ctest(1)

Synopsis

```
ctest [<options>]
```

Description

The “ctest” executable is the CMake test driver program. CMake-generated build trees created for projects that use the ENABLE_TESTING and ADD_TEST commands have testing support. This program will run the tests and report results.

Options

`-C <cfg>, --build-config <cfg>`
: Choose configuration to test.

  Some CMake-generated build trees can have multiple build configurations in the same tree. This option can be used to specify which one should be tested. Example configurations are “Debug” and “Release”.

`-V,--verbose`
: Enable verbose output from tests.

  Test output is normally suppressed and only summary information is displayed. This option will show all test output.

`-VV,--extra-verbose`
: Enable more verbose output from tests.

  Test output is normally suppressed and only summary information is displayed. This option will show even more test output.

`--debug`
: Displaying more verbose internals of CTest.

  This feature will result in a large number of output that is mostly useful for debugging dashboard problems.

`--output-on-failure`

Output anything outputted by the test program if the test should fail. This option can also be enabled by setting the environment variable `CTEST_OUTPUT_ON_FAILURE`.

`-F`

Enable failover.

This option allows ctest to resume a test set execution that was previously interrupted. If no interruption occurred, the -F option will have no effect.

`-j <jobs>, --parallel <jobs>`

Run the tests in parallel using the given number of jobs.

This option tells ctest to run the tests in parallel using given number of jobs. This option can also be set by setting the environment variable `CTEST_PARALLEL_LEVEL`.

`--test-load <level>`

While running tests in parallel (e.g. with `-j`), try not to start tests when they may cause the CPU load to pass above a given threshold.

When `ctest` is run as a Dashboard Client this sets the `TestLoad` option of the CTest Test Step.

`-Q,--quiet`

Make ctest quiet.

This option will suppress all the output. The output log file will still be generated if the –output-log is specified. Options such as –verbose, –extra-verbose, and –debug are ignored if –quiet is specified.

`-O <file>, --output-log <file>`

Output to log file

This option tells ctest to write all its output to a log file.

`-N,--show-only`

Disable actual execution of tests.

This option tells ctest to list the tests that would be run but not actually run them. Useful in conjunction with the -R and -E options.

`-L <regex>, --label-regex <regex>`

Run tests with labels matching regular expression.

This option tells ctest to run only the tests whose labels match the given regular expression.

`-R <regex>, --tests-regex <regex>`

Run tests matching regular expression.

This option tells ctest to run only the tests whose names match the given regular expression.

`-E <regex>, --exclude-regex <regex>`

Exclude tests matching regular expression.

This option tells ctest to NOT run the tests whose names match the given regular expression.

`-LE <regex>, --label-exclude <regex>`

Exclude tests with labels matching regular expression.

This option tells ctest to NOT run the tests whose labels match the given regular expression.

`-D <dashboard>, --dashboard <dashboard>`

Execute dashboard test

This option tells ctest to act as a CDash client and perform a dashboard test. All tests are <Mode><Test>, where Mode can be Experimental, Nightly, and Continuous, and Test can be Start, Update, Configure, Build, Test, Coverage, and Submit.

`-D <var>:<type>=<value>`

Define a variable for script mode

Pass in variable values on the command line. Use in conjunction with -S to pass variable values to a dashboard script. Parsing -D arguments as variable values is only attempted if the value following -D does not match any of the known dashboard types.

`-M <model>, --test-model <model>`

Sets the model for a dashboard

This option tells ctest to act as a CDash client where the TestModel can be Experimental, Nightly, and Continuous. Combining -M and -T is similar to -D

`-T <action>, --test-action <action>`

Sets the dashboard action to perform

This option tells ctest to act as a CDash client and perform some action such as start, build, test etc. Combining -M and -T is similar to -D

`--track <track>`

Specify the track to submit dashboard to

Submit dashboard to specified track instead of default one. By default, the dashboard is submitted to Nightly, Experimental, or Continuous track, but by specifying this option, the track can be arbitrary.

`-S <script>, --script <script>`

Execute a dashboard for a configuration

This option tells ctest to load in a configuration script which sets a number of parameters such as the binary and source directories. Then ctest will do what is required to create and run a dashboard. This option basically sets up a dashboard and then runs ctest -D with the appropriate options.

