after the `add-coverage` subcommand.

- `--purecov-verbose`

Enables verbose diagnostic messages in the Purecov parser.

- `--strip-path <strip-path>`

Specifies an additional strip-path to use when merging coverage. This option can be specified multiple times.

see *Test Advisor 2020.12 User and Administrator Guide* [↗](#).

- `--suiteName <suiteName>`

Identifies the coverage data as belonging to the named suite.

- `--testDuration`

The duration (in ms) of the test.

- `--testName <testName>`

Identifies the coverage data as belonging to the named test.

- `--testSource`

The path to the 'sourcefile' for the test. This could be a test script, a Makefile, or a source file of a unit test.

If the argument ends with a colon (:) followed by one or more digits, then the digits shall be taken as the test source line number for the test, and only the portion of the argument before the colon shall be used for the test source filename. A default line number of 1 is used if no line number is provided.

- `--testSourceEncoding`

The encoding of the file provided to `--testSource`.

- `--testStatus [pass | fail | unknown]`

Identifies the status of the test identified by `suitename` and `testname`. Must be one of `pass`, `fail` or `unknown`.

- `--teststart <teststart>`

Identifies start time of the test identified by `suitename` and `testname`.

See Appendix A, *Accepted date/time formats* for proper formatting of the `<teststart>` argument.

- `--windriver-run`

Reads the specified file as a file containing coverage data in Wind River Coverage Run format. For implementation details, see [Wind River VxWorks with Bullseye](#).

- `--verbose`

Enables verbose diagnostic messages.

#### add-other-hosts

Adds all translation units from emit repositories in the current intermediate directory but associated with host names other than the current one. In general, an intermediate directory can contain several emits, each associated with a specific host name. This option copies all of the TUs from emits associated with other hosts into the emit associated with the current host. This sub-command can be used to aggregate the results of a distributed build into a single intermediate directory.

#### add-scm-annotations --input <input\_file>

Adds the SCM annotations for the source files in the specified input file to the source files in the intermediate directory. The input file can be the output file of the `cov-manage-emit dump-scm-annotations` option or the `cov-extract-scm` [option](#).

This option reads input from standard input when `<input_file>` is `"-"`. Otherwise, it reads from the specified file.

#### **Note**

If you pipe the output of `cov-extract-scm` directly to `cov-manage-emit`, for example:

```
cov-extract-scm --input input.txt --output - |
cov-manage-emit --dir idir add-scm-annotations --input -
```

This will always generate at least one error and the first line of output will read `Extracting SCM data for ### files`.

#### add-test-capture-run [OPTIONS]

Adds a new test capture run to your intermediate directory. Note that in most use cases this is not required. The valid command line options are as follows:

- `--build-id <build ID>`

Specifies the build ID of the test capture run.

For more information about `--build-id` usage, see the description for the `cov-build --build-id` option.

- `--set-as-current`

Sets the test capture run as the "current" run in the intermediate directory. Any addition of coverage that does not specify a test capture run will use this test capture run.

- `--success <value>`

Specify if this test capture run was successful. Valid values are:

- `true`
- `false`
- `unknown`
- `unset`

- `--test-capture-run-tag <tag>`

Specifies a custom tag to allow for each selection of this test capture run. For example:

```
"linux-build"
```

- `--test-capture-run-timestamp <timestamp>`

Specifies the timestamp to use for the test capture run for this invocation. If it is not specified, the current time will be used, which is the typical use case. This option is only provided as a way for multiple test runs to use the same test capture run.

See Appendix A, *Accepted date/time formats* for proper formatting of the `<timestamp>` argument.

`add-to-library --dir <intermediate_directory>`

Adds an existing intermediate directory created by a `cov-build` command to the intermediate directory library specified with the `--idir-library` option. The intermediate directory library will be created if it does not exist.

The added intermediate directory is uniquely identified within the library by directory name, which is the last non-empty component of the path to the intermediate directory.

For example:

```
> cov-manage-emit --idir-library libdir add-to-library --dir /home/user/my-idir
```

The added intermediate directory is uniquely identified by "my-idir". This directory can then be referenced within the library using "my-idir".

`check-compatible`

Checks if an emit version is compatible with the current Coverity Analysis tools. For example:

```
cov-manage-emit --dir idir check-compatible
```

This option returns 0 if it is compatible, 1 if it is not compatible, and 2+ if there is an error.

#### check-integrity

Checks database integrity. If this check fails, print errors to stdout and exits with a non-zero code. Otherwise, exits with 0.

#### compute-coverability <command\_options>

Computes coverability of lines in files that are missing coverage data in the intermediate directory. The computed coverability is stored in the intermediate directory. Valid command line options:

- `--trace-log <filename>`

Writes trace log messages to the specified file. Given a filename of - (a hyphen), the messages will be written to `stderr`. Any other value is interpreted as a file to open and write the log messages to.

- `--verbose`

Enables verbose diagnostic messages.

#### delete-bytecode

Removes all file contents from the database. This is useful if you need to provide an intermediate directory to Coverity technical support and want to make sure that source code is excluded.

#### delete-coverage

Removes all coverage data from the database. This is useful if you need to provide an intermediate directory to Coverity technical support and want to make sure that coverage data is excluded.

To delete an individual test, see `delete-test-coverage`.

#### delete-scm-annotations

Removes all SCM annotations from the database. This is useful if you need to provide an intermediate directory to Coverity technical support and want to make sure that SCM data is excluded.

#### delete-source

Removes only source contents, including all webapp archive files (JSP, XML, and so forth). This option does not remove Java class, JAR files, or .NET bytecode. This is useful if you need to provide an intermediate directory to Coverity technical support and want to make sure that the source content is excluded.

#### delete-test-coverage [OPTIONS]

Deletes coverage for a specific test across all TUs affected by the test. The test to be deleted is uniquely identified by the following command options:

- `--suite <suite name>`

Specifies the suite name belonging to the test. Suite names are available through the `list-tests` sub-command.

- `--testname <testname>`

Specifies the test name belonging to the test. Test names are available through the `list-tests` sub-command.

- The test capture run, which you can specify with the `-test-capture-run-id` command line option or by the combination of the following command line options:

- `--build-id <build ID>`

Specifies the build ID of the test capture run.

For more information about `--build-id` usage, see the description for the `cov-build --build-id` option.

- `--test-capture-run-tag <tag>`

Specifies the tag used to identify the test capture run.

- `test-capture-run-timestamp <timestamp>`

Specifies the time when the test capture run occurred.

Information about test capture runs is available through the `list-test-capture-runs` sub-command. For more details on these options, including valid `<timestamp>` formats, see the `update-test-capture-run` sub-command.

#### Examples:

```
> cov-manage-emit --dir idir delete-test-coverage --suiteName SuiteAlpha \
--testname TestAlpha --test-capture-run-id 5
```

```
> cov-manage-emit --dir idir delete-test-coverage --suiteName SuiteBeta \
--testname TestBeta --test-capture-run-timestamp "2010-01-04T13:53:08" \
--build-id 100 --test-capture-run-tag "CaptureTag"
```

To delete coverage for all tests, `delete-coverage`.

#### `dump-scm-annotations --output <output_file>`

Output all SCM annotations stored in the intermediate directory. This command creates a cache of the SCM annotations that is intended to be reapplied in the future using `add-scm-annotations`.

This option places output on a standard output when `<output_file>` is `"-"`. Otherwise, it outputs to the specified file.

#### `export-json-build <options>`

Exports a captured build to a JSON file for later use with `import-json-build`.

Required options for `export-json-build`:

- `--output-file <filename>`, `-of <filename>`

Specify the path to and file name for the JSON file to be exported.

Optional options for `export-json-build`:

- `--schema-version <value>`

Specifies the schema version to use for the JSON file. See the output of `cov-manage-emit list-json-schema-versions` for a listing of valid values, as well as details about version differences. If not specified, the latest schema version will be used.

- `--strip-path <path>`

Strips the prefix for the exported file names and paths in the exported JSON file. This may be specified multiple times, but only the first matching strip path for any given path will be stripped. Accordingly, specify multiple strip paths from most specific to least for best results.

`import-json-build <options>`

Imports a JSON build file, which can be generated via the `export-json-build` sub-command, to the intermediate directory specified to `cov-manage-emit`. Note that the imported directory will not be useful for analysis results until at least a partial capture has been performed *after* the import. This option is primarily used in combination with `cov-run-desktop` to enable `cov-run-desktop` to work without requiring a full native build capture.

Required options for `import-json-build`:

- `--input-file <filename>, -if <filename>`

Specify the path to and file name for the exported JSON file to import.

Optional options for `import-json-build`:

- `--compilation-log <log file>`

Specify a log file to dump diagnostic output from this command to. If this is not specified, then the output from the `import-json-build` command goes to stdout.

- `--parallel <number of processes>, -j <number of processes>`

Specify the number of `cov-translate` processes to use simultaneously for the import. Note that 'auto' can be specified to allow `cov-manage-emit` to automatically determine the number of processes to use based on the detected hardware.

`list-builds`

Reports total number of successful and failed builds.

`list-compiled-classes`

Lists the classes contained in the emit that have been compiled. Use with `--java` or `--cs` to limit the results to one of the languages. The output is CSV-formatted and written to standard output and is designed to be specified into the `cov-build --java-instrument-classes <filename>` command.

The CSV file format has two columns:

- Column 1 - The name of the class.
- Column 2 - The full path to the source file for the class.

This sub-command applies only to Java and .NET, not to C/C++.

`list-coverage-known <command_options>`

Lists the files contained in the emit which have corresponding coverage data included in the emit. The valid command line options are:

- `--output <output_file>`

The list is written to standard output when `<output_file>` is "-". Otherwise, the list is written to the specified file.

- `--filename-regex <regex>`

Includes a file for consideration if the regular expression (`regex`) matches the name of the file; this is not case-sensitive. For the purpose of turning a file name into a string that can then be matched against a `regex`, the following normalizations are applied:

- The name is made absolute, including the drive letter on Windows systems.
- The forward-slash character ("/") separates name components.
- When no drive letter is present, the name begins with a forward-slash character ("/"); otherwise, a forward-slash character ("/") follows the drive letter.
- `--count`

Reports the number of files that would have been reported. If used with `--filename-regex`, it reports the number of matching files only.

`list-coverage-unknown <command_options>`

Lists the source files contained in the intermediate directory which do not have corresponding coverage data included in the intermediate directory. The valid command line options are:

- `--output <output_file>`

The list is written to standard output when `<output_file>` is the dash character ("-"). Otherwise, the list is written to the specified file.

- `--filename-regex <regex>`

Includes a file for consideration if the regular expression (`regex`) matches the name of the file; this is not case-sensitive.

For the purpose of turning a file name into a string that can then be matched against a `regex`, the following normalizations are applied:

- The name is made absolute, including the drive letter on Windows systems.

- The forward-slash character ("/") separates name components.
- When no drive letter is present, the name begins with a forward-slash character ("/"); otherwise, a forward-slash character ("/") follows the drive letter.
- `--count`  
Reports the number of files that would have been reported. If used with `--filename-regex`, it reports the number of matching files only.

#### list-functions-v1

Lists the functions in the intermediate directory. The valid command line options are:

- `--function-pattern <pattern>`

Restrict output to only those functions which match `<pattern>`. `<pattern>` follows the syntax described in the Translation unit pattern matching section, however the following predicates are used instead of the predicates described in that section:

- `mangled_name(<regex>)`: The function is included if its mangled name matches the given regex.
- `unmangled_name(<regex>)`: The function is included if its unmangled name matches the given regex.
- `filename(<regex>)`: The function is included if it is in a file whose stripped name matches the given regex.
- `impacted_since(<date>)`: The function is included if it was impacted on or after the given date. [FM]
- `impacted()`: Like `impacted_since(<date>)`, but uses the `code-version-date` in the emit as the given date. [FM]
- `directly_impacted_since(<date>)`: The function is included if it was directly impacted on or after the given date. [FM]
- `directly_impacted()`: Like `directly_impacted_since(<date>)`, but uses the `code-version-date` in the emit as the given date. [FM]
- `indirectly_impacted_since(<date>)`: The function is included if it was indirectly impacted on or after the given date. Functions without indirect impact dates are not included. [FM]
- `indirectly_impacted()`: Like `indirectly_impacted_since(<date>)`, but uses the `code-version-date` in the emit as the given date. [FM]
- `scm_modified_since(<date>)`: The function is included if it was modified on or after the given date. SCM data is used to determine modification. Functions without SCM data are included. [FM]

- `has_scm_data()`: The function is included if it has SCM data. [FM]
- `covered_by_suitename(<regex>)`: The function is included if it was executed by a test whose `suitename` matches the given regex. [TM]
- `covered_by_testname(<regex>)`: The function is included if it was executed by a test whose `testname` matches the given regex. [TM]
- `covered_by_test(<suite_regex>, <test_regex>)`: The function is included if it was executed by a test whose `suitename` matches `<suite_regex>` and whose `testname` matches `<test_regex>`. [TM]

Predicates marked with [FM] use function metrics generated by Test Advisor analysis, and require that such analysis be run prior to querying. Typically this is done by running `cov-analyze` with either the `--test-advisor` or `--enable-test-metrics` option.

Predicates marked with [TM] use test metrics information, and require that such analysis be run prior to querying. Typically this is done by running `cov-analyze` with the `--enable-test-metrics` option.

Function metrics and test metrics which have been generated in an intermediate directory using `cov-analyze` can be re-used within that directory for the purpose of this query. The `import` command of `cov-manage-history` should be run on the intermediate directory to allow this reuse.

- `--output-fields <fields>`

Specifies the fields for each function to include in the output. `<fields>` is a comma-separated list of keywords from among the following:

- `mangled_name`: The mangled name of the function.
- `unmangled_name`: The unmangled name of the function.
- `filename`: The name of the file containing the function.
- `impact_date`: The impact date of the function. [FM]
- `direct_impact_date`: The direct impact date of the function. [FM]
- `indirect_impact_date`: The indirect impact date of the function. [FM]
- `scm_modified_date`: The SCM modified date of the function. [FM]
- `covering_tests`: The tests that cover the function. Requires the `--json` option be specified, see below. [TM]
- `default`: The default output fields. This is equivalent to `"mangled_name,unmangled_name,filename"`, and is the default if the `--output-fields` option is not specified.

[FM] and [TM] indicate output fields which use function metrics and test metrics, respectively, from the results of Test Advisor analysis. See `--function-pattern` above for details.

Output is in CSV format by default, unless the `--json` option is given. Each line except the first corresponds to a function, and contains the output fields in the order specified. The first line is a header indicating the field names.

- `--json`

Output in JSON format. The output is a JSON array, where each element corresponds to a function. Each element of this array is an object whose name/value pairs correspond to the specified `--output-fields`.

This option is required when `covering_tests` is included in the `--output-fields` option. Specifying the `covering_tests` output field will add an element named `covering_tests` to each function object, whose value is an array representing the tests that cover the function. Each element of this array is an object with the following name/value pairs:

- `suitename`: The suite name of the test.
- `testname`: The test name of the test.

- `--strip-path <path>`

The strip-path to use when evaluating filenames. Normally this is not required, but can be used to override the default.

All dates must match a format described in Appendix A, *Accepted date/time formats*.

#### list-json-schema-versions

List valid `schema-version` values for use with `export-json-build`. A brief description of version differences will accompany each value.

#### list-scm-known <command\_options>

Lists the files contained in the emit that have corresponding SCM data included in the emit. The valid command line options are:

- `--output <output_file>`

The list is written to standard output when `<output_file>` is the dash character ("-"). Otherwise, the list is written to the specified file.

- `--filename-regex <regex>`

Includes a file for consideration if the regular expression (`regex`) matches the name of the file; this is not case-sensitive.

For the purpose of turning a file name into a string that can then be matched against a `regex`, the following normalizations are applied:

- The name is made absolute, including the drive letter on Windows systems.

- The forward-slash character ("/") separates name components.
- When no drive letter is present, the name begins with a forward-slash character ("/"); otherwise, a forward-slash character ("/") follows the drive letter.
- `--count`  
Reports the number of files that would have been reported. If used with `--filename-regex`, it reports the number of matching files only.

`list-scm-unknown <command_options>`

Lists the source files contained in the intermediate directory that do not have corresponding SCM annotations included in the intermediate directory. The valid command line options are:

- `--output <output_file>`  
The list is written to standard output when `<output_file>` is the dash character ("-"). Otherwise, the list is written to the specified file.
- `--filename-regex <regex>`  
Includes a file for consideration if the regular expression (`regex`) matches the name of the file; this is not case-sensitive.  
  
For the purpose of turning a file name into a string that can then be matched against a `regex`, the following normalizations are applied:
  - The name is made absolute, including the drive letter on Windows systems.
  - The forward-slash character ("/") separates name components.
  - When no drive letter is present, the name begins with a forward-slash character ("/"); otherwise, a forward-slash character ("/") follows the drive letter.
- `--count`  
Reports the number of files that would have been reported. If used with `--filename-regex`, it reports the number of matching files only.

`list-test-capture-runs`

Lists the test capture runs currently stored in the intermediate directory or intermediate directory library.

`list-tests <command_options>`

Lists all tests which have been stored in the intermediate directory. The valid command line options are:

- `--count`  
Reports the number of tests that would have been reported. If used with `--suite-name`, it reports the number of tests for suite only.

- `--suite` <name>

Restricts the tests that are listed or counted to those that belong to the given named suite.

#### list-tests-v2

Lists the tests in the intermediate directory. The valid command line options are:

- `--test-pattern` <pattern>

Restrict output to only those tests which match <pattern>. <pattern> follows the syntax described in the Translation unit pattern matching section, however the following predicates are used instead of the predicates described in that section:

- `suite`( <regex> ): The test is included if its `suite` matches the given regex.
- `test`( <regex> ): The test is included if its `test` matches the given regex.
- `covers_function_mangled_name`( <regex> ): The test is included if it covers at least one line of a function whose mangled name matches the given regex. [TM]
- `covers_function_unmangled_name`( <regex> ): The test is included if it covers at least one line of a function whose unmangled name matches the given regex. [TM]
- `covers_filename`( <regex> ): The test is included if it covers at least one line of a file whose stripped filename matches the given regex. [TM]

Predicates marked with [TM] use test metrics information, and require that such analysis be run prior to querying. Typically this is done by running `cov-analyze` with the `--enable-test-metrics` option.

Test metrics which have been generated in an intermediate directory using `cov-analyze` can be re-used within that directory for the purpose of this query. The `import` command of `cov-manage-history` should be run on the intermediate directory to allow this reuse.

- `--output-fields` <fields>

Specifies the fields for each test to include in the output. <fields> is a comma-separated list of keywords from among the following:

- `suite`: The suite name of the test.
- `test`: The test name of the test.
- `run_date`: The date of the latest run of the test.
- `status`: The status of the latest run of the test. This is a string from the set {"pass", "fail", "unknown"}.
- `duration_ms`: The (integer) duration of the latest run of the test in milliseconds.
- `source_filename`: The stripped filename of the source of the test.

- `source_line`: The 1-based line number of the source of the test.
  - `test_capture_run_id`: The (integer) id of the `TestCaptureRun` for the test.
  - `covered_functions`: The tests that cover the function. Requires the `--json` option be specified, see below. [TM]
  - `default`: The default output fields. See below.
- [TM] indicates output fields which use test metrics from the results of Test Advisor analysis. See `--test-pattern` above for details.

Output is in CSV format by default, unless the `--json` option is given. Each line except the first corresponds to a test, and contains the output fields in the order specified. The first line is a header indicating the field names.

- `--json`

Output in JSON format. The output is a JSON array, where each element corresponds to a test. Each element of this array is an object whose name/value pairs correspond to the specified `--output-fields`.

This option is required when `covered_functions` is included in the `--output-fields` option. Specifying the `covered_functions` output field will add an element named `covered_functions` to each test object, whose value is an array representing the functions covered by the test. Each element of this array is an object with the following name/value pairs:

- `mangled_name`: The mangled function name.
  - `unmangled_name`: The unmangled function name.
  - `filename`: The name of the file containing the function.
- `--strip-path <path>`

The `strip-path` to use when evaluating filenames. Normally this is not required, but can be used to override the default.

All dates must match a format described in Appendix A, *Accepted date/time formats*.

#### query-build-id

Outputs the current build ID for this intermediate directory.

#### remove-coverability <command\_options>

This option is deprecated as of the 2020.12 release. Use `delete-coverage` instead.

Removes computed coverability of lines in files in the intermediate directory. This is effectively the inverse of `compute-coverability`. Valid command options:

- `--verbose`

Enables verbose diagnostic messages.

`remove-from-library --dir <directory_name>`

Removes an intermediate directory from the intermediate directory library specified with the `--idir-library` option.

For example, the following command will remove the intermediate directory "my-idir" from the library "libdir".

```
> cov-manage-emit --idir-library libdir remove-from-library --dir my-idir
```

`repair`

Repairs database integrity. This operation might cause data loss, such as discarding translation units that are damaged.

`reset-host-name`

If the specified intermediate directory has data associated with a single host name other than the current host name, changes the host name associated with the emit database to the current host name.

`set-build-id`

Sets the build ID of the specified intermediate directory. The valid command line options are:

`--build-id <build-id>`

Set the build ID of the specified intermediate directory to `<build-id>`.

`--build-id-file <build-id-file>`

`<build-id-file>` is a file containing a build ID. Set the build ID of the specified intermediate directory to the value contained by this file.

`update-test [OPTIONS]`

Updates an existing test in your intermediate directory.

The required options are '`--testname`' and '`--status`'. By default, the test capture run marked `current` will be used unless options are given to select a specific test capture run of the test to update. If no test capture run has been marked `current`, this command will display a message and exit. The valid command line options are:

`--build-id <build ID>`

Specifies the build ID of the test capture run for the test.

For more information about `--build-id` usage, see the description for the `cov-build --build-id` option.

`--status <value>`

Specify if this test was successful. Valid values are:

- `pass`
- `fail`
- `unknown`

`--suite-name <SUITE-NAME>`

Specifies the suite name of the test to update.

`--test-capture-run-id <TCR ID>`

Specifies the test capture run ID of the test capture run for the test to use. You can determine the test capture run ID for a specific run by using the `list-test-capture-runs` sub-command.

`--test-capture-run-tag <tag>`

Specifies the custom tag of the test capture run for the test to update. For example:

```
"linux-build"
```

`--test-capture-run-timestamp <timestamp>`

Specifies the timestamp to use for the test capture run of the test for this invocation.

See Appendix A, *Accepted date/time formats* for proper formatting of the `<timestamp>` argument.

`--testname <TESTNAME>`

Specifies the test name of the test to update.

#### update-test-capture-run

Updates an existing test capture run in your intermediate directory. The valid command line options are:

- `--build-id <build ID>`

Specifies the build ID of the test capture run.

For more information about `--build-id` usage, see the description for the `cov-build --build-id` option.

- `--success <value>`

Specify if this test capture run was successful. Valid values are:

- `true`
- `false`
- `unknown`
- `unset`
- `--set-as-current`

Sets the test capture run as the "current" run in the intermediate directory. Any addition of coverage that does not specify a test capture run will use this test capture run.

- `--test-capture-run-id <TCR ID>`

Specifies the test capture run ID of the test capture run to use. You can determine the test capture run ID for a specific run by using the `list-test-capture-runs` sub-command.

- `--test-capture-run-tag <tag>`

Specifies a custom tag to allow for each selection of this test capture run. For example:

```
"linux-build"
```

- `--test-capture-run-timestamp <timestamp>`

Specifies the timestamp to use for the test capture run for this invocation. If it is not specified, the current time will be used, which is the typical use case. This option is only provided as a way for multiple test runs to use the same test capture run.

See Appendix A, *Accepted date/time formats* for proper formatting of the `<timestamp>` argument.

### Translation unit sub-commands with optional filtering

The following filtering sub-commands work on translation units. By default, all translation units are included in the results. You can optionally restrict the translation units used in these operations with the `--tu` and/or `--tu-pattern` options.

The options for listing emit database information and recompiling also support restricting the translation units with the `--tu` and/or `--tu-pattern` options.

`add <int_dir>`

Add (copy) all translation units from a specified intermediate directory (`<int_dir>`) into the current one (the one specified with the `--dir` option). If `--tu` and/or `--tu-pattern` are specified, then those filters are interpreted as applying to the source emit, and only the matching subset is copied.

`link-file <out_file>`

Create a file (`<out_file>`) with a description of the specified translation units as a link file, which can be used as input to `cov-link`.

`list`

List all translation units in the intermediate directory. Each translation unit is identified by its numeric ID, which is listed along with its primary source file name.

`list-json`

List all translation units in the intermediate directory as a standards-compliant JSON array. The translation units are identified by a numeric ID, which is listed along with the following fields:

- `id`: The unique numeric translation unit ID.
- `primaryFilename`: The primary source file name.
- `primaryFileSizeInBytes`: The size of the primary source file in bytes.
- `primaryFileHash`: MD5 hash of the contents of the primary source file.
- `language`: String describing the translation unit language.
- `userLanguage`: The user-specified translation unit language.
- `hasASTs`: Boolean. 'true' if the intermediate directory contains an AST for this translation unit, otherwise 'false'.

- `mspchTuFilename` (optional): The Microsoft precompiled header file which was created when this translation unit was built. This field is only displayed when a Microsoft precompiled header was created.
- `mspchId` (optional): The translation unit ID for the included Microsoft precompiled header. This field is only displayed when a precompiled header was used.

Example output:

```
[
 {
 "id" : 1,
 "primaryFilename" : "/home/build/project/tu1.cpp",
 "primaryFileSizeInBytes" : 92,
 "primaryFileHash" : "6f700a28a47e79cddff8fba60cac7098",
 "language" : "C++",
 "userLanguage" : "C++",
 "hasASTs" : true
 },
 {
 "id" : 2,
 "primaryFilename" : "c:/project/stdafx.cpp",
 "primaryFileSizeInBytes" : 122,
 "primaryFileHash" : "3827e3e7426ce0bdebb7e51c94d2a680",
 "language" : "C++",
 "userLanguage" : "C++",
 "hasASTs" : false,
 "mspchTuFilename" : "c:/project/stdafx.pch"
 }
]
```

#### Note

The output of this command may contain additional attributes that are not documented here. For maximum interoperability, please ignore any attribute that is not documented.

`extract-files --output-dir <dir> [--strip-path <path>]... [--regex <regex> | <filename>...]`

Extracts files present in the emit directory to the specified output directory.

The original directory, optionally stripped by any `--strip-path` arguments will be made relative to the specified output directory (on Windows, the drive letter if any, is always removed). Specify the files to extract by including either a regular expression or a list of filenames. If you include the `--regex` option, all files whose name matches the given regular expression are extracted: for this purpose, the file names are represented using a '/' separator. If a *tu* filter (`--tu`, `--tu-pattern`) is provided, only files referenced by the filtered TUs are included.

### Translation unit sub-commands with required filtering

The following filtering sub-commands work on translation units. You must supply the translation units used in these operations with the `--tu` and/or `--tu-pattern` options. The `TU list` sub-command identifies the TUs available for your required filter.

**delete**

Delete all TUs that satisfy the specified translation unit filter.

**preprocess**

Similar to `recompile`, except that when `cov-emit` is invoked, it is passed the `-E` (preprocess) and `--output_defs` options, which results in preprocessing only. The emit database is not modified by this operation.

The preprocessed output file (which is the stdout of `cov-emit`) is stored in the `preprocessed` subdirectory of the `c/output` subdirectory of the intermediate directory. The name of the file is the name of the primary source file for the TU, minus any path information, minus any file extension, plus either `.i` or `.ii`.

This option works only for C/C++ source code, not Java.

**print-compilation-info [<options>]**

For the specified translation units, print the command lines for `cov-emit`, `cov-translate` (if it was run), and `cov-build` (if it was run).

The options are:

- `--detailed` - provides all process details except environment variables.
- `--print-env` - provides the environment variable definitions for the process.

**print-compilation-time**

Prints the invocation time of any `cov-build`, `cov-emit`, `cov-emit-cs`, `cov-emit-java`, `cov-emit-vb`, `cov-translate`, or for a given translation unit (TU) to be easily accessible.

**Usage examples**

```
cov-manage-emit --dir dir -tu <TU#> print-compilation-time
```

```
cov-manage-emit --dir dir -tp <pattern> print-compilation-time
```

**Output example:**

```
cov-manage-emit --dir idir -tp "success()" print-compilation-time
```

```
Looking for translation units
|0-----25-----50-----75-----100|

Translation unit:
1 -> /Users/emoriarty/Testing/BZ55606/test.cpp
cov-emit invocation time (seconds): 1
cov-translate invocation time (seconds): 1
cov-build invocation time (seconds): 2
Translation unit:
2 -> /Users/emoriarty/Testing/BZ55606/test.cpp
cov-emit invocation time (seconds): 2
cov-translate invocation time (seconds): 2
```

```
cov-build invocation time (seconds): 2
Translation unit:
3 -> /Users/emoriarty/Testing/BZ55606/test.cpp
cov-emit invocation time (seconds): 2
cov-translate invocation time (seconds): 2
cov-build invocation time (seconds): 2
Translation unit:
4 -> /Users/emoriarty/Testing/BZ55606/test.cpp
cov-emit invocation time (seconds): 2
cov-translate invocation time (seconds): 2
cov-build invocation time (seconds): 2
```

### print-source

For the specified translation units, list the name and the contents of the primary source file associated with the TU. This option also reports, in parentheses, the internal row ID of the source file. It accepts the same command options as `print-source-files-contents`.

### print-source-files

For the specified translation units, list the names of all the source files associated with the TU. Also reports, in parentheses, the internal row ID of the source file.

### print-source-files-contents

For the specified translation units, list the names and contents of all the source files associated with the TU. Also reports, in parentheses, the internal row ID of the source file.

`print-source-files-contents` has the following options:

- `--scm-annotations` - Prefixes each source line with the change record (or commit record) that contributed most recently to the line. The change record data that is as follows:
  - Date and time that the change record was applied according to the SCM system.
  - The author (username) attributed to the change record.
  - The revision of the change, which is an identifier for the change record provided by the SCM system.
- `--coverage` - Outputs and prefixes each line of the TU being changed. Each line is printed with a marker identifying whether or not that line has been covered by a test (as seen by the Test Advisor `cov-build` command). The notations is as follows:
  - "+" - The line has been covered by one or more tests.
  - "-" - The line has not been covered by any tests.
  - " " - The line is not eligible for coverage (that is, the line does not represent executable code).
  - "?" - The line does not have coverage data available.

For example:

```
Translation unit:
```

```
1 -> ./sample.cpp
 Primary SF : ./sample.cpp (row ID 1)
 /* Sample that generates interesting coverage lines */
 /* (c) 2015 Synopsys, Inc. All rights reserved worldwide. */
 #
 +#void call_seven_times()
 #{
 +#}
 #
 +#void call_three_times()
 #{
 +#}
 #
 +#void call_ten_times()
 #{
 +#}
 #
 -#void call_zero()
 #{
 -#}
 #
 +#int main()
 #{
 +# int x = 0;
 +# for (int i = 0; i < 10; ++i) {
 # x += i
 +# * 3; /* multi-line statement, may be wrong */
 +# if (i < 7) {
 +# call_seven_times();
 # } else {
 +# call_three_times();
 # }
 +# call_ten_times();
 +# if (x > 300) {
 -# call_zero();
 # }
 # }
 +# return 0;
 #}
```

- `--build-id <build ID>`

Specifies the build ID of the test capture run.

For more information about `--build-id` usage, see the description for the `cov-build --build-id` option.

- `--test-capture-run-id <TCR ID>`

Specifies the test capture run ID of the test capture run to use. You can determine the test capture run ID for a specific run by using the `list-test-capture-runs` sub-command.

- `--test-capture-run-tag <tag>`

Specifies a custom tag to allow for each selection of this test capture run. For example:

```
"linux-build"
```

- `--test-capture-run-timestamp <timestamp>`

Specifies the timestamp to use for the test capture run for this invocation. If it is not specified, the current time will be used, which is the typical use case. This option is only provided as a way for multiple test runs to use the same test capture run.

See Appendix A, *Accepted date/time formats* for proper formatting of the `<timestamp>` argument.

- `--suiteName <suiteName>`

Identifies the coverage data as belonging to the named suite. This option is an accepted option when `--coverage` is used.

- `--testName <testName>`

Identifies the coverage data as belonging to the named test. This option is an accepted option when `--coverage` is used.

#### print-source-files-stats

For the specified translation units, list the names of all of the source files associated with the TU. Also reports the internal row ID of the source file (in parentheses) followed by statistics for that source file. The statistics listed include:

- The file contents time stamp, size, and MD5 sum
- The count of blank lines
- The count of comment lines
- The count of code lines
- The count of code lines with inline comments

Example output is as follows:

```
1 -> /example_dir/a.cpp
Primary SF : /example_dir/a.cpp (row ID 1)
 Timestamp: 2013-07-19 11:37:35
 Size: 25
 MD5 sum: 7edc175dc475923c51c579924b724a8c
 Blank lines: 1
 Comment lines: 0
 Code lines: 2 (1 with inline comments)
```

#### print-tuid

Prints the TU ids for the TU requested using either `-tu` or `--tu-pattern`. Unlike most commands that will error if an invalid tu is specified, `print-tuid` will silently ignore it. For example:

```

$ cov-manage-emit --dir foo --tu-pattern 'success()' print-tuid
Looking for translation units
|0-----25-----50-----75-----100|
|*****|
1
3
4

$ cov-manage-emit --dir foo --tu-pattern 'success()' print-tuid -of tuids.txt
Looking for translation units
|0-----25-----50-----75-----100|
|*****|
$ cat tuids.txt
1
3
4

```

### Listing emit database information

The `find` sub-command lists information stored in the emit DB such as symbol names, locations, and definitions. By default, all translation units are included in the results. You can optionally restrict the translation units used in these operations with the `--tu` and/or `--tu-pattern` options.

What is being matched by the regular expression (regex) is, in C++, the mangled name of the symbol (according to the IA64 C++ ABI, see <http://mentoreembedded.github.io/cxx-abi/abi-examples.html#mangling>), of which the actual identifier is always a substring. In C, what is matched is just the identifier.

`find <regular expression> [OPTIONS]`

There are four kinds of symbols: functions, classes, global variables, and enumerations. If you specify

```
find <regular expression>
```

then the listing for the matching regex command is the symbol name, the kind of entity, the declaration location, and the definition TU.

You can control the information that is returned by using the following options:

`--kind {f | c | e | g}`

Restricts the search to certain types of entities. The choices are `f`, `c`, `e` and `g`, for function, class, enum, and global, respectively.

`--print-callees`

For a function, lists the set of functions it calls. Does not list information on other entities.

`--print-codexm`

Lists the entity's abstract syntax tree (AST) definition as CodeXM patterns.

CodeXM is a specialized language used to write customized checkers that run using the Coverity engine.

**--print-definitions**

Lists the entity's definition syntax by pretty-printing the AST definition.

**--print-debug**

Lists the entity's AST definition in debug (indented tree) mode.

The `find` sub-command accepts multiple operands and applies each of them as an inclusive filter when searching for symbols. In the following example, the first invocation of `cov-manage-emit` displays all symbols (`global_1` and `global_2`).

```
$ cat t.c
int global_1 = 1;
int global_2 = 2;

$ cov-emit --dir covint t.c
Emit for file '/tmp/t.c' complete.

$ cov-manage-emit --dir covint find .
Matching global: global_1
 declared at:
 /tmp/t.c:1:5-/tmp/t.c:1:12
 defined in TU 1 with row 1
Matching global: global_2
 declared at:
 /tmp/t.c:2:5-/tmp/t.c:2:12
 defined in TU 1 with row 2
```

The following two examples supply a regex command that selects exactly one of those symbols.

```
$ cov-manage-emit --dir covint find global_1
Matching global: global_1
 declared at:
 /tmp/t.c:1:5-/tmp/t.c:1:12
 defined in TU 1 with row 1

$ cov-manage-emit --dir covint find global_2
Matching global: global_2
 declared at:
 /tmp/t.c:2:5-/tmp/t.c:2:12
 defined in TU 1 with row 2
```

The following invocation specifies multiple regex commands that select both symbols.

```
$ cov-manage-emit --dir covint find global_1 global_2
Matching global: global_1
 declared at:
 /tmp/t.c:1:5-/tmp/t.c:1:12
 defined in TU 1 with row 1
Matching global: global_2
 declared at:
 /tmp/t.c:2:5-/tmp/t.c:2:12
 defined in TU 1 with row 2
```

## Emit Server sub-commands

These commands allow for starting and stopping an emit server on the specified intermediate directory/intermediate directory library. These commands all take an intermediate directory or an intermediate directory library.

You may only specify one of `--dir` or `--idir-library`, but not both.

### start-server [OPTIONS]

Starts an emit server for the specified intermediate directory or intermediate directory library. Valid options are:

- `--port <port-number>`

Specifies the port number to bind the server. The default is 15772.

- `--interface <ip-address-to-run-on>`

Specifies the IP address to which you want the server to bind.

- `--gcov-cache-size <size-in-mb>`

Specifies the size of the cache in MB to use for gcov data. The default is 500MB.

- `--force-start`

Forces the start of the server. This is useful if the previous emit server was not cleanly shut down and the PID file remains from the previous run.

