

# Gcovr User Guide

<b>COLLABORATORS</b>
----------------------

	<i>TITLE :</i> Gcovr User Guide		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		September 2, 2013	

## Contents

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Getting Started</b>	<b>1</b>
2.1	Tabular Output of Code Coverage . . . . .	2
2.2	Tabular Output of Branch Coverage . . . . .	3
2.3	XML Output . . . . .	3
2.4	HTML Output . . . . .	4
<b>3</b>	<b>The <code>gcovr</code> Command</b>	<b>5</b>
<b>4</b>	<b>Installation</b>	<b>6</b>
<b>5</b>	<b>Status and Future Plans</b>	<b>6</b>
<b>6</b>	<b>Acknowledgements</b>	<b>7</b>
<b>A</b>	<b>Testing Gcovr</b>	<b>7</b>

## Abstract

Gcovr provides a utility for managing the use of the GNU `gcov` utility and generating summarized code coverage results. This command is inspired by the Python `coverage.py` package, which provides a similar utility in Python. The `gcovr` command produces either compact human-readable summary reports, machine readable XML reports (in `Cobertura` format) or simple HTML reports. Thus, `gcovr` can be viewed as a command-line alternative to the `lcov` utility, which runs `gcov` and generates an HTML-formatted report. This documentation describes Gcovr 3.0.

---

## 1 Overview

Gcovr is a Python package that includes a self-contained `gcovr` command. Gcovr is an extension of `gcov`, a GNU utility that summarizes the lines of code that are executed - or "covered" - while running an executable. The `gcovr` command interprets `gcov` data files to summarize code coverage in several formats:

- Text output with coverage statistics indicated with summary statistics and lists of uncovered line, and
- XML output that is compatible with the Cobertura code coverage utility.

The [Gcovr Home Page](http://gcovr.com) is <http://gcovr.com>. This webpage contains links for documentation in [HTML](#), [PDF](#), and [EPUB](#) formats. The [Gcovr Home Page](#) also includes developer resources (e.g. [automated test results](#)). Gcovr is available under the [BSD](#) license.

The Gcovr User Guide provides the following documentation:

- [Getting Started](#): Some simple examples that illustrate how to use Gcovr
- [The gcovr Command](#): Description of command-line options for `gcovr`
- [Installation](#): How to install Gcovr
- [Status and Future Plans](#): Comments on the past, present and future of Gcovr

## 2 Getting Started

The `gcovr` command provides a summary of the lines that have been executed in a program. Code coverage statistics help you discover untested parts of a program, which is particularly important when assessing code quality. Well-tested code is a characteristic of high quality code, and software developers often assess code coverage statistics when deciding if software is ready for a release.

The `gcovr` command can be used to analyze programs compiled with GCC. The following sections illustrate the application of `gcovr` to test coverage of the following program:

```
1 // example1.cpp
2
3 #include <iostream>
4
5 #define MACRO()    if (1<0) foo(-1); else foo(1);
6
7 int foo(int param)
8 {
9     if (param)
10    {
11        return 1;
12    }
13    else
14    {
15        return 0;
16    }
17 }
18
19 void bar(int param)
20 {
21     if (param)
22     {
23         std::cout << "param not null." << std::endl;
24     }
25     else
26     {
```

```

27         std::cout << "param is null." << std::endl;
28     }
29 }
30
31
32 int main(int argc, char* argv[])
33 {
34     MACRO ()
35
36     foo(0);
37
38     return 0;
39 }

```

This code executes several subroutines in this program, but some lines in the program are not executed.

## 2.1 Tabular Output of Code Coverage

We compile `example1.cpp` with the GCC compiler as follows:

```
g++ -fprofile-arcs -ftest-coverage -fPIC -O0 example1.cpp -o program
```

Note that we compile this program without optimization, because optimization may combine lines of code and otherwise change the flow of execution in the program. Additionally, we compile with the `-fprofile-arcs -ftest-coverage -fPIC` compiler options, which add logic to generate output files that can be processed by the `gcov` command.

The compiler generates the `program` executable. When we execute this command:

```
./program
```

the files `example1.gno` and `example1.gda` are generated. These files are processed with by `gcov` to generate code coverage statistics. The `gcovr` command calls `gcov` and summarizes these code coverage statistics in various formats. For example:

```
gcovr -r .
```

generates a text summary of the lines executed:

File	Lines	Exec	Cover	Missing
example1.cpp	14	9	64%	19-29
TOTAL	14	9	64%	

Each line of this output includes a summary for a given source file, including the number of lines instrumented, the number of lines executed, the percentage of lines executed, and a summary of the line numbers that were not executed. To improve clarity, `gcovr` uses an aggressive approach to grouping uncovered lines and will combine uncovered lines separated by "non-code" lines (blank, freestanding braces, and single-line comments) into a single region. As a result, the number of lines listed in the "Missing" list may be greater than the difference of the "Lines" and "Exec" columns.