`-SP <script>, --script-new-process <script>`

Execute a dashboard for a configuration

This option does the same operations as -S but it will do them in a separate process. This is primarily useful in cases where the script may modify the environment and you do not want the modified environment to impact other -S scripts.

`-A <file>, --add-notes <file>`

Add a notes file with submission

This option tells ctest to include a notes file when submitting dashboard.

`-I [Start,End,Stride,test#,test#|Test file], --tests-information`

Run a specific number of tests by number.

This option causes ctest to run tests starting at number Start, ending at number End, and incrementing by Stride. Any additional numbers after Stride are considered individual test numbers. Start, End,or stride can be empty. Optionally a file can be given that contains the same syntax as the command line.

`-U, --union`

Take the Union of -I and -R

When both -R and -I are specified by default the intersection of tests are run. By specifying -U the union of tests is run instead.

`--rerun-failed`

Run only the tests that failed previously

This option tells ctest to perform only the tests that failed during its previous run. When this option is specified, ctest ignores all other options intended to modify the list of tests to run (-L, -R, -E, -LE, -I, etc). In the event that CTest runs and no tests fail, subsequent calls to ctest with the –rerun-failed option will run the set of tests that most recently failed (if any).

`--repeat-until-fail <n>`

Require each test to run `<n>` times without failing in order to pass.

This is useful in finding sporadic failures in test cases.

`--max-width <width>`

Set the max width for a test name to output

Set the maximum width for each test name to show in the output. This allows the user to widen the output to avoid clipping the test name which can be very annoying.

`--interactive-debug-mode [0|1]`

Set the interactive mode to 0 or 1.

This option causes ctest to run tests in either an interactive mode or a non-interactive mode. On Windows this means that in non-interactive mode, all system debug pop up windows are blocked. In dashboard mode (Experimental, Nightly, Continuous), the default is non-interactive. When just running tests not for a dashboard the default is to allow popups and interactive debugging.

`--no-label-summary`

Disable timing summary information for labels.

This option tells ctest not to print summary information for each label associated with the tests run. If there are no labels on the tests, nothing extra is printed.

`--build-and-test`

Configure, build and run a test.

This option tells ctest to configure (i.e. run cmake on), build, and or execute a test. The configure and test steps are optional. The arguments to this command line are the source and binary directories. By default this will run CMake on the Source/Bin directories specified unless –build-nocmake is specified. The –build-generator option *must* be provided to use –build-and-test. If –test-command is specified then that will be run after the build is complete. Other options that affect this mode are –build-target –build-nocmake, –build-run-dir, –build-two-config, –build-exe-dir, –build-project,–build-noclean, –build-options

`--build-target`

Specify a specific target to build.

This option goes with the –build-and-test option, if left out the all target is built.

`--build-nocmake`

Run the build without running cmake first.

Skip the cmake step.

`--build-run-dir`

Specify directory to run programs from.

Directory where programs will be after it has been compiled.

`--build-two-config`

Run CMake twice

`--build-exe-dir`

Specify the directory for the executable.

`--build-generator`

Specify the generator to use.

`--build-generator-platform`

Specify the generator-specific platform.

`--build-generator-toolset`

Specify the generator-specific toolset.

`--build-project`

Specify the name of the project to build.

`--build-makeprogram`

Override the make program chosen by CTest with a given one.

`--build-noclean`
Skip the make clean step.

`--build-config-sample`
A sample executable to use to determine the configuration

A sample executable to use to determine the configuration that should be used. e.g. Debug/Release/etc

`--build-options`
Add extra options to the build step.

This option must be the last option with the exception of –test-command

`--test-command`
The test to run with the –build-and-test option.

`--test-output-size-passed <size>`
Limit the output for passed tests to <size> bytes.

`--test-output-size-failed <size>`
Limit the output for failed tests to <size> bytes.

`--test-timeout`
The time limit in seconds, internal use only.

`--tomorrow-tag`
Nightly or experimental starts with next day tag.

This is useful if the build will not finish in one day.

`--ctest-config`
The configuration file used to initialize CTest state when submitting dashboards.