Examples:

To start an emit server on a single intermediate directory:

```
cov-manage-emit --dir idir start-server [--port 15772] [--interface host_or_ip]
```

To start an emit server on an intermediate directory library:

```
cov-manage-emit --idir-library idir-lib start-server [--port 15772] [--interface host_or_ip]
```

### stop-server

Stops a running emit server for the specified intermediate directory or intermediate directory library.

Examples:

To stop a running emit server on a single intermediate directory:

```
cov-manage-emit --dir idir stop-server
```

To stop an emit server on an intermediate directory library:

```
cov-manage-emit --idir-library idir-lib stop-server
```

### query-server

Query an intermediate directory or intermediate directory library to verify that an emit server is already running. The output will be in a JSON format, for example:

```
[
 {
 "host" : "localhost",
 "pid" : 654321,
 "port" : 33445
 }
]
```

If no server is running, the JSON file will be empty.

Examples:

To check if an emit server is running for a given single intermediate directory:

```
cov-manage-emit --dir dir query-server
```

To check if an emit server is running for a given intermediate directory library:

```
cov-manage-emit --idir-library idir-lib query-server
```

## Recompiling

The `recompile` sub-commands repeat a `cov-emit` compilation. You can use this option, for example, with updated `cov-emit` binary or compiler configuration settings to attempt to compile inputs that have previously failed. This is similar to `cov-build --replay`.

You can modify the translation units that are recompiled with the `--tu` and/or `--tu-pattern` options.

The `recompile` sub-commands are:

### parse-source-only-tus [OPTIONS]

Recompiles source-only TUs from the intermediate directory (those that were added through `cov-build --record-with-source` and that have not been recompiled already).

This subcommand works only for C/C++ source code.

### recompile [OPTIONS]

Recompile the set of TUs specified by the filter. For each TU to be recompiled, invoke `cov-emit` with the command line, environment settings, and current directory recorded in the emit repository. Source files are re-read from the file system.

#### Note

Note that this subcommand will not correctly recompile your selected TUs if the intermediate directory has been moved since running `cov-emit`. If you have moved your intermediate directory to a new location or separate machine, use `recompile-from-dir` and specify the new `--dir` location.

### recompile-from-dir [OPTIONS]

Recompiles translation units from source contained within the emit directory. Replaying from the emit will have the same results, regardless of changes to the files in the filesystem (including deletion).

This option is similar to `cov-build --replay-from-emit`, but it allows you to perform finer-grained filtering of the TUs being replayed. For example:

```
cov-manage-emit --dir idir --tu 10 recompile-from-dir
```

This subcommand works only for C/C++ source code.

### replay-from-script -if <json\_file> [OPTIONS]

Reads a JSON script produced by Incredibuild, builds a list of compile commands, and executes each of the compile commands against `cov-translate` for accelerating Windows code builds using Incredibuild.

The `-if <json_file>` option points to the json script file that is described in the `replay_from_script` command.

For more information, see "Using IncrediBuild" in the Coverity Analysis 2020.12 User and Administrator Guide. [🔗](#).

#### **Note**

`--record-only` works the same as "`cov-build --record-only`," recording the build to be replayed later.

### retranslate [OPTIONS]

Run `cov-translate` on the set of TUs specified by the filter.

For each TU to be recompiled, invoke `cov-translate` using the command line, environment settings, and current directory recorded in the emit repository. Does not work with a TU compiled directly by `cov-emit`.

Invocation of `cov-translate` requires a Coverity configuration. By default, the configuration that was used during the initial compilation will be used, but this can be overridden by specifying a configuration on the `cov-manage-emit` command line.

This subcommand works only for C/C++ source code.

#### **Note**

Note that this subcommand will not correctly retranslate your selected TUs if the intermediate directory has been moved since running `cov-emit`.

### retranslate-or-emit [OPTIONS]

Run `cov-translate` on the set of TUs specified by the filter.

Similar to the `retranslate` option, except that in the case of a TU where `cov-emit` was invoked directly without `cov-translate`, invokes `cov-emit` instead of using `cov-translate`.

This subcommand works only for C/C++ source code.

 **Note**

Note that this subcommand will not work correctly if the intermediate directory has been moved since running `cov-emit`.

The `recompile` sub-command [OPTIONS] are as follows:

`--compilation-log <log_file>`

Saves diagnostic messages from `cov-translate` and `cov-emit` to `<log_file>` (instead of the default of standard output and standard error). Also displays a progress ticker bar.

`--desktop`

Used in conjunction with Desktop Analysis to perform recompilation faster by disabling bytecode decompilation in Java, C#, and Visual Basic builds.

`--do-decomp`

Used in conjunction with Desktop Analysis to perform recompilation in Java builds where bytecode decompilation is enabled.

`--emit-complementary-info`

Enables emitting of complementary information for compliance checkers such as MISRA checkers. Selecting this option results in a slower build capture but a faster analysis, and it should be applied when using compliance checkers. The default value is `--no-emit-complementary-info`

 **Note**

Enabling the `--emit-complementary-info` option prior to running an analysis is likely to turn up additional defects.

.

Any analysis involving `--coding-standard-config` requires the information generated during `cov-build` when including the `--emit-complementary-info` option. The `cov-build` command will take longer, so this option should only be used when `cov-analyze` is used with `--coding-standard-config`.

If `cov-build` did not include the `--emit-complementary-info` option and `cov-analyze` does include `--coding-standard-config`, `cov-analyze` automatically re-runs every `cov-emit` command (for the Translation Units to be analyzed). This excludes the native build and the `cov-translate` overhead, but it will add significant overhead to `cov-analyze`. Note that analysis will fail if the emit database does not include source; that is re-emit is not possible.

`--name <name>`

Associates any new TUs created with a build named `<name>`. New TUs are not created by `parse-source-only-tus` or `recompile-from-dir`. These commands will reuse the existing TUs, so this option will have no effect. TUs will also not be created if the TUs are already up to date.

`--parallel <number_of_processes> , -j <number_of_processes>`

Spawn up to `<number_of_processes>` processes to run the recompilations. This option accepts the number of processes, or `auto` which sets the number of replay processes to the number of logical processors in the machine (`-j 0` is also accepted and is the same as `auto`).

## Decompiling

The `decompile-binary-tus-from-dir` subcommands repeat a decompilation recorded in the emit.

`decompile-binary-tus-from-dir` [OPTIONS]

Decompiles translation units from byte code source contained within the emit directory. Replaying from the emit will have the same results, regardless of changes to the files in the filesystem (including deletion)..

This option is similar to `cov-build --replay-decomp`, but it allows you to perform finer-grained filtering of the TUs that are being replayed. For example:

```
cov-manage-emit --dir idir --tu 10 decompile-binary-tus-from-dir
```

The `decompile` sub-commands are:

`--compilation-log <log_file>`

Saves diagnostic messages to `<log_file>` (instead of the default of standard output and standard error). Also displays a progress ticker bar.

`--disable-decomp-bodies`

Disable decompiling method bodies of the byte code.

## Translation unit pattern matching

The argument to `--tu-pattern` is a string that acts as a filter on translation units. Alternatively, to use a file name for a pattern, specify `@<filename>`. Each pattern in this file must be on a separate line.

To get useful information about the translation units in an emit repository, use the `list` sub-command.

A pattern has the following syntax:

```
[!] <function>("<regex>" | '<regex>') [|] <function>("<regex>")
[&& <function>("<regex>")]
```

When combining patterns, the precedence from lowest to highest, is OR (`|`), AND (`&&`), and NEG (`!`). OR and AND are left-associative. You can use parentheses to group expressions to override precedence or associativity. The `regex` is a Perl regular expression. A backslash in a quoted string is interpreted as a regular expression metacharacter, and not as a string literal metacharacter. You can use single or double quotes to pass the string properly from the shell to the command. The Perl `regex` is used for partial matches; for full matches use the beginning of line (`^`) and end of line (`$`) symbols.

The values for `<function>` for where to apply the regular expression are:

`all`

No argument. Matches all compilations. Used when all TUs desired and `tu-pattern` required.

`arg`

Matches if the `regex` matches any of the native compiler command line elements, including the native compiler executable itself.

**build\_arg:**

Matches if the `regex` matches any argument to `cov-build`, including the `cov-build` executable name.

**build\_name("<regex>")**

Matches if `cov-build --name <name>` compiled the translation unit and `<name>` is matched by `<regex>`.

**cov\_emit\_arg**

Matches if the `regex` matches any argument to the Coverity compiler front end, such as `cov-emit`, including the executable name.

**file**

Matches if the `regex` matches the name of the primary source file. For the purpose of turning a file name into a string that can then be matched against a `regex`, the following normalizations are applied:

- The name is converted to an absolute pathname. On Windows, this includes the drive letter.
- On Windows, all letters are lower-cased, including the drive letter (this applies to all names in translation units created on Windows).
- The forward-slash character (`/`) separates name components.
- When no drive letter is present, the name begins with `/`; otherwise, a `/` follows the drive letter.

For example:

```
--tu-pattern "file('test\.c$')"
```

**failure**

No argument. Matches if the compilation was unsuccessful (exit code `!= 0`). Used by `cov-build --replay-failures`.

**header**

Matches if the `regex` matches the name of any header file, which is defined to be a source file included in the TU other than the primary source file.

**lang("<lang>")**

The `lang` pattern matches TUs with one of the following specified language patterns:

- C
- C++
- C#
- CUDA
- .NET bytecode
- Fortran
- Go

- HTML
- Java
- JavaScript
- JSX
- JVM bytecode
- Kotlin
- Objective-C
- Objective-C++
- PHP
- Python 2
- Python 3
- Ruby
- Scala
- Swift
- Text
- TypeScript
- Visual Basic
- Vue.js SFC

**link\_file**

The `regex` is not interpreted as a regular expression, but rather as the name of a file, which should be the output of `cov-link` (or `cov-manage-emit link_file`).

**success**

No argument. Matches if the compilation was successful (exit code = 0).

**Shared options**

**--debug, -g**

Turn on basic debugging output.

**--info**

Displays certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

`--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>`

Set the detail level of command messages. Higher is more verbose (more messages). The default is 1. Use `--verbose 0` to disable progress bars.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## Examples

List build information from an intermediate directory:

```
> cov-manage-emit --dir apache_2111 \
 list-builds
```

List all translation unit information from an intermediate directory:

```
> cov-manage-emit --dir apache_2111 \
 list
```

List information from an intermediate directory for the translation unit with the ID 6:

```
> cov-manage-emit --dir apache_2111 --tu 6 \
 list
```

List all information on all entities:

```
> cov-manage-emit --dir apache_2111 \
 find *.*
```

List only callee information for all entities:

```
> cov-manage-emit --dir apache_2111 \
 find --print-callees *.*
```

List all information for entity `uninit`:

```
> cov-manage-emit --dir apache_2111 \
 find --print-callees uninit
```

```
find '^uninit$'
```

List the definition of entity `uninit`:

```
> cov-manage-emit --dir apache_2111 \
 find '^uninit$' --print-definitions
```

List the source files of TU 1:

```
> cov-manage-emit --dir apache_2111 \
 --tu 1 print-source-files
```

List TUs where `bar.cc` or `foo.cc` is the primary source file:

```
> cov-manage-emit --dir apache_2111 \
 --tu-pattern "file('bar.\cc$') || file('foo.\cc$')" list
```

List TUs using patterns specified in the file `files`:

```
> cov-manage-emit --dir apache_2111 \
 --tu-pattern @files list
```

Recompile the TUs in the emit database:

```
> cov-manage-emit --dir apache_2111 recompile
```

Recompile and put diagnostic information in the `211_log.txt` file:

```
> cov-manage-emit --dir apache_2111 \
 recompile --compilation-log 211_log.txt
```

Recompile only TU 1:

```
> cov-manage-emit --dir apache_2111 \
 --tu 1 recompile --compilation-log 211_log.txt
```

Recompile translation units where `test.c` is the primary source file:

```
> cov-manage-emit --dir apache_2111 \
 --tu-pattern "file('test\c$')" recompile
```

List source files missing SCM annotations from the intermediate directory:

```
> cov-manage-emit --dir apache_2111 \
 list-scm-unknown --output files-without-scm-age-data.txt
```

List source files under `/builds` missing SCM annotations from the intermediate directory:

```
cov-manage-emit --dir apache_2111 list-scm-unknown \
 --output files-without-scm-annotation-data.txt \
 --filename-regex '^/builds/'
```

List source files with existing SCM annotations in the intermediate directory:

```
> cov-manage-emit --dir apache_2111 \
 list-scm-known
```

```
list-scm-known --output files-with-scm-age-data.txt
```

List source files under `/builds` with existing SCM annotations from the intermediate directory:

```
cov-manage-emit --dir apache_2111 \
 list-scm-known --output files-with-scm-annotations.txt \
 --filename-regex '^/builds/'
```

Count the source files under `/builds` with existing SCM annotations from the intermediate directory:

```
cov-manage-emit --dir apache_2111 \
 list-scm-known --output count-files-with-scm-annotations.txt \
 --filename-regex '^/builds/' --count
```

Add SCM annotations for source files in the intermediate directory:

```
> cov-manage-emit --dir apache_2111 \
 add-scm-annotations --input scm-age-data.txt
```

Dump SCM annotations for source files in the intermediate directory:

```
> cov-manage-emit --dir apache_2111 \
 dump-scm-annotations --output scm-age-data.txt
```

Remove all SCM annotations from the intermediate directory:

```
cov-manage-emit --dir apache_2111 delete-scm-annotations
```

Add test coverage data from a single gcda file to the intermediate directory:

```
> cov-manage-emit --dir apache_2111 \
 add-coverage --suite-name FooSuite --test-name FooTest \
 --teststart "2012-03-19 07:12:11" --verbose \
 --gcda test.gcda
```

Add test coverage data in batch mode using the batch script `coverage-batch.txt`:

```
> cov-manage-emit --dir apache_2111 \
 add-coverage --batch coverage-batch.txt
```

Compute the missing coverage and add it to the intermediate directory:

```
> cov-manage-emit --dir apache_2111 compute-coverability
```

Remove all coverage from the intermediate directory:

```
cov-manage-emit --dir apache_2111 delete-coverage
```

Remove the coverage that was added by `compute-coverability` from the intermediate directory:

```
> cov-manage-emit --dir apache_2111 remove-coverability
```

List all tests that have results stored in the intermediate directory:

```
> cov-manage-emit --dir apache_2111 list-tests
```

List all tests for suite `MyModuleTests` that have results stored in the intermediate directory:

```
cov-manage-emit --dir apache_2111 list-tests --suiteName MyModuleTests
```

Count all tests that have results stored in the intermediate directory:

```
cov-manage-emit --dir apache_2111 list-tests --count
```

List source files missing coverage data from the intermediate directory:

```
cov-manage-emit --dir apache_2111 \
 list-coverage-unknown --output files-without-coverage-data.txt
```

List source files under `/builds` missing coverage data from the intermediate directory:

```
cov-manage-emit --dir apache_2111 \
 list-coverage-unknown --output files-without-coverage-data.txt \
 --filename-regex '^/builds/'
```

List source files with existing coverage data in the intermediate directory:

```
cov-manage-emit --dir apache_2111 \
 list-coverage-known --output files-with-coverage-data.txt
```

List source files under `/builds` with existing coverage data from the intermediate directory:

```
cov-manage-emit --dir apache_2111 \
 list-coverage-known --output files-with-coverage-data.txt \
 --filename-regex '^/builds/'
```

Count the source files under `/builds` with existing coverage data from the intermediate directory:

```
cov-manage-emit --dir apache_2111 \
 list-coverage-known --output count-files-with-coverage-data.txt \
 --filename-regex '^/builds/' --count
```

## See Also

[cov-build](#)

[cov-extract-scm](#) 

[cov-import-scm](#) 

---

## Name

cov-preprocess Preprocess a C/C++ source file.

## Synopsis

```
cov-preprocess [OPTIONS] <files>
```

## Description

The `cov-preprocess` command preprocesses a source file. This command reads the command-line argument from `<intermediate_directory>/emit` and outputs the preprocessed source file or files into the `<intermediate_directory>/output/preprocessed` directory. Coverity recommends that you use the `--output-file` option to change the default output behavior whenever you need for the combined output path and filename to be an invariant (for example, in scripts) because the internal structure of the intermediate directory might change in future releases. However, note that using this option necessitates that only one file is preprocessed per invocation of `cov-preprocess`.

If you do not use an absolute path name to specify a file name, `cov-preprocess` searches for the specified file name in `<intermediate_directory>/emit`. To speed up the search time, use full path names to files that you want to preprocess.

Preprocessing expands all preprocessor directives such as `#include`, and expands macro definitions. A preprocessed source file is self-contained and can be compiled by itself with no additional files.



### Note

`cov-preprocess` doesn't read source files from the emit or re-create source folders. Therefore the source files, directory structure, and working environment of the original build captured in the emit must exist and be used when `cov-preprocess` runs.

## Options

`--config <coverity_config.xml>` , `-C <coverity_config.xml>`

Uses the specified configuration file instead of the default configuration file located at `<install_dir_sa>/config/coverity_config.xml`.

`--diff, -d`

Preprocess the file with both the native compiler and with `cov-emit`, and then attempt to find relevant differences between the preprocessed files using some heuristics.

`--diff-only`

Determine the relevant differences between two files that are already preprocessed using some heuristics.

`--dir <intermediate_directory>`

Path name to an intermediate directory that is used to store the results of the build and analysis.

`-if <source_file>`

Specify an input file that is preprocessed as if it were the file that was compiled.

**--native**

Preprocess with the native compiler instead of with `cov-emit`.

**--no-lines, -n**

Do not put `#line` directives in the preprocessed output file.

**--no-retranslate, -nr**

Do not re-translate the command line from the original compiler when attempting to preprocess with `cov-emit`. This can be faster, but it will not work with template compiler configurations.

**--output-file <output-file>, -of <output\_file>**

Specify the path to and file name for the output file. Coverity recommends that you use this option instead of relying on the default output behavior.

**--tu <translation\_unit\_id(s)>, -tu <translation\_unit\_id(s)>**

A set of translation units (TUs), named by their numeric id attribute(s). A translation unit approximately maps to the output from a single run of a compiler. This option requires a comma-separated list of id(s), and `--tu` may be specified multiple times. The union of all these identifier sets is the set of TUs to operate on subsequently, for operations that work on TUs. It is an error if any of the specified IDs do not correspond to any existing translation unit.

## Shared options

**--debug, -g**

Turn on basic debugging output.

**--info**

Displays certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

**--ident**

Displays the version of Coverity Analysis and build number.

**--tmpdir <tmp>, -t <tmp>**

Specifies the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

**--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>**

Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.

- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects`(*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

### **See Also**

`cov-translate`

---

## Name

cov-record-source Record Java Web applications and output them to an intermediate directory.

## Synopsis

```
cov-record-source --dir <intermediate_dir> [OPTIONS]
```

## Description

The `cov-record-source` command records Java Web application files and outputs them to an intermediate directory (this command does not attempt to parse or emit the files).

The Java Web applications can be in the following forms:

- Web Archive (.war file)
- Enterprise Archive (.ear file)
- A directory with the unpacked contents of either
- Any combination of the above

When recording a build that contains a Java Web application that you want to analyze, you must run the `cov-record-source` command in addition to the `cov-build --record-with-source` command to properly record the Java Web application.

## Options

`--findears <directory_list>`

See the `cov-emit-java --findears` option.

`--findears-unpacked <directory_list>`

See the `cov-emit-java --findears-unpacked` option.

`--findjars <jar_containing_directories>`

See the `cov-emit-java --findjars` option.

`--findwars <directory_list>`

See the `cov-emit-java --findwars` option.

`--findwars-unpacked <directory_list>`

See the `cov-emit-java --findwars-unpacked` option.

`--webapp-archive <archive_file_or_dir>, --war <archive_file_or_dir>, --ear <archive_file_or_dir>`

See the `cov-emit-java --webapp-archive` option.

## Exit codes

This command returns the following exit codes:

- 0: The command successfully completed the requested task.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

### **See Also**

cov-emit-java

cov-build

---

## Name

`cov-run-desktop` Analyze locally changed files on a developer's desktop.

## Synopsis

```
cov-run-desktop [OPTIONS] [FILES]
```

## Description

The `cov-run-desktop` command performs an expedited local analysis by considering only the files or translation units specified by the user, subject to various command line options. When running `cov-run-desktop`, you must either include the specific file name(s) at the end of the command, pass the `--analyze-scm-modified` option to let your Source Code Management system (SCM) specify which files to analyze, or analyze previously captured source, by using the `--analyze-captured-source` option. Translation unit selection has additional information on this process.

You can create a `coverity.conf` file to share compiler configurations and other desktop analysis settings to multiple users of the same code base. This configuration file can also be used by individual users to maintain these settings locally (see the *Coverity Desktop Analysis 2020.12: User Guide* [🔗](#) for details).

The analysis performed is the same as when using `cov-analyze`, so `cov-run-desktop` accepts most of the same options as those two commands.

Because `cov-run-desktop` normally does not analyze an entire code base, it relies on summaries stored in Coverity Connect to get information about the code that is not analyzed locally. This requires that a periodic (perhaps nightly) full analysis be run in order to populate the summary information. Additionally, after the main analysis phase is complete, `cov-run-desktop` normally contacts the Coverity Connect server to determine which defects were newly introduced, and to retrieve triage data (Classification, Severity, Owner, etc.) for existing issues. `cov-run-desktop` can also operate in "disconnected" mode, relying on previously downloaded summary and issue data, if any.

By default, `cov-run-desktop` produces issue output in a text format that is intended to imitate the typical output syntax of compiler error messages. Most editors and IDEs will then automatically allow you to navigate to the corresponding locations in the source code. `cov-run-desktop` can also write the issues in JSON format so that a user-provided tool can consume and present them in a user-defined way.

The exit code of `cov-run-desktop` is 0 when the analysis completes successfully and 2 or greater if there is an error. The `--exit1-if-defects` option causes `cov-run-desktop` to exit with code 1 when defects are present.

## Translation unit selection

In order to improve the speed of the analysis, `cov-run-desktop` limits the number of files or translation units considered. In this context, a translation unit refers to any primary source file from the intermediate directory, along with any files included by that primary source file. The selection process is defined using various options that affect what code will be analyzed.

There are two methods to select which translation units will be analyzed, and it is required that the user specify one. The first method is to explicitly pass a list of source files to be analyzed at the end of the command line:

```
cov-run-desktop [OPTIONS] FILE1 [FILE2 [...]]
```

For C and C++, the files will typically be `.c` or `.cpp` source files. If you want to analyze a header file, see *Coverity Desktop Analysis 2020.12: User Guide* [🔗](#).

This can also be accomplished by listing your chosen source files in a response file, and then specifying it on the command line with the `@@<response_file>` syntax.

The second method of translation unit selection is to query your source code management (SCM) system for the set of files that have been modified locally and analyze those. This method is designated with the `--analyze-scm-modified` option. See `--analyze-scm-modified` and [A note on Git superprojects](#) for more information on this method.

 **Note**

You will receive an error if you attempt to pass more than one of these methods in the same command.

## A note on Git superprojects

Git superprojects are unsupported by Desktop Analysis, and will cause errors when used with `--analyze-scm-modified`.

You may be able to workaround this issue by creating a script to access Git using submodules, and specify that script with the `--scm-tool` option. This is an advanced usecase, and should only be attempted by experienced users.

## Regular expressions

All `cov-run-desktop` options that call for a regular expression (`regex`) follow Perl syntax. The regular expression is case sensitive, and is considered a match if it matches a substring (i.e. full string match requires explicit anchors).

## Options categories

The options accepted by `cov-run-desktop` fall into several categories:

 **Note**

Each of the items below are linked to other sections in this document. Many link to the relevant definition in the `cov-run-desktop` section, while the analysis options link to `cov-analyze`.

Options that affect what code will be analyzed

- `--analyze-captured-source`

- `--analyze-scm-modified`
- `--analyze-untracked-files`
- `--ignore-modified-file-regex`
- `--ignore-modified-non-psf`
- `--ignore-untracked-file-regex`
- `--modification-date-threshold`
- `--restrict-modified-file-regex`
- `--restrict-untracked-file-regex`
- `--tu-pattern`

#### Options for using your Source Code Management (SCM) system

- `--analyze-untracked-files`
- `--ignore-untracked-file-regex`
- `--restrict-untracked-file-regex`
- `--scm`
- `--scm-project-root`
- `--scm-tool`
- `--scm-tool-arg`

#### Options that control the analysis

- `--aggressiveness-level`
- `--all`
- `--android-security`
- `--checker-option`
- `--concurrency (C/C++)`
- `--debug`
- `--debug-flags (C/C++)`
- `--disable`

- `--disable-android-security`
- `--disable-default`
- `--disable-fb (Java)`
- `--disable-fnptr (C/C++)`
- `--disable-misra (C/C++)`
- `--disable-parse-warnings (C/C++/Swift)`
- `--distrust-all`
- `--distrust-mobile-other-app`
- `--distrust-mobile-other-privileged-app`
- `--distrust-mobile-same-app`
- `--distrust-mobile-user-input`
- `--distrust-console`
- `--distrust-database`
- `--distrust-environment`
- `--distrust-filesystem`
- `--distrust-http`
- `--distrust-http-header`
- `--distrust-js-client-cookie`
- `--distrust-js-client-external`
- `--distrust-js-client-html-element`
- `--distrust-js-client-http-referer`
- `--distrust-js-client-http-header`
- `--distrust-js-client-other-origin`
- `--distrust-js-client-url-query-or-fragment`
- `--distrust-network`
- `--distrust-rpc`

- `--distrust-servlet`
- `--distrust-system-properties`
- `--enable`
- `--enable-audit-mode`
- `--enable-callgraph-metrics`
- `--enable-constraint-fpp`
- `--enable-fb (Java)`
- `--enable-fnptr (C/C++)`
- `--enable-parse-warnings (C/C++/Swift)`
- `--enable-single-virtual (C/C++)`
- `--enable-virtual (C/C++)`
- `--extend-checker`
- `--extend-checker-option`
- `--fb-exclude (Java)`
- `--fb-include (Java)`
- `--fb-max-mem (Java)`
- `--fnptr-models (C/C++)`
- `--hfa (C/C++)`
- `--ident`
- `--ignore-deviated-findings`
- `--info`
- `--inherit-taint-from-unions (C/C++)`
- `--jobs`
- `--max-loop (C/C++)`
- `--max-mem`
- `--misra-config (C/C++)`

- `--model-file`
- `--no-field-offset-escape` (C/C++)
- `--not-tainted-field` (C#, Java, Visual Basic)
- `--override-worker-limit` (C/C++)
- `--parse-warnings-config` (C/C++/Swift)
- `--paths` (C/C++)
- `--redirect`
- `--rule` (C/C++)
- `--security` (C/C++)
- `--security-file`
- `--skip-android-app-sanity-check` (Java Android)
- `--strip-path`
- `--tainted-field` (Java)
- `--tmpdir`
- `--trust-all`
- `--trust-mobile-other-app`
- `--trust-mobile-other-privileged-app`
- `--trust-mobile-same-app`
- `--trust-mobile-user-input`
- `--trust-console`
- `--trust-database`
- `--trust-environment`
- `--trust-filesystem`
- `--trust-http`
- `--trust-http-header`
- `--trust-js-client-cookie` )

- `--trust-js-client-external`
- `--trust-js-client-html-element`
- `--trust-js-client-http-referer`
- `--trust-js-client-http-header`
- `--trust-js-client-other-origin`
- `--trust-js-client-url-query-or-fragment`
- `--trust-network`
- `--trust-rpc`
- `--trust-servlet`
- `--trust-system-properties`
- `--tu-pattern (C/C++)`
- `--use-reference-settings`
- **[Deprecated]** `--user-model-file`: Use `--model-file` instead.
- `--verbose`
- `--wait-for-license`
- `--webapp-security-aggressiveness-level (C#, Java, Visual Basic)`
- `--webapp-security (C#, Java, JavaScript, Visual Basic)`
- `--whole-program`

Options to specify connection and triage information for the Coverity Connect server

- `--auth-key-file`
- `--certs`
- `--connect-timeout`
- `--disconnected`
- `--host`
- `--mark-fp`
- `--mark-int`

- --on-new-cert
- --password
- --port
- --reference-snapshot
- --set-new-defect-owner
- --set-new-defect-owner-limit
- --set-new-defect-owner-to
- --ssl
- --stream
- --user

#### Output and filtering options

- --add-ignore-modified-file-regex
- --add-restrict-modified-file-regex
- --category-regex
- --checker-regex
- --cid
- --component-not-regex
- --component-regex
- --confine-to-scope
- --custom-triage-attribute-not-regex
- --custom-triage-attribute-regex
- --exit1-if-defects
- --file-regex
- --first-detected-after
- --first-detected-before
- --function-regex

- `--ignore-modified-file-regex`
- `--impact-regex`
- `--include-missing-locally`
- `--json-output-v7 <filename>`
- `--kind-regex`
- `--lang`
- `--language-regex`
- `--local-status-not-regex`
- `--local-status-regex`
- `--merge-key-regex`
- `--MISRA-category-regex`
- `--no-default-triage-filters`
- `--no-text-output`
- `--occurrences`
- `--ownerLdapServerName-regex`
- `--print-path-events`
- `--present-in-reference`
- `--relative-paths`
- `--relative-to`
- `--restrict-modified-file-regex`
- `--sort`
- `--subcategory-regex`
- `--text-output`
- `--text-output-style`
- `--triage-attribute-not-regex`
- `--triage-attribute-regex`

## Options

`--add-ignore-modified-file-regex <regex>`

Specify a `<regex>` to ignore in addition to what is specified by `--ignore-modified-file-regex` and the `coverity.conf` file.

`--add-restrict-modified-file-regex <regex>`

Specify a `<regex>` to add to those specified by `--restrict-modified-file-regex` and the `coverity.conf` file. This means that filtered translation units must match both regular expressions specified by `--add-restrict-modified-file-regex` and `--restrict-modified-file-regex`.

`--allow-suffix-match`

Restores backward compatibility (pre-8.7) in specifying files to analyze on the command line. Specifically, this option allows specifying files for analysis using any unique path suffix already captured in the emit. Normally (without this option), the specified path must exist relative to the current working directory or as an absolute path, as one would expect from a command line command, so this option is only recommended for backward compatibility.

This functionality can be made more permanent through a `coverity.conf` setting. See *Coverity Desktop Analysis 2020.12: User Guide* [↗](#) for details.

`--analyze-captured-source`

Selects for analysis all files previously captured. This option is an alternative to listing files for analysis on the `cov-run-desktop` command line.

`--analyze-scm-modified`

Specifies the translation units to be analyzed as those that have been modified locally, as referenced against your Source Code Management (SCM) system. When `--analyze-scm-modified` is passed, you must also pass the `--scm` option, or, preferably, set `"settings.scm.scm"` in `coverity.conf`. This option is an alternative to listing files for analysis on the `cov-run-desktop` command line.

`--analyze-untracked-files <boolean>`

When true, files reported as untracked by the SCM will be analyzed. This option is false by default.

`--auth-key-file <token>`

Specifies the file name of an authentication key that has previously been retrieved. This option has no effect in disconnected mode.

`--build`

Runs the build command specified in the `coverity.conf` file under `cov-build`. This is necessary so the Coverity tools know how to compile all of the source files in your project. If you add new source files or change how they are compiled, you need to re-run `cov-run-desktop --build`. This command will automatically configure compilers (`cov-run-desktop --configure`) if they have not been configured yet.

`--category-regex <regex>`

Filters the list of returned issues to include only those whose `checkerProperties` [↗](#) is not null, and where `checkerProperties.category` matches the specified regular expression.

`--certs <filename>`

In addition to CA certificates obtained from other trust stores, use the CA certificates in the given `<filename>`. For information on the new SSL certificate management functionality, please see *Coverity Platform 2020.12 User and Administrator Guide* [↗](#)

`--checker-regex <regex>`

Filters the list of returned issues to include only those whose `checkerName` [↗](#) matches the specified regular expression.

`--cid <rangeFilter>`

Filters the list of returned issues to include only those whose `stateOnServer` [↗](#) is not null, and whose `stateOnServer.cid` is not null and matches the range filter. The range filter should match one of the following formats:

`<int>`

A single CID matching the value in `<int>`.

`<int> "-"`

A range of CIDs including `<int>` and all greater values.

`"-" <int>`

A range of CIDs including `<int>` and all lower values.

`<int1> "-" <int2>`

A range of CIDs including `<int1>`, `<int2>`, and all CID values in between.

`--clean`

Runs the clean command specified in the `coverity.conf` file.

`--code-base-dir <code_base_dir>`

Specifies the value of the "code\_base\_dir" variable. This is the directory that contains the `coverity.conf` file, and will generally be your SCM project root directory.

By default, `cov-run-desktop` searches upward in the file tree from where it is invoked to find a `coverity.conf` file. If one is found, then the directory containing that file is the `code_base_dir`. Otherwise, the invocation directory is the `code_base_dir`.

`--component-not-regex <regex>`

Filters the list of returned issues to include only those whose `stateOnServer` [↗](#) is null or for those which have no component names that match the specified regular expression.

`--component-regex <regex>`

Filters the list of returned issues to include only those whose `stateOnServer` [↗](#) is null or for which at least one component name matches the specified regular expression.

`--config <cov_config_file>`

The name of the file where the compiler configuration information will be stored. By default, this is the value of the `compiler_config_file` in `coverity.conf`.

**--configure**

When this option is specified, `cov-run-desktop` invokes `cov-configure` on each of the `CompilerConfiguration` elements in the `coverity.conf` file.

**--confine-to-scope <boolean>**

Filters the list of returned issues to include only those whose `mainEventFilePathname` [↗](#) is one of the "analysis scope" files. This means that any defects found in files outside of the analysis scope will not be returned. When false, no such filtering is done.

This option is true by default.

 **Note**

For information on how analysis scope is defined, see the *Coverity Desktop Analysis 2020.12: User Guide* [↗](#).

**--connect-timeout <timeout>**

Allows users to change (in seconds) the connection timeout to the given duration. The default connection timeout is set to 60 seconds.

**--cov-user-dir <cov\_user\_dir>**

Specifies the value of the "cov\_user\_dir" variable. This corresponds to a directory where user-specific and application-specific settings are stored. By default, this is "%APPDATA%/Coverity" on Windows and "\$HOME/.coverity" on unix.

**--create-auth-key**

Creates an authentication key file and writes it to the `auth_key_file` location specified in `coverity.conf`.

**--custom-triage-attribute-not-regex <attrName> <regex>**

Filters the list of returned issues to include only those whose `stateOnServer` [↗](#) is null, or whose `CustomTriage` [↗](#) does not contain a key equal to the specified attribute name, or that does contain such a key but with a corresponding value that does not match the specified regular expression.

For example, `--custom-triage-attribute-not-regex "customAttr" "customVal"` will return only those issues that *do not* have a custom attribute named "customAttr" with a value containing "customVal" as a substring.

**--custom-triage-attribute-regex <attrName> <regex>**

Filters the list of returned issues to include only those whose `stateOnServer` [↗](#) is null, or whose `CustomTriage` [↗](#) contains a key exactly equal to the specified attribute name and for which the corresponding value matches the specified regular expression.

For example, `--custom-triage-attribute-regex "customAttr" "customVal"` will return only those issues that have a custom attribute named "customAttr" with a value containing "customVal" as a substring.

**--data-dir**

Specifies the directory in which `cov-run-desktop` stores its data. The default value is `$(code_base_dir)/data-coverity`. The value is stored in a variable, `$(data_dir)`.

NOTE: You must use `coverity.conf` 6 or above to use the `data_dir` option.

`--dir`

Specifies the intermediate directory. Required if no `coverity.conf` file is present.



**Note**

Note that Desktop Analysis can only be run on an intermediate directory created on the same machine, and in the same source code directory, as the analysis will take place (i.e. the build and analysis processes must take place on the same machine and directory, and the intermediate directory must not be moved).

`--disable-misra`

Ignores any MISRA analysis configuration when `cov-run-desktop` uses analysis settings from the chosen reference snapshot. This is useful when the reference snapshot includes unwanted or unnecessary MISRA analysis results, but you still want to use the same settings for local analysis.

`--disconnected`

When specified, `cov-run-desktop` operates in "disconnected" mode. Related options will be accepted but ignored while disconnected.

`--enable-audit-mode`

Enables audit-mode analysis, which is intended to expose more potential security vulnerabilities by considering additional potential data sources that could be used in an exploit.

Using this option usually reports more defects that are less likely to represent true vulnerabilities. Audit mode analysis will take noticeably longer to complete: It analyzes all functions that are present in the source, not just those that are present in the call tree. This level of testing can be useful for auditors and for any users who want to see the maximum number of defects.

The `--enable-audit-mode` option has the following effects:

- It enables additional audit-mode checkers that normally are off by default: for example, `SQL_NOT_CONSTANT` and `INSECURE_COOKIE`.

For a list of all Audit Mode checkers, refer to the "Checker Enablement and Option Defaults by Language" chapter in the HTML version of the *Coverity Checker Reference*.

- It sets the `--webapp-security-aggressiveness-level` to `high`.
- It sets `--distrust-all`.
- For tainted dataflow security checkers, it introduces additional audit-mode sources of untrusted (tainted) data, in order to model potential attacks. Such sources include all function parameters, and the return value from external functions (those that are not visible in the source code or bytecode, and for which no model exists).

