Gcovr User Guide

COLLABORATORS			
	TITLE : Gcovr User Guide		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 6, 2016	

Contents

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; the development of gcovr was motivated by the need for text and XML reports. This documentation describes Gcovr 3.3.

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 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:

```
// example1.cpp

int foo(int param)
{
    if (param)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int main(int argc, char* argv[])
{
    foo(0);

    return 0;
}
```

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.gcno` and `example1.gcda` 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:

```
../../../scripts/gcovr -r .
```

generates a text summary of the lines executed:

```
------------------------------------------------------------------------------
                           GCC Code Coverage Report
Directory: .
------------------------------------------------------------------------------
File                                       Lines    Exec  Cover   Missing
------------------------------------------------------------------------------
example1.cpp                                   7       6    85%   7
------------------------------------------------------------------------------
TOTAL                                          7       6    85%
------------------------------------------------------------------------------
```

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:

```
../../../scripts/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:

```
------------------------------------------------------------------------------
                           GCC Code Coverage Report
Directory: .
------------------------------------------------------------------------------
File                                     Branches   Taken  Cover   Missing
------------------------------------------------------------------------------
```

```
example1.cpp                                        2       1     50%   5
------------------------------------------------------------------------------
TOTAL                                               2       1     50%
------------------------------------------------------------------------------
```

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:

```
../../../scripts/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="0.5" line-rate="0.8571428571428571"
 timestamp="1470497828" version="gcovr 3.3">
 <sources>
  <source>.</source>
 </sources>
 <packages>
  <package branch-rate="0.5" complexity="0.0" line-rate="0.8571428571428571"
   name="">
   <classes>
    <class branch-rate="0.5" complexity="0.0" filename="example1.cpp"
     line-rate="0.8571428571428571" name="example1_cpp">
     <methods/>
     <lines>
      <line branch="false" hits="1" number="3"/>
      <line branch="true" condition-coverage="50% (1/2)" hits="1" number="5">
       <conditions>
        <condition coverage="50%" number="0" type="jump"/>
       </conditions>
      </line>
      <line branch="false" hits="0" number="7"/>
      <line branch="false" hits="1" number="11"/>
      <line branch="false" hits="1" number="15"/>
      <line branch="false" hits="1" number="17"/>
      <line branch="false" hits="1" number="19"/>
     </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.