This option tells CTest to use different initialization file instead of CTestConfiguration.tcl. This way multiple initialization files can be used for example to submit to multiple dashboards.

`--overwrite`
Overwrite CTest configuration option.

By default ctest uses configuration options from configuration file. This option will overwrite the configuration option.

`--extra-submit <file>[;<file>]`

Submit extra files to the dashboard.

This option will submit extra files to the dashboard.

`--force-new-ctest-process`

Run child CTest instances as new processes

By default CTest will run child CTest instances within the same process. If this behavior is not desired, this argument will enforce new processes for child CTest processes.

`--schedule-random`

Use a random order for scheduling tests

This option will run the tests in a random order. It is commonly used to detect implicit dependencies in a test suite.

`--submit-index`

Legacy option for old Dart2 dashboard server feature. Do not use.

`--timeout <seconds>`

Set a global timeout on all tests.

This option will set a global timeout on all tests that do not already have a timeout set on them.

`--stop-time <time>`

Set a time at which all tests should stop running.

Set a real time of day at which all tests should timeout. Example: 7:00:00 -0400. Any time format understood by the curl date parser is accepted. Local time is assumed if no timezone is specified.

`--http1.0`

Submit using HTTP 1.0.

This option will force CTest to use HTTP 1.0 to submit files to the dashboard, instead of HTTP 1.1.

`--no-compress-output`

Do not compress test output when submitting.

This flag will turn off automatic compression of test output. Use this to maintain compatibility with an older version of CDash which doesn't support compressed test output.

`--print-labels`
Print all available test labels.

This option will not run any tests, it will simply print the list of all labels associated with the test set.

`--help,-help,-usage,-h,-H,/?`
Print usage information and exit.

Usage describes the basic command line interface and its options.

`--version,-version,/V [<f>]`
Show program name/version banner and exit.

If a file is specified, the version is written into it. The help is printed to a named <f>ile if given.

`--help-full [<f>]`
Print all help manuals and exit.

All manuals are printed in a human-readable text format. The help is printed to a named <f>ile if given.

`--help-manual <man> [<f>]`
Print one help manual and exit.

The specified manual is printed in a human-readable text format. The help is printed to a named <f>ile if given.

`--help-manual-list [<f>]`
List help manuals available and exit.

The list contains all manuals for which help may be obtained by using the `--help-manual` option followed by a manual name. The help is printed to a named <f>ile if given.

`--help-command <cmd> [<f>]`
Print help for one command and exit.

The **cmake-commands(7)** manual entry for `<cmd>` is printed in a human-readable text format. The help is printed to a named <f>ile if given.

`--help-command-list [<f>]`
List commands with help available and exit.

The list contains all commands for which help may be obtained by using the `--help-command` option followed by a command name. The help is printed to a named <f>ile if given.

`--help-commands [<f>]`

Print cmake-commands manual and exit.

The **cmake-commands(7)** manual is printed in a human-readable text format. The help is printed to a named <f>ile if given.

`--help-module <mod> [<f>]`

Print help for one module and exit.

The **cmake-modules(7)** manual entry for `<mod>` is printed in a human-readable text format. The help is printed to a named <f>ile if given.

`--help-module-list [<f>]`

List modules with help available and exit.

The list contains all modules for which help may be obtained by using the `--help-module` option followed by a module name. The help is printed to a named <f>ile if given.

`--help-modules [<f>]`

Print cmake-modules manual and exit.

The **cmake-modules(7)** manual is printed in a human-readable text format. The help is printed to a named <f>ile if given.

`--help-policy <cmp> [<f>]`

Print help for one policy and exit.

The **cmake-policies(7)** manual entry for `<cmp>` is printed in a human-readable text format. The help is printed to a named <f>ile if given.

`--help-policy-list [<f>]`

List policies with help available and exit.

The list contains all policies for which help may be obtained by using the `--help-policy` option followed by a policy name. The help is printed to a named <f>ile if given.

`--help-policies [<f>]`

Print cmake-policies manual and exit.

The **cmake-policies(7)** manual is printed in a human-readable text format. The help is printed to a named <f>ile if given.

`--help-property <prop> [<f>]`

Print help for one property and exit.