`--exit1-if-defects <boolean>`

When true, `cov-run-desktop` exits with code 1 when defects are present in the analysis, as long as there are no errors present that cause a higher exit code. This option is false by default.

**--extend-checker <executable>**

This option will cause the specified `executable` to be invoked as an additional checker for Desktop Analysis. The executable can be either an absolute or relative path.

This option can be specified more than once to enable multiple Extend checkers.

**--extend-checker-option <executable> <checker\_name>:<option>[:<option\_value>]**

Passes a checker option for an Extend checker, which must be specified with `--extend-checker`. The specified `executable` must exactly match the `executable` passed to `--extend-checker`.

Example:

```
> cov-run-desktop [options] --extend-checker MY_CHECKER.exe --extend-checker-option MY_CHECKER.exe MY_CHECKER:option_a:true
```

**--file-regex <regex>**

Filters the list of returned issues to include only those whose `mainEventFilePathname` [🔗](#) matches the specified regular expression.

**--first-detected-after <date>**

Filters the list of returned issues to include only those that were first detected after the specified date. The value of `<date>` must follow one of the following formats:

- `YYYY-MM-DD`: Specifies a year (YYYY), month (MM), and day (DD). Note that midnight in the local time zone (that is, `T00:00<local>`) is implicit.
- `YYYY-MM-DD[ T]hh:mm(:ss)`: Specifies a time of day along with the date. The time format accepts hours (hh), minutes (mm), and seconds (ss). Note that the local time zone is implicit.
- `YYYY-MM-DD[ T]hh:mm(:ss)Z`: Specifies the Greenwich Mean Time (GMT) zone for the specified time and date. Here, `Z` refers to "Zulu", which signifies GMT.
- `YYYY-MM-DD[ T]hh:mm(:ss)[+-]hh:mm`: Specifies the date along with an offset (`[+-]`) to the specified local time of day.

**--first-detected-before <date>**

Filters the list of returned issues to include only those that were first detected before the specified date. For the date format, see the `--first-detected-after` option.

**--function-regex <regex>**

Filters the list of returned issues to include only those whose `functionDisplayName` [🔗](#) matches the specified regular expression.

**--host <hostname>**

Specifies the DNS hostname or IP address of the machine where Coverity Connect is running. This option is required, or must be specified in `coverity.conf`, unless operating in disconnected mode. If disconnected, this option has no effect.

**--ignore-deviated-findings**

Set this option to prevent reporting defects that are deviated with annotations.

Any defects or false positives annotated using the `#pragma Coverity` compliance directive will be suppressed and will not be reported by Coverity Connect. *All* recorded deviations in the current project version are then written to a CSV file. For more information see the *Coverity Checker Reference*.

`--ignore-modified-file-regex <regex>`

When specified, any file whose name matches `<regex>` will be treated as not modified. This option may only be specified once, but `<regex>` may use the `|` operator to ignore multiple files. See Translation unit selection for more information.

`--ignore-modified-non-psf <boolean>`

When specified as `true`, a translation unit will be considered modified only if its primary source file is modified. This option is `false` by default.

`--ignore-uncapturable-inputs <boolean>`

When `true`, allows analysis to proceed even if Coverity is unable to process some input files for analysis. Specifically, if some file has not been in a captured a compilation and no automatic way of capturing it is configured (for example, filesystem capture), the file will be ignored if this option is `true`. If `false`, such input files will cause an error. Although this option can be used as a crude alternative to the "modified file" regex options, it is not recommended for that purpose. Although not recommended, this functionality can be made more permanent through a `coverity.conf` setting. See *Coverity Desktop Analysis 2020.12: User Guide* [🔗](#) for details.

`--ignore-untracked-file-regex <regex>`

Untracked files are ignored if they match the `<regex>`.

`--impact-regex <regex>`

Filters the list of returned issues to include only those whose `checkerProperties` [🔗](#) is not null, and where `checkerProperties.impact` matches the specified regular expression.

`--include-missing-locally <boolean>`

When this option is set to `true`, the list of returned issues includes issues that are missing in the local analysis but present in the reference snapshot; that is, their local status is "missing". Defaults to `--include-missing-locally false`.

`--json-output-v7 <filename>`

When present, `cov-run-desktop`'s output is written to the specified file in JSON output [🔗](#). You can include either an absolute path or a path relative to the location in which you execute the command. If you want the file name to end in `.json`, you must include it in the `<filename>`.

`--json-output-v7` is the recommended JSON output option, as it contains the most complete set of information. Earlier versions, v1-v6, are supported for backward compatibility.

`--kind-regex <regex>`

Filters the list of returned issues to include only those whose `checkerProperties` [🔗](#) is not null, and where the `issueKinds` value matches the specified regular expression.

Matching is done using a single string, which is a comma-separated concatenation of all the `issueKinds`, in alphabetical order. For example, `"QUALITY, SECURITY"`.

`--lang <language>`

Write event messages in the specified language. Currently, the supported values are `en` (for English), `ja` (for Japanese), and `zh-cn` (for Simplified Chinese). The default language is English (`en`).

`--lang-regex <language>`

This option specifies a regex and matches it against the defect's language. These languages include the following: C, C++, C#, CUDA, Fortran, Java, JavaScript, Objective-C, Objective-C++, PHP, Python, Ruby, Scala, Swift, Text, and VisualBasic.

`--local-status-not-regex <local status>`

This option can be used only when `--include-missing-locally` is set to `true`. Filters the list of returned issues to exclude those issues whose local status matches the specified regular expression. The valid values for the local status are as follows:

`local`

The issue's CID is present in a recent run, but not present on the server.

`missing`

The issue's CID is present on the server, but not in a recent run.

`present`

The issue's CID is present in both a recent run and on the server.

Here is an example of using `--local-status-not-regex`:

```
cov-run-desktop --include-missing-locally true --local-status-not-regex present
<analysis options>
```

`local-status-regex <local status>`

This option can be used only when `--include-missing-locally` is set to `true`. Filters the list of returned issues to include only those issues whose local status matches the specified regular expression. The valid values for the local status are as follows:

`local`

The issue's CID is present in a recent run, but not present on the server.

`missing`

The issue's CID is present on the server, but not in a recent run.

`present`

The issue's CID is present in both a recent run and on the server.

Here is an example of using `--local-status-regex`:

```
cov-run-desktop --include-missing-locally true --local-status-regex missing
<analysis options>
```

- 
- `--mark-fp <cid> <explanation>`  
Sets the Classification of the specified CID to "False Positive". The value of `<cid>` is the defect's CID, and `<explanation>` is a string which explains why this defect is a False Positive.
- `--mark-int <cid> <explanation>`  
Sets the Classification of the specified CID to "Intentional". The value of `<cid>` is the defect's CID, and `<explanation>` is a string which explains why this defect is Intentional.
- `--merge-key-regex <regex>`  
Filters the list of returned issues to include only those whose `mergeKey` [↗](#) matches the specified regular expression.
- `--MISRA-category-regex <regex>`  
Filters defects that have a MISRA category which matches the specified `<regex>`. Possible categories are Advisory, Required, and Mandatory.
- `--modification-date-threshold <date_time>`  
Specifies the modification date threshold to use instead of the default. Only those files modified on or after the specified date will be included in the analysis. The value of `<date_time>` should match the `YYYY-MM-DD[ T]hh:mm(:ss)` format, where date and time are separated by a space or `T`. The time, specified by `hh:mm:ss`, is optional, and seconds (`:ss`) are not required. If time is not specified, the default value is midnight (00:00) of the specified date.
- `--no-default-triage-filters`  
By default, `cov-run-desktop` has three active issue filters, which have the effect of suppressing issues triaged as uninteresting, or in a component named to hold third-party code:
- `--component-not-regex "[Tt]hird.*[Pp]arty"`
  - `--triage-attribute-not-regex "classification" \`  
`"False Positive|Intentional|No Test Needed|Tested Elsewhere"`
  - `--triage-attribute-not-regex "action" "Ignore"`
- If `--no-default-triage-filters` is specified, then all three of these filters are deactivated. If `--component-regex` or `--component-not-regex` is specified, then the first filter is deactivated. If `--triage-attribute-regex` or `--triage-attribute-not-regex` is specified for "classification" or "action", then the respective filter for that attribute is deactivated.
- `--no-text-output`  
When present, `cov-run-desktop` does not print the compiler-like textual output.
- `--occurrences <range>`  
When there are multiple occurrences for a given defect, each instance is given an occurrence number, `O`, from 1 to `N`. The `--occurrences <range>` option, specifies the valid values for `O`.
- Ranges are as described for the `--cid` option.
- `--on-new-cert <trust|distrust>`  
Indicates whether to trust (with `trust-first-time`) self-signed certificates, presented by the server, that haven't been seen before. For information on the new SSL certificate management functionality, please see *Coverity Platform 2020.12 User and Administrator Guide* [↗](#)
-

`--ownerLdapServerName-regex <regex>`

Filters the list of returned issues to include only those whose `stateOnServer`  is null, or whose `ownerLdapServerName` matches the specified regular expression.

`--password <password>`

Specifies the password for connecting to the Coverity Connect server.

`--port <port_number>`

Specifies the HTTP or HTTPS port of the Coverity Connect server. The default value is 8080. If `--ssl` is present, the default value is 8443. This option has no effect in disconnected mode.

`--present-in-reference <boolean>`

Filters the list of returned issues to include only those whose `stateOnServer`  is null, or whose value for `presentInReferenceSnapshot` is equal to the specified boolean.

`--print-path-events <boolean>`

When true, path events are printed in the text output. When false, path events are not printed. This option is true by default.

`--reference-snapshot <specification>`

Specifies how `cov-run-desktop` should select a reference snapshot from the selected stream. The `<specification>` must be one of the following values:

`id:<ID>`

Use the snapshot with the matching `<ID>`. `--reference-snapshot` will return an error if the ID is invalid or if it is not in the selected stream.

`date:<date_time>`

Use the snapshot that was created closest to, but not after, the specified `<date_time>`. The value of `<date_time>` should match the `YYYY-MM-DD[ T]hh:mm(:ss)` format, where date and time are separated by a space or `T`. The time, specified by `hh:mm:ss`, is optional, and seconds (`:ss`) are not required.

`--reference-snapshot` will return an error if there is no snapshot with summary data that was created before the specified `<date_time>`.

`latest`

Use the snapshot with the latest code-version date (that also contains summary data) in the specified stream.

 **Note**

This is not necessarily the most recently committed snapshot, since it is possible to commit a snapshot with an arbitrary code version date.

`idir-date`

Use the snapshot created closest to, but not after, the creation date of the intermediate directory.

This is the default option.

**scm**

This option will query the SCM to determine the version that was most recently checked out or updated, and then use the closest snapshot.

The `--scm` option, or the `"settings.scm.scm"` attribute in `coverity.conf`, is required when using this specification.

**--relative-paths <boolean>**

When true, compiler-like output paths are printed as relative paths, relative to the directory specified by `--relative-to`. If `--relative-to` is not specified, the current working directory is used.

**--relative-to <path>**

When `--relative-paths` is true, compiler-like output paths are printed relative to the specified `<path>`. If not specified, the current working directory is used.

**@@<response\_file>**

Specify a response file that contains a list of additional command line arguments, such as a list of files for analysis. Each line in the file is treated as one argument, regardless of spaces, quotes, etc. The file is read using the platform default character encoding. Using a response file is recommended when the list of input XML files is long or automatically generated.

Optionally, you can choose a different encoding, by specifying it after the first "@". For example:

```
cov-run-desktop [OPTIONS] @UTF-16@my_response_file.txt
```

You must use a supported Coverity encoding, listed under the `cov-build --encoding` option.

**--restrict-modified-file-regex <regex>**

When specified, only files whose name matches `<regex>` (and whose timestamp satisfies the modification date threshold) will be included in the analysis. This option may only be specified once, but `<regex>` may use the "|" operator to include multiple files.

**Note**

`--ignore-modified-file-regex` takes precedence if used in tandem with `--restrict-modified-file-regex`.

**--restrict-untracked-file-regex <regex>**

Only untracked files that match the `<regex>`, and do not match `--ignore-untracked-file-regex`, will be analyzed.

**--scm <scm\_type>**

Specifies the name of the source control management system. For this option to function correctly, your source files must remain in their usual locations in the checked-out source tree. If the files are copied to a different location after checkout, the SCM query will not work.

Possible `scm_type` values:

- Accurev: `accurev`
- Azure DevOps Server (ADS): `ads`

Windows only.

- ClearCase: `clearcase`
- CVS: `cvs`
- GIT: `git`
- Mercurial: `hg`
- Perforce: `perforce`
- Plastic: `plastic|plastic-distributed`.

Use `plastic` when working in a non- or partially-distributed Plastic configuration. Use `plastic-distributed` when working in a fully-distributed Plastic configuration.

- SVN: `svn`
- Team Foundation Server (TFS): `tfs`

Windows only.

For usage information for the `--scm` option, see `cov-extract-scm`.

 **Note**

The following commands or setup utilities must be run before `cov-run-desktop` in order to successfully communicate with the SCM server:

- `accurev`:

Login command

- `perforce`

The environment variable `P4PORT` should be set to the value expected by the p4 tool.

- `tfs` or `ads`:

Windows credentials in Credential Manager to access the TFS or ADS server

`--scm-param`

Specify extra arguments to be passed to the SCM tool in a context-aware manner. For usage information of the `--scm` option, see `cov-extract-scm`.

`--scm-project-root <scm_root_path>`

Specifies a path that represents the root of the source control repository. Use this option when `cov-run-desktop` is being run from a directory other than the root of the source control repository. All paths returned by `--get-modified-files` will be relative to this path.

This option behaves the same as `cov-extract-scm --scm-project-root`. See `cov-extract-scm` for additional details.

`--scm-tool <scm_tool_path>`

Specifies the path to an executable that interacts with the source control repository. If the executable name is given, it is assumed that it can be found in the path environment variable. If not provided, the command uses the default tool for the specified `--scm` system.

This option behaves the same as `cov-extract-scm --scm-tool`. See `cov-extract-scm` for additional details.

`--scm-tool-arg <scm_root_path>`

This option has been deprecated. Instead of using `--scm-tool-arg arg1`, use `--scm-param tool_arg=arg1`. Specifies additional arguments that are passed to the SCM tool, specified in the `--scm-tool` option, that gathers the last modified dates. The arguments are placed before the command and after the tool. This option can be specified multiple times.

This option behaves the same as `cov-extract-scm --scm-tool-arg`. See `cov-extract-scm` for additional details.

`--set-new-defect-owner <boolean>`

When true, and in connected mode, sets the owner for newly detected defects that exist locally as the current user. True by default.

See also, `--set-new-defect-owner-limit`.

`--set-new-defect-owner-limit <limit>`

Set the limit on the number of defects to assign to the current user. If the number of discovered defects is more than the limit, then skip the assignment. The default limit is 100.

 **Note**

`--set-new-defect-owner` and `--set-new-defect-owner-limit` have no effect on the following platforms:

- FreeBSD
- Itanium
- NetBSD

`--set-new-defect-owner-to <user>`

When used with `--set-new-defect-owner`, this specifies the user to whom any new defects will be assigned. The default is the current user.

Note that the specified `<user>` must already exist in the Coverity Connect database.

`--setup`

This option is intended as a single step to get a new user ready for Desktop Analysis. It creates an authentication key (if one has not already been created), runs the clean command, if applicable, and then captures a full build, if applicable. No build is captured if the `--skip-build` option is used with

`--setup` or if the configured build command in `coverity.conf` is empty. The clean command, if configured, is only executed if a build is to be captured. Since it is common to capture a build, it is an error to leave the build command unspecified and run `--setup` without `--skip-build`.

`--skip-build`

Used only with `--setup` to omit capturing a build, even if one is specified in `coverity.conf`. This can be useful if your project contains compiled and interpreted code, but you intend only to analyze interpreted (filesystem capture) code. This functionality can be made more permanent by specifying an empty build command in `coverity.conf`. See *Coverity Desktop Analysis 2020.12: User Guide* [🔗](#) for usage.

`--sort <sort_spec>`

Specifies the sort order for text output. The `<sort-spec>` accepts the values listed below. To sort on more than one attribute, you can use a non-empty, comma-separated list of values. Additionally, to specify ascending or descending sort order for any attribute, you can add `:a` or `:d` (respectively) directly after the attribute name. All attributes, except `cid`, are ordered in ascending order by default.

The available sort attributes are:

- `cid`: ID number assigned by Coverity Connect to each issue.
- `occurrence`: The number of the occurrence among all those in the output set that have the same merge key.
- `occurrences`: The total number of issue occurrences.
- `mergeKey`: An internal identifier used to assign CIDs to issues. This is mainly useful when the CID is missing, either because `cov-run-desktop` is in disconnected mode or because the Coverity Connect server is a subscriber that is disconnected from its coordinator.
- `checker`: The name of the checker that found this issue.
- `file`: The complete path to the file that contains the issue.
- `line`: The file's line number where the issue is located.
- `function`: The name of the function that contains the issue.
- `impact`: The issue's impact, as determined by Coverity Connect: High, Medium, or Low
- `category`: Description of the nature of the software issue.
- `subcategory`: The sub-category of the defect reported by `<checker>`.
- `present`: True if the issue is present in the specified snapshot, otherwise false.
- `ownerLdapServerName`: The LDAP server of the defect owner.
- `component`: The name of the component that contains this issue.
- `firstDetected`: The date and time in which the issue was first detected by the analysis.

- `classification`: The value of the issue's classification attribute.
- `action`: The specified action to be taken on the issue.
- `fixTarget`: Target milestone for fixing an issue.
- `severity`: The value of the issue's severity attribute.
- `legacy`: True if the issue is marked as a `legacy` issue, otherwise false.
- `MISRACategory`: MISRA categories are sorted in order from least to most stringent: Advisory, Required, Mandatory. If this option is not specified, MISRA defects are not sorted by category.
- `owner`: The user assigned to the issue.
- `externalReference`: An internal identifier used by your company to track the issue.
- `customTriage[<attribute>]`: The value of any custom triage attributes. Within the bracket-enclosed `<attribute>` name, use two consecutive right brackets ( `]` ) to encode a single right bracket ( `]` ).

For example, `--sort file,classification:d` will order the results first by ascending file name, then descending Classification.

`--ssl`

When present, use SSL encryption for all communication with Coverity Connect. This option has no effect in disconnected mode.

`--stream <stream_name>`

Specifies the Coverity Connect stream which contains the relevant snapshot and triage information. This option has no effect in disconnected mode.

`--subcategory-regex <regex>`

Filters the list of returned issues to include only those whose `checkerProperties` [is not null](#), and where `checkerProperties.subcategoryShortDescription` matches the specified regular expression.

`--text-output <filename>`

Write the text output to the specified file rather than writing it to the console.

`--text-output-style <style>`

Specifies the style of the text output. There are two accepted styles:

- `oneline` - Each occurrence and event is written as a single line of output. This format works best with vi type editors.
- `multiline` - Each occurrence and event is split across multiple lines. This format works best with Emacs editors.
- `msvs` - Similar to `multiline`, but prints locations as `<file(line)>` instead of `<file:line>`; for use with Visual Studio.

If unspecified, the `multiline` style is used by default.

`--triage-attribute-not-regex <attrName> <regex>`

Filters the list of returned issues to include only those whose `stateOnServer` is null, or whose `Triage` does not contain a key equal to the specified attribute name, or that does contain such a key but with a corresponding value that does not match the specified regular expression.

For example, `--triage-attribute-not-regex "severity" "Minor"` will return only those issues that *do not* have a "severity" attribute with a value containing "Minor" as a substring.

`--triage-attribute-regex <attrName> <regex>`

Filters the list of returned issues to include only those whose `stateOnServer` is null, or whose `Triage` contains a key exactly equal to the specified attribute name and for which the corresponding value matches the specified regular expression.

For example, `--triage-attribute-regex "classification" "Bug"` will return only those issues that have a "classification" attribute with the value, "Bug" (or containing the substring "Bug").

`--tu-pattern <pattern>`

When specified, only those translation units which match `<pattern>` will be present in the analysis.

`--upgrade <version>`

Launches an assisted upgrade of Coverity Analysis to the version specified. The value of `<version>` should be the desired Coverity Analysis version number (2020.12 for example). This will download the required installer from Coverity Connect, run it, and provide instructions to invoke the new version. The existing installation with *not* be overwritten.

`--url <path>`

Allows you to connect to a CIM instance that has a context path in its HTTP(S) URL. You can use this option instead of the `--host`, or `--port` options. The `--url` option is provided to accommodate the use of a context path and to deal with setting up Coverity Connect behind a reverse proxy.

Use HTTPS or HTTP to connect to Coverity Connect HTTPS or HTTP port. For `http`, the default port is 80; for `https`, the default port is 443. For example:

```
https://example.com/coverity
```

```
https://cimpop:8008
```

```
http://cim.example.com:8080
```

 **Note**

You may not use the `commit://` scheme in the URL.

`--use-reference-settings <boolean>`

If true, `cov-run-desktop` will use the analysis settings downloaded from the reference snapshot. True by default.

`--user <user_name>`

Specifies the Coverity Connect user name. If unspecified, the default is the value for the environment variable, "COV\_USER", "USER", or "USERNAME", if specified. If none of these is specified, and

"settings.server.username" is not specified in the `coverity.conf` file, then the `--user` option is required.

This option has no effect in disconnected mode.

#### `--whole-program`

Some checkers, such as Application Security checkers and `IDENTIFIER_TYPO`, are only effective when analyzing all source files in the program. By default, these "whole-program" checkers are not available to `cov-run-desktop`.

Specifying `--whole-program` allows `cov-run-desktop` to run whole-program checkers.

### Example 1

```
> cov-run-desktop file1.c file2.c
```

This will analyze `file1.c` and `file2.c`, assuming a compilation of those files has previously been captured using `cov-build`.

### Example 2

```
> cov-run-desktop file1.js
```

This will analyze `file1.js`

### Example 3

```
> cov-run-desktop --analyze-scm-modified
```

This will query your SCM to find out which files have been modified locally and analyze those.

### Example 3

```
> cov-run-desktop --dir idir --host my_server --port 8080 \
 --auth-key-file keyfile --stream my_stream \
 --triage-attribute-regex Owner user1 \
 --text-output-style oneline \
 file1.c file2.c
```

This example analyzes both `file1.c` and `file2.c`, obtains a reference snapshot from the "my\_stream" stream on `my_server:8080`, authenticates using "keyfile" (which must have been previously created using `cov-run-desktop`), filters the results for those assigned to `user1`, and writes the defects to the console in the `oneline` format, which uses one line of text for each defect and event. Notice that many of the options specified here could instead have been put into a `coverity.conf` file for convenience.

### Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.

- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects`(*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

### **See Also**

`cov-build`

`cov-configure`

`cov-manage-im`

`cov-manage-emit`

---

## Name

`cov-run-fortran` Run the Coverity Fortran Syntax Analysis tool on user's Fortran code.

## Synopsis

```
cov-run-fortran --dir <intermediate_directory> [CONTROL-OPTIONS] [--] [ANALYSIS-OPTIONS-AND-FILES]
```

## Description

The `cov-run-fortran` command runs the Coverity Fortran Syntax Analysis tool on user's code and puts the output into a form compatible with Coverity Connect. The `cov-commit-defects` command can then be used to upload the results to Coverity Connect. The options described in this command reference are of two kinds: Control and Analysis. Control options are interpreted by the `cov-run-fortran` command and used to control Coverity Fortran Syntax Analysis and the subsequent translation. Analysis options are passed through to Coverity Fortran Syntax Analysis verbatim. This command reference provides an overview of the analysis options. Refer to the Coverity Fortran Syntax Analysis User Guide [for additional details](#).

Coverity Fortran Syntax Analysis is a static analysis tool designed to give detailed feedback on syntax and usage that is not compatible with the selected compiler and language level. Coverity Fortran Syntax Analysis supports many current and legacy configurations through pre-written compiler configuration files. It is especially useful for porting and regularization efforts, since compiler incompatibilities can be discovered by selecting the target compiler's configuration file.

The control options simplify the selection of a configuration file, and specify the directories where the intermediate and output files are to be stored. The analysis options indicate which source and library files to process and control other features of the Coverity Fortran Syntax Analysis tool.

The main output of `cov-run-fortran` is the file `FC.errors.xml`, which is written into the `output` section of the intermediate directory specified by the `--dir` option. The `emit-db` is updated with filename and error summary information.

The exit code of `cov-run-fortran` is 0 when the analysis completes successfully and 2 or greater if there is an error.

## Fortran Inputs

Coverity Fortran Syntax Analysis analyzes the specified Fortran source files in two phases, a local phase and a global phase. During the local phase, it analyzes each file separately and emits defects as it finds them. It also collects symbol information to be used during the global analysis phase. During the global phase, it uses the stored information to report on issues that span program unit boundaries.

Coverity Fortran Syntax Analysis respects C-style preprocessor (i.e. `cpp`) directives and `INCLUDE` statements to determine when source code is being imported. Coverity Fortran Syntax Analysis emulates the target compiler by importing those files in the expected manner. Coverity Fortran Syntax Analysis resolves paths relative to the current working directory; thus, the environment in which `cov-run-fortran` is run should mimic the actual compilation environment as closely as possible.

Coverity Fortran Syntax Analysis also respects `USE` statements appearing in the code and uses these to import symbols and interfaces from the modules so named. Modules must be analyzed in a bottom-up order, so that types, symbols and interfaces imported from a given module can be checked for consistency at their point of use. Coverity Fortran Syntax Analysis automatically calculates and uses a bottom-up processing order. However, it is still necessary to list in the `cov-run-fortran` command the source files and/or library files containing all of the modules used.

Coverity Fortran Syntax Analysis library files provide a summary of the interface information extracted from module and non-module source files. They perform the dual purpose of fulfilling module references and providing information about intrinsic or library functions provided by the compilation environment (compiler and OS). Multiple library files can be referenced in one `cov-run-fortran` invocation. See the Coverity Fortran Syntax Analysis User Guide [🔗](#) for more information on creating and using libraries.

As with other Coverity analysis tools, `cov-run-fortran` will accept some or all of its input from response files. Command line arguments of the form `@@<response_file>` cause the command interpreter to read options from the named response file.

## Fortran Outputs

In addition to the error XML file produced by `cov-run-fortran`, Coverity Fortran Syntax Analysis produces a number of outputs that may be useful. These include a module dependency file, an annotated listing with defect annotations shown in-line, and a function/procedure cross-reference (i.a. a callgraph). See the Coverity Fortran Syntax Analysis User Guide [🔗](#) for detailed information on these additional outputs.

## Options usage

As shown in the command synopsis, `cov-run-fortran` control options and analysis options separated by a `--` argument. Control options govern the operation of `cov-run-fortran` as a whole, while analysis options govern only the analysis. Analysis options include the names of the source files on which Coverity Fortran Syntax Analysis operates as well as library file names and supplementary output file names.

Analysis options may act globally or locally. When listed before any source file name, the effect of the option is global. It acts upon all of the listed source files unless explicitly overridden. When specified within the source file list, the effect of an option is local. It acts only upon the immediately following source file name. Analysis options can be negated by prefixing the option name with `-n` or `-no-` option syntax.

## Options categories

The options accepted by `cov-run-fortran` fall into several categories as summarized below.



### Note

Each of the items below is linked to other sections in this document. Many link to the relevant definition in this section, while common analysis options link to corresponding entries in the `cov-analyze` section.



### Note

Analysis options must follow all control options and are separated from control options by `--`.

## Control options that select the compiler configuration

Configuration options are used to select the configuration file that controls compiler emulation. The `--configuration` and `--config-path` options can be used to select a configuration file by name, including a user-supplied configuration file. The remaining configuration options can be used to select from the matrix of pre-written configuration files.

Configurations are filtered in the following order: `--platform`; `--vendor`; `--version`; `--level`. Very few configuration files provide platform information, so it does not make a good selector. In most cases, it is sufficient to select the desired compiler emulation using just the `--vendor` and `--version` options. Where necessary, the `--platform` and `--level` options can be used to refine this initial selection.

- `--configuration`
- `--config-path`
- `--level`
- `--list-configs`
- `--platform`
- `--vendor`
- `--version`

## Control options used in other Coverity analysis tools

- `--append`
- `--dir`
- `--strip-path`
- `--security-file`

## Other control options

- `--impact`

## Analysis options that affect the analysis of individual program units

The following analysis options affect the kinds of defects that are reported:

 **Note**

Analysis options must be separated from control options by a double-dash (`--`).

- `-acqintf`
- `-allc`

- -cntl
- -cond
- -cpp
- -declare
- -dp
- -externals
- -f03
- -f08
- -f18
- -f77
- -f90
- -f95
- -ff
- -i2
- -i4
- -i8
- -intent
- -intrinsic
- -obsolescent
- -r8
- -relax
- -save
- -specific
- -standard

### **Analysis options that affect the analysis of whole programs**

The following analysis options affect the kinds of defects that are reported:

 **Note**

Analysis options must be separated from control options by a double-dash (--).

- -ancmpl
- -anprg
- -anref

### **Analysis options that affect additional outputs**

The following analysis options do not affect any of the defects that are reported:

 **Note**

Analysis options must be separated from control options by a double-dash (--).

- -l
- -plen
- -pwid
- -refstruct
- -moddep
- -shinc
- -shsub
- -shsrc
- -shsngl
- -shprg
- -shref
- -shcom
- -shmodtyp
- -shmodvar
- -shmoddep

### **Analysis options used for library usage and maintenance**

The following analysis options are used when generating or maintaining libraries:

- -create

- `-include`
- `-library`
- `-update`

### Miscellaneous analysis options

The following options (except for `-idep`, `-log` and `-report`) affect the kinds of defects that are reported:

- `-define`
- `-I`
- `-idep`
- `-informative`
- `-log`
- `-report`
- `-rigorous`
- `-truncate`
- `-warnings`

### Options

#### `-acqintf`

(analysis, global and local) During the local analysis phase, use the interfaces of previously-analyzed subprograms to validate calls and function invocations whose interfaces are not explicitly provided. If negated, the actual argument lists of subprogram invocations will only be verified during the global analysis phase.

This option should be specified when analyzing an unrelated set of program units, and when interfaces have been updated but the a library file presenting these interfaces has not yet been updated.

Default: `-nacqintf`

#### `-allc`

(analysis, global and local) Analyze all columns of each input record. If negated and the `-ff` option is not in effect, only columns 1 to 72 (after expansion of any tabs) will be analyzed.

Default: `-nallc`

#### `-ancmpl`

(analysis, global only) The complete program is analyzed. Unreferenced procedures, unreferenced and undefined common blocks, unreferenced and undefined common block objects, unreferenced

modules, unreferenced and undefined public module variables, unreferenced public module constants and unreferenced public module derived types are flagged. If the `-anref` and `-rigorous` options are also in effect, the call tree will be traversed to detect unsaved common blocks and modules with unsaved public data which are not saved in the root of referencing program units.

Default: `-nancmpl`

`-anprg`

(analysis, global only) Verify the consistency of the whole program.

Default: `-anprg`

`-anref`

(analysis, global only) Analyze the reference structure.

Default: `-anref`

`--append`

(control) Append defects from this analysis run to the defects from the last analysis run.

This option is intended for combining the Coverity Fortran Syntax Analysis output from different code sets into a single emit. No attempt is made to cull duplicate file or defect entries.

`-cntl <c>`

(analysis, global and local) Allow a maximum of `c` continuation lines in a statement. The value of `c` must be 999 or less.

Default: depends on the selected compiler emulation

`-cond`

(analysis, global and local) Process debug lines (lines with a `D` in the first column). If this option is negated, debug lines are treated as comments.

Default: `-ncond`

`--config-path`

(control) Specifies an alternate path where Coverity Fortran Syntax Analysis should look for its configuration files. By default, these are in `fortran/share/` relative to the `cov-analysis` installation directory.

`--configuration`

(control) The configuration option can be used to directly specify the name of the configuration file to be used by Coverity Fortran Syntax Analysis.

For a list of the available configurations, use `cov-run-fortran --list-configs`. For a list of supported compiler emulations and their corresponding configuration filenames, consult Appendix A.1 in the Coverity Fortran Syntax Analysis User Guide [🔗](#).

`-cpp`

(analysis, global and local) Interpret C-style preprocessor directives as if the Fortran sources are first run through the C preprocessor `cpp`.

Default: `-cpp` for files whose filename extension begins with `.F`; `-ncpp` otherwise.

**-create**

(analysis, global) Create a new library file. If more than one library file is specified, the library file to be created must be the first in the list.

Default: `-ncreate`

**-declare**

(analysis, global and local) Generate a warning for all variables that have not been explicitly declared in a type statement.

Default: `-ndeclare`

**-define <symbols>**

(analysis, global) Define metasymbols for conditional compilation. The list of `<symbols>` must be comma-, semicolon- or colon-separated.

Default: `-ndefine`

**--dir <intermediate\_directory>**

Path name to an intermediate directory that is used to store the results of the build and analysis.

**-dp**

(analysis, global and local) Map all real objects to double precision and all double precision objects to REAL(16). Map all complex objects to double complex and all double complex to COMPLEX(16).

Default: `-ndp`

**-externals**

(analysis, global and local) Flag referenced external procedures which have not been declared `external`.

Default: `-nexternals`

**-f77**

(analysis, global and local) Validate the syntax for conformance to the FORTRAN 77 standard. All nonstandard syntax will be flagged. Note that this option by itself does not enable FORTRAN 77 syntax validation. It is also necessary to select a configuration that supports FORTRAN 77 syntax. Both of these steps are taken if the configuration control options include the `--level` option.

Default: `-nf77`

**-f90**

(analysis, global and local) Validate the syntax for conformance to the Fortran 90 standard. All nonstandard syntax will be flagged. Note that this option by itself does not enable Fortran 90 syntax validation. It is also necessary to select a configuration that supports Fortran syntax. Both of these steps are taken if the configuration control options include the `--level` option.

Default: `-nf90`

-f95

(analysis, global and local) Validate the syntax for conformance to the Fortran 95 standard. All nonstandard syntax will be flagged. Note that this option by itself does not enable Fortran 95 syntax validation. It is also necessary to select a configuration that supports Fortran syntax. Both of these steps are taken if the configuration control options include the `--level` option.

Default: `-nf95`

-f03

(analysis, global and local) Validate the syntax for conformance to the Fortran 2003 standard. All nonstandard syntax will be flagged. Note that this option by itself does not enable Fortran 2003 syntax validation. It is also necessary to select a configuration that supports Fortran 2003 syntax. Both of these steps are taken if the configuration control options include the `--level` option.

Default: `-nf03`

-f08

(analysis, global and local) Validate the syntax for conformance to the Fortran 2008 standard. All nonstandard syntax will be flagged. Note that this option by itself does not enable Fortran 2008 syntax validation. It is also necessary to select a configuration that supports Fortran 2008 syntax. Both of these steps are taken if the configuration control options include the `--level` option.

Default: `-nf08`

-f18

(analysis, global and local) Validate the syntax for conformance to the Fortran 2018 standard. All nonstandard syntax will be flagged. Note that this option by itself does not enable Fortran 2018 syntax validation. It is also necessary to select a configuration that supports Fortran 2018 syntax. Both of these steps are taken if the configuration control options include the `--level` option.

Default: `-nf18`

-ff

(analysis, global and local) Specifies that source code is in the free source form. The exact interpretation depends on the compiler configuration and language level options selected.

For files with the filename extension `f90`, `f95`, `f03`, `f2003`, `f03`, `F2008`, `F90`, `F95`, `F03`, `F2003`, `F03`, or `F2008` the default is `-ff`. For all other files, the default is `-nff`.

-I <paths>

(analysis, global) Set directories of include files. Path names must be separated by commas or colons with no embedded spaces.

Default: `-nI`

-i2

(analysis, global and local) Default integers occupy 2 bytes.

-i4

(analysis, global and local) Default integers occupy 4 bytes.

- i8**  
(analysis, global and local) Default integers occupy 8 bytes.
- idep <d>**  
(analysis, global) Generate a file listing all referenced include files.  
  
Default: `-ndep`
- impact <impact>**  
(control) Selects the impact level of the issues formatted for input into Coverity Connect. Valid values are Audit, Low, Medium and High. A lower impact level selects all higher impact levels as well.  
Default: High
- include, -include -, -include <sub\_list>**  
(analysis, global) From the library file, include the subroutines named in <sub\_list> in the analysis. The <sub\_list> must be a comma-, semicolon- or colon-separated list that does not contain any spaces. If <sub\_list> is omitted, then all subroutines are included.  
  
Default: `-ninclude`
- informative**  
(analysis, global) Show informative messages.  
  
Default: `-ninformative`
- intent**  
(analysis, global and local) Flag parameters for which no INTENT attribute has been specified.  
  
Default: `-nintent`
- intrinsic**  
(analysis, global and local) Flag referenced intrinsic procedures which have not been declared intrinsic.  
  
Default: `-nintrinsic`
- l, -l <list-file>**  
(analysis, global only) Specified that a merged list file is desired and optionally supplies the list file name.  
  
In the first form, the file name is omitted. The listing is written to a file whose base name is the same as that of the first source file and has the filename extension `.lst`.  
  
In the second form, the file name is a single hyphen. The listing file is written to the standard output.  
  