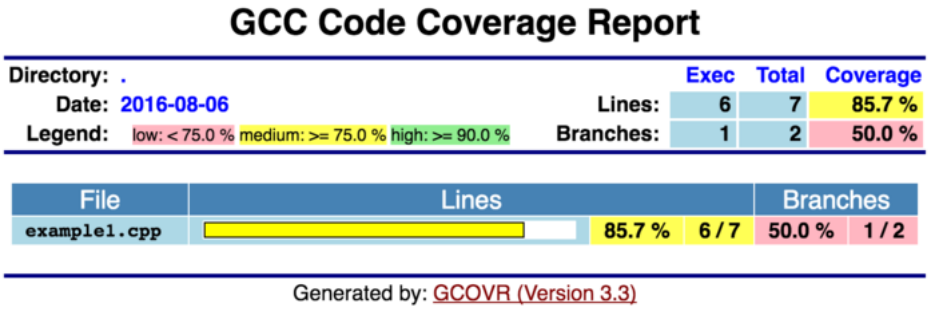
The `--xml` option generates a denser XML output, and the `--xml-pretty` option generates an indented XML output that is easier to read. Note that the XML output contains more information than the tabular summary. The tabular summary shows the percentage of covered lines, while the XML output includes 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:

```
../../../scripts/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:

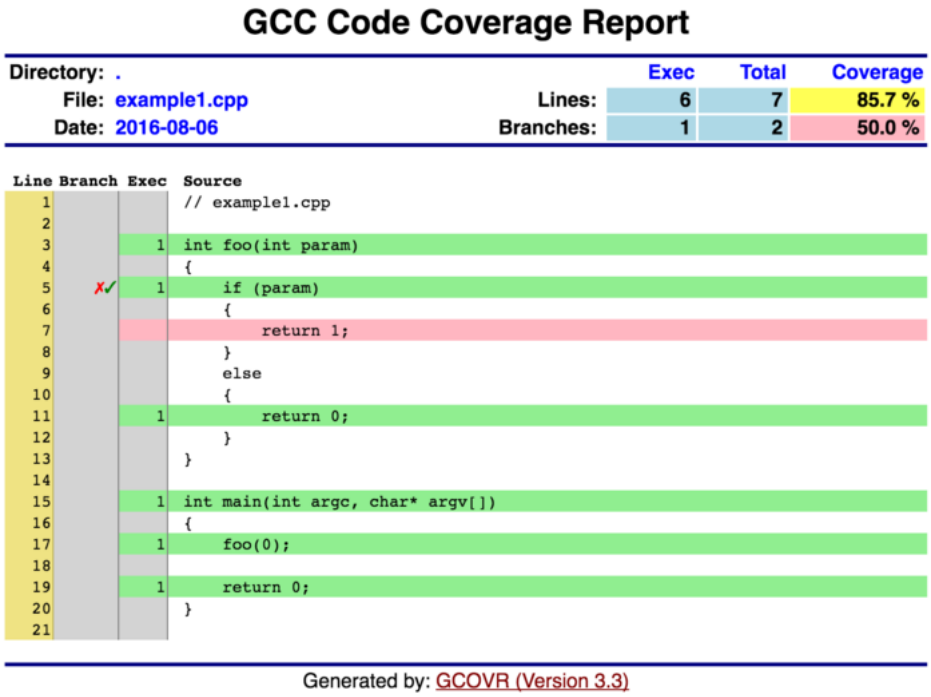


The default behavior of the `--html` option is to generate HTML for a single webpage that summarizes the coverage for all files. The HTML is printed to standard output, but the `-o` (`--output`) option is used to specify a file that stores the HTML output.

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:

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

This generates the following HTML page for the file `example1.cpp`:



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:
  -h, --help            show this help message and exit
  --version             Print the version number, then exit
  -v, --verbose         Print progress messages
  --object-directory=OBJDIR
                        Specify the directory that contains the gcov data
                        files.  gcovr must be able to identify the path
                        between the *.gcda files and the directory where gcc
                        was originally run.  Normally, gcovr can guess
                        correctly.  This option overrides gcovr's normal path
                        detection and can specify either the path from gcc to
                        the gcda file (i.e. what was passed to gcc's '-o'
                        option), or the path from the gcda file to gcc'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 for source files.  This is
                        also 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.
  --html-absolute-paths
                        Set the paths in the HTML report to be absolute
                        instead of relative
  -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'].
  --exclude-unreachable-branches
                        Exclude from coverage branches which are marked to be
                        excluded by LCOV/GCOV markers or are determined to be
```

```
                      from lines containing only compiler-generated "dead"
                      code.
  --exclude-directories=EXCLUDE_DIRS
                      Exclude directories from search path that match this
                      regular expression
  -g, --use-gcov-files  Use preprocessed gcov files for analysis.
  -s, --print-summary   Prints a small report to stdout with line & branch
                      percentage coverage
```

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 pip installed, then you can install `Gcovr` from the PyPI network servers by executing

```
pip install gcovr
```

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

The `gcovr` script has been tested with many different versions of CPython 2.7, 3.4, and 3.5, and PyPy 2.7 and 3.5.

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 3.3 release:

- ja11sop
- Jörg Kreuzberger
- Arvin Schnell
- Andrew Stone
- Attie Grande
- Mikael Salson
- Kai Blaschke
- mikkleini
- trapzero
- B Brian Bauer
- William Hart

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/tests` 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 pip installed, then you can install this package from PyPI by executing

```
pip install pyutilib.th
```