The **cmake-properties(7)** manual entries for `<prop>` are printed in a human-readable text format. The help is printed to a named <f>ile if given.

`--help-property-list [<f>]`

List properties with help available and exit.

The list contains all properties for which help may be obtained by using the `--help-property` option followed by a property name. The help is printed to a named <f>ile if given.

`--help-properties [<f>]`

Print cmake-properties manual and exit.

The **cmake-properties(7)** manual is printed in a human-readable text format. The help is printed to a named <f>ile if given.

`--help-variable <var> [<f>]`

Print help for one variable and exit.

The **cmake-variables(7)** manual entry for `<var>` is printed in a human-readable text format. The help is printed to a named <f>ile if given.

`--help-variable-list [<f>]`

List variables with help available and exit.

The list contains all variables for which help may be obtained by using the `--help-variable` option followed by a variable name. The help is printed to a named <f>ile if given.

`--help-variables [<f>]`

Print cmake-variables manual and exit.

The **cmake-variables(7)** manual is printed in a human-readable text format. The help is printed to a named <f>ile if given.

Dashboard Client

CTest can operate as a client for the CDash software quality dashboard application. As a dashboard client, CTest performs a sequence of steps to configure, build, and test software, and then submits the results to a CDash server.

Dashboard Client Steps

CTest defines an ordered list of testing steps of which some or all may be run as a dashboard client:

`Start`
: Start a new dashboard submission to be composed of results recorded by the following steps. See the CTest Start Step section below.

`Update`
: Update the source tree from its version control repository. Record the old and new versions and the list of updated source files. See the CTest Update Step section below.

`Configure`
: Configure the software by running a command in the build tree. Record the configuration output log. See the CTest Configure Step section below.

`Build`
: Build the software by running a command in the build tree. Record the build output log and detect warnings and errors. See the CTest Build Step section below.

`Test`
: Test the software by loading a `CTestTestfile.cmake` from the build tree and executing the defined tests. Record the output and result of each test. See the CTest Test Step section below.

`Coverage`
: Compute coverage of the source code by running a coverage analysis tool and recording its output. See the CTest Coverage Step section below.

`MemCheck`
: Run the software test suite through a memory check tool. Record the test output, results, and issues reported by the tool. See the CTest MemCheck Step section below.

`Submit`
: Submit results recorded from other testing steps to the software quality dashboard server. See the CTest Submit Step section below.

Dashboard Client Modes

CTest defines three modes of operation as a dashboard client:

`Nightly`
: This mode is intended to be invoked once per day, typically at night. It enables the `Start`, `Update`, `Configure`, `Build`, `Test`, `Coverage`, and `Submit` steps by default. Selected steps run even if the `Update` step reports no changes to the source tree.

`Continuous`
: This mode is intended to be invoked repeatedly throughout the day. It enables the `Start`, `Update`, `Configure`, `Build`, `Test`, `Coverage`, and `Submit` steps by default, but exits after the `Update` step if it reports no changes to the source tree.

`Experimental`
: This mode is intended to be invoked by a developer to test local changes. It enables the `Start`, `Configure`, `Build`, `Test`, `Coverage`, and `Submit` steps by default.

Dashboard Client via CTest Command-Line

CTest can perform testing on an already-generated build tree. Run the `ctest` command with the current working directory set to the build tree and use one of these signatures:

```
ctest -D <mode>[<step>]
ctest -M <mode> [ -T <step> ]...
```

The `<mode>` must be one of the above Dashboard Client Modes, and each `<step>` must be one of the above Dashboard Client Steps.

CTest reads the Dashboard Client Configuration settings from a file in the build tree called either `CTestConfiguration.ini` or `DartConfiguration.tcl` (the names are historical). The format of the file is:

```
# Lines starting in '#' are comments.
```

```
# Other non-blank lines are key-value pairs.
<setting>: <value>
```

where `<setting>` is the setting name and `<value>` is the setting value.

In build trees generated by CMake, this configuration file is generated by the `CTest` module if included by the project. The module uses variables to obtain a value for each setting as documented with the settings below.