In the third form, the listing filename is supplied. The listing is written to the specified file.
- level <level>**  
(control) Specifies the language level (standard) used to select a Coverity Fortran Syntax Analysis configuration file. Available values are: `£77 £90 £95 £03 £08 £18`

The `--level` option selects the minimum language level that must be supported by the selected compiler configuration. We assume that a compiler supporting a given language level also supports all prior levels. Thus, for example, a compiler supporting Fortran 95 will also be selected if `--level f90` is specified.

When this configuration control option is provided, the corresponding language level option (`-f77`, `-f90`, `-f95`, etc.) is also supplied to Coverity Fortran Syntax Analysis.

`-library <filename>`

(global) The filename specified is a \FCK\ library file.

Default: `-nlibrary`

`-log`

(global) Show defines and undefines of metavariables.

Default: `-nolog`

`--list-configs`

(control) The `--list-configs` option reads all of the available configurations from the Coverity Fortran Syntax Analysis installation and prints out summary information in tabular form. Each entry contains the name of the configuration followed by a tuple containing its platform, vendor, version and language level. Any of these that are unspecified are omitted from the tuple.

`-moddep, -moddep <file>`

(analysis, global only) Generate a file containing the module dependency structure in XML format. If no filename is specified, `fckmd.xml` is used.

`-obsolescent`

(analysis, global and local) Flag all syntax elements marked as obsolescent in the Fortran standard that is in effect.

Default: `-nobsolescent`

`--platform <platform>`

(control) Specifies the target architecture or operating system used to select a Coverity Fortran Syntax Analysis configuration file. This option can be used to narrow the configuration selection when a compiler has different features that depend on the platform. Usually, the `--vendor` and `--version` options are sufficient to select the desired compiler emulation. Platform values include:  
**convex cray dec fujitsu hp9000 hpvms hp ibm rs6000 unisys vax vms**

`-plen <lines>`

(analysis, global only) Specifies the page length of the output listing. Default:62

`-pwid <columns>`

(analysis, global only) Specifies the page width of the output listing. Default:100

`-r8`

(analysis, global and local) Map all default reals to double precision. Map all default complex objects to double complex.

**-refstruct, -refstruct <file>**

(analysis, global only) Generate a file containing the reference structure of the program. The output is stored in XML format. If no filename is specified, `fckrs.xml` is used.

**-relax**

(analysis, global and local) Relax type checking on integers, logicals and Holleriths. No messages will be produced for type conflicts between logicals and integers, for the use of relational operators on logicals, and for the use of logical operators on integers. Hollerith (character) constants can be used in expressions and mixed with logicals, integers and reals.

Default: `-nrelax`

**-report <filename>**

(analysis, global) Generate a report file with the given name. The default filename extension is `.rpt`. If the filename is omitted, `fck.rpt` is used.

Default: `-nrpt`

**-rigorous**

(analysis, global) Flag less robust and less portable code at the expense of more informative messages. This option removes the limit on the number of messages displayed per line, so should be used with caution.

Default: `-nrigorous`

**-save**

(analysis, global and local) Assume that all variables are saved by default.

Default: `-nsave`

**--security-file <license file>, -sf <license file>**

Path to a valid Coverity Analysis license file. If not specified, this path is given by the `<security_file>` tag in the Coverity configuration or by `license.dat` (located in the Coverity Analysis `<install_dir>/bin` directory). A valid license file is required to run the analysis.

**-shcom, -shcom <com\_list>**

(analysis, global and local) In the listing file, show cross-references of common-block objects. The `com_list` names the common blocks to be displayed. If omitted, all common blocks are displayed.

Default: `-nshcom`

**-shinc**

(analysis, global and local) In the listing file, show included source as well.

Default: `-shinc`

**-shmoddep, -shmoddep <mod\_list>**

(analysis, global only) Show module dependencies. If the `mod_list` is supplied, dependencies of the named modules are shown. Otherwise, dependencies are shown for all modules.

Default: `-nshmoddep`

`-shmodtyp, -shmodtyp <mod_list>`  
(analysis, global only) Show cross-reference listings of public module derived types. If the `mod_list` is supplied, cross-references for the named modules are shown. Otherwise, cross-references are shown for all modules.

Default: `-nshmodtyp`

`-shmodvar, -shmodvar <mod_list>`  
(analysis, global only) Show cross-reference listings of public module data. If the `mod_list` is supplied, cross-references for the named modules are shown. Otherwise, cross-references are shown for all modules.

Default: `-nshmodvar`

`-shprg`  
(analysis, global only) Show cross-reference listings for the program.

Default: `-shprg`

`-shref, -shref <root-list>`  
(analysis, global only) Show the complete reference structure of the referenced procedures. If supplied, the `root-list` provides the roots for the reference structure. If omitted, the main program is used as the root.

Default: `-shref`

`-shsngl`  
(analysis, global and local) In the program unit cross-references, show unreferenced constants, namelist groups and procedures that are declared in include files or modules. Also, show unreferenced common-block objects and unreferenced imported module variables.

Default: `-shsngl`

`-shsrc`  
(analysis, global and local) In the listing file, show the source code. To list source code, the `-shsub` option must also be in effect.

Default: `-shsrc`

`-shsub`  
(analysis, global and local) In the listing file, show the code and cross-references of program units and subprograms. The listing of source code lines can be suppressed using the `-nshsrc` option.

Default: `-shsub`

`-specific`  
(analysis, global and local) Flag all referenced procedures that are invoked using type-specific (i.e. non-generic) names.

Default: `-nspecific`

`-standard`

(analysis, global and local) Validate all syntax for conformance to the selected language level (standard).

Default: `-nstandard`

`--strip-path <path>, -s <path>`

Strips the prefix of a file name path in error messages and references to your source files. If you specify the `--strip-path` option multiple times, you strip all of the prefixes from the file names, in the order in which you specify the `--strip-path` argument values.

(control) This option is also available with `cov-commit-defects` and `cov-analyze`.

The leading portion of the path is omitted if it matches a value specified by this option. For example, if the actual full pathname of a file is `/foo/bar/baz.c`, and `--strip-path /foo` is specified, then the name attribute for the file becomes `/bar/baz.c`.

 **Note**

Coverity recommends using this option for a number of reasons:

You can enhance end-to-end performance of the path stripping process by using this option during the analysis of your source code, rather than when committing the analysis results to Coverity Connect;.

It shortens paths that Coverity Connect; displays. It also allows your deployment to be more portable if you need to move it to a new machine in the future.

`-truncate`

(analysis, global) Truncate names to 6 significant characters.

Default: `-ntruncate`

`-update`

(analysis, global) Update the library file. If the library file does not exist, it will be created.

Default: `-nupdate`

`--vendor`

Specifies the compiler vendor to be used in selecting a Coverity Fortran Syntax Analysis configuration file. Vendor values include: **absoft cd control convex cray cv compaq cyber dec digital equip res digres apollo domain ftn salford fujitsu gnu hp hewlett ibm intel lahey ms microsoft nag ndp ps pathscale pdp pgi princeton prime prospero rm ryan sgi silicon sun solaris oracle unisys watcom**

`--version <ver>`

Specifies the compiler version to be used in selecting a Coverity Fortran Syntax Analysis configuration file. Valid values for the `--version` option are vendor-specific.

This option should be specified only when emulation of a particular compiler version is desired. Moreover, not all configuration files provide version information, so specifying a particular version might select the empty set.

`-warnings`  
(analysis, global) Show warnings.

Default: `-warnings`

### Example 1

```
> cov-run-fortran --dir idir -- file1.f file2.f
```

This will use Coverity Fortran Syntax Analysis to analyze `file1.f` and `file2.f` using the default configuration file. Coverity Fortran Syntax Analysis defects will be written into the `output/` subdirectory of the emit directory `idir` in a file called `FC.errors.xml`. This file is suitable for upload to Coverity Connect using `cov-commit-defects`.

### Example 2

```
> cov-run-fortran --dir idir --vendor intel --version 14.1 -- file1.f
```

This will use Coverity Fortran Syntax Analysis to analyze `file1.f`, selecting the configuration file that will emulate the Intel Fortran compiler version 14.1. Coverity Fortran Syntax Analysis will accept all of the standard language constructs and extensions normally accepted by the Intel Fortran compiler of that version. It will report incompatibilities with the language recognized that compiler, including obsolescent and deleted features and features supported by other compilers but not by that version of the Intel Fortran compiler.

### Example 3

```
> cov-run-fortran --dir idir --vendor intel --version 14.1 --level f95 -- file1.f
```

This will use Coverity Fortran Syntax Analysis to analyze `file1.f`, selecting the configuration file that will emulate the Intel Fortran compiler version 14.1. Coverity Fortran Syntax Analysis will accept all of the standard language constructs and extensions normally accepted by the Intel Fortran compiler of that version. However, the `--level f95` flag causes all language elements incompatible with the Fortran 95 standard (obsolescent, deleted, or supported only in a newer standard) to be flagged.

### Example 4

```
> cov-run-fortran --dir idir -- -ff file1.f
```

This will use Coverity Fortran Syntax Analysis to analyze `file1.f` using the default compiler configuration, but forcing the interpretation of the input according to the free-form source form.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.

- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects`(*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

### See Also

Coverity Fortran Syntax Analysis User Guide [🔗](#)

`cov-import-results`

`cov-commit-defects`

`cov-manage-emit`

---

## Name

cov-security-da Run a dynamic analysis for the security checkers.

## Synopsis

```
cov-security-da
 [--da-max-mem <mem>]
 --dir <idir>
 [--tu <id>]
 [--tu-pattern <translation-unit-pattern>]...
```

## Description

The `cov-security-da` command runs a dynamic analysis of Java and .NET bytecode and a separate dynamic analysis of JavaScript templates. The Java/.NET analysis invokes certain string-manipulation functions to detect whether they correctly escape or sanitize unsafe values. The JavaScript template analysis dynamically renders observed template files to detect interpolation sites that could be vulnerable to XSS.



### Note

Coverity Security Dynamic Analysis for C# and Visual Basic requires a Windows 64-bit or Linux 64-bit system that supports .NET Core 3.1.

The output of the bytecode analysis primarily affects the XSS checker for Java, C#, and Visual Basic, and the output of the template-DA analysis primarily affects the XSS checker for Javascript.

By default, `cov-build` and `cov-capture` invoke `cov-security-da` as a final step. This command needs to be invoked manually in the following situations:

- When `cov-build` or `cov-capture` was invoked using the `--no-security-da` option
- After invoking `cov-emit-java`—for example, to capture a Web application archive (WAR) file
- When the intermediate directory has been manually modified

Because the Java/.NET dynamic analysis requires compiled bytecode, it cannot be used with a Java file system capture. Similarly, the JavaScript template analysis will not be invoked if a suitable JavaScript project can't be identified.

## Options

`--da-max-mem`

[Java Web application security option] Sets the JVM heap size of the VM that is running the dynamic analyzer, a component of the Java Web application security analysis. The option accepts an integer that specifies a number of megabytes (MB). The default value is 1024.

`--dir <intermediate directory>`

Specifies the intermediate directory to emit to. This option is required: You must specify a directory.

**--no-bytecode-da**

Disables all bytecode analysis for Java, C#, and Visual Basic.

**--run-template-da-on-emit**

Specify to have `cov-security-da` attempt to re-run the template-DA on an existing intermediate directory that contains one or more captured JavaScript projects.

**--template-da-timeout**

Specify the maximum amount of time to take (in ms) when analyzing a single JavaScript template file. Defaults to 1 minute.

**--tu <id>, --tu-pattern <translation\_unit\_pattern>**

Restricts the dynamic analysis to specific translation units, identified either by numeric ID (`--tu`) or by pattern (`--tu-pattern`).

The `--tu-pattern` option can be specified multiple times. Both `--tu` and `--tu-pattern` can be specified on a single command line. The tool runs on the union of the translation units indicated by all such options.

It is an error if at least one `--tu-pattern` argument is specified but no translation unit matches any of the specified patterns.

For more information, see "Translation unit pattern matching".

## See Also

`cov-build`

`cov-capture`

---

## Name

`cov-test-configuration` Test command-line translations of a configuration by making assertions about the translations.

## Synopsis

```
cov-test-configuration [OPTIONS] <input_script.json>
```

## Description

The `cov-test-configuration` command parses the given input script, invokes `cov-translate` on the native compiler commands, and asserts that the given facts about the translated command lines are true. The translation takes place according to the configuration generated by the `cov-configure` command.

If all tests pass, it exits with 0. Otherwise, it exits with a non-zero code.

### Note

The `cov-translate` command uses the same configuration as this command and translates native command-line options into `cov-emit` options but does *NOT* execute `cov-emit`. See `cov-translate --dry-run`.

Any command names referenced by the input script must be executable in the current environment.

**Input script format.** The input script must be a well-formed, valid JSON text file encoded in UTF-8. See [www.json.org](http://www.json.org)  for more information on JSON.

Sample input script format:

```
[
{
 "section": "My Section Label",
 "tests": [
 {
 "given_input" : "gcc -m64 -c foo.c",
 "assert_that_output": {
 "contains": ["-w", "-D__NO_INLINE__", "--incompat_proto"],
 "omits": ["-m32"]
 }
 }
],
 ... // more tests here
}
, ... // more sections here
]
```

The expected structure is the following:

- A list of one or more `section` objects.

- Each `section` object is composed of:
  - `section`: A label naming the section of tests.
  - `tests`: A list of one or more `test` objects.
- Each `test` object is composed of:
  - `given_input`: The native command line to translate.
  - `assert_that_output`: An `assertion` object applied to translated command line.
- An `assertion` object is composed of one or more of the following `assertion` operators:
  - `contains`: List of one or more command-line options that are required in translation.
  - `omits`: List of one or more command-line options that are forbidden in translation.

Each test translates the `given_input` with `cov-translate` and checks that all the `assertion` operators pass when applied to the translation.

Example 1:

```
> cov-configure --config myTest/coverity_config.xml --msvc
> cov-test-configuration --config myTest/coverity_config.xml MyTests.json
```

Output of the `cov-test-configuration` example:

```
Section [0] My Section Label
Tests run: 1, Failures: 0, Errors: 0

Sections run: 1, Tests run: 1, Failures: 0, Errors: 0
```

In this example, all tests passed.

Example 2:

```
> cov-test-configuration --config myTest/coverity_config.xml OtherTests.json
```

Output of the `cov-test-configuration` example:

```
Section [0] My Section Label
Tests run: 1, Failures: 0, Errors: 0

Section [1] Microsoft C/C++
Tests run: 5, Failures: 2, Errors: 1
```

```
Section [1] Microsoft C/C++: Test [0]: Assertion [contains][2] failed
Given input: cl -c foo.c
Expected : output contains bob
Actual : cov-emit.exe ... --c --microsoft --no_alternative_tokens \
-w --ignore_calling_convention --microsoft_version 1300 \
--no_stdarg_builtin -D_USE_ATTRIBUTES_FOR_SAL=0 foo.c

Sections run: 2, Tests run: 6, Failures: 2, Errors: 0
```

The example shows two test failures. Failures are assertion operators that failed. Errors are failures that were encountered when executing `cov-translate`.

The ellipsis ( . . . ) in the example represents other arguments that `cov-translate` adds to make `cov-emit` imitate the `cl` input more precisely.

## Shared options

`--config <coverity_config.xml>, -c <coverity_config.xml>`

Uses the specified configuration file instead of the default configuration file located at `<install_dir_sa>/config/coverity_config.xml`.

`--debug, -g`

Turn on basic debugging output.

`--ident`

Displays the version of Coverity Analysis and build number.

`--info`

Displays certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

`--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>`

Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects`(*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

### **See Also**

`cov-configure`

`cov-emit`

`cov-translate`

---

## Name

`cov-translate` Translate native compiler command line into a valid command line for a Coverity compiler.

## Synopsis

```
cov-translate [OPTIONS] <compile-command>
```

## Description

This feature translates the native compiler command line given and invokes the Coverity compiler with the translated command line. The translation is done according to the configuration generated by the `cov-configure` command.

### Note

If you change the set of compilation options used by your build process, delete the `<intermediate_directory>` and capture a full build from scratch. Otherwise, the translation units captured using the old options will remain in the emit. The new translation units will not replace the old translation units due to the changed compilation options.

## Example

Suppose you have a single file `t.c` that is compiled with `gcc` as:

```
> gcc -DFOO=BAR -c t.c
```

In this case, invoking this command with `cov-translate` in front will call `cov-emit` in the appropriate way to compile `t.c`, assuming that `gcc` has been configured with `cov-configure`:

```
> cov-translate --dir /tmp/emit gcc -DFOO=BAR -c t.c
```

Output of the `cov-translate` example:

```
cov-emit --dir /tmp/emit ... --gcc -w -DFOO=BAR t.c
Emit for file 't.c' complete.
```

In the previous example, `...` represents other arguments that are added to provide system include directories, preinclude files, and other command-line arguments that `cov-translate` adds to make `cov-emit` imitate `gcc` more precisely.

## Options

`--add-arg <arg>`

Directly adds `<arg>` to the argument list passed to the Coverity compiler. These arguments are passed before other arguments detected by `cov-translate`.

`--add-arg-no-file <arg>`

Adds an argument `<arg>` to the invocations of the Coverity compiler. This argument is not stored in the response file, and therefore is not used by `cov-build --replay` or `cov-preprocess`.

**--auto-diff**

Only applies to C/C++. This option turns on automatic diagnosis of parsing problems by comparing preprocessed files generated by the native compiler versus `cov-emit`. It can be useful to determine incompatibilities (especially of include order and macro definitions), but it might interfere with the build for some compilers that are not fully supported.

**--build-id <build-id>, --build-id-file <build-id-file>**

Specifies a build ID to be used by this invocation of `cov-translate`. The build ID is a string that uniquely identifies a build. Every intermediate directory is associated with a build ID.

When passing `--build-id`, the `<build-id>` argument is used verbatim as the build ID. When passing `--build-id-file`, the `<build-id-file>` argument is opened as a file and its contents are read; those contents are then used as a build ID.

If `--c-coverage` is passed to `cov-translate`, exactly one of these switches also needs to be passed to `cov-translate`.

**--c-coverage <tool>**

Compatible with C and C++ builds only. Enables C and C++ coverage collection using the specified tool. The only tool currently available is "function", which enables the Function Coverage Instrumentation

If `--c-coverage` is passed to `cov-translate`, `--c-coverage-log-file` must also be passed.

**--c-coverage-log-file <log-file-path>**

Specifies a log file for the enabled C/C++ coverage tool to log to. This argument is required if `--c-coverage` has been passed to `cov-translate`.

**--chase-symlinks**

Follows symbolic links when determining file names to report.

This option is not supported for use with Clang compilers.

**--clean-preprocessed**

Only applies to C/C++. This option cleans up preprocessed files left behind by `--preprocess-first` after each file is compiled. Usually `--preprocess-first` leaves preprocessed files in the source code directory. This option only makes sense if `--preprocessed-first` is also specified.

**--cygpath <path>**

Specify the path to the directory, which contains the bin directory of the Cygwin installation, if it is not in the PATH environment variable.

**--cygwin**

On Windows, indicates that the build is done with Cygwin. This option allows Cygwin-style paths to be used in the native build command. However, you must use Windows-style paths for all Coverity Analysis commands.

**--dir <intermediate\_dir>**

Pathname to an intermediate directory that is used to store the emit repository and output directory.

Unless you are running `cov-translate` for debugging purposes, you should always use the `--dir` option. Without it, the command will not emit output an intermediate directory.

Note that you can specify this pathname in `coverity_config.xml`. For details on using this file, see on the `<prevent>` tag in the section "Using the `<prevent>` tag to specify directories and emit options" [🔗](#) in *Coverity Analysis 2020.12 User and Administrator Guide*.

**--dryrun, -n**

Prints out the Coverity compiler command line it would normally run without actually running it. This option can help expedite your configuration process. The option is analogous to the `make -n` command.

**--emit-cmd-line-id**

This option is deprecated as of the 4.4 release.

**--emit-complementary-info**

C, C++ only. Enables emitting of complementary information for compliance checkers such as MISRA checkers. Selecting this option results in a slower build capture but a faster analysis, and it should be applied when using compliance checkers. The default value is `--no-emit-complementary-info`

 **Note**

Enabling the `--emit-complementary-info` option prior to running an analysis is likely to turn up additional defects.

Any analysis involving `--coding-standard-config` requires the information generated during `cov-build` when including the `--emit-complementary-info` option. The `cov-build` command will take longer, so this option should only be used when `cov-analyze` is used with `--coding-standard-config`.

If `cov-build` did not include the `--emit-complementary-info` option and `cov-analyze` does include `--coding-standard-config`, `cov-analyze` automatically re-runs every `cov-emit` command (for the Translation Units to be analyzed). This excludes the native build and the `cov-translate` overhead, but it will add significant overhead to `cov-analyze`. Note that analysis will fail if the emit database does not include source; that is re-emit is not possible.

**--emit-parse-errors**

Deprecated. Use the `--enable_PARSE_ERROR` option in `cov-analyze` instead. Specifies that compile failures should be made visible in Coverity Connect, appearing as defects.

**--emulate-string <regex>, -s <regex>**

Must be used along with the `--run-compile` option. The `--emulate-string` option specifies a regex that, if matched on the command line, causes `cov-translate` to only run the native compiler command line assigned without attempting to call the Coverity compiler. This is useful for files such as `confctest.c`, which are compiled by the `cov-configure` feature to test the native compiler's output for certain strings. When the output for `cov-translate` is interspersed with these strings, it causes `cov-configure` to fail.

--encoding <enc>

Specifies the encoding of source files. Use this option when the source code contains non-ASCII characters so that Coverity Connect can display the code correctly. The default value is US-ASCII. Valid values are the ICU-supported encoding names:

US-ASCII

UTF-8

UTF-16

UTF-16BE

UTF-16 Big-Endian

UTF-16LE

UTF-16 Little-Endian

UTF-32

UTF-32BE

UTF-32 Big-Endian

UTF-32LE

UTF-32 Little-Endian

ISO-8859-1

Western European (Latin-1)

ISO-8859-2

Central European

ISO-8859-3

Maltese, Esperanto

ISO-8859-4

North European

ISO-8859-5

Cyrillic

ISO-8859-6

Arabic

ISO-8859-7

Greek

ISO-8859-8

Hebrew

ISO-8859-9

Turkish

ISO-8859-10  
Nordic

ISO-8859-13  
Baltic Rim

ISO-8859-15  
Latin-9

Shift\_JIS  
Japanese

EUC-JP  
Japanese

ISO-2022-JP  
Japanese

GB2312  
Chinese (EUC-CN)

ISO-2022-CN  
Simplified Chinese

Big5  
Traditional Chinese

EUC-TW  
Taiwanese

EUC-KR  
Korean

ISO-2022-KR  
Korean

KOI8-R  
Russian

windows-1251  
Windows Cyrillic

windows-1252  
Windows Latin-1

windows-1256  
Windows Arabic

-E  
For C/C++ only. Only preprocess the source file.

**--fail-stop, -fs**

Returns an error code if the Coverity compiler fails. By default, Coverity compiler parse failures are ignored and the return code is a success.

**--force**

Passes `--force` to the Coverity compiler, which cause emits to happen for files with timestamps that have not changed. See `--force` in `cov-emit` for other Coverity compiler documentation.

**--no-caa-info**

Does not collect the information required for Coverity Architecture Analysis in the intermediate directory.

**--no-emit**

Parses, but does not emit source files. This is useful primarily for debugging purposes and is not intended for general use.

**--no-emit-complementary-info**

For C/C++ only. This option disables the emitting of complementary information for compliance checkers such as MISRA checkers.

**--no-headers**

For C/C++ only. This option does not emit header files, only the primary source file(s). Because this option can cause problems in C++ programs, you should use it only if directed by Coverity support.

**--no-parallel-translate**

Compatible with C and C++ builds only. Disables `cov-translate` parallelization. This will prevent `cov-translate` from running in parallel, regardless of the degree of parallelization requested, either directly to `cov-build`, `cov-translate`, through configuration files, or native command line translation.

This can also be added as a Coverity compiler argument in a configuration file (as it is not actually passed to the Coverity compiler). For example:

```
<prepend_arg>--no-parallel-translate</prepend_arg>
```

**--no-preprocess-next**

For C/C++ only. This option disables the `--preprocess-next` option.

**--parallel-translate=<number\_of\_processes>**

Compatible with C and C++ builds only. This option instructs `cov-translate` to run Coverity compiler in parallel when multiple files are seen on a single native compiler invocation. This is similar to Make's `-j` switch and to the Microsoft Visual C/C++ `/MP` switch. It specifies the `<<number_of_processes>>` to be greater than zero to explicitly set the number of processes to spawn in parallel, or zero to auto-detect based on the number of CPUs. When specified directly to `cov-build` or `cov-translate`, this option will override any settings set in the configuration files or translated through the native command line.

This can also be added as a Coverity compiler argument in a configuration file (though it is not actually passed to the Coverity compiler). For example:

```
<prepend_arg>--parallel-translate=4</prepend_arg>
```

`--preinclude <file.h>, -pi <file.h>`

For C/C++ only. This option specifies that `<file.h>` should be preincluded before all other source and header files when invoking `cov-emit`. This is equivalent to `cov-emit's --preinclude` argument.

`--preprocess-first`

Compatible with C and C++ builds only. Not supported for Clang compilers. This option uses the native compiler to preprocess source files and then invokes `cov-emit` to compile the output of the native processor. By default, `cov-emit` (which is invoked by `cov-translate`) otherwise tries to preprocess and parse each source file.

Using this option can address some cases in which hard-to-diagnose causes for macro predefinitions are different, or for header files that cannot be found by `cov-emit`. Usually, `cov-configure` attempts to intelligently guess the native compiler's predefined macros and built-in include directories, but sometimes `cov-configure` guesses incorrectly. Using the `--preprocess-first` option circumvents the problem, but at the cost of losing macro information during analysis. Using `--preprocess-first` does not always work because it requires rewriting the native compiler command line, which the native compiler may or may not like.

`--preprocess-next`

Compatible with C and C++ builds only. Not supported for Clang compilers. This option uses `cov-emit` to preprocess source files. If that attempt fails, or if `cov-emit` encounters a parse error, this option preprocesses the files with the native preprocessor, and invokes `cov-emit` to compile the output of the native processor. This offers the benefit of using the higher-fidelity `cov-emit` preprocessor, while also providing a fallback in case of errors.

This option can be disabled with `--no-preprocess-next` option (the latter has precedence over the former). See `--preprocess-first` for information about the effects of using the native preprocessor.

`--print-native`

Prints out the native compiler command line.

`--record-only, -ro`

This option is available for `cov-emit` only. It records the compilation command in the emit directory, and does not automatically attempt to parse or emit the code. Later, `cov-build` can be run with the `--replay` option to actually parse and emit the code.

`--redirect stdout|stderr,<filename>, -rd stdout|stderr,<filename>`

Redirects either `stdout` or `stderr` to `<filename>`.

`--run-compile`

In addition to calling the Coverity compiler, this option also runs the native compiler. The `--run-compile` option should only be used if you are attempting to manually integrate `cov-translate` into your build system. It should never be used with `cov-build`.

`--timings`

Passes `-#` to `cov-emit`, which prints out timing information for various stages of parsing and emitting for every file compiled. This is a debugging option not intended for general use.

## Shared options

- `--config <coverity_config.xml> , -C <coverity_config.xml>`  
Uses the specified configuration file instead of the default configuration file located at `<install_dir_sa>/config/coverity_config.xml`.
- `--debug, -g`  
Turn on basic debugging output.
- `--debug-flags <flag> [, <flag>]`  
Controls the amount of debugging output produced during a build. These flags can be combined on the command line using a comma as a delimiter.
- Valid flags are `translate` and `translate-phases`. For example, `--debug-flags translate`.
- `--ident`  
Displays the version of Coverity Analysis and build number.
- `--info`  
Displays certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.
- `--tmpdir <tmp>, -t <tmp>`  
Specifies the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.
- `--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>`  
Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## See Also

`cov-build`

cov-configure

cov-emit

cov-emit-cs

cov-emit-vb

cov-emit-java

cov-emit-swift

---

## Name

cov-upgrade-static-analysis Upgrade an old Coverity Analysis release to the latest version.

## Synopsis

```
cov-upgrade-static-analysis { --use-existing-release | --use-new-release } --old-release <dir> [--old-config <file>...]
[OTHER OPTIONS]
```

## Description

The `cov-upgrade-static-analysis` command upgrades an old Static Analysis or Coverity Analysis release to Coverity Analysis version 2020.12. Run this command from the `<install_dir_sa>/bin` directory of the new release of Coverity Analysis. Also, the exact upgrade process differs slightly based on file permission and web server process owner issues.

There are two modes in which you can run this command.

- In the first (and preferred) mode of operation, specified with the `--use-new-release` option, the configuration and the database in the old release is moved into the new release.
- In the second mode, specified with the `--use-existing-release` option, the old release is upgraded in place. When the upgrade is completed, the new release is installed in the location that was formerly occupied by the old release.

## Options

### Basic options

`--old-config <file>`

The location of an old Coverity Analysis non-default configuration file (`coverity_config.xml`) that for the old release. Specifying the path to all configuration files referenced on a typical command line to the old release allows the upgrade to re-write any configuration files that are relevant when invoking the programs in the new release. Repeat this option for each configuration file.

`--old-release <dir>, -or <dir>`

The location of the old Coverity Analysis release. This should be the full path to the directory containing the version file, which may be the textual `VERSION` file, the XML `VERSION.xml` file, or both.

`--use-existing-release`

Upgrade an existing Coverity Analysis release in-place.

Do not use this option if you are upgrading to the current version of Coverity Analysis .

`--use-new-release`

Copy settings and the database from the old Coverity Analysis release to the new release. Leave the old release untouched. Note that any files in the old release that were added by the user are added to the new release automatically when the upgrade is complete. This is the preferred mode of operation.

## Other options

--help, -h

Provide help information on the command.

--log <file>

The absolute path and file name for where to save the upgrade log. The default log file is <install\_dir\_sa>/bin/coverity\_upgrade.log.

--pedantic

If the command returns with warnings, return a non-zero exit code. The default behavior is to return non-zero codes only when there are errors.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

---

## Name

cov-wizard Launches a GUI-based utility for configuring Coverity Analysis.

## Synopsis

```
cov-wizard <command>
```

## Description

The `cov-wizard` command runs a GUI-based Coverity Analysis utility for setting up compilers, configuring and running the build process, running the analysis for quality and security issues, and committing the results to Coverity Connect. Java code and C/C++ code. For information about Coverity Wizard, see the *Coverity Wizard 2020.12 User Guide* [🔗](#).

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects`(*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

---

## Coverity Analysis Ant Tasks

---

## Name

`covanalyzeandcommit` Analyze Java source code and class files, and then commit the results to Coverity Connect.

## Synopsis

```
<covanalyzeandcommit
 dir="int_dir"
 [OPTIONAL_ATTRIBUTES]>
 <[OPTIONAL_ELEMENTS]/>
</covanalyzeandcommit>
```

## Description

The `covanalyzeandcommit` analyzes Java source code and class files that have been previously stored in an intermediate directory by capturing a build with the `cov-build` command and/or by adding them manually using the `cov-emit-java` command. Then it commits the results to Coverity Connect. The commit process occurs if the analysis had either no errors or only recoverable errors. Otherwise, the commit process is skipped.

## Attributes

`additionalanalysisoptions="options"`

Most users are unlikely to use this attribute, which provides a string of additional, space-separated options to pass to the analysis command. You might use it to pass options that are not available as analysis-related attributes to this Ant task.

This attribute should not be used with any options that contain spaces, such as filenames with spaces.

Example:

```
<covanalyzeandcommit additionalanalysisoptions="--append false --max-mem 8000"/>
```

See `cov-analyze` for a complete list of options.

`additionalcommitoptions="options"`

Most users are unlikely to use this attribute, which provides a string of additional, space-separated options to pass to `cov-commit-defects`. You might use it to pass options that are not available as commit-related attributes to this Ant task.

This attribute should not be used with any options that contain spaces, such as filenames with spaces.

Example:

```
<covanalyzeandcommit additionalcommitoptions="--debug false --ticker-mode none"/>
```

See `cov-commit-defects` for a complete list of options.

`all="true"`

Enables Coverity Analysis for Java checkers that are disabled by default.

Exception: Web application security checkers (such as XSS) are not affected by this option. To enable them, see `--webapp-security`.

`analysis="false"`

Allows you to turn off the analysis process, which normally takes place prior to the commit process. In this way, you can commit the results of a prior analysis to Coverity Connect without running another analysis first. Defaults to true.

`binpath="<install_dir>/bin"`

Specifies the directory containing `cov-analyze` and `cov-commit-defects`. Use this attribute if the Ant task fails to find these commands. Without this attribute, the Ant task searches for these based on the PATH environment variable and/or the location of `coverity-anttask.jar`.

`commit="false"`

Allows you to turn off the commit process. In this way, you can perform an analysis without committing results to Coverity Connect afterward. Defaults to true.

`config="coverity_config.xml"`

Uses the specified configuration file instead of the default configuration file located at `<install_dir>/config/coverity_config.xml`. This file applies to both the analysis and the commit processes.

`dataport="cim_commit_port"`

Specifies the commit port of the Coverity Connect server.

`dir="int_dir"`

Pathname to an intermediate directory that is used to store the emit repository and output directory.

If you specify `"."`, it uses the current directory as the intermediate directory.

`disabledefault="true"`

Disables default checkers. This option is useful if you want to disable all default checkers and then enable only a few with the `--enable` option.

For a list of checkers that are disabled through this option, see the `--enable` option documentation for the `cov-analyze` command.

`failonerror="true|false"`

If true, the build process will succeed only if both `cov-analyze` and `cov-commit-defects` exit without return codes that indicate failure. Otherwise, the Ant task will always succeed. Defaults to true.

`spotbugs="true|false"`

Explicitly enables (`true`) or disables (`false`) SpotBugs analysis. This attribute corresponds to the `enable-fb` (when true) and `disable-fb` (when false) options to `cov-analyze`. SpotBugs

analysis is enabled by default. Further, you can use the `disable checker` element to disable individual SpotBugs bug patterns. However, `enable checker` *will not* enable them.

This attribute supports the analysis process.

`host="server_hostname"`

You should use the `--url` option instead of this option. This option is deprecated and will be removed in a future release.

Specify the server hostname to which to send the results. The server must have a running instance of Coverity Connect.

If unspecified, the default is the `host` element from the XML configuration file.

 **Note**

- If you're running `cov-commit-defects` on a Linux OS, or using `--ssl`, you must enter the full host and domain name for the `--host` parameter:

```
--host <server_hostname.domain.com>
```

- The `--host` switch, while still supported, now produces a deprecation warning that it may be removed in a later release. The `--url` syntax is the preferred replacement.

`httpport="port_number"`

Used with the `--host` option to specify the HTTP port on the Coverity Connect host. This port is used to connect to the `--dataport` commit port.

The Commit port is determined using one of the following methods, listed in order of priority (the first applicable item will be used):

1. The Commit port specified with `--dataport`.
2. The HTTP port specified with `--port`. `cov-commit-defects` connects to this port to retrieve the dataport using HTTP if `--ssl` is absent or HTTPS if `--ssl` is present.
3. The HTTPS port specified with `--https-port`. `cov-commit-defects` connects to this port using HTTPS to retrieve the dataport.
4. The Commit port, specified with the `cim/commit/port` element from the XML configuration file.
5. The HTTP or HTTPS port specified with the `cim/port` or `cim/https_port` element, respectively, from the XML configuration file.
6. HTTP port 8080 without `--ssl` or 8443 with `--ssl` is used to retrieve the dataport from Coverity Connect.

 **Note**

If you are committing to an SSL-enabled instance of Coverity Connect, you might encounter an error message when you define the `--port` option (for example, `--port 8443`). Use the `--https-port` option instead.

`parallelthreads="N"`

Allows you to control the number of analysis workers that run in parallel. This number is limited by the terms of your license. The default value for the `-j` option is 1.

This attribute supports the analysis process.

`password="password"`

You should use the `--url` option instead of this option. This option is deprecated and will be removed in a future release.

Specify the password for either the current user name, or the user specified with the `--user` option. For security reasons, the password transmitted to the Coverity Connect is encrypted. If unspecified, the default is (in order of precedence):

1. The password from the `--url` option.
2. The `password` element from the XML configuration file.
3. The environment variable `COVERITY_PASSPHRASE`.
4. The password in the file pointed to by the environment variable `COVERITY_PASSPHRASE_FILE`.

 **Note**

The passphrase can be stored in a file without any other text, such as a newline character.

 **Warning**

On multi-user systems, such as Linux, users can see the full command line of all commands that all users execute. For example, if a user uses the `ps -Awf` command, identifying information such as usernames, process identities, dates and times, and full command lines display.

This attribute supports the commit process.

`resultproperty="property_name"`

The name of a property to store the return code of the command. It provides the maximum value of the `cov-analyze` and `cov-commit-defects` return code. Only used if `failonerror=false`.

This attribute supports the analysis process.

`stream="stream_name"`

Specifies a stream name to which to commit these defects.

If the stream option is not specified, the `stream` element from the XML configuration file is used.

If the stream is associated with a specific language and you attempt to commit results from other languages to that stream, the commit will fail. However, in Coverity Connect, it is possible to associate a stream with multiple languages even if the stream was previously associated with a single programming language.

**strippath**

Strip the prefix from all file names in error messages and file references committed. This might make commits from multiple users match, even if the code is located in a different location.

