Command Line Reference for cpptestcc

The following options are available:

You can run the following command to print out the available options to the console: `cpptestcc -help`

-compiler <name|path>

Specifies the name of the compiler configuration you want to use for code analysis and instrumentation. See Supported Compilers for the list of supported compilers or use the `-list-compilers` command line option to print out the list of supported compilers to the console.

Configuration file format (see -psrc): `cpptestcc.compiler <name>`

Examples:

- `cpptestcc -compiler gcc_3_4`
- `cpptestcc -compiler vc_11_0`

-listcompilers

Prints out the names of all supported compiler configurations.

Configuration file format (see -psrc): `cpptestcc.listCompilers`

-include <file|pattern> and -exclude <file|pattern>

Includes into or excludes from the instrumentation scope all the file(s) that match the specified pattern.

Final filtering is determined only after all include/exclude entries have been specified in the order of their specification.

The following wildcards are supported:

- ? - Any character
- * - Any sequence of characters

To prevent shells from expanding * wildcards to the list of files or directories, you can use the `regex:` prefix to specify the value.

These options can be specified multiple times.

Configuration file format (see -psrc): `cpptestcc.include <path|pattern>`

Example 1:

Sample project layout:

```
<project root>
 + external_libs
 + src
 + include
```

If your project has the above layout, the following command will exclude all the files in the `external_libs` directory from instrumentation scope:

```
cpptestcc -include regex:*/<project root>/* -exclude regex:*/<project root>/external_libs <other command line options>
```

Example 2:

Sample project layout:

```
<project root>
<sourcefiles>.cpp
<headerfiles>.hpp
```

If your project has the above layout, the following command will only instrument the header files (the source files will not be instrumented):

```
cpptestcc -include regex:* -exclude regex:*.cpp <remaining part of cmd>
```

-ignore <pattern>

Specifies the source files that will be ignored during processing. The files that match the specified pattern will be compiled, but they will not be parsed or instrumented.

> **-ignore vs. -exclude**
>
> The `-ignore` option completely removes the specified file from processing so that it is not parsed by the coverage engine.
>
> The `-include/-exclude` filters are applied after source code is parsed, which allows you to selectively instrument or not instrument header files.

You can use the `-ignore` option to reduce build time overhead by ignoring coverage analysis on some sections of the code (such as external libraries) or to ignore specific file that expose parse errors or other problems during processing.

The following wildcards are supported:

- ? - Any character
- * - Any sequence of characters

To prevent shells from expanding * wildcards to the list of files or directories, you can use the `regex:` prefix to specify the value.

This option can be specified multiple times.

Configuration file format (see -psrc): `cpptestcc.ignore <path|pattern>`

Example:

```
cpptestcc -ignore "*/Lib/*" <remaining part of cmd>
cpptestcc -ignore regex:*/file.c <remaining part of cmd>
cpptestcc -ignore c:/proj/file.c <remaining part of cmd>
cpptestcc -ignore "*/MyLib/*.cpp" -ignore file.cpp <remaining part of cmd>
```

-line-coverage

Enables collecting line coverage.

Runtime coverage results are being written to the results log as the code is executed. This imposes some overhead on the tested code execution time, it but it allows you to ensure that that coverage data is collected even if the application crashes.

Configuration file format (see -psrc): `cpptestcc.lineCoverage`

-optimized-line-coverage

Enables collecting optimized line coverage.

Runtime coverage results are stored in memory and then written to the results log either after the application finishes or on user request. This results in better performance, but results may be lost if the application crashes.

Configuration file format (see -psrc): `cpptestcc.optimizedLineCoverage`

-coverage-early-init

Enables initializing the coverage module at the beginning of the application entry point.

Configuration file format (see -psrc): `cpptestcc.coverageEarlyInit [true|false]`

-coverage-auto-finalization

If enabled, collecting coverage will be automatically finalized at application exit. This option is enabled by default.

Configuration file format (see -psrc): `cpptestcc.coverageAutoFinalization [true|false]`

-optimized-coverage-corruption-detection

Enables corruption detection algorithms for optimized coverage metrics.

Configuration file format (see -psrc): `cpptestcc.optimizedCoverageCorruptionDetection [true|false]`

-template-coverage

Enables collecting coverage for template classes and functions

Configuration file format (see -psrc): `cpptestcc.templateCoverage [true|false]`

-workspace <path>

Specifies a custom directory where information about code structure will be stored during code analysis and instrumentation. The `cpptestcc` tool will use the information to generate the final coverage report.

By default, the information is stored in the working directory of the current compilation. If your compilation uses more than one working directory, we recommend that you specify a custom directory to ensure that all coverage data is stored in the same location. The workspace location must be the same for `cpptestcc` and `cpptestcli`.

Configuration file format (see -psrc): `cpptestcc.workspace <path>`

-psrc <file>

Specifies the path to a configuration file where you can configure additional `cpptestcc` options.

By default, `cpptestcc` attempts to read the `.psrc` file located in either the current working directory or in the user HOME directory. This option allows you to specify a custom location of the configuration file.

If an option is configured both in the command line and in the configuration file, `cpptestcc` will use the value specified in the command line.

Configuration file format: `pscom.psrc <file>`

-help

Prints out the help message and exits.

Configuration file format (see -psrc): `cpptestcc.help`