Dashboard Client via CTest Script

CTest can perform testing driven by a `cmake-language(7)` script that creates and maintains the source and build tree as well as performing the testing steps. Run the `ctest` command with the current working directory set outside of any build tree and use one of these signatures:

```
ctest -S <script>
ctest -SP <script>
```

The `<script>` file must call CTest Commands commands to run testing steps explicitly as documented below. The commands obtain Dashboard Client Configuration settings from their arguments or from variables set in the script.

Dashboard Client Configuration

The Dashboard Client Steps may be configured by named settings as documented in the following sections.

CTest Start Step

Start a new dashboard submission to be composed of results recorded by the following steps.

In a CTest Script, the `ctest_start()` command runs this step. Arguments to the command may specify some of the step settings. The command first runs the command-line specified by the `CTEST_CHECKOUT_COMMAND` variable, if set, to initialize the source directory.

Configuration settings include:

`BuildDirectory`
: The full path to the project build tree.

- CTest Script variable: **CTEST_BINARY_DIRECTORY**
- **CTest** module variable: **PROJECT_BINARY_DIR**

SourceDirectory
: The full path to the project source tree.

- CTest Script variable: **CTEST_SOURCE_DIRECTORY**
- **CTest** module variable: **PROJECT_SOURCE_DIR**

CTest Update Step

In a CTest Script, the **ctest_update()** command runs this step. Arguments to the command may specify some of the step settings.

Configuration settings to specify the version control tool include:

BZRCommand
: `bzr` command-line tool to use if source tree is managed by Bazaar.

- CTest Script variable: **CTEST_BZR_COMMAND**
- **CTest** module variable: none

BZRUpdateOptions
: Command-line options to the `BZRCommand` when updating the source.

- CTest Script variable: **CTEST_BZR_UPDATE_OPTIONS**
- **CTest** module variable: none

CVSCommand
: `cvs` command-line tool to use if source tree is managed by CVS.

- CTest Script variable: **CTEST_CVS_COMMAND**
- **CTest** module variable: `CVSCOMMAND`

CVSUpdateOptions
: Command-line options to the `CVSCommand` when updating the source.

- CTest Script variable: **CTEST_CVS_UPDATE_OPTIONS**
- **CTest** module variable: `CVS_UPDATE_OPTIONS`

GITCommand
: `git` command-line tool to use if source tree is managed by Git.

- CTest Script variable: **CTEST_GIT_COMMAND**
- **CTest** module variable: GITCOMMAND

GITUpdateCustom
: Specify a custom command line (as a semicolon-separated list) to run in the source tree (Git work tree) to update it instead of running the GITCommand.

  - CTest Script variable: **CTEST_GIT_UPDATE_CUSTOM**
  - **CTest** module variable: CTEST_GIT_UPDATE_CUSTOM

GITUpdateOptions
: Command-line options to the GITCommand when updating the source.

  - CTest Script variable: **CTEST_GIT_UPDATE_OPTIONS**
  - **CTest** module variable: GIT_UPDATE_OPTIONS

HGCommand
: hg command-line tool to use if source tree is managed by Mercurial.

  - CTest Script variable: **CTEST_HG_COMMAND**
  - **CTest** module variable: none

HGUpdateOptions
: Command-line options to the HGCommand when updating the source.

  - CTest Script variable: **CTEST_HG_UPDATE_OPTIONS**
  - **CTest** module variable: none

P4Client
: Value of the -c option to the P4Command.

  - CTest Script variable: **CTEST_P4_CLIENT**
  - **CTest** module variable: CTEST_P4_CLIENT

P4Command
: p4 command-line tool to use if source tree is managed by Perforce.

  - CTest Script variable: **CTEST_P4_COMMAND**
  - **CTest** module variable: P4COMMAND

P4Options
: Command-line options to the P4Command for all invocations.

- CTest Script variable: **CTEST_P4_OPTIONS**
- **CTest** module variable: CTEST_P4_OPTIONS

P4UpdateCustom
: Specify a custom command line (as a semicolon-separated list) to run in the source tree (Perforce tree) to update it instead of running the P4Command.

- CTest Script variable: none
- **CTest** module variable: CTEST_P4_UPDATE_CUSTOM

P4UpdateOptions
: Command-line options to the P4Command when updating the source.