`strippath` is a nested element which expects one attribute, `prefix`. The value of the `prefix` attribute is the prefix that will be stripped from file names. This is an example of the `strippath` element: `<strippath prefix="/path/to/project/root">`

If specified multiple times, strips all of the prefixes from each filename, in the order the `strippath` elements are supplied.

This nested element supports the analysis process.

**target="target\_name"**

Target platform for this project (for example, `i386`).

**user="user\_name"**

You should use the `--url` option instead of this option. This option is deprecated and will be removed in a future release.

Specifies the user name that is shown in Coverity Connect as having committed this snapshot. If unspecified, the default is:

1. The username specified by the `--url` option, if any.
2. The `user` element from the XML configuration file.
3. The environment variable `COV_USER`.
4. The environment variable `USER`.
5. The name of the operating system user invoking the command (where supported).
6. The UID of the operating system user invoking the command (where supported).
7. `admin`.

**version="version"**

This snapshot's project version.

**Elements**

The following elements must be enclosed by the `covanalyzejava` element.

```
<checkeroption checker="checker_name" option="option_name" val="value"/>
 Pass option option_name (with optional value value) to a specific checker checker_name.
```

For example:

```
<checkeroption checker="NULL_RETURNS" option="check-bias" value="5"/>
```

```
<disable checker="checker_name" />
```

Disable `checker_name`. This can be specified multiple times. See also `--list-checkers` and `--disable-default`. For example, to disable cross-references in defects in the code browser, specify `<disable checker="XREFS" />`.

```
[<disable checker="checker_name" />
```

```
<enable checker="checker_name" />
```

Enable `checker_name`. The checker name is case insensitive. This can be specified multiple times. See also the `<disable checker="checker_name">` element and the `disabledefault="true"` attribute.

```
[<enable checker="checker_name" />
```

## Examples

Analyzing and committing results.

```
<target name="loadtask" description="Load the covanalyzejava task">
 <taskdef resource="com/coverity/anttask.xml" classpath="${anttask.jar}" />
</target>

<target name="build.analyzeandcommit" description="Analyze and commit results"
 depends="loadtask">
 <covanalyzeandcommit
 dataport="${env.CIM_COMMIT_PORT}"
 dir="${env.PREVENTINTDIR}"
 host="${env.CIM_SERVER}"
 password="coverity"
 stream="${env.SA_TEST_PROJECT}"
 user="admin"
 version="1.2 rc 7" />
</target>
```

Enabling all but one of the default checkers for the analysis.

```
<target name="analyzeandcommit.no.forwardnull" depends="loadtask">
 <property environment="env1" />
 <echo message="Current PATH = ${env1.PATH}" />
 <covanalyzeandcommit
 dataport="${env.CIM_COMMIT_PORT}"
 dir="${env.SA_INT_DIR}"
 disabledefault="true"
 host="${env.CIM_SERVER}"
 password="coverity"
 stream="${env.SA_TEST_PROJECT}"
 user="admin"
 version="1.2">
 <enable checker="FORWARD_NULL" />
 </covanalyzeandcommit>
</target>
```

**See Also**

cov-analyze

covbuild

---

## Name

covbuild Intercept all calls to the compiler invoked by the build system using an Ant task.

## Synopsis

```
<covbuild
 dir="int_dir"
 [OPTIONAL_ATTRIBUTES]>
</covbuild>
```

## Description

The `covbuild` task calls Ant with a specified build file and target, and it captures any compilations under this call.

## Attributes

`antargs`

Passes command-line arguments to Ant.

`antfile="build.xml"`

Specifies the location of the build file that is called by Ant. The default is the current Ant build file.

`binpath="<install_dir>/bin"`

Specifies the directory containing `cov-build`. Use this attribute if the Ant task fails to find this command. Without this attribute, the Ant task searches for `cov-build` based on the `PATH` environment variable and/or the location of `coverity-anttask.jar`.

`covbuildargs="options"`

Passes space-delimited options to `cov-build`.

`dir="dir"`

Specifies the intermediate directory into which the build is captured.

`inheritAll="false"`

When this attribute is set to `true`, the properties passed to the Ant invocation that runs the `covbuild` task will be passed on to the Ant invocation made by the `covbuild` task (similar to the `inheritAll` attribute of the built-in Ant task, which ships as part of Ant). Unlike the attribute of the built-in Ant task, the `covbuild` attribute defaults to `false`.

`target="target"`

Specifies a target in the build file (see `antfile`). Defaults to the default target of the specified Ant build file.

## Examples

Note that the following are equivalent.

- Ant:

```
<covbuild dir="idir"
 antfile="build0.xml"
 target="build"
/>
```

- Command line:

```
> "cov-build --dir idir ant -f build0.xml build"
```

#### Additional examples:

```
<target name="build.default" depends="loadtask">
 <property environment="env4"/>
 <echo message="Current PATH = ${env4.PATH}"/>
 <covbuild
 dir="${env.SA_INT_DIR}"
 target="build1"/>
</target>

<target name="build.alternative" depends="loadtask">
 <property environment="env4"/>
 <echo message="Current PATH = ${env4.PATH}"/>
 <covbuild
 dir="${env.SA_INT_DIR}"
 antfile="alternative.xml"
 target="build-alternative"/>
</target>

<target name="build.executable" depends="loadtask">
 <echo message="binpath = ${build.binpath}"/>
 <covbuild
 binpath="${build.binpath}"
 dir="${env.PREVENTINTDIR}"
 target="build1"/>
</target>
```

#### See Also

[cov-build](#)

[covanalyzeandcommit](#)

---

## Test Advisor Commands

---

## Name

`cov-emit-server` Start an emit server to collect coverage.

## Synopsis

```
cov-emit-server
 [--dir|--idir-library] <directory> [--port <port number>]
 [--interface <ip-to-bind-to>]
 [--gcov-cache-size <size in MB>]
 [--force-start]
```

## Description

`cov-emit-server` allows for the collection of coverage from remote machines. `cov-emit-server` can also be used to collect coverage on a single machine. For some tests, this can greatly improve the performance of a coverage collection run. See `cov-build` for information on what coverage tools are supported.



### Note

In general, you should not launch `cov-emit-server` directly, `cov-manage-emit` should be used to control the server. For more information, see "Emit Server sub-commands" for `cov-manage-emit`.

`cov-emit-server` creates a log file in `<directory>/cov-emit-server.log`.

## Options

`--dir <int_dir>`

Specifies the intermediate directory in which the emit server will start. Note that `--dir` and `--idir-library` are mutually exclusive, you can only specify one or the other.

`--idir-library <idir-library>`

Specifies the intermediate directory library in which the emit server will start. Note that `--dir` and `--idir-library` are mutually exclusive, you can only specify one or the other. If `<idir-library>` does not exist, it will be created for you.

`--port <port_number>`

Specify the port to listen on. Default is 15772.

`--interface <ip-address-to-use>`

Specify the IP address to listen on. This must be an IPv4 address.

`--gcov-cache-size <size-in-mb>`

Specify the size of the cache to use for collecting gcov data files. The default size is 500MB.

`--force-start`

Force starting the server, even if the pid file exists. This is useful if the previous server did not shut down cleanly.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## Examples

Start an emit server on an intermediate directory:

```
cov-emit-server --dir idir
```

Start an emit server, using a custom port on an external interface:

```
cov-emit-server --dir idir --port 12345 --interface 10.0.0.1
```

Start an emit server on an idir library:

```
cov-emit-server --idir-library my-idir-library
```

## See Also

`cov-build`

`cov-manage-emit`

---

## Name

cov-emit-server-control Control a running emit server instance

## Synopsis

```
cov-emit-server-control
 [--dir <idir>|--idir-library <directory>|--interface <ip-address>]
 [--port <port number>]
 [--start-suite <suitename>|--start-test <testname>|--stop-test <testname>]
 [--max-wait-time <timeout-seconds>] [--status <test-status>]
 [--log <logfile>]
```

## Description

`cov-emit-server-control` controls a running instance of `cov-emit-server`. This is used to start and stop tests when using Remote Test Separation.

## Options

- `--dir <idir>`  
Specifies the intermediate directory of the emit server to control. Exactly one of `--dir`, `--idir-library`, or `--interface` must be specified.
- `--idir-library <directory>`  
Specifies the intermediate directory library of the emit server to control. Exactly one of `--dir`, `--idir-library`, or `--interface` must be specified.
- `--interface <ip-address>`  
Specifies the IP address on which the emit server is listening. Exactly one of `--dir`, `--idir-library`, or `--interface` must be specified.
- `--port <port_number>`  
Specifies the port number on which the emit server is listening. This must be used when `--interface` is given.
- `--start-suite <suitename>`  
Start a testsuite with the given suitename. Exactly one of `--start-suite`, `--start-test`, or `--stop-test` must be specified.
- `--start-test <testname>`  
Start a test with the given testname. Exactly one of `--start-suite`, `--start-test`, or `--stop-test` must be specified.
- `--stop-test <testname>`  
Stop a test with the given testname. Exactly one of `--start-suite`, `--start-test`, or `--stop-test` must be specified.
- `--max-wait-time <timeout-seconds>`  
Specifies the timeout to use when communicating with the emit server. If unspecified, a timeout of 30 seconds is used.

`--status <test-status>`

Specifies the test status to assign to the currently running test. This option should be used with the `--stop-test` option. Acceptable values for `test-status` are "pass", "fail", or "unknown".

`--log <logfile>`

Specifies where diagnostic information shall be logged. When `logfile` is "-", logging is written to stdout. If not specified, no logging is performed.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## Examples

Start a testsuite named "mysuite" on the emit server running on intermediate directory "idir":

```
cov-emit-server-control --dir idir --start-suite mysuite
```

Start a test name "mytest" on the emit server running on IP address 127.0.0.1, port 6405:

```
cov-emit-server-control --interface 127.0.0.1 --port 6405 --start-test mytest
```

## See Also

`cov-emit-server`

---

## Name

cov-manage-history Manages historical data needed by Test Advisor

## Synopsis

```
cov-manage-history --dir <intermediate_directory> download | import
[command_options]
```

## Description

The `cov-manage-history` command queries a running Coverity Connect instance for the history of functions previously committed to a specified stream. This history is downloaded and stored in a given intermediate directory.



### Note

The `cov-manage-history` command now accepts authorized tokens. It also replaces the `cov-download-history` command, which was deprecated as of the 7.0 release.

## Options

### Common options

`--dir <intermediate_directory>`

Pathname to the intermediate directory to store the history.

`--java`

Deprecated in version 7.0: Specify that the stream contains Java data. Starting in 7.0, the command detects the source code language automatically, so this option is not needed anymore.

### Individual commands

`download`

Replaces the functionality in `cov-download-history` command. This command has the following options:

- `--auth-key-file`

This option specifies the location of an authentication key file that was previously created. It is used to connect to the Coverity Connect server.

- `--authenticate-ssl`

This is equivalent to `--on-new-cert distrust`.

- `--certs <filename>`

In addition to CA certificates obtained from other trust stores, use the CA certificates in the given `<filename>`. For information on the new SSL certificate management functionality, please see *Coverity Platform 2020.12 User and Administrator Guide* [↗](#)

- `--connect-timeout <n>`

Sets the timeout for establishing connections to `<n>` seconds. If a connection to Coverity Connect cannot be established within this time, the transaction is aborted. This timeout cannot be disabled. The default value is 60 seconds.

- `--host <coverityconnect_host>`

`--port <coverityconnect_port>`

The hostname and port of the Coverity Connect instance to download history from. `--port` is an optional property. If `--port` is not specified on the command line, the default is 8080 without `--ssl` and 8443 with `--ssl`.

- `--max-retries <n>`

Sets the number of times to retry failed or aborted requests with Coverity Connect to `<n>`. Note that this does not include the initial attempt, so a setting of 1 results in at most 2 request attempts. A setting of 0 means to never retry failed requests. The default value is 1.

- `--merge`

Normally, the download command overwrites any previously downloaded or imported history in the intermediate directory with the newly-downloaded history. Specifying this option causes `cov-manage-history` to instead merge the newly-downloaded history with any existing history. This allows history from multiple streams to be combined for use in analysis.

- `--on-new-cert <trust | distrust>`

Indicates with `--ssl` whether to trust (with `trust-first-time`) self-signed certificates, presented by the server, that the application has not seen before.

- `--response-timeout <n>`

Sets the response timeout to `<n>` seconds. For every request for data sent to Coverity Connect, if a response is not received within this time, the request is aborted. A setting of 0 disables this timeout. The default value is 300 seconds.

- `--sleep-before-retry <n>`

Sets the time to sleep before retrying a failed or aborted request with Coverity Connect to `<n>` seconds. A setting of 0 disables this sleep. The default value is 1 second.

- `--ssl`

Enables SSL encryption for communication with Coverity Connect.

- `--stream <stream>`

The name of the stream on the Coverity Connect instance to query.

- `--unstrip-path <unstrip_path>`

This option is deprecated in 2020.12. Use the `--strip-path` option for `cov-analyze` for C/C++ or Java.

- `--url <path>`

Allows you to connect to a CIM instance (to download history) that has a context path in its HTTP(S) URL. You can use this option instead of the `--host`, or `--port` options. The `--url` option is provided to accommodate the use of a context path and to deal with setting up Coverity Connect behind a reverse proxy.

Use HTTPS or HTTP to connect to Coverity Connect HTTPS or HTTP port. For `http`, the default port is 80; for `https`, the default port is 443. For example:

```
https://example.com/coverity
```

```
https://cimpop:8008
```

```
http://cim.example.com:8080
```

 **Note**

You may not use the `commit://` scheme in the URL.

- `--user <username>`  
`--password <password>`

The username and password used to log into the Coverity Connect instance. These will be encrypted if `--ssl` is used.

### import

Responsible for copying the analysis history data from one intermediate directory to another intermediate directory (or the same, if the directory is to be reused). It has the following option:

- `--from-dir <int_dir>`

The source intermediate directory. This command will copy the analysis history data from the source intermediate directory into the current intermediate directory.

 **Note**

You should NOT use the `--from-dir` in your production environment to copy Test Advisor history from an intermediate directory for the first time after a version upgrade (for example, from 7.0.x to 7.5.0).

Instead, you should commit the "old" intermediate directory to Coverity Connect and then download the Test Advisor history using the `download` option to `cov-manage-history`. For example:

```
cov-manage-history download --host <host_name> --stream <stream_name> \
 --user <username> --password <password>
```

Normally, when the analysis history is generated by `cov-analyze` in an intermediate directory, it can only be committed to an instance of Coverity Connect. If the history is to be re-used within the same intermediate directory (for example, during a subsequent run of analysis on the same intermediate directory), then the `import` command should be used to make that analysis history available for re-use. To do this, the same intermediate directory should be specified for both the `--dir` and `--from-dir` options.

## Shared options

`--verbose <0, 1, 2, 3, 4>`, `-V <0, 1, 2, 3, 4>`

Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects` (*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## See Also

`cov-extract-scm`

`cov-import-scm`

`cov-manage-emit`

---

## Name

`cov-patch-bullseye` Patch a Bullseye small runtime for use with Test Advisor.

## Synopsis

```
cov-patch-bullseye --bullseye-dir <path> [--output-dir <path>]
```

## Description

The `cov-patch-bullseye` allows you to patch your Bullseye small runtime for Test Advisor coverage. You must build the Bullseye small runtime before you patch it and use it in `cov-build`. For usage instructions, see *Test Advisor 2020.12 User and Administrator Guide*. [🔗](#) in the *Test Advisor 2020.12 User and Administrator Guide*.

## Options

`--bullseye-dir <path>`

Path to the Bullseye installation directory.

`--output-dir <path>`

Path where the modified Bullseye small runtime files will be placed. If this option is not specified, it will default to `<path>/coverity`, where `<path>` is the value from `--bullseye-dir`.

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects`(*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## See Also

`cov-build`

---

## Dynamic Analysis Commands

---

## Name

`cov-start-da-broker` Start the Dynamic Analysis Broker.

## Synopsis

```
cov-start-da-broker --dir <intermediate_directory> --host
<cim_server_host_name> --user <user_name> --password <password> --stream
<dynamic_stream> { } [OPTIONS]
```

## Description

The `cov-start-da-broker` command starts the Dynamic Analysis Broker, a process that receives defect data from Dynamic Analysis Agents and forwards the data to the Coverity Connect. In addition to the command line, some command line options can be set in environment variables, and most can be set via configuration file. See the *Dynamic Analysis 2020.12 Administration Tutorial* for more information.

## Options

The list below describes the command line options for `cov-start-da-broker`.

- `--broker-port <port>`  
Listen on this port for Agent's connections (default 4422).
- `--config-file <file>, -cf <file>`  
Specify the path and filename of the Java properties configuration file. This is where you put your configuration file properties.
- `--dataport <port_number>, -dp <port_number>, --port <port_number>`  
Specify the Coverity Connect server port for source and defect commits (Default: 9090).
- `--dir <intermediate_directory>`  
Required option that specifies an intermediate directory that was created through the `cov-build` command. The `cov-build` command captures your source and classes. For more information, see `cov-build` in *Coverity Analysis 2020.12 User and Administrator Guide* [🔗](#).
- `--help, -h`  
Print a help message and exit.
- `--host <hostname>, -r <hostname>`  
Specify the host name or IP address of the Coverity Connect server.
- `--only-listen, -ol`  
[Deprecated] This option is deprecated as of version 7.6.0 and will be removed from a future release.  
  
Listens for Dynamic Analysis Agents and send their defects to Coverity Connect, but do not commit the source code to Coverity Connect (Default: False, which means the broker listens for agents and commits source code.)
- `--only-source-commit, -osc`  
[Deprecated] This option is deprecated as of version 7.6.0 and will be removed from a future release.

Commit source code to a source stream, but do not listen for Agents. (Default: False, which means the broker commits source code and listens for Agents.)

- `--only-test-connection, -n`  
Validate the command line and configuration, test the connection to the Coverity Connect server, verify that the source and dynamic streams exist, and exit regardless of other command line options. (Default: False, which means the broker operates normally in a non-test mode.)
- `--output-dir <directory>`  
Change the output directory from the default of `cda_data`. This directory stores a run directory for each invocation of the Broker.
- `--password <password>, -pa <password>`  
Provide a Coverity Connect password.
- `--rundir <run_directory>`  
Specify a location for the Dynamic Analysis Broker run directory. This is where the Broker puts its log files and information files such as `broker.started` and `broker.ok`. If the run directory you specify already exists, the Broker moves the old one to `old_<run_dir>_<number>`, where `<number>` is the first number such that the name isn't taken. This option overrides the default behavior where the Broker creates a run directory with a unique name as a new subdirectory under the output directory.
- `--run-prefix <prefix>, -rp <prefix>`  
Change the prefix of the run directory from the default of "broker" to `<prefix>`. The run directory is a sub-directory of the output directory where the Broker stores logs and other information from this run.
- `--security-file <license_path>, -sf <license_path>`  
Select a path to a file that contains a valid Coverity Analysis license that is required to commit source code. This option is unnecessary if the license is properly installed. Not available as an Ant attribute. (Default: `<install_dir_ca>/bin/license.dat`)
- `--shutdown-after <seconds>`  
Specify the number of seconds to wait before the Broker shuts down when no Agents are connected. This convenience option ensures that unused Broker processes terminate when they are no longer required. Specifying `never` means to never shut down automatically. Default: 600 seconds.
- `--stream <dynamic_stream>`  
Specify the name of the Dynamic Analysis stream to commit defects and source code.
- `--user <user_name>, -u <user_name>`  
User name for Coverity Connect server. If no user name is specified, the Broker uses the name of the operating system user that invokes it. (Default: Operating System user name.)
- `--version`  
Print version information.

## Examples

The `cov-start-da-broker` command lines below are entered on a single line without returns.

Example command line (dynamic commit and source commit):

```
cov-start-da-broker --host cim.example.com --dataport 9090 --user jdoe
--password secret --stream my_project --dir my_intermediate_dir
```

Example command line (source commit, no dynamic commit):

```
cov-start-da-broker --host cim.example.com --dataport 9090 --user jdoe
--password secret --stream my_project --dir my_intermediate_dir --only-source-commit
```

Example command line (dynamic commit, no source commit):

```
cov-start-da-broker --host cim.example.com --dataport 9090 --user jdoe
--password secret --stream my_project --dir my_intermediate_dir --only-listen
```

---

## Name

`cov-stop-da-broker` Stop the Dynamic Analysis Broker.

## Synopsis

This command allows you to manually shutdown a Dynamic Analysis Broker process.

```
cov-stop-da-broker [clean | early | dirty] [--broker-port <port_number>] [--broker-host
<hostname_or_IP>]
```

## Description

The `cov-stop-da-broker` command sends a shutdown request to the Dynamic Analysis Broker (see the *Dynamic Analysis 2020.12 Administration Tutorial* for an explanation of the Broker). If this command is not executed, the Broker automatically terminates 10 minutes after the last Dynamic Analysis Agent disconnects, although the automatic shutdown period can be changed with the `--shutdown-after` option.



### Note

Running `cov-start-da-broker` with the `--only-source-commit` option causes the Broker to exit after finishing its source commit. Running `cov-stop-da-broker` is not necessary in this case.

## Options

The list below describes the options for the `cov-start-da-broker` command.

```
[clean|dirty|early]
```

Specify how the Dynamic Analysis Broker shuts down. Can be one of three values: `clean`, `dirty` or `early`.

`clean`

(Default) Tells the Broker to exit once all of the following are true:

- The Broker has finished committing source code to Coverity Connect
- All Dynamic Analysis Agents have disconnected from the Broker.
- The Broker has finished sending all of the defects that the Dynamic Analysis Agents reported to Coverity Connect

`dirty`

Tell the Broker to exit immediately. The Broker terminates the source commit if it has not completed, closes connections to Agents (which will themselves exit if their `failfast` option is true), and drops any defects that it has not sent to Coverity Connect. If the source commit already finished successfully before the dirty shutdown, some defects from this run of the Broker may appear in Coverity Connect, otherwise, no defects from this run of the Broker will appear in Coverity Connect.

### early

Tell the Broker to close its connections to Agents (which will themselves exit if their `failfast` option is true) and then exit when all of the following are true:

- The Broker has finished committing source code to Coverity Connect.
- The Broker has finished sending all of the defects that the Agents reported (before having their connections to the Broker forcibly closed) to Coverity Connect.

`--broker-host <host_or_IP>`

Specify the hostname or IP address on which the Broker is running. Default: localhost

`--broker-port <port-number>`

Specify the port on which the Broker is listening. Default: 4422

## Examples

This example sends a shut down request to the Broker on localhost:4422.

```
cov-stop-da-broker
```

This example sends an early shutdown request to the Broker on host `broker.example.com` on port 1234.

```
cov-stop-da-broker early --brokerhost broker.example.com --brokerport 1234
```

An example using Ant:

```
<cov-stop-da-broker
 shutdowntype="early"
 brokerhost="broker.example.com"
 brokerport="1234" />
```

---

## Dynamic Analysis Ant Tasks

---

## Name

cov-dynamic-analyze-java Use Ant task to run a Java program with Dynamic Analysis Agent enabled.

## Description

This task runs a Java program with the Dynamic Analysis Agent enabled. It functions as a replacement for any existing `java` task. You can replace `<java>` with this task in an existing build script.

This task sets the `fork` parameter to `true`. You cannot run Dynamic Analysis from within the same virtual machine as the Ant build.

You can disable Dynamic Analysis's functionality by setting the `enabled` parameter to `false`. In this mode, the task passes through all functionality to the `java` task, and all parameters that are specific to Dynamic Analysis are ignored.

The Dynamic Analysis distribution includes an `antlib.xml` file, so you can enable this task by adding the following line to the Ant build script:

```
<typedef resource="com/coverity/anttask.xml"
 classpath="<install_dir_ca>/library/coverity-anttask.jar"/>
```

This task extends the `java` ant task, so it supports all parameters that the `java` task supports.

## Attributes

Below are the Dynamic Analysis Ant attributes for `cov-dynamic-analyze-java` and `cov-dynamic-analyze-junit`. Default values are in parenthesis.

`detect_deadlocks="<boolean>"`  
Detect deadlocks. (True)

`agentoptions="<boolean>"`  
Specify a list of Dynamic Analysis Agent options as you would pass on the command line in the form of `option=value,option=value...`. This attribute allows you to specify Agent options for which there is no Ant attribute. Do not specify an Agent option both in this string and via an Ant attribute because doing so has undefined consequences. (Not specified.)

`detect_races="<boolean>"`  
Detect race conditions. (True.)

`detect_resource_leaks="<boolean>"`  
Detect resource leaks. (True.)

`broker_host="<host_or_IP>"`  
Specify the hostname or IP address on which you ran the Broker. (localhost)

`broker_port="<port_number>"`  
Specify the port on which the Broker is listening. This is set with the `broker-port` option to `cov-start-da-broker`. If you intend to run more than one Broker instance simultaneously on the same machine, it is a good practice to use non-default ports to avoid collisions. (4422)

`enabled=<boolean>`

(No agent option.) Enable Dynamic Analysis. If set to `False`, then a `cov-dynamic-analyze-java` task behaves like a `java` task, and a `cov-dynamic-analyze-junit` task behaves like a `javaunit` task. (`True`.)

`exclude_instrumentation="<colon_separated_list_of_prefixes>"`

Exclude classes from being watched by Dynamic Analysis to speed up Dynamic Analysis. However, Dynamic Analysis does not detect defects in excluded code nor as a result of actions performed in excluded code (such as field access or lock acquisitions).

This option consists of a colon-separated list of prefixes of the fully qualified names to exclude. For example:

`"exclude-instrumentation=A.B"` excludes any class whose name starts with `"A.B"`, such as `"A.B"`, `"A.B.c"`, or `"A.Bc"`.

`"exclude-instrumentation=A.B."` (with a period) excludes `"A.B.c"`, but not `"A.B"` nor `"A.Bc"`.

(The default is to exclude nothing.)

`failfast="<boolean>"`

If `True`, the Dynamic Analysis Agent exits the program it is watching if the Dynamic Analysis Agent loses its connection to the Broker or has other problems. If `False`, the Dynamic Analysis Agent quietly allows the program to continue running, even if Dynamic Analysis Agent cannot run properly. (`False`.)

`instrument_only="<colon_separated_list_of_prefixes>"`

Specify a colon-separated list of classes watched by Dynamic Analysis. Dynamic Analysis excludes all other classes (same as if they were all specified as options to `exclude-instrumentation`). As with `exclude-instrumentation` above, using this option might speed up Dynamic Analysis, but at the cost of missing defects. This option consists of a colon-separated list of prefixes of fully qualified names to include. For example:

`"instrument-only=A.B"` includes any class whose name starts with `"A.B"`, such as `"A.B"`, `"A.B.c"`, or `"A.Bc"`.

`"instrument-only=A.B."` includes `"A.B.c"`, but not `"A.B"` nor `"A.Bc"`.

(The default is to include everything.)

`instrument_arrays="<boolean>"`

Watch reads and writes into arrays to report race conditions. (`False`.)

`instrument_collections="<boolean>"`

Detect race conditions on collections. For example suppose `map` is a `java.util.Map` and one thread executes `map.put("key", "value")` without holding any locks. If Dynamic Analysis sees another thread access this `map`, it reports a `RACE_CONDITION`. (`True`.)

`override_security_manager="<boolean>"`

Install a permissive security manager (`java.lang.SecurityManager`) that allows all operations except the installation of other security managers. This option exists because a restrictive security

manager causes Dynamic Analysis to fail. Setting this option to `true` might allow Dynamic Analysis of your program to proceed. If this option is `false`, running Dynamic Analysis on your application might require adjusting your security policy or excluding classes that run within the restrictive security manager (see the `exclude-instrumentation` and `instrument-only` options). (False.)

`repeat_connect="<non-negative_number>"`

If this option is set to a number greater than zero, and the initial attempt to connect to the Broker fails, then the Agent tries to reconnect to the Broker that number of times, with a one second pause between attempts, before giving up. For best results, set this option to something greater than zero when starting both the Broker and Agents from a script or build file. (0 when running the Agent from the command line and 20 when running it through the `cov-dynamic-analyze-java` or `cov-dynamic-analyze-junit` Ant tasks.)

## Nested elements

There are no additional nested elements beyond what the `java` or `junit` tasks support.

## Examples

The following example runs the program `test.Main` with Dynamic Analyzer instrumentation enabled:

```
<cov-dynamic-analyze-java classname="test.Main" detect_deadlocks="false">
 <arg value="-h">
 <classpath>
 <pathelement location="dist/test.jar"/>;
 <pathelement path="{java.class.path}"/>
 </classpath>
</cov-dynamic-analyze-java>
```

---

## Name

cov-dynamic-analyze-junit Runs tests from the JUnit testing framework with Dynamic Analysis enabled.

## Description

This task runs tests from the JUnit testing framework with Dynamic Analysis enabled. It functions as a replacement for any existing JUnit task. You can replace `<junit>` with `<cov-dynamic-analyze-junit>` in an existing build script.

This task sets the `fork` parameter to `true`. You cannot run Dynamic Analysis from within the same virtual machine as the Ant build. However, all the tests can be run within a single forked VM by setting the `forkmode` parameter of the `junit` task to `once`.

Disable Dynamic Analysis's functionality by setting the `enabled` parameter to `false`. In this mode, the task passes through all functionality to the `junit` task and all Dynamic Analysis-specific parameters are ignored.

The Dynamic Analysis distribution includes an `antlib.xml` file, you can enable this task by adding the following line to the Ant build script:

```
<typedef resource="com/coverity/anttask.xml"
classpath="<install_dir_ca>/library/coverity-anttask.jar"/>
```

This task extends the `junit` ant task. It supports all parameters that by the `junit` task supports. For the complete list of supported parameters, see the Ant `junit` task documentation.

## Attributes

For the available `cov-dynamic-analyze-junit` attributes, see `cov-dynamic-analyze-java` (*Coverity 2020.12 Command Reference*).

## Nested elements

There are no additional nested elements beyond what the `java` or `junit` tasks support.

## Examples

The following example runs the test that is defined in `my.test.TestCase` with Dynamic Analysis instrumentation enabled:

```
<cov-dynamic-analyze-junit>
 <test name="my.test.TestCase"/>
</cov-dynamic-analyze-junit>
```

The following example runs the test that is defined in `my.test.TestCase` with Dynamic Analysis instrumentation enabled. At the end of the test, a one-line summary prints. A detailed report of the test is in `TEST-my.test.TestCase.txt`. The build process stops if the test fails.

```
<cov-dynamic-analyze-junit printsummary="yes" haltonfailure="yes">
 <formatter type="plain"/>
 <test name="my.test.TestCase"/>
```

```
</cov-dynamic-analyze-junit>
```

The following example runs `my.test.TestCase` in the same VM with Dynamic Analysis instrumentation disabled.

```
<cov-dynamic-analyze-junit enabled="no">
 <test name="my.test.TestCase"/>
</cov-dynamic-analyze-junit>
```

---

## Name

cov-start-da-broker Start the Dynamic Analysis Broker using an Ant task.

## Synopsis

```
<cov-start-da-broker ATTRIBUTES/>
```

## Description

The `cov-start-da-broker` Ant task starts the Dynamic Analysis Broker, which receives defects from Dynamic Analysis instances and sends them to Coverity Connect. See *Dynamic Analysis 2020.12 Administration Tutorial* for details on what the Broker does and how it works.

## Attributes

The list below describes the Ant attributes for `cov-start-da-broker`.

`brokerport="<port>"`

Listen on this port for Agent's connections (default 4422).

`configfile="<file>"`

Specify the path and filename of the Java properties configuration file. This is where you put your configuration file properties.

`dataport="<port_number>"`

Specify the Coverity Connect server port for source and defect commits (Default: 9090).

`dir="<intermediate_directory>"`

Required option that specifies an intermediate directory that was created through the `cov-build` command. The `cov-build` command captures your source and classes. For more information, see `cov-build` in *Coverity Analysis 2020.12 User and Administrator Guide* [↗](#).

`host="<hostname>"`

Specify the host name or IP address of the Coverity Connect server.

`onlylisten="<boolean>"`

[Deprecated] This option is deprecated as of version 7.6.0 and will be removed from a future release.

Listens for Dynamic Analysis Agents and send their defects to Coverity Connect, but do not commit the source code to Coverity Connect (Default: False, which means the broker listens for agents and commits source code.)

`onlysourcecommit="<boolean>"`

[Deprecated] This option is deprecated as of version 7.6.0 and will be removed from a future release.

Commit source code to a source stream, but do not listen for Agents. (Default: False, which means the broker commits source code and listens for Agents.)

`onlytestconnection="<boolean>"`

Validate the command line and configuration, test the connection to the Coverity Connect server, verify that the source and dynamic streams exist, and exit regardless of other command line options. (Default: False, which means the broker operates normally in a non-test mode.)

outputdir="<directory>"

Change the output directory from the default of `cda_data`. This directory stores a run directory for each invocation of the Broker.

password="<password>"

Provide a Coverity Connect password.

rundir="<run\_directory>"

Specify a location for the Dynamic Analysis Broker run directory. This is where the Broker puts its log files and information files such as `broker.started` and `broker.ok`. If the run directory you specify already exists, the Broker moves the old one to `old_<run_dir>_<number>`, where `<number>` is the first number such that the name isn't taken. This option overrides the default behavior where the Broker creates a run directory with a unique name as a new subdirectory under the output directory.

runprefix="<prefix>"

Change the prefix of the run directory from the default of "broker" to `<prefix>`. The run directory is a sub-directory of the output directory where the Broker stores logs and other information from this run.

shutdownafter="<seconds>"

Specify the number of seconds to wait before the Broker shuts down when no Agents are connected. This convenience option ensures that unused Broker processes terminate when they are no longer required. Specifying `never` means to never shut down automatically. Default: 600 seconds.

stream="<dynamic\_stream\_name>"

Specify the name of the Dynamic Analysis stream to commit defects and source code.

user="<user\_name>"

User name for Coverity Connect server. If no user name is specified, the Broker uses the name of the operating system user that invokes it. (Default: Operating System user name.)

## Examples

- Commits the source code to Coverity Connect. This target uses `cov-start-da-broker` with `onlysourcecommit="true"`. Splitting the source commit from the dynamic defect commit – especially in Ant build files – is a good practice because it ensures that the whole target will fail if the source commit fails.

```
<property name="demos.idir" location="example-idir"/>
<target name="source.commit" depends="build-project">
 <cov-start-da-broker
 failonerror="true"
 configFile="${config.file}"
 dir="${demos.idir}"
 onlySourceCommit="true"
 />
</target>
```

- Starts the Broker in the background. This target uses `cov-start-da-broker` with `onlylisten="true"` to listen for Dynamic Analysis Agent connections and to stream their defects to Coverity Connect.

```
<target name="broker.listen">
 <cov-start-da-broker
 failonerror="true"
 configfile="${config.file}"
 onlylisten="true"
 />
</target>
```

**Sample configuration file:**

```
host=cim.example.com
dataport=9090
user=jdoe
password=secret
stream=Example-dynamic
```

---

## Name

`cov-stop-da-broker` Stop the Dynamic Analysis Broker using an Ant task.

## Synopsis

```
<cov-stop-da-broker ATTRIBUTES/>
```

## Description

The `cov-stop-da-broker` Ant task stops the Dynamic Analysis Broker, which receives defects from Dynamic Analysis instances and sends them to Coverity Connect.

## Attributes

The list below describes the attributes for the `cov-start-da-broker` Ant task.

`shutdowntype="<broker-shutdown-type>"`

Specify how the Dynamic Analysis Broker shuts down. Can be one of three values: `clean`, `dirty` or `early`.

`clean`

(Default) Tells the Broker to exit once all of the following are true:

- The Broker has finished committing source code to Coverity Connect
- All Dynamic Analysis Agents have disconnected from the Broker.
- The Broker has finished sending all of the defects that the Dynamic Analysis Agents reported to Coverity Connect

`dirty`

Tell the Broker to exit immediately. The Broker terminates the source commit if it has not completed, closes connections to Agents (which will themselves exit if their `failfast` option is true), and drops any defects that it has not sent to Coverity Connect. If the source commit already finished successfully before the dirty shutdown, some defects from this run of the Broker may appear in Coverity Connect, otherwise, no defects from this run of the Broker will appear in Coverity Connect.

`early`

Tell the Broker to close its connections to Agents (which will themselves exit if their `failfast` option is true) and then exit when all of the following are true:

- The Broker has finished committing source code to Coverity Connect.
- The Broker has finished sending all of the defects that the Agents reported (before having their connections to the Broker forcibly closed) to Coverity Connect.

`brokerhost="<host_or_IP>"`

Specify the hostname or IP address on which the Broker is running. Default: localhost

brokerport="<port\_number>"

Specify the port on which the Broker is listening. Default: 4422

## Examples

The example below shows a clean shutdown of the Broker.

```
<target name="broker.shutdown">
 <cov-stop-da-broker shutdowntype="clean"/>
</target>
```

---

## Coverity Connect Commands

---

## Name

cov-admin-db Administer Coverity Connect.