The `-r` option specifies the root directory for the files that are being analyzed. This allows `gcovr` to generate a simpler report (without absolute path names), and it allows system header files to be excluded from the analysis.

Note that `gcov` accumulates statistics by line. Consequently, it works best with a programming style that places only one statement on each line. In `example1.cpp`, the `MACRO` macro executes a branch, but `gcov` cannot discern which branch is executed.

## 2.2 Tabular Output of Branch Coverage

The `gcovr` command can also summarize branch coverage using the `--branches` option:

```
gcovr -r . --branches
```

This generates a tabular output that summarizes the number of branches, the number of branches taken and the branches that were not completely covered:

File	Branches	Taken	Cover	Missing
example1.cpp	6	4	66%	39
TOTAL	6	4	66%	

## 2.3 XML Output

The default output format for `gcovr` is to generate a tabular summary in plain text. The `gcovr` command can also generate an XML output using the `--xml` and `--xml-pretty` options:

```
gcovr -r . --xml-pretty
```

This generates an XML summary of the lines executed:

```
<?xml version="1.0" ?>
<!DOCTYPE coverage
  SYSTEM 'http://cobertura.sourceforge.net/xml/coverage-03.dtd'>
<coverage branch-rate="0.6666666666667" line-rate="0.642857142857"
  timestamp="1376247833" version="gcovr 3.0 (r2887)">
  <sources>
    <source>.</source>
  </sources>
  <packages>
    <package branch-rate="0.6666666666667" complexity="0.0"
      line-rate="0.642857142857" name="">
      <classes>
        <class branch-rate="0.6666666666667" complexity="0.0"
          filename="example1.cpp" line-rate="0.642857142857" name="example1_cpp">
            <methods/>
            <lines>
              <line branch="false" hits="1" number="32"/>
              <line branch="false" hits="1" number="34"/>
              <line branch="false" hits="1" number="36"/>
              <line branch="false" hits="1" number="38"/>
              <line branch="false" hits="2" number="7"/>
              <line branch="true" condition-coverage="100% (2/2)" hits="2" number="9">
                <conditions>
                  <condition coverage="100%" number="0" type="jump"/>
                </conditions>
              </line>
              <line branch="false" hits="1" number="11"/>
              <line branch="false" hits="1" number="15"/>
              <line branch="false" hits="0" number="19"/>
              <line branch="false" hits="0" number="21"/>
              <line branch="false" hits="0" number="23"/>
              <line branch="true" condition-coverage="50% (2/4)" hits="3" number="39">
                <conditions>
                  <condition coverage="50%" number="0" type="jump"/>
                </conditions>
              </line>
            </lines>
          </class>
        </classes>
      </package>
    </packages>
  </coverage>
```

```

    </conditions>
  </line>
  <line branch="false" hits="0" number="27"/>
  <line branch="false" hits="0" number="29"/>
</lines>
</class>
</classes>
</package>
</packages>
</coverage>

```

This XML format is in the [Cobertura XML](#) format suitable for import and display within the [Jenkins](#) and [Hudson](#) continuous integration servers using the [Cobertura Plugin](#).

The `--xml` option generates a denser XML output, and the `--xml-pretty` option generates an indented XML output that is easier to read. Note that the XML output contains more information than the tabular summary. The tabular summary shows the percentage of covered lines, while the XML output includes branch statistics and the number of times that each line was covered. Consequently, XML output can be used to support performance optimization in the same manner that `gcov` does.

## 2.4 HTML Output

The `gcovr` command can also generate a simple HTML output using the `--html` option:

```
gcovr -r . --html -o example1.html
```

This generates a HTML summary of the lines executed. In this example, the file `example1.html` is generated, which has the following output:

GCC Code Coverage Report					
Directory: .			Exec	Total	Coverage
Date: 2013-08-11			Lines: 9	14	64.3 %
Legend: low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %			Branches: 4	6	66.7 %
File	Lines		Branches		
example1.cpp	<div><div></div></div>	64.3 %	9 / 14	66.7 %	4 / 6
Generated by: <a href="#">GCOVR (Version 3.0 (r2887))</a>					

The default behavior of the `--html` option is to generate HTML for a single webpage that summarizes the coverage for all files. The HTML is printed to standard output, but the `-o` (`--output`) option is used to specify a file that stores the HTML output.