- CTest Script variable: **CTEST_P4_UPDATE_OPTIONS**
- **CTest** module variable: CTEST_P4_UPDATE_OPTIONS

SVNCommand
: svn command-line tool to use if source tree is managed by Subversion.

- CTest Script variable: **CTEST_SVN_COMMAND**
- **CTest** module variable: SVNCOMMAND

SVNOptions
: Command-line options to the SVNCommand for all invocations.

- CTest Script variable: **CTEST_SVN_OPTIONS**
- **CTest** module variable: CTEST_SVN_OPTIONS

SVNUpdateOptions
: Command-line options to the SVNCommand when updating the source.

- CTest Script variable: **CTEST_SVN_UPDATE_OPTIONS**
- **CTest** module variable: SVN_UPDATE_OPTIONS

UpdateCommand
: Specify the version-control command-line tool to use without detecting the VCS that manages the source tree.

- CTest Script variable: **CTEST_UPDATE_COMMAND**
- **CTest** module variable: <VCS>COMMAND when UPDATE_TYPE is <vcs>, else UPDATE_COMMAND

UpdateOptions
: Command-line options to the UpdateCommand.

- CTest Script variable: **CTEST_UPDATE_OPTIONS**
- **CTest** module variable: <VCS>_UPDATE_OPTIONS when UPDATE_TYPE is <vcs>, else UPDATE_OPTIONS

UpdateType
: Specify the version-control system that manages the source tree if it cannot be detected automatically. The value may be bzr, cvs, git, hg, p4, or svn.

  - CTest Script variable: none, detected from source tree
  - **CTest** module variable: UPDATE_TYPE if set, else CTEST_UPDATE_TYPE

UpdateVersionOnly
: Specify that you want the version control update command to only discover the current version that is checked out, and not to update to a different version.

  - CTest Script variable: **CTEST_UPDATE_VERSION_ONLY**

Additional configuration settings include:

NightlyStartTime
: In the Nightly dashboard mode, specify the "nightly start time". With centralized version control systems (cvs and svn), the Update step checks out the version of the software as of this time so that multiple clients choose a common version to test. This is not well-defined in distributed version-control systems so the setting is ignored.

  - CTest Script variable: **CTEST_NIGHTLY_START_TIME**
  - **CTest** module variable: NIGHTLY_START_TIME if set, else CTEST_NIGHTLY_START_TIME

CTest Configure Step

In a CTest Script, the **ctest_configure()** command runs this step. Arguments to the command may specify some of the step settings.

Configuration settings include:

ConfigureCommand
: Command-line to launch the software configuration process. It will be executed in the location specified by the BuildDirectory setting.

- CTest Script variable: CTEST_CONFIGURE_COMMAND
- CTest module variable: CMAKE_COMMAND followed by PROJECT_SOURCE_DIR

CTest Build Step

In a CTest Script, the ctest_build() command runs this step. Arguments to the command may specify some of the step settings.

Configuration settings include:

DefaultCTestConfigurationType
: When the build system to be launched allows build-time selection of the configuration (e.g. Debug, Release), this specifies the default configuration to be built when no -C option is given to the ctest command. The value will be substituted into the value of MakeCommand to replace the literal string ${CTEST_CONFIGURATION_TYPE} if it appears.

  - CTest Script variable: CTEST_CONFIGURATION_TYPE
  - CTest module variable: DEFAULT_CTEST_CONFIGURATION_TYPE, initialized by the CMAKE_CONFIG_TYPE environment variable

MakeCommand
: Command-line to launch the software build process. It will be executed in the location specified by the BuildDirectory setting.

  - CTest Script variable: CTEST_BUILD_COMMAND
  - CTest module variable: MAKECOMMAND, initialized by the build_command() command

UseLaunchers
: For build trees generated by CMake using one of the Makefile Generators or the Ninja generator, specify whether the CTEST_USE_LAUNCHERS feature is enabled by the CTestUseLaunchers module (also included by the CTest module). When enabled, the generated build system wraps each invocation of the compiler, linker, or custom command line with a “launcher” that communicates with CTest via environment variables and files to report granular build warning and error information. Otherwise, CTest must “scrape” the build output log for diagnostics.