## Synopsis

```
cov-admin-db backup [--dir] <file_or_dir> [--force] [--no-overwrite] [-j
<number_of_processors>] [--debug]
```

```
cov-admin-db check-integrity [--install-dir <install_dir>][--debug]
```

```
cov-admin-db optimize [--debug]
```

```
cov-admin-db psql [--debug]
```

```
cov-admin-db reset-admin-password [--debug]
```

```
cov-admin-db restore <file_or_dir> [--force] [--no-overwrite] [-j
<number_of_processors>] [--debug]
```

```
cov-admin-db scramble --input-dump <file_or_dir> --output-dir <dir_name> [--
debug]
```

```
cov-admin-db tune {[--read]|[--show-profile]|[--suggest]|[--write]}[--debug]
```

```
cov-admin-db upgrade-schema [--debug]
```

## Log File

This command generates a log file: <install\_dir\_cc>/logs/cov-admin-db.log.

## Description

The `cov-admin-db` command performs various operations on Coverity Connect as described in the following table.

Subcommand	DB type	Operation
backup	--	Backs up the database to an archive file or directory. This option only works with the embedded database.  See the section called “Backing up and Restoring a Database”.
check-integrity	embedded	Checks the integrity of your database by verifying tables, sequences, columns, constraints, and indexes.  See the section called “Checking Database Integrity”.
optimize	embedded	Improves database use of indexes and statistics.  See the section called “Optimizing the Database”.
psql	external	Runs the <code>psql</code> command on the database, allowing you to issue queries interactively to PostgreSQL.

Subcommand	DB type	Operation
	embedded	
reset-admin-password	external	Changes password for the <code>admin</code> account to that specified at the prompt.
restore	--	Restores data to the embedded database, using the specified archive file or directory.  See the section called “Backing up and Restoring a Database”.
scramble	external embedded	Strips all sensitive information from your database backup so that Coverity support can troubleshoot it.  See the section called “Preparing the Connect Database to Send to Synopsys”.
tune	external embedded	Allows you to tune PostgreSQL and Java JVM settings for the Coverity Connect database for optimal performance. Tune options allow you to read tune settings, display the server profile, display suggested settings, and to apply the suggested tune settings.  See the section called “Tuning Coverity Connect for Your Environment”.
upgrade-schema	external embedded	Upgrades an archived schema from a previous version of Coverity Connect to make it compatible with the current version of Coverity Connect.

## General Options

--debug

Displays debug output.

## Backing up and Restoring a Database

Use the `backup` and `restore` subcommands to back up and restore your database. These variants only work with the embedded database.

```
cov-admin-db backup [--dir] <file_or_dir> [--force] [--no-overwrite] [-j
<number_of_processors>] [--debug]
```

```
cov-admin-db restore <file_or_dir> [--force] [--no-overwrite] [-j
<number_of_processors>] [--debug]
```

- `Backup` backs up the database to an archive file or directory.
- `Restore` restores data to the embedded database, using the specified archive file or directory. Before you use this subcommand, run the `cov-im-ctl maintenance` command.

In addition to restoring the data, this command upgrades the schema to make the schema compatible with the current version of Coverity Connect. The archive file is not modified during this process.

 **Caution**

Use caution when restoring a database with this command because it deletes all data from the current database.

With both backup and restore, you have a choice of two different formats: a file or a directory. Files are both more convenient to work with than directories and faster for backups and restores.

Option	Use
<code>--dir</code>	With <code>Backup</code> : The directory where you want the database backed up.  With <code>Restore</code> : The directory from which data is restored to the database.
<code>--force</code>	Suppresses user inputs during the backup and restore process to help automate the procedure.
<code>--no-overwrite</code>	Specifies that the following will not be overwritten: <ul style="list-style-type: none"> <li>• An existing backup file with the <code>backup</code> command</li> <li>• The contents of a database with the <code>restore</code> command</li> </ul> If you do not specify this option, you must manually answer the questions during the backup/restore process. All questions are skipped if <code>--force</code> is present.
<code>-j</code> <code>&lt;number_of_processors&gt;</code>	Use with either subcommand to control the level of parallelism and reduce the amount of time for the operation. The <code>&lt;number_of_processors&gt;</code> attribute sets the number of core processors available on your system.  This setting has no effect for file backup/restore. It defaults to 4 for directory-based backup/restore. 2 or 3 is recommended.

## Checking Database Integrity

Use the `check-integrity` subcommand to check the integrity of your database: the command verifies that tables, sequences, columns, constraints, and indexes have the intended definitions.

 **Note**

This operation automatically runs before the `cov-admin-db backup` command is executed and after the `cov-admin-db restore` command is executed.

```
cov-admin-db check-integrity [--install-dir <install_dir_name>] [--debug]
```

Use the `--install-dir` option to specify another Coverity Connect installation to check. The default location is `<install-dir-CC>`. The subcommand is compatible with all versions of Coverity Connect.

## Optimizing the Database

Use the `optimize` subcommand to optimize database use of indexes and statistics.

You can use this subcommand without putting Coverity Connect in `Maintenance` mode.

To reclaim the maximum amount of space, you can back up and restore the embedded database after optimizing. For information about this process, see "Administering the Coverity Connect database" in the *Coverity Platform 2020.12 User and Administrator Guide*.

## Preparing the Connect Database to Send to Synopsys

You might need to send a backup of your Connect database to Coverity support. You have two options for doing so securely:

- Remove the source code from the database before sending. Your Support representative can explain how you do that.
- Anonymize the database.

This operation, called *scrambling*, not only removes source code, but destroys all names of items in the database (for example, users, components, and streams), replacing them with small numbers. Because the data is overwritten, there is no way to "un-scramble" a scrambled database.

Scrambling has disadvantages: 1) It takes longer than removing source. 2) Depending on the problem that Support needs to diagnose, you might need to work with Support to match a few of the scrambler-generated numbers to their unscrambled names.

Use the `scramble` subcommand to anonymize your database before sending it to Coverity Support. This command works with all supported CIM version backups.

The syntax of the `scramble` command is as follows:

```
cov-admin-db scramble --input-dump <file_or_dir> --output-dir <output_dir>
[--debug]
```

Use the `--input-dump` option to specify the name of the input backup file or directory. This is the file you want scrambled. The output of the scrambling operation is a database backup directory, `<output_dir>`.

The basic workflow is as follows:

1. Run the `cov-admin-db backup` command to create a backup of your database.
2. To avoid impacting the resources used by your production Connect instance, install Connect on a different host. Use the embedded database option when installing.
3. On the instance installed above, run the `cov-admin-db scramble` command to scramble the database backup.

The `cov-admin-db scramble` command takes the file or directory you provide with the `--input-dump` option, and restores it into a temporary database. It then scrambles information in the temporary database.

4. The `cov-admin-db scramble` command then backs up the stripped database to the directory specified by the `--output-dir` option.
5. You can now make a tar or zip archive of the `<output_dir>` directory and send it to Coverity Support.

 **Note**

Like removing source, scrambling takes significant disk space, both for the two backups and for the temporary database. The temporary database will consume the same amount of disk space as the production database.

## Tuning Coverity Connect for Your Environment

The `cov-admin-db tune` command is used by the installer at install time to adjust your Coverity Connect database and JVM settings for best performance.

After the product is installed, you can use the `cov-admin-db tune` subcommand to retune Coverity Connect. You might want to retune for reasons like the following:

- Your environment has changed, and you want to retune the system in response to those changes.
- To restrict use of available resources.

Command syntax provides the following options:

```
cov-admin-db tune {[--read]|[--show-profile]|[--suggest]|[--write]}[--debug]
```

Option	Use
<code>--read</code>	Reads the current tune settings and displays them.
<code>--show-profile</code> [[<key>=<value>]...]	Displays the profile that describes the current environment: hardware and OS settings.  Use the <code>&lt;key=value&gt;</code> expression to override current profile settings.
<code>--suggest</code> [[<key>=<value>]...]	Calculates optimal Coverity Connect tuning values based on the current profile and displays what would be written to the settings files with a subsequent <code>cov-admin-db --write</code> command.  Use the <code>&lt;key=value&gt;</code> expression to provide alternate values for profile settings.
<code>--write</code> [[<key>=<value>]...]	Calculates optimal Coverity Connect tuning values based on current profile settings and writes them to the settings files.

Option	Use
	Use the <key=value> expression to provide alternate values for profile settings.  You must restart Coverity Connect for these settings to take effect.

The subcommand assembles a description of the available resources, called a *profile*, which is a collection of settings. The profile isn't stored, but it is used with the `--show-profile`, `--suggest`, and `--write` options.

Profile settings include the following:

Setting	Meaning
<code>isExternalDb</code>	Boolean: true if the database disk is external (not embedded).
<code>isSsd</code>	Boolean: true if the database disk is an SSD.
<code>mode</code>	Use "default".
<code>os</code>	"Windows" or "Linux"
<code>physicalMemory</code>	Physical memory in gigabytes.
<code>processorCount</code>	Number of cores.
<code>tomcatMemoryFraction</code>	A number between 0 and 1 that indicates the proportion of memory to allocate to the Coverity Connection Web application.

Profile settings are derived from three sources (arranged from lowest to highest precedence):

1. The current environment detected by the application, which is overridden by
2. Values provided on the `cov-admin-db tune` command line using <key=value>, which are overridden by
3. Values set using environment variables

For example, the following sequence of commands illustrate how profile settings can be overridden:

```
$ cov-admin-db tune --show-profile
isExternalDb = false
isSsd = false
mode = default
os = Linux
physicalMemory = 16g
processorCount = 12
```

Entering the following command:

```
$ cov-admin-db tune --show-profile processorCount=10
```

Displays the profile as follows:

```
isExternalDb = false
isSsd = false
mode = default
os = Linux
physicalMemory = 16g
processorCount = 10
```

Executing the following commands

```
$ export physicalMemory=10g
```

```
$ cov-admin-db tune --show-profile physicalMemory=20g
```

Displays the physical memory of 10g because the environment variable setting overrides the command line setting.

```
isExternalDb = false
isSsd = false
mode = default
os = Linux
physicalMemory = 16g
processorCount = 10
```

How you use tune options depends on your use case:

- If your environment has changed and you simply want to retune your system for these changes, use the `cov-admin-db tune --write` command.
- If you want to retune for testing or to limit the use of available resources, use the `cov-admin-db tune --write < key=value >` command.



#### Note

Remember to restart Coverity Connect after retuning.

### Understanding the Difference Between `--write` and `--suggest`

This section explains how system settings are derived from the system profile, and how you use the `--suggest` and `--write` options to display or change these settings.

1. As mentioned before, the system's profile is assembled from three sources: current settings, values set on the `cov-admin-db` command line, and values set using environment variables.

To display the profile use the `--show-profile` option.

2. A calculator is chosen based on the profile's `mode` setting. (The `default` mode is normally the one used.)
3. The profile is provided as input to the calculator.
4. Based on this input, the calculator outputs a collection of settings for the JVM and database.
  - `--suggest` outputs these settings to the console.

- `--write` writes these settings to their configuration files.

## Upgrading the Database

Use the `upgrade-schema` subcommand to upgrade your database schema: The command modifies the existing schema and data to make it compatible with the current version of Coverity Connect.

The `upgrade-schema` subcommand supports an external PostgreSQL database or the embedded database. Because the `cov-admin-db restore` command and the installer always upgrade the schema, you need to run `cov-admin-db upgrade-schema` only when using an external database.

---

## Name

`cov-archive` Export streams to an archive file; import streams from an archive file; get information about an archive file.

## Synopsis

This command has three variants depending on whether you want to export, import, or get information. A fourth variant returns help information about one of the first three variants.

```
cov-archive [--debug] export-streams [--remove [--silent]] --archive
<archive-file> [--project <project-name>...]... [--stream <stream-
name>...]...
```

```
cov-archive [--debug] import-streams --archive <archive-file> [--cluster-
config <cluster-config-file>]
```

```
cov-archive [--debug] list --archive <archive-file>
```

```
cov-archive help [<command>]
```

```
cov-archive -h
```

```
cov-archive --help
```

## Description

The `cov-archive` commands allow you, the system administrator, to export a set of streams into an archive file and optionally delete the exported streams, import streams from an archive, or get information about an archive file (its identifier, version, the date and time of creation, the streams contained, and so on) or a cluster config file (the identifier of the corresponding archive file, the identifier of the corresponding Coverity Connect coordinator, the date and time of creation). You can also get help about `cov-archive` command options.

You may import an archive into a Coverity Connect instance that has the same or a newer version as the Coverity Connect instance used to create the archive. You can check the Coverity Connect version used to create the archive using the `cov-archive list` command.

The `cov-archive` command generates a log file `<CC_install_dir>/logs/cov-archive.log`.

### Important

You may use the `import-streams` command of `cov-archive` only while Coverity Connect is in maintenance mode. You can run the `export-streams` command while Coverity Connect is operational.

### Important

The command `import-streams` has exactly two outcomes: it either completes successfully or fails without doing any observable changes to the Coverity Connect database:

- If the command completes without reporting an error, then the outcome is success.

- If the command reports an error, then the outcome may be either success or failure. In the majority of cases the reported error means that the command failed, but in some rare cases (for example, if the command issued a commit but did not receive a response for some reason) the only way to tell whether the outcome is success or failure is by checking whether the database contains a stream from the archive or not. Such checking may be done by executing the command again—if the first command succeeded, then the second one fails by reporting that the streams already exist.

## Options

`--archive <archive-file>`

The path of the file to which you are exporting or from which you are importing. The path is either absolute or relative to the shell's current working directory.

`--cluster-config <cluster-config-file>`

The path to the cluster config file. The path is either absolute or relative to the shell's current working directory. This file is created as a result of completing the first step of the two-step import into a subscriber Coverity Connect instance and is required to do the second step. This option must be specified for each of the two steps. If this option is not specified when the command is executed on the coordinator (step one), then the command imports an archive into the coordinator instead of producing a cluster config file.

Importing into a standalone/coordinator Coverity Connect instance is a straightforward action. Importing into a subscriber Coverity Connect instance is a two-step process.

Step one is executed on the coordinator instance in the Coverity Connect cluster that contains the target subscriber instance and produces a cluster config file required to do the second step.

Step two is executed on the target subscriber instance only after this instance catches up with the changes introduced on the coordinator by the first step. These changes are specified by the cluster config file. Importing is refused if the data specified in the file is not present in the target subscriber. You may see whether the subscriber caught up with its coordinator by navigating to "Help">"System Diagnostics">"Cluster" and looking at the "Last synchronized" timestamp.

Note that it is allowed to have the same streams in different Coverity Connect instances in a cluster. This means streams from the archive are allowed to exist in the coordinator when doing step one, and the streams existing in the coordinator are allowed to be imported to its subscriber when doing step two.

Before deciding to import into a coordinator/subscriber instance you should make sure that the data replication between them is happening successfully. This can be done informally by checking that the aforementioned "Last synchronized" timestamp is not too old, e.g., that its value is within the last 24 hours. Do not import into a coordinator/subscriber if there appears to be an issue with the data replication process between them because doing so may only further complicate the issue.

`<command>`

One of the following: `export-streams`, `import-streams`, `list` or `help`. If nothing is specified, it is interpreted as `help`.

`--project <project-name>`

The name of the project from which you want to export streams. You must specify either a project or a stream; you may specify both project and stream names.

`--remove`

Remove the exported streams from the database, when exporting successfully completes.

Note that the data belonging to the streams is deleted in the background while Coverity Connect is running. This does not prevent using the `--remove` option while Coverity Connect is in maintenance mode: the data will eventually be deleted once Coverity Connect starts. You can run `vacuum full` later if you need to return the freed up storage space to the OS. We recommend setting `cim.cleanup.stream.delay.min = 2` in `cim.properties` if you have a significantly higher value specified explicitly in `cim.properties` and you are going to delete large number of streams. See *Coverity Platform User and Administrator Guide* for more details about this property.

`--silent`

Suppress confirmation of the action. This option may be specified only when using the `--remove` option.

`--stream <stream-name>`

The name of the stream you want to export. You must specify either a project or a stream; you may specify both project and stream names.

## Limitations and considerations

Please observe the following limitations and considerations in using the `cov-archive` command.

### Limitations

- The following data is not exported: cross references, issue categorization maps, component maps, components, licenses and users ("Committed by") associated with snapshots, "Bind Password" of LDAP configurations (it must be set after importing either manually or via the Web Services API - see the *Web Services API Reference* for more information).
- You cannot import individual streams from an archive.

### Considerations

- Imported triage stores (and associated data) are not merged. Instead, a new triage store is created in the target database.
- Attribute definitions in the target database must be the same (and defined using the same index order), otherwise the import will fail.
- LDAP configurations are merged based on their display name. If the target database has a configuration with the same UUID but a different display name, the import will fail.
- Users are imported in a disabled state, and user preferences are not exported.
  - If a user in the archive is already present in the target database, that user will not be disabled and preferences will be preserved.

- Users are merged if they have the same login name, deleted status, and LDAP configuration (which may or may not be merged, as described above).
- Only stream-level user/group role assignments are exported. Role permissions are not exported because importing them may change access to other entities in the target Coverity Connect instance. When importing role assignments, each role is imported (without its permissions) if it does not exist in the target database, otherwise the existing role is used.
- CIDs of imported issues are preserved unless they are empty or they belong to a different pre-existing issue.

## Exit codes

- 0: The command successfully completed.
- 1: Either an error occurred or help was requested.

## Examples

The following command displays the usage help for the command `export-streams`.

```
cov-archive help export-streams
```

The following command exports a stream named `s1` and all streams linked to the projects named `p1` and `p2` to a file `../s1_p1_p2.covarch`.

```
cov-archive export-streams --stream s1 --project p1 --project p2
--archive ../s1_p1_p2.covarch
```

The following command imports streams from the file `../s1_p1_p2.covarch` and writes detailed logs.

```
cov-archive --debug import-streams --archive ../s1_p1_p2.covarch
```

The two commands that follow import streams from the file `../s1_p1_p2.covarch` into a subscriber in a two-step process.

Step one: Execute on the coordinator.

```
cov-archive import-streams --archive ../s1_p1_p2.covarch --cluster-config ../
s1_p1_p2.covclustcfg
```

Step two: Wait until the target subscriber catches up with the coordinator, then execute on the subscriber.

```
cov-archive import-streams --archive ../s1_p1_p2.covarch --cluster-config ../
s1_p1_p2.covclustcfg
```

---

## Name

`cov-get-certs` Create a file of trusted self-signed certificates.

## Synopsis

```
cov-get-certs --host <host> --port <port> --certs <certs_file>
```

## Description

In situations where Coverity Connect is a client of other services (email, Bugzilla, JIRA, LDAP) and one or more of those services uses a self-signed certificate, `cov-get-certs` is used to transfer that server's self-signed certificate to Coverity Connect's trust store of CA root certificates, thus enabling Coverity Connect to connect using SSL to the service.

Before the Coverity 8.0 release, `cov-get-certs` was needed for all Java applications. Now it is needed only for Coverity Connect

If you want to edit the certificate file, use your JRE's `keytool` command. The password for the certificates file is `changeit`.

## Options

`--host <server-host>`

The server hostname, which Coverity Connect is a client of.

`--port <server-port>`

The host server's HTTPS connection port.

`--certs <cert_file>`

The name of the certificate file. Because this defaults to Coverity Connect's Java trust store (at `<Coverity Connect install directory>/jre/lib/security/cacerts`), you don't normally need to use this option.

## Example

```
cov-get-certs --host example.com --port 8443
```

---

## Name

cov-im-ctl Start or stop Coverity Connect.

## Synopsis

```
cov-im-ctl {maintenance | start | status | stop} [-w <seconds>]
```

## Description

The `cov-im-ctl` command performs various operations, including starting or stopping Coverity Connect.

On Windows systems, when Coverity Connect is installed as a service, the `cov-im-ctl.exe` program is often unnecessary because Coverity Connect starts and stops automatically when the system boots up or shuts down. When Coverity Connect is installed as a service, any administrator can use this command.

When Coverity Connect is not installed as a service, only the user who installed Coverity Connect is able to use this program to start or stop it.

## Options

`maintenance`

Place the embedded database in maintenance mode. Use this option, for example, before you restore a database from backup.

`start`

Start Coverity Connect.

`status`

Provide status information about Coverity Connect.

`stop`

Stop Coverity Connect.

`-w <seconds>`

Wait at least the specified number of seconds for a response to this request.

## Examples

Check to see if Coverity Connect is running:

```
> cov-im-ctl status
```

Stop Coverity Connect:

```
> cov-im-ctl stop
```

Start Coverity Connect:

```
> cov-im-ctl start
```

---

## Name

cov-import-cert Import certificate for SSL communication for coordinator/subscribers.

## Synopsis

```
cov-import-cert <cert_file> <truststore_file>
```

## Description

`cov-import-certs` allows you add the certificate into the user's own truststore, to allow authentications between a coordinator and a subscriber. If the truststore does not exist, one will be created. For more information, see *Configuring Coverity Connect enterprise clusters* in the *Coverity Platform 2020.12 User and Administrator Guide* [↗](#).

## Options

<cert\_file>

The name of the certificate file.

<truststore\_file>

The truststore file shared for SSL communication. If the truststore does not exist, one will be created.

---

## Name

cov-manage-im Manage and query defects, projects, and streams in Coverity Connect.

## Synopsis

```
cov-manage-im --mode defects [MODE OPTIONS][CONNECTION OPTIONS][SHARED
OPTIONS]
```

```
cov-manage-im --mode projects [MODE OPTIONS][CONNECTION OPTIONS][SHARED
OPTIONS]
```

```
cov-manage-im --mode streams [MODE OPTIONS][CONNECTION OPTIONS][SHARED
OPTIONS]
```

```
cov-manage-im --mode triage [MODE OPTIONS][CONNECTION OPTIONS][SHARED
OPTIONS]
```

```
cov-manage-im --mode motd [MODE OPTIONS][CONNECTION OPTIONS][SHARED OPTIONS]
```

```
cov-manage-im --mode commit [MODE OPTIONS][CONNECTION OPTIONS][SHARED
OPTIONS]
```

```
cov-manage-im --mode notification [MODE OPTIONS][CONNECTION OPTIONS][SHARED
OPTIONS]
```

```
cov-manage-im --mode auth-key [MODE OPTIONS][CONNECTION OPTIONS][SHARED
OPTIONS]
```

## Description

The `cov-manage-im` command modifies and queries information for defects, projects, and streams in Coverity Connect. This command also outputs logging information to `<install_dir_cc>/logs/cim.log`.

This command has the following modes of operation:

- Defects mode
- Projects mode
- Streams mode
- Triage mode
- MOTD mode
- Commit mode
- Notification mode

- Authentication key mode

The `cov-manage-im` command can operate on (update or delete) the set of objects that were matched by a query within a single command line.

Each `cov-manage-im` mode accepts CONNECTION options that allow you specify connection settings such as host name, port number, and so forth, on the command line.

Alternatively, you can use the `coverity_config.xml` file, which is a configuration file that you can edit to store connection options for `cov-manage-im`. If you run the `cov-manage-im` command from a Coverity Analysis or Coverity Analysis package, you can create a default version of this file at `<install_dir_sa>/config/coverity_config.xml` by running the `cov-configure` command. After you run the `cov-configure` command, you must add the elements as in the example that follows, if you want to include connection options in the configuration file instead of on the command line. If you run the `cov-manage-im` command from a Coverity Connect package, you must manually create the default `coverity_config.xml` file, and move it to `<install_dir_cc>/config/coverity_config.xml`.

The following example element in the `coverity_config.xml` file defines connection options for Coverity Connect:

```
<!DOCTYPE coverity SYSTEM "coverity_config.dtd">
<coverity>
 <config>
 <cim>
 <host>cim.company.com</host>
 <port>8443</port>
 <!-- HTTPS port -->

 <client_security>
 <user>test</user>
 <password>secret</password>
 <ssl>yes</ssl>
 <certs>/pathto/.certs</certs>
 </client_security>
 </cim>
 </config>
</coverity>
```



#### Note

Glob arguments are patterns used for filter expressions. In glob patterns, `*` matches zero or more characters, and `?` matches exactly one character.

## Defects mode

Query and update defects in Coverity Connect.

### Synopsis

```
--mode defects --show <SCOPE> [<FILTER>] [<OUTPUT>] [<OTHER>]
```

```
--mode defects --update <SCOPE> [<FILTER>] <SET> [<OTHER>]
```

## Defects mode options

In general, you can specify options in any order. The exception is when you add more than one project or stream within a single command. In this case, you must specify the options for the properties of each new stream or project at the same time.

The <OTHER> option listed in the synopsis refers to sets of command line options that are common to all modes. The options are:

- Common OUTPUT options
- CONNECTION options
- Shared options

### OPERATION options

Exactly one OPERATION option is required:

`--show`

Output a comma separated value (CSV) list of defects from the specified scope that matches the filter criteria. Use the `--fields` option to control the display of the defect fields and their order.

`--update`

Update attributes of defects from the specified scope that match the filter criteria. At least one SET option is required. FILTER options are not required.

### SCOPE

Define the project or the set of streams to show or update. All defect operations require a scope definition.

When the same defect occurs in multiple streams, and a project scope is specified, the defects are represented as a *merged defect* for display and filtering purposes. For example, if the same defect occurs in two separate streams, and the action attribute values are different, Coverity Connect calculates a merged action. The merged action is then displayed for the merged defect. FILTER options match against the merged defect.

The SCOPE options are:

`--project <name>`

Specify the name of a project that contains the defects that you want to show or update. You can only scope for one project.

`--stream <name>`

Specify the name of a stream, or streams.

### FILTER

FILTER options focus the set of defects in the given scope that will be operated on. You can specify multiple instances of each filter, with the exception of the `--file` and `--function` filters, which you can only specify once.

If you specify multiple instances of the same filter, the values are effectively ORed together. Different filter options are effectively ANDed together. For example, `--action a --action b --severity c --severity d` matches defects of the criteria `(action = a OR b) AND (severity = c OR d)`.

The FILTER options are:

`--cid <cid-set>`

Operate on a single CID or set of CIDs. `<cid-set>` can be:

- A single CID. For example, `--cid 10118`.
- A range of CIDs. Denote range with a hyphen (-). For example, `--cid 10203-10209`.
- A comma-separated list of single CIDs and ranges of CIDs. For example, `--cid 10118,10119,10203-20109,10388`.

`--action <action>`

Operate on defects whose action attribute value exactly matches the `<action>` string. Valid strings match the list of Action values on the Coverity Connect *Attribute Details* screen. The standard (unedited) values in Coverity Connect are:

- Undecided
- Fix Required
- Fix Submitted
- Modeling Required
- Ignore

Use `Various` for `<action>` to select merged defects that have different action attribute values.

`--classification|--class <class>`

Operate on defects whose classification attribute value exactly matches the `<class>` string.

`<class>` must be one of the following:

- Unclassified
- Pending
- False Positive
- Intentional
- Bug
- Various

Use `Various` for `<classification>` to select merged defects that have different classification attribute values.

For Test Advisor policy violations, the classification must be one of the following:

- Unclassified
- Pending
- Untested
- No Test Needed
- Tested Elsewhere
- Various

`--severity <severity>`

Operate on defects whose severity attribute value exactly matches the `<severity>` string. Valid strings match the list of severity levels on the Coverity Connect *Attribute Details* screen. The standard (unedited) values in Coverity Connect are:

- Unspecified
- Major
- Moderate
- Minor

Use `Various` for `<severity>` to select merged defects that have different severity attribute values.

`--status <status>`

Operate on defects whose status attribute value exactly matches the `<status>` string.

`<status>` must be one of the following:

- New
- Triaged
- Dismissed
- Fixed

`--component <comp_map_name>.<comp_name>`

Operate on defects found by a specified component name. The name of the component map and component must match the `<comp_map_name>.<comp_name>` string.

`--component-not <comp_map_name>.<comp_name>`

Causes defects found by a specified component name to be excluded from the result. The name of the component map and component must match the `<comp_map_name>.<comp_name>` string. You can specify multiple component names.

`--checker <checker>`

Operate on defects found by a specific checker. The name of the checker must match the `<checker>` string. For example:

```
--checker FORWARD_NULL
```

`--external-reference|--ext-ref <ext-reference>`

Operate on defects whose external reference exactly matches the `<ext-reference>` string.

`--language|--lang < lang>`

Operate on a stream, or streams, based on language. `<lang>` can be one of the following:

- C/C++, `cpp`
- Java, `java`
- C#, `cs`
- `dynamic_java`
- Mixed, `mixed`
- Other, `other`

`--mergekey <mergekey>`

Operate on defects whose mergekey exactly matches the `<mergekey>` string. You may specify multiple mergekeys in a comma-separated list.

`--owner <owner>`

Operate on defects whose owner exactly matches the `<owner>` string.

Use `Unassigned` for `<owner>` to update defects that are not yet assigned.

`--file <file>`

Operate on defects whose file matches the `<file>` glob pattern. `<file>` refers to the terminal part of a filename, not a full path. You can specify this option only once.

The glob `<file>` `<pattern>` refers to the entire pathname. Pathnames are separated by a slash on all platforms. For example, `--file *.java` will match all Java files, and `--file Win.java` will match only `Win.java` at the root of the source tree. But `--file */Win.java` will match all files named `Win.java` in all directories.

`--function <function>`

Operate on defects whose function matches the `<function>` glob pattern. You can specify this option only once.

`--legacy <status>`

Operate on defects whose legacy status matches the `<status>` string. Legacy status can be any of the following values:

- `True`

- False
- Various

**--newest [snapshotId]**

Operate on defects which occur only in the project's most recent snapshot.

An optional parameter, [snapshotId], will compare the newest snapshot with the older specified snapshot. The snapshot ID number must match the number in [snapshotId] exactly.

**OUTPUT options**

OUTPUT options are not required and are only valid with the `--show` option.

**--newest [snapshotId]**

Outputs those defects which occur only in the project's most recent snapshot.

An optional parameter, [snapshotId], will compare the newest snapshot with the older specified snapshot. The snapshot ID number must match the number in [snapshotId] exactly.

**--output fields**

Display the list of valid field names for this mode. These field names can then be used with the `--fields` option.

**--page**

Sets the number of defects that are pulled per batch. The default number of defects is 100. If you set the `--page` option to 1000 then there will be fewer queries and performance can improve. For example, by setting the `--page` option to 1000 you will get 20 queries rather than 200, so you will get 10 times fewer.

Example:

```
cov-manage-im --host localhost --port 8080 --user admin --password coverity --mode defects --show --project foo --page 1000
```

**Output options**

You can also specify Output options that are common to all modes.

**SET options**

SET options update defect attributes of the selected defects defined with the FILTER options. You can use only one `--set` option for each defect attribute, such as action, severity, classification, and so forth. However, you can specify `--set` options for different defect attributes on the same command line.

**--set action:<action>**

Set the action defect attribute for defects that match the filter criteria to <action>. The action attribute used for <action> must already exist in the Coverity Connect database.

**--set {classification|class}:<class>**

Set the classification attribute of defects that match the filter criteria to <class>.

The <class> string must be one of the following:

- Unclassified
- Pending
- False Positive
- Intentional
- Bug

For Test Advisor policy violations, the classification must be one of the following:

- Unclassified
- Pending
- Untested
- No Test Needed
- Tested Elsewhere
- Various

`--set severity:<severity>`

Set the severity defect attribute of defects that match the filter criteria to `<severity>`. The severity attribute used for `<severity>` must already exist in the Coverity Connect database.

`--set owner:<owner>`

Set the owner of defects that match the filter criteria to `<owner>`. The owner attribute used for `<owner>` must already exist in the Coverity Connect database.

`--set comment:<comment>`

Add a comment to the defects that match the filter criteria.

`--set {ext-ref|external-reference}:<reference>`

Add an external reference to the defects that match the filter criteria.

`--set legacy:<status>`

Set the legacy status for the defect. The acceptable values are:

- True
- False

## Defects mode examples

The first example shows the four connection options that must contain values either on the command line, or in the XML configuration file (host, port, user, password). These connection options are intentionally dropped from most of the subsequent examples to reduce the length of the command lines. When the connection options are not specified, assume that the values are retrieved from the default XML configuration file.

### Show examples

Show all (merged) defects in project X.

```
> cov-manage-im --host cim.company.com --port 8080 --user test \
 --password secret --mode defects --show --project X
```

Show all open defects in project X

```
> cov-manage-im ---mode defects --show --project X \
 --status New --status Triaged
```

Show all defects with the specified mergekey in the stream named Y.

```
> cov-manage-im --mode defects --show --stream Y --mergekey
46941efa13559f40754b0d90dd99f2d2
```

List the defect fields that can be passed to --fields in defects mode.

```
> cov-manage-im --mode defects --show --project P --output fields
```

Control what defect fields are shown by specifying some of these fields with the --fields option.

```
> cov-manage-im --mode defects --show --project P \
 --fields cid,action,severity
```

Show particular (merged) defects in project X, filtering with different CID specifiers.

```
> cov-manage-im --mode defects --show --project X --cid 123
 > cov-manage-im --mode defects --show --project X --cid 123,456
 > cov-manage-im --mode defects --show --project X --cid 1-4
 > cov-manage-im --mode defects --show --project X --cid 1-4,18,25-30
```

Show all Triaged defects in stream Y that are classified as Bugs.

```
> cov-manage-im --mode defects --show --stream Y \
 --status Triaged --class Bug
```

Show all open defects in stream Y with no owner.

```
> cov-manage-im --mode defects --show --stream Y \
 --status New --status Triaged --owner Unassigned
```

Show all open defects in "client" source files.

```
> cov-manage-im --mode defects --show --stream Y \
 --status New --status Triaged --file "client-*.c"
```

Show all open defects listed by CID in all components.

```
cov-manage-im --mode defects --show --project project1 \
 --fields cid,component
```

### Update examples

Assign all unassigned defects in streams X and Y to jdoe

---

## cov-manage-im

---

```
> cov-manage-im --mode defects --update --stream X \
 --stream Y --owner Unassigned --set owner:jdoe
```

Set the classification of CID 10002 in project P to False Positive.

```
> cov-manage-im --mode defects --update --project P \
 --cid 10002 --set "class:False Positive"
```

Set all the attributes of CID 10002 in project P.

```
> cov-manage-im --mode defects --update --project P \
 --cid 10002 --set "action:Fix Required" --set class:Bug \
 --set severity:Major --set owner:jdoe \
 --set "comment:This appears to be real." \
 --set ext-ref:yyy
```

Set the classification of defects with the specified mergekey in stream Y to False Positive.

```
> cov-manage-im --mode defects --update --stream Y \
 --mergekey 46941efa13559f40754b0d90dd99f2d2 --set "class:False Positive"
```

Mark all defects in MyStream which were introduced in the most recent snapshot as Legacy=true.

```
> cov-manage-im --mode defects --stream MyStream --update --newest --set legacy:true
```

Mark all defects in MyStream which were introduced after snapshot 10006 as Legacy=true.

```
> cov-manage-im --mode defects --stream MyStream --update --newest 10006 --set
 legacy:true
```

## Projects mode

Query, add, delete, and update projects in Coverity Connect.

### Synopsis

```
--mode projects --show [<FILTER>] [<OUTPUT>] [<OTHER>]
```

```
--mode projects --add <SET> [<OTHER>]
```

```
--mode projects --delete <FILTER> [<OTHER>]
```

```
--mode projects --update <FILTER> <SET> [<OTHER>]
```

```
--mode projects --update <FILTER> <REMOVE> [<OTHER>]
```

### Projects mode options

In general, you can specify options in any order. The exception is when you add more than one project within a single command. In this case, you must specify the options for the properties of each new project at the same time.

The <OTHER> option listed in the synopsis refers to sets of command line options that are common to all modes. The options are:

- Common OUTPUT options
- CONNECTION options
- Shared options

#### OPERATION options

Specify exactly one OPERATION option.

##### --show

Output a comma separated value (CSV) list of projects that match the filter criteria. Use the `--fields` option to control the display of the project fields and their order. FILTER options are not required with `--show`.

##### --add

Add one or more new projects. A project name is required for each project that is added. You can add multiple projects with a single command line by specifying groups of SET options for each new project.

##### --delete

Delete the project or projects that match the filter criteria. At least one FILTER option is required.

##### --update

Update the attributes of projects that match the filter criteria. At least one FILTER option and one SET/REMOVE option is required.

#### FILTER options

FILTER options focus the set of projects that are operated on. You can specify multiple instances of each filter, except for `--name` and `--description`.

If you specify multiple instances of the same filter, the values are effectively ORed together. Different filter options are effectively ANDed together. For example, `--stream a --stream b --description c --description d` matches projects of the criteria  $(stream = a \text{ OR } b) \text{ AND } (description = c \text{ OR } d)$ .

The FILTER options are:

##### --name <glob>

Operate on a project, or projects, whose name matches the specified glob pattern.

##### --description|--desc <glob>

Operate on a project, or projects, whose description matches the specified glob pattern.

##### --stream <glob>

Operate on projects that are associated with a stream whose name matches the glob pattern.

#### OUTPUT options

OUTPUT options are not required and are only valid with the `--show` operation.

The OUTPUT options are:

**--output fields**

Display the list of valid field names for the Projects mode. The field names can then be used with the `--fields` option.

**--output streams**

Display information about each stream associated with the project, rather than the information about the project itself.

**Output Options**

You can also specify Output options that are common to all modes.

**SET options**

The SET options apply changes to project attributes. Use `--add` to set the attributes of new projects, or `--update` to update the attributes of existing projects. At least one SET option is required with `--update`.