The `--html-details` option is used to create a separate web page for each file. Each of these web pages includes the contents of file with annotations that summarize code coverage. Consider the following command:

```
gcovr -r . --html --html-details -o example2.html
```

This generates the following HTML page for the file `example2.cpp`:



## GCC Code Coverage Report

Directory: .	Exec	Total	Coverage
File: <b>example2.cpp</b>	Lines: 6	7	85.7 %
Date: <b>2013-08-11</b>	Branches: 1	2	50.0 %

Line	Exec	Source
1		// example2.cpp
2		
3	1	int foo(int param)
4		{
5	1	if (param)
6		{
7		return 1;
8		}
9		else
10		{
11	1	return 0;
12		}
13		}
14		
15	1	int main(int argc, char* argv[])
16		{
17	1	foo(0);
18		
19	1	return 0;
20		}
21		

Generated by: [GCOVR \(Version 3.0 \(r2887\)\)](#)

Note that the `--html-details` option can only be used with the `-o` (`--output`) option. For example, if the `--output` option specifies the output file `coverage.html`, then the web pages generated for each file will have names of the form `coverage.<filename>.html`.

### 3 The gcovr Command

The `gcovr` command recursively searches a directory tree to find `gcov` coverage files, and generates a text summary of the code coverage. The `--help` option generates the following summary of the `gcovr` command line options:

Usage: `gcovr [options]`

A utility to run `gcov` and generate a simple report that summarizes the coverage

Options:

<code>-h, --help</code>	show this help message and exit
<code>--version</code>	Print the version number, then exit
<code>-v, --verbose</code>	Print progress messages
<code>--object-directory=OBJDIR</code>	Specify the directory that contains the <code>gcov</code> data files. <code>gcovr</code> must be able to identify the path between the <code>*.gda</code> files and the directory where <code>gcc</code> was originally run. Normally, <code>gcovr</code> can guess correctly. This option overrides <code>gcovr</code> 's normal path detection and can specify either the path from <code>gcc</code> to the <code>gda</code> file (i.e. what was passed to <code>gcc</code> 's <code>'-o'</code> option), or the path from the <code>gda</code> file to <code>gcc</code> 's

```

original working directory.
-o OUTPUT, --output=OUTPUT      Print output to this filename
-k, --keep                      Keep the temporary *.gcov files generated by gcov.  By
                                default, these are deleted.
-d, --delete                    Delete the coverage files after they are processed.
                                These are generated by the users's program, and by
                                default gcovr does not remove these files.
-f FILTER, --filter=FILTER      Keep only the data files that match this regular
                                expression
-e EXCLUDE, --exclude=EXCLUDE   Exclude data files that match this regular expression
--gcov-filter=GCOV_FILTER       Keep only gcov data files that match this regular
                                expression
--gcov-exclude=GCOV_EXCLUDE     Exclude gcov data files that match this regular
                                expression
-r ROOT, --root=ROOT            Defines the root directory.  This is used to filter
                                the files, and to standardize the output.
-x, --xml                      Generate XML instead of the normal tabular output.
--xml-pretty                    Generate pretty XML instead of the normal dense
                                format.
--html                         Generate HTML instead of the normal tabular output.
--html-details                 Generate HTML output for source file coverage.
-b, --branches                 Tabulate the branch coverage instead of the line
                                coverage.
-u, --sort-uncovered           Sort entries by increasing number of uncovered lines.
-p, --sort-percentage          Sort entries by decreasing percentage of covered
                                lines.
--gcov-executable=GCOV_CMD      Defines the name/path to the gcov executable [defaults
                                to the GCOV environment variable, if present; else
                                'gcov'].

```

The following sections illustrate the use of these command line options.

## 4 Installation

Gcovr requires virtually no installation. The `gcovr` command can be downloaded and used directly without installing additional files.

If you have [setuptools](#) or [distribute](#) installed, then you can install Gcovr from PyPI by executing

```
easy_install gcovr
```

This places the `gcovr` executable in the `bin` or `Scripts` directory for your python installation.

The `gcovr` script has been tested with many different versions of Python: 2.5 - 3.2. Note that this script has only been tested with the CPython implementation.

## 5 Status and Future Plans

The Gcovr 3.0 release is the first release that is hosted a GitHub. Previous Gcovr development was hosted at Sandia National Laboratories as part of the FAST project. However, Gcovr is now widely used outside of Sandia, and GitHub will facilitate the integration of contributions from a wider set of developers.

## 6 Acknowledgements

The following developers contributed to the Gcovr 3.0 release:

- William Hart
- John Siirola

The Gcovr documentation is generated using [AsciiDoc](#).

We would like to thank the following organizations for providing web hosting and computing resources: GitHub and Sandia National Laboratories. The development of Gcovr has been partially supported by Sandia National Laboratories. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

## A Testing Gcovr

In the `gcovr/test` directory, you can execute

```
python test_gcovr.py
```

to launch all tests. By default, this test script executes test suites on a variety of code configurations that reflect different use-cases for `gcovr`.

You can execute a specific test suite by giving its name as an argument to this test script. For example, the command

```
python test_gcovr.py GcovrXml
```

executes the `GcovrXml` test suite, which tests `gcovr` with XML output.

To run the `test_gcovr.py` script, you will need to install the [pyutilib.th](#) package. If you have [setuptools](#) or [distribute](#) installed, then you can install this package from PyPI by executing

```
easy_install pyutilib.th
```