  - CTest Script variable: CTEST_USE_LAUNCHERS
  - CTest module variable: CTEST_USE_LAUNCHERS

CTest Test Step

In a CTest Script, the `ctest_test()` command runs this step. Arguments to the command may specify some of the step settings.

Configuration settings include:

`TestLoad`
: While running tests in parallel (e.g. with `-j`), try not to start tests when they may cause the CPU load to pass above a given threshold.

  - CTest Script variable: `CTEST_TEST_LOAD`
  - `CTest` module variable: `CTEST_TEST_LOAD`

`TimeOut`
: The default timeout for each test if not specified by the `TIMEOUT` test property.

  - CTest Script variable: `CTEST_TEST_TIMEOUT`
  - `CTest` module variable: `DART_TESTING_TIMEOUT`

CTest Coverage Step

In a CTest Script, the `ctest_coverage()` command runs this step. Arguments to the command may specify some of the step settings.

Configuration settings include:

`CoverageCommand`
: Command-line tool to perform software coverage analysis. It will be executed in the location specified by the `BuildDirectory` setting.

  - CTest Script variable: `CTEST_COVERAGE_COMMAND`
  - `CTest` module variable: `COVERAGE_COMMAND`

`CoverageExtraFlags`
: Specify command-line options to the `CoverageCommand` tool.

  - CTest Script variable: `CTEST_COVERAGE_EXTRA_FLAGS`
  - `CTest` module variable: `COVERAGE_EXTRA_FLAGS`

CTest MemCheck Step

In a CTest Script, the `ctest_memcheck()` command runs this step. Arguments to the command may specify some of the step settings.

Configuration settings include:

`MemoryCheckCommand`
: Command-line tool to perform dynamic analysis. Test command lines will be launched through this tool.

- CTest Script variable: `CTEST_MEMORYCHECK_COMMAND`
- `CTest` module variable: `MEMORYCHECK_COMMAND`

`MemoryCheckCommandOptions`
: Specify command-line options to the `MemoryCheckCommand` tool. They will be placed prior to the test command line.

- CTest Script variable: `CTEST_MEMORYCHECK_COMMAND_OPTIONS`
- `CTest` module variable: `MEMORYCHECK_COMMAND_OPTIONS`

`MemoryCheckType`
: Specify the type of memory checking to perform.

- CTest Script variable: `CTEST_MEMORYCHECK_TYPE`
- `CTest` module variable: `MEMORYCHECK_TYPE`

`MemoryCheckSanitizerOptions`
: Specify options to sanitizers when running with a sanitize-enabled build.

- CTest Script variable: `CTEST_MEMORYCHECK_SANITIZER_OPTIONS`
- `CTest` module variable: `MEMORYCHECK_SANITIZER_OPTIONS`

`MemoryCheckSuppressionFile`
: Specify a file containing suppression rules for the `MemoryCheckCommand` tool. It will be passed with options appropriate to the tool.

- CTest Script variable: `CTEST_MEMORYCHECK_SUPPRESSIONS_FILE`
- `CTest` module variable: `MEMORYCHECK_SUPPRESSIONS_FILE`

Additional configuration settings include:

`BoundsCheckerCommand`

Specify a `MemoryCheckCommand` that is known to be command-line compatible with Bounds Checker.

- CTest Script variable: none
- **`CTest`** module variable: none

`PurifyCommand`

Specify a `MemoryCheckCommand` that is known to be command-line compatible with Purify.

- CTest Script variable: none
- **`CTest`** module variable: `PURIFYCOMMAND`

`ValgrindCommand`

Specify a `MemoryCheckCommand` that is known to be command-line compatible with Valgrind.

- CTest Script variable: none
- **`CTest`** module variable: `VALGRIND_COMMAND`

`ValgrindCommandOptions`

Specify command-line options to the `ValgrindCommand` tool. They will be placed prior to the test command line.

- CTest Script variable: none
- **`CTest`** module variable: `VALGRIND_COMMAND_OPTIONS`

CTest Submit Step

In a CTest Script, the **`ctest_submit()`** command runs this step. Arguments to the command may specify some of the step settings.