**--set name:<name>**

Specify a name for a new project with the `--add OPERATION` option. This option is required for each project being added with `--add`. For example:

```
--add --set name:Project1
```

Update the name of an existing project using the `--update OPERATION` option. For example:

```
--update --name A --set name:B
```

Project names must be between 1 and 256 characters and are case insensitive. Project names can not contain the following characters:

- : (colon)
- \* (asterisk)
- / (forward slash)
- \ (back slash)
- ` (backtick)
- ' (single quote)
- " (double quote)

**--set {description|desc}:<description>**

Specify a description for a new project using the `--add OPERATION` option, or update the description of an existing project using the `--update OPERATION` option.

**--insert stream:<name>**

Associate an existing stream with a new project using the `--add OPERATION` option, or with an existing project using the `--update OPERATION` option.

This option does not create the specified stream, or streams. The streams must already exist in Coverity Connect. You can specify multiple `--insert stream` options to associate multiple streams with a project with a single command.

A stream has two types of associations with a project:

- Primary, in which a given stream is associated with a designated primary project.
- Linked, in which a non-primary project is associated with the given project through a stream link.

See [Default Triage Scope](#) for more information about primary projects and stream links.

When a stream is associated with a primary project (`ProjectA`) and then inserted into another project (`ProjectB`), its association with `ProjectA` is changed from a primary association to a stream link. `ProjectB` becomes the stream's primary project. Although `cov-manage-im` cannot explicitly create stream links, this mechanism can be used to create a stream link.

To help you locate primary and linked associations, stream listings in Streams mode have a column called `Primary Project`. This column contains the name of the primary project that is associated with the stream (if any). Additionally, in Projects mode, you can specify `is-stream-linked` in the `--fields` option. This produces a column that displays `yes` if the stream has a linked association, or `no` if it has a primary association.

#### REMOVE options

The REMOVE options remove stream associations from projects. Remove options work for both primary and linked streams.

These options are only valid with the `--update OPERATION` option.

At least one FILTER option must be specified to prevent accidental bulk removals.

The REMOVE options are:

`--remove stream:<name>`

Remove a stream association from the specified projects. Streams are not removed from the Coverity Connect. Only the project's association with the stream is removed.

The `<name>` argument must exactly match a stream name.

You can specify multiple `--remove stream` options.

`--clear streams`

Remove all stream associations from the selected project(s).

#### Projects mode examples

The first example shows the four connection options that must contain values either on the command line, or in the XML configuration file (host, port, user, password). These connection options are intentionally dropped from most of the subsequent examples to reduce the length of the command lines.

When the connection options are not specified, assume that the values are retrieved from the default XML configuration file.

### Show examples

Show all projects.

```
> cov-manage-im --host cim.company.com --port 8080 --user test \
 --password secret --mode projects --show
```

Show all projects that contain a stream named x or y.

```
> cov-manage-im --mode projects --show --stream x --stream y
```

List the fields that can be passed to `--fields` in projects mode.

```
> cov-manage-im --mode projects --show --output fields
```

Next, control what fields are shown by specifying some of these fields with the `--fields` option.

```
> cov-manage-im --mode projects --show --fields project
```

Show stream associations for all projects.

```
> cov-manage-im --mode projects --show --output streams
```

Show stream associations for projects where the project name starts with 'a'.

```
> cov-manage-im --mode projects --show --output streams --name "a*"
```

### Add examples

Add a new project with minimal attributes specified.

```
> cov-manage-im --mode projects --add --set name:"HelloWorld"
```

Add a new project with all attributes specified

```
> cov-manage-im --mode projects --add \
 --set name:"hello world" \
 --set desc:"A full project" \
 --insert stream:mystream
```

Add two new projects at the same time.

```
> cov-manage-im --mode projects --add \
 --set name:proj1 \
 --set desc:"First project" \
 --insert stream:mystream \
 --set name:proj2 \
 --set desc:"Second project"
```

### Delete examples

Delete a project named `old-project`.

```
> cov-manage-im --mode projects --delete --name old-project
```

### Update examples

Rename project A to B and update description at the same time.

```
> cov-manage-im --mode projects --update --name A \
 --set name:B --set "desc:This is now a B project"
```

Add stream associations to project P's existing stream associations.

```
> cov-manage-im --mode projects --update --name P \
 --insert stream:x
```

Remove all stream associations named x from project P .

```
> cov-manage-im --mode projects --update --name P --remove stream:x
```

Remove all stream associations from project P

```
> cov-manage-im --mode projects --update --name P \
 --clear streams
```

### Streams mode

Query, add, delete, and update streams in Coverity Connect.

#### Synopsis

```
--mode streams --show [<FILTER>] [<OUTPUT>] [<OTHER>]
```

```
--mode streams --add <SET> [<OTHER>]
```

```
--mode streams --delete <FILTER> [<OTHER>]
```

```
--mode streams --update <FILTER> <SET> [<OTHER>]
```

#### Streams mode options

In general, you can specify options in any order. The exception is when you add more than one stream within a single command. In this case, you must specify the options for the properties of each new stream at the same time.

The <OTHER> option listed in the synopsis refers to sets of command line options that are common to all modes. The options are:

- Common OUTPUT options
- CONNECTION options
- Shared options

#### OPERATION options

Specify exactly one OPERATION option in Streams mode.

**--show**

Output a comma separated value (CSV) list of streams that match the filter criteria. Use the `--fields` option to control the display of the stream fields and their order. FILTER options are not required with `--show`.

**--add**

Add new streams. A minimum of one stream name is required for each stream that you add.

You can add multiple streams with a single command by specifying groups of SET options for each new stream.

**--delete**

Delete the streams that match the filter criteria. At least one FILTER option is required.

**--update**

Update the name or description of the streams that match the filter criteria. A stream's language can not be updated after a stream has been created. At least one FILTER option and one SET option is required.

**FILTER options**

FILTER options focus the set of streams that are operated on. You can specify multiple instances of each filter, except for `--name` and `--description`.

If you specify multiple instances of the same filter, the values are effectively ORed together. Different filter options are effectively ANDed together. For example, `--project a --project b --language c --language d` matches streams of the criteria `(project = a OR b) AND (language = c OR d)`.

**--name <glob>**

Operate on a stream, or streams, whose name matches the glob pattern.

**--language|--lang <lang>**

Operate on a stream, or streams, based on language. `lang` can be one of:

- C/C++, `cpp`
- Java, `java`
- C#, `cs`
- `dynamic_java`
- Mixed, `mixed`
- Other, `other`

**--description|--desc <glob>**

Operate on a stream, or streams, whose description matches the glob pattern.

**--stream <glob>**

Operate on a stream, or streams, whose name matches the glob pattern.

`--project <glob>`

Operate on streams that are associated with projects whose name matches the glob pattern.

#### OUTPUT options

OUTPUT options are optional and are valid only with the `--show` option.

`--output fields`

Display the list of valid field names for this mode. These field names can then be used with the `--fields` option.

#### Output Options

You can also specify Output options that are common to all modes.

#### SET options

The SET options apply changes to stream attributes. Use `--add` to set the attributes of new streams, or `--update` to update the attributes of existing streams. At least one SET option is required with `--update`.

`--set {component-map|cmap}:<component-map>`

Specify a component map to associate with a stream using the `--add OPERATION` option.

`--set {description|desc}:<description>`

Specify a description for a new stream using the `--add OPERATION` option, or update the description of a stream using the `--update OPERATION` option.

`--set desktopAnalysis:{disabled|enabled}`

This option allows (or prohibits) the stream to provide data for Desktop Analysis. Only streams that have specifically enabled Desktop Analysis can be used as reference streams for Desktop Analysis users.

To set this option for newly created streams, use `--add`, for example `--mode streams --add --set desktopAnalysis:enabled`.

To set this option for existing streams use `--update`, for example `--mode streams --update --set desktopAnalysis:enabled`.

These options are disabled by default.

For more information, see the *Coverity Platform 2020.12 User and Administrator Guide* [🔗](#)

`--set expiration:{disabled|enabled}`

This option allows you to set Coverity Connect to automatically delete streams after a period of inactivity. Only streams that are specifically configured for this feature are eligible for automatic deletion.

To set this option for newly created streams, use `--add`, for example `--mode streams --add --set expiration:enabled`.

To set this option for existing streams use `--update`, for example `--mode streams --update --set expiration:enabled`.

These options are disabled by default.

For more information, see "Designating a stream for auto-deletion of expired streams" in the *Coverity Platform 2020.12 User and Administrator Guide* [🔗](#)

`--set {language|lang}: <lang>`

Specify a language for a new stream using the `--add OPERATION` option. `--set language` can be used for each stream you add. If you do not set a language for the stream, it defaults to `mixed`. You can not update the language of an existing stream with this option. You should choose the default (`mixed`) for each new stream that you create. The other languages are provided for backward compatibility for previously released Coverity Connect versions (in which implicitly designated languages were required for streams). The valid languages are:

- `Mixed`, `mixed`
- `C/C++`, `cpp`
- `Java`, `java`
- `C#`, `cs`
- `dynamic_java`
- `Other`, `other`

 **Note**

`Other` can be used when creating a stream with the `--add` option. `Other` is among the languages that may be shown when displaying a stream with the `--show` option.

`--set name:<name>`

Specify a name for a new stream with the `--add OPERATION` option. This option is required for each stream added with `--add`. For example

```
--add --set name:Stream1
```

Update the name of an existing stream using the `--update OPERATION` option. For example:

```
--update --name A --set name:B
```

Stream names must be between 1 and 256 characters and are case sensitive. Stream names can not contain the following characters:

- `:` (colon)
- `*` (asterisk)
- `/` (forward slash)
- `\` (backslash)

- ` (backtick)
- ' (single quote)
- " (double quote)

`--set ownerAssignmentOption:{default_component_owner|scm|none}`

This option allows you to set owner assignment options for a stream. If you do not set the `ownerAssignmentOption` for the stream, it defaults to `default_component_owner`. You can update this value to any of the entries mentioned above.

`--set triage:<triage-store>`

Specify a triage store to which the stream will belong. If the triage store is not specified, it defaults to the default triage store.

### Streams mode examples

The first example shows the four connection options that must contain values either on the command line, or in the XML configuration file (host, port, user, password). These connection options are intentionally dropped from most of the subsequent examples to reduce the length of the command lines. When the connection options are not specified, assume that the values are retrieved from the default XML configuration file.

#### Note

The name of the built-in triage store is *Default Triage Store*. Use this triage store unless a different one has been set up.

### Show examples

Show all streams.

```
> cov-manage-im --host cim.company.com --port 8080 --user test \
 --password secret --mode streams --show
```

Show all streams whose name starts with "linux-".

```
> cov-manage-im --mode streams --show --name "linux-*
```

Show all streams whose language is Java.

```
> cov-manage-im --mode streams --show --lang Java
```

List the fields that can be passed to `--fields` in streams mode.

```
> cov-manage-im --mode streams --show --output fields
```

Control the fields that are shown by specifying some of these fields with the `--fields` option.

```
> cov-manage-im --mode streams --show \
 --fields stream,language,desktop-analysis
```

Displays the attributes of stream `mystream`, including the stream's language, `other`.

```
> cov-manage-im --mode streams --show \
 --name mystream
```

### Add examples

Add a new C/C++ stream with minimal attributes specified.

```
> cov-manage-im --mode streams --add \
 --set name:HelloWorld \
 --set lang:mixed \
 --set triage:mytriagestore
```

Add a new stream with all attributes specified, and desktop analysis enabled.

```
> cov-manage-im --mode streams --add \
 --set name:HelloWorld \
 --set lang:mixed \
 --set triage:mytriagestore \
 --set "desc:My stream" \
 --set desktopAnalysis:enabled
```

Add two new streams at the same time

```
> cov-manage-im --mode streams --add \
 --set name:stream1 \
 --set lang:mixed \
 --set desc:"My new stream" \
 --set triage:mytriagestore \
 --set name:stream2 \
 --set desc:"My other new stream" \
 --set triage:mytriagestore
```

Associate the stream1 stream with the component1 component map.

```
> cov-manage-im --mode streams --update --name stream1 \
 -- set component-map:component1
```

Adds a new stream with other.

```
> cov-manage-im --mode streams --add --set --name:mystream \
 lang:other --set triage:mytriagestore
```

### Delete examples

Delete the stream named old-stream (it can only be deleted if defects have NOT yet been committed).

```
> cov-manage-im --mode streams --delete --name old-stream
```

### Update example

Rename stream A to B and update description at the same time.

```
> cov-manage-im --mode streams --update --name A \
 --set name:B --set "desc:This is now a B stream"
```

## Triage mode

Query, add, delete, and update triage stores in Coverity Connect.

### Synopsis

```
--mode triage --show [<FILTER>][<OUTPUT>] [<OTHER>]
```

```
--mode triage --add <SET> [<OTHER>]
```

```
--mode triage --delete <FILTER> [<OTHER>]
```

```
--mode triage --update <FILTER> <SET> [<OTHER>]
```

### Triage mode options

In general, you can specify options in any order. The exception is when you add more than one triage store within a single command. In this case, you must specify the options for the properties of each new triage store at the same time.

The `<OTHER>` option listed in the synopsis refers to sets of command line options that are common to all modes. The options are:

- Common OUTPUT options
- CONNECTION options
- Shared options

#### OPERATION options

Specify exactly one OPERATION option in Triage mode.

##### --show

Output a comma separated value (CSV) list of triage stores and their descriptions that match the filter criteria. Use the `--fields` option to control the display of the triage store fields and their order. FILTER options are not required with `--show`.

##### --add

Add new triage stores. A minimum of one triage store name is required for each triage store that you add.

You can add multiple triage stores with a single command by specifying groups of SET options for each new triage stores.

##### --delete

Delete the triage stores that match the name you specify.

##### --update

Update the name or description of the triage store that match the filter criteria. At least one FILTER option and one SET option is required.

### FILTER options

FILTER options focus the set of triage stores that are operated on. You can specify multiple instances of each filter.

`--name <glob>`

Operate on a triage store, or triage stores, that match the name.

### SET options

The SET options apply changes to triage store names and descriptions. Use `--add` to create a new triage store or description, or `--update` to update the triage store. At least one SET option is required with `--update`.

`--set name:<name>`

Specify a name for a new triage store with the `--add OPERATION` option. This option is required for each triage store added with `--add`. For example

```
--add --set name:triagestore1
```

Update the name of an existing triage store using the `--update OPERATION` option. For example:

```
--update --name triagestore1 --set name:triagestore2
```

Names must be between 1 and 256 characters and are case sensitive. Names can not contain the following characters:

- `:` (colon)
- `*` (asterisk)
- `/` (forward slash)
- `\` (backslash)
- ``` (backtick)
- `'` (single quote)
- `"` (double quote)

`--set {description|desc}:<description>`

Specify an optional description for a new triage store using the `--add OPERATION` option, or update the description of a triage stream using the `--update OPERATION` option.

### Triage mode examples

The first example shows the four connection options that must contain values either on the command line, or in the XML configuration file (host, port, user, password). These connection options are intentionally dropped from most of the subsequent examples to reduce the length of the command lines. When the connection options are not specified, assume that the values are retrieved from the default XML configuration file.

**Note**

The name of the built-in triage store is *Default Triage Store*. Use this triage store unless a different one has been set up.

**Show examples**

Show all triage stores and descriptions.

```
> cov-manage-im --host cim.company.com --port 8080 --user test \
 --password secret --mode triage --show
```

Show individual triage store named mytriagestore .

```
> cov-manage-im --mode triage --show --name mytriagestore
```

**Add examples**

Add a new triage store and description.

```
> cov-manage-im --mode triage --add \
 --set name:mytriagestore \
 --set desc:"This is my new triage store"
```

**Update examples**

Change a triage store name and its description.

```
> cov-manage-im --mode triage --update --name mytriagestore \
 --set name:yourtriagestore \
 --set desc:"This is your new triage store"
```

**Delete example**

Delete the store triage named yourtriagestore (it can only be deleted if it does contain any streams)

```
> cov-manage-im --mode triage --delete --name yourtriagestore
```

**MOTD mode**

Sets and gets a message of the day for a Coverity Connect instance.

**Synopsis**

```
--mode motd --show [<OUTPUT>] [<OTHER>]
```

```
--mode motd --update <SET> [<OTHER>]
```

**MOTD mode options**

In general, you can specify options in any order.

The <OTHER> option listed in the synopsis refers to sets of command line options that are common to all modes. The options are:

- Common OUTPUT options
- CONNECTION options
- Shared options

#### OPERATION options

Specify exactly one OPERATION option in motd mode.

`--show`  
Outputs the current message of the day.

`--update`  
Updates the message of the day.

#### OUTPUT options

OUTPUT options are optional and are valid only with the `--show` option.

`-no-headers`  
Displays the message of the day without a header if `enabled` is specified.

#### SET options

The SET options apply changes the message of the day. Use `--update` to update the message of the day. At least one SET option is required with `--update`.

`--set message:"<message>"`  
Specify a message. For example

```
--set message:"this is the message of the day"
```

### MOTD mode examples

The first example shows the four connection options that must contain values either on the command line, or in the XML configuration file (host, port, user, password). These connection options are intentionally dropped from most of the subsequent examples to reduce the length of the command lines. When the connection options are not specified, assume that the values are retrieved from the default XML configuration file.

#### Show examples

Show message of the day.

```
> cov-manage-im --host cim.company.com --port 8080 --user test \
 --password secret --mode motd --show -no-headers
```

#### Update example

Change the message of the day.

```
> cov-manage-im --mode motd --update --set message:"hello, Hello, HELLO"
```

## Commit mode

Gets, enables, and disables the commit gate, which permits or prevents commits to the Coverity Connect database.

## Synopsis

```
--mode commit --update --set status:<enabled>|<disabled> [<OTHER>]
```

## Commit mode options

In general, you can specify options in any order.

The <OTHER> option listed in the synopsis refers to sets of command line options that are common to all modes. The options are:

- Common OUTPUT options
- CONNECTION options
- Shared options

### OPERATION options

Specify exactly one OPERATION option in commit mode.

`--update`

Updates the commit gate status.

### SET options

The SET options apply changes to the commit gate. Use `--update` to update the commit gate. At least one SET option is required with `--update`.

`--set status:{enabled | disabled}`

Opens the commit gate (enabled) or closes the commit gate (disabled).

## Commit mode examples

The first example shows the four connection options that must contain values either on the command line, or in the XML configuration file (host, port, user, password). These connection options are intentionally dropped from most of the subsequent examples to reduce the length of the command lines. When the connection options are not specified, assume that the values are retrieved from the default XML configuration file.

## Update examples

Enable the commit gate.

```
> cov-manage-im --mode commit --update --set status:enabled
```

Disable the commit gate.

```
> cov-manage-im --mode commit --update --set status:disabled
```

## Notification mode

Manually triggers a notification on a specific Coverity Connect view.

### Synopsis

```
--mode notification --execute --view <viewName>
```

For shared views, only the user who shared the view can trigger the notification. An error will occur if the user with whom the view is shared attempts to trigger the notification through this command option.

For help configuring notification settings for a view in Coverity Connect, see *Coverity Platform 2020.12 User and Administrator Guide* [↗](#).

### Notification mode options

In general, you can specify options in any order.

The options are:

- Common OUTPUT options
- CONNECTION options
- Shared options

#### OPERATION options

Specify exactly one OPERATION option in commit mode.

`--execute`

Triggers the firing of the notification email. `--execute` requires the inclusion of the `--view` filter.

#### FILTER options

FILTER options focus the set of defects that are operated on.

`--view <viewName>`

Specifies the Coverity Connect view for which the notification will be sent.

### Notification mode example

This example shows the command to trigger a notification on the Coverity Connect view, *myView*.

```
> cov-manage-im --mode notification --execute --view myView
```

## Authentication key mode

Create or revoke an authentication key for secure communication with the Coverity Connect server.

## Synopsis

```
--mode auth-key --create --output-file <filename> [<SET>]
```

```
--mode auth-key --revoke <auth-key-ID>
```

## Authentication key mode options

In general, you can specify options in any order.

### SET options

The SET options apply certain attributes to the authentication key created.

`--set description:<description>`

Sets the description of the authentication key. If not provided, the description is an empty string.

`--set expiration:<dateTime>`

Sets the expiration date for the authentication key. There are four accepted syntaxes for `<dateTime>`:

- YYYY-MM-DD

The authentication key will expire on the date specified.

- YYYY-MM-DD[ T]hh:mm(:ss)

The authentication key will expire on the date and time specified. Date and time may be separated by "T" or a space, and seconds are optional.

- "after\_<N>\_days"

The authentication key will expire <N> days in the future.

- "after\_<N>.<M>\_days"

The authentication key will expire <N>.<M> days in the future.

### Note

See the *Coverity Platform 2020.12 User and Administrator Guide* [for](#) important information about authentication key restrictions.

## Authentication key mode examples

### Create example

This example creates an authentication file named "myFile" with the description, "test user authentication file." This authentication file will expire in 90 days.

```
> cov-manage-im --host cim.company.com --port 8080 --user test \
 --password secret --mode auth-key --create --output-file myFile \
 \
```

```
--set description:"test user authentication file" \
--set expiration:"after_90_days"
```

### Revoke example

This example revokes the authentication key with ID 12345.

```
> cov-manage-im --mode auth-key --revoke 12345
```

### Common OUTPUT options

The following OUTPUT options are common to all modes.

#### --fields

Specify the fields to output with a `--show OPERATION` option. The list of fields are specified as a comma-separated value (CSV) list.

To display a list of field names that are valid for a given mode, use `--show --output fields` in that mode. For example:

```
> cov-manage-im --mode projects --show --output fields
```

The following examples generate a comma-separated list of the values in the specified fields.

```
> cov-manage-im --mode projects \
--show --fields project,description,creation-date,last-modified-date
```

```
> cov-manage-im --mode projects \
--show --output streams --fields project,stream-name,is-stream-linked
```

```
> cov-manage-im --mode streams \
--show --fields stream,language,description
```

```
> cov-manage-im --mode defects --stream MySampleStream \
--show --fields action,cid,checker
```

The order in which the fields are specified in the CSV list is the order in which they display. The same field can be listed multiple times.

#### --no-headers|-nh

Do not print field headers with `--show` operation.

#### --separator <sep>

Use `<sep>` to separate CSV values instead of a comma.

#### --output-file|-of <file>

Write output to a file named by `<file>`.

### CONNECTION options

The following CONNECTION options are common to all modes. You can also store connection details in the `<install_dir_sa>/config/coverity_config.xml` [p. 384] file.

`--auth-key-file <keyfile>`

Specify the location of your authentication key file, created in Authentication key mode. See the *Coverity Platform 2020.12 User and Administrator Guide* [\[4\]](#) for important information about authentication key restrictions.

`--certs <filename>`

In addition to CA certificates obtained from other trust stores, use the CA certificates in the given `<filename>`. For information on the new SSL certificate management functionality, please see *Coverity Platform 2020.12 User and Administrator Guide* [\[4\]](#)

`--host <server-hostname>`

Specify the Coverity Connect server hostname. To use this option, the Coverity Connect server must be running. If this option is unspecified, the default is the value from the `cim/host` element from the XML configuration file.

`--on-new-cert <trust | distrust>`

Indicates whether to trust (with `trust-first-time`) self-signed certificates, presented by the server, that the application has not seen before. For information on the new SSL certificate management functionality, please see *Coverity Platform 2020.12 User and Administrator Guide* [\[4\]](#)

`--port <server-port>`

Specify the Coverity Connect server HTTP or HTTPS connection port. To use this option, the Coverity Connect server must be running. If this option is unspecified, the default is established in the following order:

1. The value from the `cim/port` element from the XML configuration file.
2. `8080`. If `--ssl` is present, the default is `8443`.

`--ssl`

Allow Coverity Connect to use an SSL-encrypted channel.

`--url <path>`

Allows you to connect to a CIM instance that has a context path in its HTTP(S) URL. You can use this option instead of the `--host`, or `--port` options. The `--url` option is provided to accommodate the use of a context path and to deal with setting up Coverity Connect behind a reverse proxy.

Use HTTPS or HTTP to connect to Coverity Connect HTTPS or HTTP port. For `http`, the default port is 80; for `https`, the default port is 443. For example:

```
https://example.com/coverity
```

```
https://cimpop:8008
```

```
http://cim.example.com:8080
```



#### Note

You may not use the `commit://` scheme in the URL.

**--user** <user\_name>

Connect to Coverity Connect as `user_name`. If this option is unspecified, the default is established in the following order:

1. The `cim/client_security/user` element from the XML configuration file.
2. The `COV_USER` environment variable.
3. The `USER` environment variable.
4. The name of the operating system user invoking the command (if supported).

**--password** <password>

Specify the password for either the current user name, or the user specified with the `--user` option. If this option is unspecified, the default is established from the `cim/client_security/password` element from the XML configuration file.

**--userLdapServer** <domain>

Specify the domain of the user. If this option is not specified, the domain is resolved following this procedure:

1. If the user name contains "@", two possible users are considered, one with the name as given, and one with the name comprising the substring after the last "@".
2. If one and only one user name is found, then the domain is set to the domain of this user.
3. Otherwise, an error is output explaining that the domain could not be automatically set, and asking the user to explicitly specify a domain with `--userLdapServer` for users who are ldap users.

 **Note**

Coverity recommends using this parameter to avoid issues when LDAP authentication is used.

## Shared options

The following options are common to all modes of the `cov-manage-im` command.

**--config|-c** <coverity\_config.xml>

Use a specified XML configuration file instead of the default configuration file located at `<install_dir_sa>/config/coverity_config.xml`.

**--debug|-g**

Turn on basic debugging output. You must specify a mode and at least one option for proper debug reporting. This option will issue a warning if the command line that you are using is not valid; for example, if you use an unsupported option in a particular mode.

**--verbose|-V** <0, 1, 2, 3, 4>

Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1. Use `--verbose 0` to disable the output of `--show` options.

--response-file|-rf <file>

Specify command line options in <file>. These options are processed as if they are specified on the command line at the same point as the response file is specified. Multiple response files can be used on a single command line, but nested response files are not allowed.

Lines in response files starting with # are considered to be comments and are ignored.

---

**Name**

`cov-start-im` Start Coverity Connect.

**Synopsis**

`cov-start-im`

**Description**

The `cov-start-im` command starts Coverity Connect.

On Windows systems, when Coverity Connect is installed as a service, the `cov-start-im.exe` program is often unnecessary because Coverity Connect starts automatically when the system boots up. When Coverity Connect is installed as a service, any administrator can use this command.

When Coverity Connect is not installed as a service, only the user who installed Coverity Connect is able to use `cov-start-im` to start it.

---

**Name**

cov-stop-im Stop Coverity Connect.

**Synopsis**

cov-stop-im

**Description**

The `cov-stop-im` command stops Coverity Connect.

On Windows systems, when Coverity Connect is installed as a service, the `cov-stop-im.exe` program is often unnecessary because Coverity Connect stops automatically when the system shuts down. When Coverity Connect is installed as a service, any administrator can use this command.

When Coverity Connect is not installed as a service, only the user who installed Coverity Connect is able to use `cov-stop-im` to stop it.

---

## Name

cov-support Create support information package for Coverity Connect.

## Synopsis

```
cov-support [-v | -vv] [--coverity-home <coverity-base-directory>] [--with-config] [--with-logs [days]] -o|--output <outputfile>
```

## Description

The `cov-support` command creates a compressed archive containing various property and log files for Coverity Connect. Support may request that you create this archive and submit it for analysis.

If `--with-config` is specified, then the following files in the `/config` folder are included in the archive:

- `cim.properties`
- `server.xml`
- `VERSION`
- `system.properties`
- `postgresql.conf`
- `web.properties`

If `--with-logs` is specified, then log files in the `/logs`, `/postgresql`, and `/.install4j` directories are added to the archive.

## Options

`--coverity-home name_of_directory`

Specify the directory where Coverity Connect is installed, if it is installed in a directory other than the directory specified by `coverity-base-directory`.

`-o | --output name_of_file`

Destination output file. File is in tar-bzip2 format.

`--with-config`

Include configuration files in the support archive.

`--with-logs <days>`

Include log files in the support archive for the past number of days.

`-v`

Enable verbose logging information (for debug purposes)

`-vv`

Enable very verbose logging information (for trace purposes)

## Example

To obtain the config files and the log files for five days, and put them in an archive named `support-archive`:

```
> cov-support -v --with-config --with-logs 5 -o support-archive
```

---

# CVSS Report

---

## Name

`cov-generate-cvss-report` Generate a CVSS report from an existing CVSS report configuration.

## Synopsis

```
cov-generate-cvss-report <configuration-file> [--company-logo <path-to-company-logo>] [--help] [--on-new-cert <value>] [--output <output-path-to-pdf>] --password <spec> [--profile <profile.json>] --project <project-name> [--includeCusp][--report] [--scores]
```

## Description

The `cov-generate-cvss-report` command creates a CVSS report based on a configuration file and defect information in Coverity Connect.

### Note

When generating a report, it is recommended that you use the `--profile <security-profile-file>` and `--scores` options in the same command as the `--output <output-file>` and `--report` options.

You can also generate a report without using the `--profile <security-profile-file>` and `--scores` options, but you should ensure that the `CVSS_*` attributes are updated before generating the report.

## Options

### Optional arguments:

`--company-logo`

Path to the company logo file.

`--help`

Shows the help message and exits.

`--includeCusp`

Add 16 CUSP CWE Id's to the SANS 25 CWE ID's list. This adds the following CWE rules to those currently supported:

- [26] CWE-770: Allocation of Resources Without Limits or Throttling
- [28] CWE-754: Improper Check for Unusual or Exceptional Conditions
- [29] CWE-805: Buffer Access with Incorrect Length Value
- [30] CWE-838: Inappropriate Encoding for Output Context
- [32] CWE-822: Untrusted Pointer Dereference
- [33] CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

- [34] CWE-212: Improper Cross-boundary Removal of Sensitive Data
- [37] CWE-841: Improper Enforcement of Behavioral Workflow
- [40] CWE-825: Expired Pointer Dereference

`--on-new-cert <value>`

`<value>` can be `trust` or `distrust`. If the server provides an untrusted self-signed certificate and the value is set to `distrust` (which is the default), an attempt to connect to the server using SSL may fail.

`--output <output-path-to-pdf>`

The name of the PDF output file. The `--output <output-file>` option replaces any existing `<output-file>` files with the same name. Must be used in conjunction with the `--report` option.

`--profile <PROFILE>.JSON`

A `.csv` file with CWE-CVSS vector mappings

`--report`

Must be specified with the `--output <output-file>` option. Can also be used with the `--profile <security-profile-file>` and `--scores` options. When specified without the `--scores` option, it generates a CVSS Report and does not update the `CVSS_*` attributes.

When specified with the `--scores` option, the `--report` option generates the CVSS Report and also updates the `CVSS_*` attributes in Coverity Connect.

`--scores`

When specified with the `--report` option, the `--scores` option updates the `CVSS_*` attributes in Coverity Connect when generating a report.

### Required arguments:

`<configuration-file>`

A `.yaml` file containing server configuration, Coverity Connect project, and other report-related parameters

 **Note**

#### (Optional)

To enable an HTTPS connection, specify the `https` protocol for the `url` property in your configuration properties file. If your Coverity server uses a self-signed certificate, you may choose to specify the `certs` property and the location of the certificate (CA) in your configuration properties file.

`--password <spec>`

Coverity Connect password specifier

The password `<spec>` argument is required, and has four forms:

1. `console`

The password is read from the keyboard without echoing to the console.

2. `file:<filename>`

The password is read from the first line of the file `<filename>`.

3. `file:-`

The password is read from standard input. This is used with pipes and redirection.

4. `env:<variable>`

The password is read from the environment variable `<variable>`.

`--project <project-name>`  
Coverity Project to assign CVSS metrics to defects

## Examples

This command uses the `config.yaml` file to generate a CVSS Report in the `MyReport.pdf` file. If the Coverity Connect server provides a self-signed certificate, it will be trusted. The Coverity Connect server's password is provided from the value of an environment variable. The command also uses the `Master_CWE_CVSS_Base_Score_Profile_V1.json` file to update the scores in Coverity Connect.

```
cov-generate-cvss-report ../config/cvss-rpt-config.properties --on-new-cert trust
--output MyReport.pdf --password env:PASSWORD_ENV_VAR
--profile ../config/Master_CWE_CVSS_Base_Score_Profile_V1.json
--project foo --report --scores
```

This command uses the `cvss-rpt-config.yaml` file to generate a CVSS Report in the `AnotherReport.pdf` file. You will be prompted to enter the Coverity Connect server's password in the console.

```
cov-generate-cvss-report ../config/cvss-rpt-config.yaml --output AnotherReport.pdf
--password console --project project-name-2 --report
```

## Copyright Notice

Copyright (c) 2020 Synopsys, Inc. [software-integrity-support@synopsys.com](mailto:software-integrity-support@synopsys.com)

---

# Coverity Integrity Report

---

## Name

`cov-generate-integrity-report` A command-line application for generating a Coverity Integrity Report.

## Synopsis

```
cov-generate-integrity-report [-c <ja | en>] [--certs <certificate file>] [-H] [-h
<Coverity Connect hostname>] [-j <project name>] [-o <report file name>] [--on-new-cert
<trust | distrust>] [-p <Coverity Connect port>] [-P <password>] [-t] [-u <user>]
```

## Description

The `cov-generate-integrity-report` command runs the Coverity Integrity Report application for generating a Coverity Integrity Report. For information about Coverity Integrity Report, see the *Coverity Platform 2020.12 User and Administrator Guide* [🔗](#).

## Options

- `-c, --locale <ja | en>`  
Locale of the report, either "ja" or "en".
- `--certs <certificate-file>`  
An optional collection of CA certificates.
- `-H, --help`  
Print this message.
- `-h, --host <Coverity Connect hostname>`  
Hostname of the Coverity Connect server.
- `-j, --project <project name>`  
Name of the project on which to report.
- `-o, --output <report file name>`  
Name of the resulting pdf file (default "integrity\_report.pdf")
- `--on-new-cert <trust | distrust>`  
Indicates whether to trust (with trust-first-time) self-signed certificates, presented by the server, that the application has not seen before. Default is `distrust`. For information on the new SSL certificate management functionality, please see *Coverity Platform 2020.12 User and Administrator Guide* [🔗](#)
- `-p, --port <Coverity Connect port>`  
Port used by the Coverity Connect server.
- `-P, --password <console>| file:<filename>| env:<variable>`  
You can set the password as an environment variable and pass the environment variable with the command:

```
export passwordENV=coverity
--password env:passwordENV
```

-t, --trial  
Skip page 3 of the report, which contains severity data.

-u, --user <user>  
User

All other options are specified in the configuration file (default `report_config.properties`).

## Example

```
cov-generate-integrity-report
```

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects`(*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

## Copyright Notice

Copyright (c) 2020 Synopsys, Inc. [software-integrity-support@synopsys.com](mailto:software-integrity-support@synopsys.com)

---

# MISRA Report

---

## Name

cov-misra-report Launches a GUI application for configuring MISRA Report.

## Synopsis

```
cov-misra-report <config.yaml>
```

## Description

The `cov-misra-report` command runs the GUI MISRA Report application for configuring and generating a MISRA Report. The MISRA Report uses analysis results for a project in Coverity Connect to evaluate a codebase and create a formatted report. The codebase is evaluated against a policy, which is a set of rules or standards for determining pass or fail. The result for each element is presented in the MISRA Compliance section in the Executive Summary of the report. For information about MISRA Report, see the *Coverity Platform 2020.12 User and Administrator Guide* [🔗](#).

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects`(*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

---

## Name

`cov-generate-misra-report` Generate a MISRA report from an existing MISRA report configuration.

## Synopsis

```
cov-generate-misra-report <configuration file> [--help] --output <output-file> --password <spec> --on-new-cert <trust|distrust>
```

## Description

The `cov-generate-misra-report` command creates a MISRA Report based on a configuration file and current issue data in Coverity Connect.

## Options

`<configuration file>`

Coverity MISRA Report configuration (config.yaml) file made by the `cov-misra-report` GUI application.

`--help`

Display the help message and exit.

`--on-new-cert <trust|distrust>`

Indicates whether to trust (with `trust-first-time`) self-signed certificates, presented by the server, that the application has not seen before.

`--output <output-file>`

Name of the PDF output file. The file will be replaced if present.

`--password <spec>`

Coverity Connect password specifier

The password `<spec>` argument is required, and has four forms:

`console`

The password is read from the keyboard without echoing to the console.

`file:<filename>`

The password is read from the first line of the file `<filename>`.

`file:-`

The password is read from the standard input. This is for use with pipes and redirection, not for keyboard input.

`env:<variable>`

The password is read from the environment variable `<variable>`.

## Examples

```
cov-generate-misra-report MyConfiguration.yaml --output MyReport.pdf --password console
```

This command will use the configuration file MyConfiguration.yaml to generate a MISRA Report in the file MyReport.pdf. You will be prompted to enter the Coverity Connect server password on the console.

### **Copyright Notice**

Copyright (c) 2020 Synopsys, Inc. software-integrity-support@synopsys.com

---

## **OWASP Web Top 10**

---

## Name

cov-generate-owasp2017-report Generate an OWASP Top Ten report.

## Synopsis

```
cov-generate-owasp2017-report <config-file> --password <pw> --project
<project-name> --help
```

## Description

The OWASP Top Ten report generator uses analysis results for a Coverity Connect project to evaluate the analyzed codebase. Based on this evaluation, it creates an OWASP Top Ten report, which details the assessments that were done, provides a summary of findings, and specifies the remediations needed. Information from this report is of special interest to application security assurance teams and their clients. .

## Options

<configuration-file>

A file named `config.yml` that specifies information used in the report title page, and notification information for the project owner.

--password <pw>

Password to log in to Coverity Connect.

--project <project-name>

Specifies the name of the Coverity project for the source code being analyzed.

--help

To display other attributes that are supported by the launcher.

## Examples

Report the line count for the Apache `regcomp.c` file:

```
cov-generate-owasp2017-report config/cvss-rpt-config.properties --password abracadabra
--project myProject
```

## Copyright Notice

Copyright (c) 2020 Synopsys, Inc. [software-integrity-support@synopsys.com](mailto:software-integrity-support@synopsys.com)

---

## Mobile OWASP Top 10

---

## Name

cov-generate-mobile-owasp-report Generate a mobile OWASP report.

## Synopsis

```
cov-generate-mobile-owasp-report <config-file> --password <pw> --project
<project-name> --help
```

## Description

The Mobile OWASP Top Ten report generator uses analysis results for a Coverity Connect project to evaluate the analyzed codebase. Based on this evaluation, it creates a Mobile OWASP Top Ten report, which details the assessments that were done, provides a summary of findings, and specifies the remediations needed. Information from this report is of special interest to application security assurance teams and their clients.

## Options

<configuration-file>

A file named `config.yaml` that specifies information used in the report title page, and notification information for the project owner.

--password <pw>

Password to log in to Coverity Connect.

--project <project-name>

Specifies the name of the Coverity project for the source code being analyzed.

--help

To display other attributes that are supported by the launcher.

## Examples

Report the line count for the Apache `regcomp.c` file:

```
cov-generate-mobile-owasp-report config/cvss-rpt-config.properties --password
abracadabra --project myProject
```

## Copyright Notice

Copyright (c) 2020 Synopsys, Inc. [software-integrity-support@synopsys.com](mailto:software-integrity-support@synopsys.com)

---

# PCI DSS

---

## Name

cov-generate-pci-dss-report Generate a PCI DSS report.

## Synopsis

```
cov-generate-pci-dss-report <config-file> --password <pw> --project <project-name> --report --help
```

## Description

The PCI DSS report generator uses analysis results for a Coverity Connect project to evaluate the analyzed codebase. Based on this evaluation, it creates a report, which details the assessments that were done, provides a summary of findings, and specifies the remediations needed. Information from this report is of special interest to application security assurance teams and their clients.

### Important

You are required to generate a CVSS report before you can use the PCI DSS report generator because the latter depends upon the vulnerability scoring system defined by CVSS. For information about generating a CVSS report, see "Coverity CVSS Report" in the *Coverity Platform User and Administrator Guide*.

## Options

<configuration-file>

A file named `config.yaml` that specifies information used in the report title page, and notification information for the project owner.

--password <pw>

Password to log in to Coverity Connect.

--project <project-name>

Specifies the name of the Coverity project for the source code being analyzed.

--report

Required.

--help

To display other attributes that are supported by the launcher.

## Examples

Report the line count for the Apache `regcomp.c` file:

```
cov-generate-pci-dss-report config/cvss-rpt-config.properties --password abracadabra --project myProject
```

## Copyright Notice

Copyright (c) 2020 Synopsys, Inc. software-integrity-support@synopsys.com

---

# Security Report

---

## Name

`cov-security-report` Launches a GUI application for configuring Security Report.

## Synopsis

```
cov-security-report <config.yaml>
```

## Description

The `cov-security-report` command runs the GUI Security Report application for configuring and generating a Security Report. The Security Report uses analysis results for a project in Coverity Connect to evaluate a codebase and create a formatted report. The codebase is evaluated against a policy, which is a set of rules or standards for determining pass or fail. The report's policy has 4 elements, and each element must pass for the policy to pass. The result for each element is presented in the Scorecard in the Executive Summary of the report. For information about Security Report, see the *Coverity Platform 2020.12 User and Administrator Guide* [🔗](#).

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects`(*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

---

## Name

`cov-generate-security-report` Generate a Security Report from an existing Security Report configuration.

## Synopsis

```
cov-generate-security-report <config file> [--auth-key-file <filename>][--company-logo <logo>] [--help] [--includeCusp] [--locale <locale>] [--on-new-cert trust|distrust] [--output <filename>] [--password <spec>] [--project <project-name>] [--user <username>]
```

## Description

The `cov-generate-security-report` command creates a Security Report based on a configuration file and current issue data in Coverity Connect.

## Options

`--auth-key-file <filename>`

Coverity Connect authentication key file.

`--company-logo <logo>`

Path to company logo file.

`<config file>`

Security Report configuration (.yaml) file made by the `cov-security-report` GUI application.

`--help`

Display the help message and exit.

`--includeCusp`

Include CWE/SANS Cusp in SANS top25 list.

`--locale <locale>`

Locale of the report: one of `en_US` (English, default), `ja_JP` (Japanese), `ko_KR` (Korean), or `zh_CN` (Simplified Chinese).

`--on-new-cert trust|distrust`

Indicates whether to trust (with `trust-first-time`) self-signed certificates, presented by the server, that the application has not seen before. If `distrust` (default), attempting to connect to the server using SSL fails if the server provides an untrusted self-signed certificate.

`--output <output-file>`

Name of the PDF output file. The file will be replaced if present.

`--password <spec>`

Coverity Connect password specifier

The password `<spec>` argument is required, and has three forms:

`console`

The password is read from the keyboard without echoing to the console.

file:<filename>

The password is read from the first line of the file *filename*. Use "-" for *filename* to read from standard input.

env:<variable>

The password is read from the environment variable <variable>.

--project <project-name>

Name of the project in Coverity Connect.

--user <username>

Coverity Connect user name.

## Examples

```
cov-generate-security-report MyConfiguration.yaml --output MyReport.pdf --password console
```

This command will use the configuration file *MyConfiguration.yaml* to generate a Security Report in the file *MyReport.pdf*. You will be prompted to enter the Coverity Connect server password on the console.

## Copyright Notice

Copyright (c) 2020 Synopsys, Inc. [software-integrity-support@synopsys.com](mailto:software-integrity-support@synopsys.com)

---

# Coverity CERT Report

---

## Name

`cov-cert-report` Launches a GUI application for configuring CERT Report.

## Synopsis

```
cov-cert-report
```

## Description

The `cov-cert-report` command runs the GUI CERT Report application for configuring and generating a CERT Report. The CERT Report uses analysis results for a project in Coverity Connect to evaluate a codebase and create a formatted report. The codebase is evaluated against a policy, which is a set of rules or standards for determining pass or fail. The report's policy has 4 elements, and each element must pass for the policy to pass. The result for each element is presented in the Scorecard in the Executive Summary of the report. For information about CERT Report, see the *Coverity Platform 2020.12 User and Administrator Guide* [🔗](#).

## Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see `cov-commit-defects`(*Coverity 2020.12 Command Reference*), `cov-analyze`, and `cov-build`.

---

## Name

`cov-generate-cert-report` Generate a CERT Report from an existing CERT Report configuration.

## Synopsis

```
cov-generate-cert-report <configuration file> [--help] --output <output-file>
--password <spec> --certs <certificate file> --on-new-cert <trust | distrust>
```

## Description

The `cov-generate-cert-report` command creates an updated CERT Report based on a configuration file and current issue data in Coverity Connect.

## Options

`--certs <filename>`

In addition to CA certificates obtained from other trust stores, use the CA certificates in the given filename.

`<configuration file>`

Coverity CERT Report configuration (.yaml) file

`--help`

Display the help message and exit.

`--on-new-cert >trust | distrust<`

Indicates whether to trust (with `trust-first-time`) self-signed certificates, presented by the server, that the application has not seen before.

`--output <output-file>`

Name of the PDF output file. The file will be replaced if present.

`--password <spec>`

Coverity Connect password specifier

The password `<spec>` argument is required, and has four forms:

`console`

The password is read from the keyboard without echoing to the console.

`file:<filename>`

The password is read from the first line of the file `<filename>`.

`file:-`

The password is read from the standard input. This is for use with pipes and redirection, not for keyboard input.

`env:<variable>`

The password is read from the environment variable `<variable>`.

## Examples

```
cov-generate-cert-report MyConfiguration.yaml --output MyReport.pdf --password console
```

This command uses the `MyConfiguration.yaml` file to generate a CERT Report in the `MyReport.pdf` file. You will be prompted to enter the Coverity Connect server password in the console.

## Copyright Notice

Copyright (c) 2020 Synopsys, Inc. [software-integrity-support@synopsys.com](mailto:software-integrity-support@synopsys.com)

---

## Appendix A. Accepted date/time formats

Many Test Advisor commands and options accept an argument for date/time. The format for these arguments must match one of the following patterns:

YYYY-MM-DD	Midnight, local time zone
YYYY-MM-DD[ T]hh:mm(:ss)?	Local time zone
YYYY-MM-DD[ T]hh:mm(:ss)?Z	UTC time zone
YYYY-MM-DD[ T]hh:mm(:ss)?[+-]hh:mm	Time zone explicitly specified

\*Dates before 1970 are not allowed.

### Examples:

2014-07-01	Midnight on July 1, 2014, local time zone
2014-07-01 13:00	1pm on July 1, 2014, local time zone
2014-07-01T13:00:30	30 sec after 1pm on July 1, 2014, local time zone
2014-07-01 13:00Z	1pm on July 1, 2014, UTC time zone
2014-07-01 13:00-07:00	1pm on July 1, 2014, Pacific Daylight Time
2014-07-01T13:00-08:00	1pm on July 1, 2014, Pacific Standard Time
2014-07-01T13:00+09:00	1pm on July 1, 2014, Japan Standard Time

---

## Appendix B. Coverity Glossary

### Table of Contents

Glossary .....	444
----------------	-----

### Glossary

#### A

Abstract Syntax Tree (AST)	A tree-shaped data structure that represents the structure of concrete input syntax (from source code).
action	In Coverity Connect, a customizable attribute used to triage a CID. Default values are Undecided, Fix Required, Fix Submitted, Modeling Required, and Ignore. Alternative custom values are possible.
Acyclic Path Count	<p>The number of execution paths in a function, with loops counted one time at most. The following assumptions are also made:</p> <ul style="list-style-type: none"><li>• <code>continue</code> breaks out of a loop.</li><li>• <code>while</code> and <code>for</code> loops are executed exactly 0 or 1 time.</li><li>• <code>do...while</code> loops are executed exactly once.</li><li>• <code>goto</code> statements which go to an earlier source location are treated as an exit.</li></ul> <p><i>Acyclic (Statement-only) Path Count</i> adds the following assumptions:</p> <ul style="list-style-type: none"><li>• Paths within expressions are not counted.</li><li>• Multiple case labels at the same statement are counted as a single case.</li></ul>
advanced triage	<p>In Coverity Connect, streams that are associated with the same always share the same triage data and history. For example, if Stream A and Stream B are associated with Triage Store 1, and both streams contain CID 123, the streams will share the triage values (such as a shared <i>Bug</i> classification or a <i>Fix Required</i> action) for that CID, regardless of whether the streams belong to the same project.</p> <p>Advanced triage allows you to select one or more triage stores to update when triaging a CID in a Coverity Connect project. Triage store selection is possible only if the following conditions are true:</p> <ul style="list-style-type: none"><li>• Some streams in the project are associated with one triage store (for example, TS1), and other streams in the project are associated with</li></ul>

another triage store (for example, TS2). In this case, some streams that are associated with TS1 must contain the CID that you are triaging, and some streams that are associated with TS2 must contain that CID.

- You have permission to triage issues in more than one of these triage stores.

In some cases, advanced triage can result in CIDs with issue attributes that are in the Various state in Coverity Connect.

See also, triage.

analysis annotation

A marker in the source code. An analysis annotation is not executable, but modifies the behavior of Coverity Analysis in some way.

Analysis annotations can suppress false positives, indicate sensitive data, and enhance function models.

Each language has its own analysis annotation syntax and set of capabilities. These are not the same as the syntax or capabilities available to the other languages that support annotations.

- For C/C++, an analysis annotation is a comment with special formatting. See code-line annotation and function annotation.
- For C# and Visual Basic, an analysis annotation uses the native C# attribute syntax.
- For Java, an analysis annotation uses the native Java annotation syntax.

Other languages do not support annotations.

annotation

See analysis annotation.

## C

call graph

A graph in which functions are nodes, and the edges are the calls between the functions.

category

See issue category.

checker

A program that traverses paths in your source code to find specific issues in it. Examples of checkers include RACE\_CONDITION, RESOURCE\_LEAK, and INFINITE\_LOOP. For details about checkers, see *Coverity 2020.12 Checker Reference*.

checker category

See issue category.

churn	A measure of change in defect reporting between two Coverity Analysis releases that are separated by one minor release, for example, 6.5.0 and 6.6.0.
CID (Coverity identifier)	See Coverity identifier (CID).
classification	A category that is assigned to a software issue in the database. Built-in classification values are Unclassified, Pending, False Positive, Intentional, and Bug. For Test Advisor issues, classifications include Untested, No Test Needed, and Tested Elsewhere. Issues that are classified as Unclassified, Pending, and Bug are regarded as software issues for the purpose of defect density calculations.
code-line annotation	<p>For C/C++, an analysis annotation that applies to a particular line of code. When it encounters a code-line annotation, the analysis engine skips the defect report that the following line of code would otherwise trigger.</p> <p>By default, an ignored defect is classified as <code>Intentional</code>. See "Models and Annotations in C/C++" in the <i>Coverity Checker Reference</i>.</p> <p>See also function annotation.</p>
code base	A set of related source files.
code coverage	The amount of code that is tested as a percentage of the total amount of code. Code coverage is measured different ways: line coverage, path coverage, statement coverage, decision coverage, condition coverage, and others.
component	A named grouping of source code files. Components allow developers to view only issues in the source files for which they are responsible, for example. In Coverity Connect, these files are specified by a Posix regular expression. See also, component map.
component map	Describes how to map source code files, and the issues contained in the source files, into components.
control flow graph	A graph in which blocks of code without any jumps or jump targets are nodes, and the directed edges are the jumps in the control flow between the blocks. The entry block is where control enters the graph, and the exit block is where the control flow leaves.
Coverity identifier (CID)	An identification number assigned to a software issue. A snapshot contains issue <i>instances</i> (or occurrences), which take place on a specific code path in a specific version of a file. Issue instances, both within a snapshot and across snapshots (even in different streams), are grouped together according to similarity, with the intent that two issues are "similar" if the same source code change would fix them both. These groups of similar issues are given a numeric identifier, the CID. Coverity

Connect associates triage data, such as classification, action, and severity, with the CID (rather than with an individual issue).

CWE (Common Weakness Enumeration)

A community-developed list of software weaknesses, each of which is assigned a number (for example, see CWE-476 at <http://cwe.mitre.org/data/definitions/476.html>). Coverity associates many categories of defects (such as "Null pointer dereferences") with a CWE number.

Coverity Connect

A Web application that allows developers and managers to identify, manage, and fix issues found by Coverity analysis and third-party tools.

## D

data directory

The directory that contains the Coverity Connect database. After analysis, the `cov-commit-defects` command stores defects in this directory. You can use Coverity Connect to view the defects in this directory. See also `intermediate` directory.

deadcode

Code that cannot possibly be executed regardless of what input values are provided to the program.

defect

See `issue`.

deterministic

A characteristic of a function or algorithm that, when given the same input, will always give the same output.

dismissed issue

Issue marked by developers as *Intentional* or *False Positive* in the Triage pane. When such issues are no longer present in the latest snapshot of the code base, they are identified as *absent dismissed*.

domain

A combination of the language that is being analyzed and the type of analysis, either static or dynamic.

dynamic analysis

Analysis of software code by executing the compiled program. See also `static analysis`.

dynamic analysis agent

A JVM agent for Dynamic Analysis that instruments your program to gather runtime evidence of defects.

dynamic analysis stream

A sequential collection of snapshots, which each contain all of the issues that Dynamic Analysis reports during a single invocation of the Dynamic Analysis broker.

## E

event

In Coverity Connect, a software issue is composed of one or more events found by the analysis. Events are useful in illuminating the context of the issue. See also `issue`.

## F

false negative	A defect in the source code that is not found by Coverity Analysis.
false path pruning (FPP)	A technique to ensure that defects are only detected on feasible paths. For example, if a particular path through a method ensures that a given condition is known to be true, then the <code>else</code> branch of an <code>if</code> statement which tests that condition cannot be reached on that path. Any defects found in the <code>else</code> branch would be impossible because they are “on a false path”. Such defects are suppressed by a false path pruner.
false positive	A potential defect that is identified by Coverity Analysis, but that you decide is not a defect. In Coverity Connect, you can dismiss such issues as false positives. In C or C++ source, you might also use code-line annotations to identify such issues as intentional during the source code analysis phase, prior to sending analysis results to Coverity Connect.
fixed issue	Issue from the previous snapshot that is not in the latest snapshot.
fixpoint	The Extend SDK engine notices that the second and subsequent paths through the loop are not significantly different from the first iteration, and stops analyzing the loop. This condition is called a fixpoint of the loop.
flow-insensitive analysis	A checker that is stateless. The abstract syntax trees are not visited in any particular order.
function annotation	For C/C++, an analysis annotation that applies to the definition of a particular function. The annotation either suppresses or enhances the effect of that function's model. See "Models and Annotations in C/C++" in the <i>Coverity Checker Reference</i> .  See also code-line annotation.
function model	A model of a function that is not in the code base that enhances the intermediate representation of the code base that Coverity Analysis uses to more accurately analyze defects.
<b>I</b>	
impact	Term that is intended to indicate the likely urgency of fixing the issue, primarily considering its consequences for software quality and security, but also taking into account the accuracy of the checker. Impact is necessarily probabilistic and subjective, so one should not rely exclusively on it for prioritization.
inspected issue	Issue that has been triaged or fixed by developers.
intermediate directory	A directory that is specified with the <code>--dir</code> option to many commands. The main function of this directory is to write build and analysis results

before they are committed to the Coverity Connect database as a snapshot. Other more specialized commands that support the `--dir` option also write data to or read data from this directory.

The intermediate representation of the build is stored in `<intermediate_directory>/emit` directory, while the analysis results are stored in `<intermediate_directory>/output`. This directory can contain builds and analysis results for multiple languages.

See also `data` directory.

intermediate representation	The output of the Coverity compiler, which Coverity Analysis uses to run its analysis and check for defects. The intermediate representation of the code is in the intermediate directory.
interprocedural analysis	An analysis for defects based on the interaction between functions. Coverity Analysis uses call graphs to perform this type of analysis. See also intraprocedural analysis.
intraprocedural analysis	An analysis for defects within a single procedure or function, as opposed to interprocedural analysis.
issue	<p>Coverity Connect displays three types of software issues: quality defects, potential security vulnerabilities, and test policy violations. Some checkers find both quality defects and potential security vulnerabilities, while others focus primarily on one type of issue or another. The Quality, Security, and Test Advisor dashboards in Coverity Connect provide high-level metrics on each type of issue.</p> <p>Note that this glossary includes additional entries for the various types of issues, for example, an inspected issue, issue category, and so on.</p>
issue category	<p>A string used to describe the nature of a software issue; sometimes called a "checker category" or simply a "category." The issue pertains to a subcategory of software issue that a checker can report within the context of a given domain.</p> <p>Examples:</p> <ul style="list-style-type: none"><li>• <code>Memory - corruptions</code></li><li>• <code>Incorrect expression</code></li><li>• <code>Integer overflow Insecure data handling</code></li></ul> <p>Impact tables in the <i>Coverity 2020.12 Checker Reference</i> list issues found by checkers according to their category and other associated checker properties.</p>

## K

**killpath** For Coverity Analysis for C/C++, a path in a function that aborts program execution. See `<install_dir_sa>/library/generic/common/killpath.c` for the functions that are modeled in the system.

For Coverity Analysis for Java, and similarly for C# and Visual Basic, a modeling primitive used to indicate that execution terminates at this point, which prevents the analysis from continuing down this execution path. It can be used to model a native method that kills the process, like `System.exit`, or to specifically identify an execution path as invalid.

**kind** A string that indicates whether software issues found by a given checker pertain to SECURITY (for security issues), QUALITY (for quality issues), TEST (for issues with developer tests, which are found by Test Advisor), or QUALITY/SECURITY. Some checkers can report quality and security issues. The Coverity Connect UI can use this property to filter and display CIDs.

## L

**latest state** A CID's state in the latest snapshot merged with its state from previous snapshots starting with the snapshot in which its state was 'New'.

**local analysis** Interprocedural analysis on a subset of the code base with Coverity Desktop plugins, in contrast to one with Coverity Analysis, which usually takes place on a remote server.

**local effect** A string serving as a generic event message that explains why the checker reported a defect. The message is based on a subcategory of software issues that the checker can detect. Such strings appear in the Coverity Connect triage pane for a given CID.

Examples:

- May result in a security violation.
- There may be a null pointer exception, or else the comparison against null is unnecessary.

**long description** A string that provides an extended description of a software issue (compare with type). The long description appears in the Coverity Connect triage pane for a given CID. In Coverity Connect, this description is followed by a link to a corresponding CWE, if available.

Examples:

- The called function is unsafe for security related code.

- All paths that lead to this null pointer comparison already dereference the pointer earlier (CWE-476).

## M

model	<p>In Coverity Analysis of the code for a compiled language—such as C, C++, C#, Java, or Visual Basic—a model represents a function in the application source. Models are used for interprocedural analysis.</p> <p>Each model is created as each function is analyzed. The model is an abstraction of the function’s behavior at execution time; for example, a model can show which arguments the function dereferences, and whether the function returns a null value.</p> <p>It is possible to write custom models for a code base. Custom models can help improve Coverity's ability to detect certain kinds of bugs. Custom models can also help reduce the incidence of false positives.</p>
modeling primitive	<p>A modeling primitive is used when writing custom models. Each modeling primitive is a function stub: It does not specify any executable code, but when it is used in a custom model it instructs Coverity Analysis how to analyze (or refrain from analyzing) the function being modeled.</p> <p>For example, the C/C++ checker CHECKED_RETURN is associated with the modeling primitive <code>_coverity_always_check_return_()</code>. This primitive tells CHECKED_RETURN to verify that the function being analyzed really does return a value.</p> <p>Some modeling primitives are generic, but most are specific to a particular checker or group of checkers. The set of available modeling primitives varies from language to language.</p>

## N

native build	The normal build process in a software development environment that does not involve Coverity products.
--------------	---------------------------------------------------------------------------------------------------------

## O

outstanding issue	Issues that are uninspected and unresolved.
outstanding defects count	The sum of security and non-security defects count.
outstanding non-security defects count	The sum of non-security defects count.
outstanding security defects count.	The sum of security defects count.

**owner** User name of the user to whom an issue has been assigned in Coverity Connect. Coverity Connect identifies the owner of issues not yet assigned to a user as *Unassigned*.

## P

**postorder traversal** The recursive visiting of children of a given node in order, and then the visit to the node itself. Left sides of assignments are evaluated after the assignment because the left side becomes the value of the entire assignment expression.

**primitive** In the Java language, elemental data types such as strings and integers are known as *primitive types*. (In the C-language family, such types are typically known as *basic types*).

For the function stubs that can be used when constructing custom models, see modeling primitive.

**project** In Coverity Connect, a specified set of related streams that provide a comprehensive view of issues in a code base.

## R

**resolved issues** Issues that have been fixed or marked by developers as *Intentional* or *False Positive* through the Coverity Connect Triage pane.

**run** In Coverity releases 4.5.x or lower, a grouping of defects committed to the Coverity Connect. Each time defects are inserted into the Coverity Connect using the `cov-commit-defects` command, a new run is created, and the run ID is reported. See also snapshot

## S

**sanitize** To clean or validate tainted data to ensure that the data is valid. Sanitizing tainted data is an important aspect of secure coding practices to eliminate system crashes, corruption, escalation of privileges, or denial of service. See also tainted data.

**severity** In Coverity Connect, a customizable property that can be assigned to CIDs. Default values are Unspecified, Major, Moderate, and Minor. Severities are generally used to specify how critical a defect is.

**sink** Coverity Analysis for C/C++: Any operation or function that must be protected from tainted data. Examples are array subscripting, `system()`, `malloc()`.

Coverity Analysis for Java: Any operation or function that must be protected from tainted data. Examples are array subscripting and the JDBC API `Connection.execute`.

snapshot	<p>A copy of the state of a code base at a certain point during development. Snapshots help to isolate defects that developers introduce during development.</p> <p>Snapshots contain the results of an analysis. A snapshot includes both the issue information and the source code in which the issues were found. Coverity Connect allows you to delete a snapshot in case you committed faulty data, or if you committed data for testing purposes.</p>
snapshot scope	<p>Determines the snapshots from which the CID are listed using the <i>Show</i> and the optional <i>Compared To</i> fields. The show and compare scope is only configurable in the <i>Settings</i> menu in <i>Issues:By Snapshot</i> views and the snapshot information pane in the <i>Snapshots</i> view.</p>
source	<p>An entry point of untrusted data. Examples include environment variables, command line arguments, incoming network data, and source code.</p>
static analysis	<p>Analysis of software code without executing the compiled program. See also dynamic analysis.</p>
status	<p>Describes the state of an issue. Takes one of the following values: <i>New</i>, <i>Triaged</i>, <i>Dismissed</i>, <i>Absent</i> <i>Dismissed</i>, or <i>Fixed</i>.</p>
store	<p>A map from abstract syntax trees to integer values and a sequence of events. This map can be used to implement an abstract interpreter, used in flow-sensitive analysis.</p>
stream	<p>A sequential collection of snapshots. Streams can thereby provide information about software issues over time and at a particular points in development process.</p>

## T

tainted data	<p>Any data that comes to a program as input from a user. The program does not have control over the values of the input, and so before using this data, the program must sanitize the data to eliminate system crashes, corruption, escalation of privileges, or denial of service. See also sanitize.</p>
translation unit	<p>A translation unit is the smallest unit of code that can be compiled separately. What this unit is, depends primarily on the language: For example, a Java translation unit is a single source file, while a C or C++ translation unit is a source file plus all the other files (such as headers) that the source file includes.</p> <p>When Coverity tools capture code to analyze, the resulting intermediate directory contains a collection of translation units. This collection includes source files along with other files and information that form the</p>

	<p>context of the compilation. For example, in Java this context includes bytecode files in the class path; in C or C++ this context includes both preprocessor definitions and platform information about the compiler.</p>
triage	<p>The process of setting the states of an issue in a particular stream, or of issues that occur in multiple streams. These user-defined states reflect items such as how severe the issue is, if it is an expected result (false positive), the action that should be taken for the issue, to whom the issue is assigned, and so forth. These details provide tracking information for your product. Coverity Connect provides a mechanism for you to update this information for individual and multiple issues that exist across one or more streams.</p> <p>See also advanced triage.</p>
triage store	<p>A repository for the current and historical triage values of CIDs. In Coverity Connect, each stream must be associated with a single triage store so that users can triage issues (instances of CIDs) found in the streams. Advanced triage allows you to select one or more triage stores to update when triaging a CID in a Coverity Connect project.</p> <p>See also advanced triage.</p>
type	<p>A string that typically provides a short description of the root cause or potential effect of a software issue. The description pertains to a subcategory of software issues that the checker can find within the scope of a given domain. Such strings appear at the top of the Coverity Connect triage pane, next to the CID that is associated with the issue. Compare with long description.</p> <p>Examples:</p> <pre>The called function is unsafe for security related code</pre> <pre>Dereference before null check</pre> <pre>Out-of-bounds access</pre> <pre>Evaluation order violation</pre> <p>Impact tables in the <i>Coverity 2020.12 Checker Reference</i> list issues found by checkers according to their type and other associated checker properties.</p>
<b>U</b>	
unified issue	<p>An issue that is identical and present in multiple streams. Each instance of an identical, unified issue shares the same CID.</p>
uninspected issues	<p>Issues that are as yet unclassified in Coverity Connect because they have not been triaged by developers.</p>

unresolved issues

Defects are marked by developers as *Pending* or *Bug* through the Coverity Connect Triage pane. Coverity Connect sometimes refers to these issues as *Outstanding* issues.

## V

various

Coverity Connect uses the term *Various* in two cases:

- When a checker is categorized as both a quality and a security checker. For example, `USE_AFTER_FREE` and `UNINIT` are listed as such in the *Issue Kind* column of the View pane. For details, see the *Coverity 2020.12 Checker Reference*.
- When different instances of the same CID are triaged differently. Within the scope of a project, instances of a given CID that occur in separate streams can have different values for a given triage attribute if the streams are associated with different . For example, you might use advanced triage to classify a CID as a *Bug* in one triage store but retain the default *Unclassified* setting for the CID in another store. In such a case, the View pane of Coverity Connect identifies the project-wide classification of the CID as *Various*.

Note that if all streams share a single triage store, you will never encounter a CID in this triage state.

view

Saved searches for Coverity Connect data in a given project. Typically, these searches are filtered. Coverity Connect displays this output in data tables (located in the Coverity Connect View pane). The columns in these tables can include CIDs, files, snapshots, checker names, dates, and many other types of data.

---

## Appendix C. Coverity Legal Notice

### Table of Contents

C.1. Legal Notice .....	456
-------------------------	-----

### C.1. Legal Notice

The information contained in this document, and the Licensed Product provided by Synopsys, are the proprietary and confidential information of Synopsys, Inc. and its affiliates and licensors, and are supplied subject to, and may be used only by Synopsys customers in accordance with the terms and conditions of a license agreement previously accepted by Synopsys and that customer. Synopsys' current standard end user license terms and conditions are contained in the `cov_EULM` files located at `<install_dir>/doc/en/licenses/end_user_license`.

Portions of the product described in this documentation use third-party material. Notices, terms and conditions, and copyrights regarding third party material may be found in the `<install_dir>/doc/en/licenses` directory.

Customer acknowledges that the use of Synopsys Licensed Products may be enabled by authorization keys supplied by Synopsys for a limited licensed period. At the end of this period, the authorization key will expire. You agree not to take any action to work around or override these license restrictions or use the Licensed Products beyond the licensed period. Any attempt to do so will be considered an infringement of intellectual property rights that may be subject to legal action.

If Synopsys has authorized you, either in this documentation or pursuant to a separate mutually accepted license agreement, to distribute Java source that contains Synopsys annotations, then your distribution should include Synopsys' `analysis_install_dir/library/annotations.jar` to ensure a clean compilation. This `annotations.jar` file contains proprietary intellectual property owned by Synopsys. Synopsys customers with a valid license to Synopsys' Licensed Products are permitted to distribute this JAR file with source that has been analyzed by Synopsys' Licensed Products consistent with the terms of such valid license issued by Synopsys. Any authorized distribution must include the following copyright notice: **Copyright © 2020 Synopsys, Inc. All rights reserved worldwide.**

**U.S. GOVERNMENT RESTRICTED RIGHTS:** The Software and associated documentation are provided with Restricted Rights. Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in subparagraph (c)(1) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of Commercial Computer Software – Restricted Rights at 48 CFR 52.227-19, as applicable.

The Manufacturer is: Synopsys, Inc. 690 E. Middlefield Road, Mountain View, California 94043.

The Licensed Product known as Coverity is protected by multiple patents and patents pending, including U.S. Patent No. 7,340,726.

#### Trademark Statement

Coverity and the Coverity logo are trademarks or registered trademarks of Synopsys, Inc. in the U.S. and other countries. Synopsys' trademarks may be used publicly only with permission from

Synopsys. Fair use of Synopsys' trademarks in advertising and promotion of Synopsys' Licensed Products requires proper acknowledgement.

Microsoft, Visual Studio, and Visual C# are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Microsoft Research Detours Package, Version 3.0.

Copyright © Microsoft Corporation. All rights reserved.

Oracle and Java are registered trademarks of Oracle and/or affiliates. Other names may be trademarks of their respective owners.

"MISRA", "MISRA C" and the MISRA triangle logo are registered trademarks of MISRA Ltd, held on behalf of the MISRA Consortium. © MIRA Ltd, 1998 - 2013. All rights reserved. The name FindBugs and the FindBugs logo are trademarked by The University of Maryland.

Other names and brands may be claimed as the property of others.

This Licensed Product contains open source or community source software ("**Open Source Software**") provided under separate license terms (the "**Open Source License Terms**"), as described in the applicable license agreement under which this Licensed Product is licensed ("**Agreement**"). The applicable Open Source License Terms are identified in a directory named `licenses` provided with the delivery of this Licensed Product. For all Open Source Software subject to the terms of an LGPL license, Customer may contact Synopsys at `software-integrity-support@synopsys.com` and Synopsys will comply with the terms of the LGPL by delivering to Customer the applicable requested Open Source Software package, and any modifications to such Open Source Software package, in source format, under the applicable LGPL license. Any Open Source Software subject to the terms and conditions of the GPLv3 license as its Open Source License Terms that is provided with this Licensed Product is provided as a mere aggregation of GPL code with Synopsys' proprietary code, pursuant to Section 5 of GPLv3. Such Open Source Software is a self-contained program separate and apart from the Synopsys code that does not interact with the Synopsys proprietary code. Accordingly, the GPL code and the Synopsys proprietary code that make up this Licensed Product co-exist on the same media, but do not operate together. Customer may contact Synopsys at `software-integrity-support@synopsys.com` and Synopsys will comply with the terms of the GPL by delivering to Customer the applicable requested Open Source Software package in source code format, in accordance with the terms and conditions of the GPLv3 license. No Synopsys proprietary code that Synopsys chooses to provide to Customer will be provided in source code form; it will be provided in executable form only. Any Customer changes to the Licensed Product (including the Open Source Software) will void all Synopsys obligations under the Agreement, including but not limited to warranty, maintenance services and infringement indemnity obligations.

The Cobertura package, licensed under the GPLv2, has been modified as of release 7.0.3. The package is a self-contained program, separate and apart from Synopsys code that does not interact with the Synopsys proprietary code. The Cobertura package and the Synopsys proprietary code co-exist on the same media, but do not operate together. Customer may contact Synopsys at `software-integrity-support@synopsys.com` and Synopsys will comply with the terms of the GPL by delivering to Customer the applicable requested open source package in source format, under the GPLv2 license. Any Synopsys proprietary code that Synopsys chooses to provide to Customer upon its request will be provided in object form only. Any changes to the Licensed Product will void all

Coverity obligations under the Agreement, including but not limited to warranty, maintenance services and infringement indemnity obligations. If Customer does not have the modified Cobertura package, Synopsys recommends to use the JaCoCo package instead.

For information about using JaCoCo, see the description for `cov-build --java-coverage` in the *Command Reference*.

#### LLVM/Clang subproject

Copyright © All rights reserved. Developed by: LLVM Team, University of Illinois at Urbana-Champaign (<http://llvm.org/>). Permission is hereby granted, free of charge, to any person obtaining a copy of LLVM/Clang and associated documentation files ("Clang"), to deal with Clang without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of Clang, and to permit persons to whom Clang is furnished to do so, subject to the following conditions: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimers. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimers in the documentation and/or other materials provided with the distribution. Neither the name of the University of Illinois at Urbana-Champaign, nor the names of its contributors may be used to endorse or promote products derived from Clang without specific prior written permission.

CLANG IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH CLANG OR THE USE OR OTHER DEALINGS WITH CLANG.

#### Rackspace Threading Library (2.0)

Copyright © Rackspace, US Inc. All rights reserved. Licensed under the Apache License, Version 2.0 (the "License"); you may not use these files except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

#### SIL Open Font Library subproject

Copyright © 2020 Synopsys Inc. All rights reserved worldwide. ([www.synopsys.com](http://www.synopsys.com)), with Reserved Font Name fa-gear, fa-info-circle, fa-question.

This Font Software is licensed under the SIL Open Font License, Version 1.1. This license is available with a FAQ at <http://scripts.sil.org/OFL>.

#### Apache Software License, Version 1.1

Copyright © 1999-2003 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The end-user documentation included with the redistribution, if any, must include the following acknowledgement: "This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)."

Alternately, this acknowledgement may appear in the software itself, if and wherever such third-party acknowledgements normally appear.

4. The names "The Jakarta Project", "Commons", and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact [apache@apache.org](mailto:apache@apache.org).
5. Products derived from this software may not be called "Apache" nor may "Apache" appear in their names without prior written permission of the Apache Group.

THIS SOFTWARE IS PROVIDED ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Apache License Version 2.0, January 2004 <http://www.apache.org/licenses/>  
Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at: <http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Results of analysis from Coverity and Test Advisor represent the results of analysis as of the date and time that the analysis was conducted. The results represent an assessment of the errors, weaknesses and vulnerabilities that can be detected by the analysis, and do not state or infer that no other errors, weaknesses or vulnerabilities exist in the software analyzed. Synopsys does NOT guarantee that all errors, weakness or vulnerabilities will be discovered or detected or that such errors, weaknesses or vulnerabilities are discoverable or detectable.

SYNOPSYS AND ITS SUPPLIERS DISCLAIM ALL WARRANTIES, CONDITIONS AND REPRESENTATIONS, EXPRESS, IMPLIED OR STATUTORY, INCLUDING THOSE RELATED

TO MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, ACCURACY OR COMPLETENESS OF RESULTS, CONFORMANCE WITH DESCRIPTION, AND NON-INFRINGEMENT. SYNOPSIS AND ITS SUPPLIERS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES, CONDITIONS AND REPRESENTATIONS ARISING OUT OF COURSE OF DEALING, USAGE OR TRADE.