Configuration settings include:

`BuildName`

Describe the dashboard client platform with a short string. (Operating system, compiler, etc.)

- CTest Script variable: **`CTEST_BUILD_NAME`**
- **`CTest`** module variable: `BUILDNAME`

`CDashVersion`

Specify the version of CDash on the server.

- CTest Script variable: none, detected from server
- CTest module variable: CTEST_CDASH_VERSION

CTestSubmitRetryCount
: Specify a number of attempts to retry submission on network failure.

- CTest Script variable: none, use the ctest_submit() RETRY_COUNT option.
- CTest module variable: CTEST_SUBMIT_RETRY_COUNT

CTestSubmitRetryDelay
: Specify a delay before retrying submission on network failure.

- CTest Script variable: none, use the ctest_submit() RETRY_DELAY option.
- CTest module variable: CTEST_SUBMIT_RETRY_DELAY

CurlOptions
: Specify a semicolon-separated list of options to control the Curl library that CTest uses internally to connect to the server. Possible options are CURLOPT_SSL_VERIFYPEER_OFF and CURLOPT_SSL_VERIFYHOST_OFF.

- CTest Script variable: CTEST_CURL_OPTIONS
- CTest module variable: CTEST_CURL_OPTIONS

DropLocation
: The path on the dashboard server to send the submission.

- CTest Script variable: CTEST_DROP_LOCATION
- CTest module variable: DROP_LOCATION if set, else CTEST_DROP_LOCATION

DropMethod
: Specify the method by which results should be submitted to the dashboard server. The value may be cp, ftp, http, https, scp, or xmlrpc (if CMake was built with support for it).

- CTest Script variable: CTEST_DROP_METHOD
- CTest module variable: DROP_METHOD if set, else CTEST_DROP_METHOD

DropSite
: The dashboard server name (for ftp, http, and https, scp, and xmlrpc).

- CTest Script variable: CTEST_DROP_SITE
- CTest module variable: DROP_SITE if set, else CTEST_DROP_SITE

DropSitePassword

The dashboard server login password, if any (for `ftp`, `http`, and `https`).

- CTest Script variable: **`CTEST_DROP_SITE_PASSWORD`**
- **`CTest`** module variable: `DROP_SITE_PASSWORD` if set, else `CTEST_DROP_SITE_PASWORD`

`DropSiteUser`
The dashboard server login user name, if any (for `ftp`, `http`, and `https`).

- CTest Script variable: **`CTEST_DROP_SITE_USER`**
- **`CTest`** module variable: `DROP_SITE_USER` if set, else `CTEST_DROP_SITE_USER`

`IsCDash`
Specify whether the dashboard server is CDash or an older dashboard server implementation requiring `TriggerSite`.

- CTest Script variable: **`CTEST_DROP_SITE_CDASH`**
- **`CTest`** module variable: `CTEST_DROP_SITE_CDASH`

`ScpCommand`
`scp` command-line tool to use when `DropMethod` is `scp`.

- CTest Script variable: **`CTEST_SCP_COMMAND`**
- **`CTest`** module variable: `SCPCOMMAND`

`Site`
Describe the dashboard client host site with a short string. (Hostname, domain, etc.)

- CTest Script variable: **`CTEST_SITE`**
- **`CTest`** module variable: `SITE`, initialized by the **`site_name()`** command

`TriggerSite`
Legacy option to support older dashboard server implementations. Not used when `IsCDash` is true.

- CTest Script variable: **`CTEST_TRIGGER_SITE`**
- **`CTest`** module variable: `TRIGGER_SITE` if set, else `CTEST_TRIGGER_SITE`

See Also

The following resources are available to get help using CMake:

Home Page

https://cmake.org

The primary starting point for learning about CMake.

Frequently Asked Questions

https://cmake.org/Wiki/CMake_FAQ

A Wiki is provided containing answers to frequently asked questions.

Online Documentation

https://cmake.org/documentation

Links to available documentation may be found on this web page.

Mailing List

https://cmake.org/mailing-lists

For help and discussion about using cmake, a mailing list is provided at cmake@cmake.org. The list is member-post-only but one may sign up on the CMake web page. Please first read the full documentation at https://cmake.org before posting questions to the list.