Mike Long's Blog

JUNE 2, 2010 · 4:02 PM

Continuous Code Coverage with gcc, googletest, and Hudson

Continuous Code Coverage with gcc, googletest, and Hudson

Heres a little recipe for getting code coverage metrics for your C++ unit tests using gcc, googletest and hudson.

Suppose we have a little class that we'd like to get under continuous integration:

```
#ifndef PROGRESSSTATE_H_
#define PROGRESSSTATE_H_

class ProgressState {
public:
  ProgressState(unsigned int target);
  virtual ~ProgressState();

  unsigned int getPercentage();
  void setValue(int value);
private:
  unsigned int m_value;
  unsigned int m_target;
};

#endif /* PROGRESSSTATE_H_ */
```

And here is the implementation:

```
#include "ProgressState.h"

ProgressState::ProgressState(unsigned int target)
 : m_value(0), m_target(target)
{}

ProgressState::~ProgressState(){}

void ProgressState::setValue(int value)
{
  if(value < 0){
    m_value = 0;
  }
  else if(value > (int)m_target){
    m_value = m_target;
  }
  else{
    m_value = value;
  }
}

unsigned int ProgressState::getPercentage()
{
  return m_value * 100 / m_target;
```

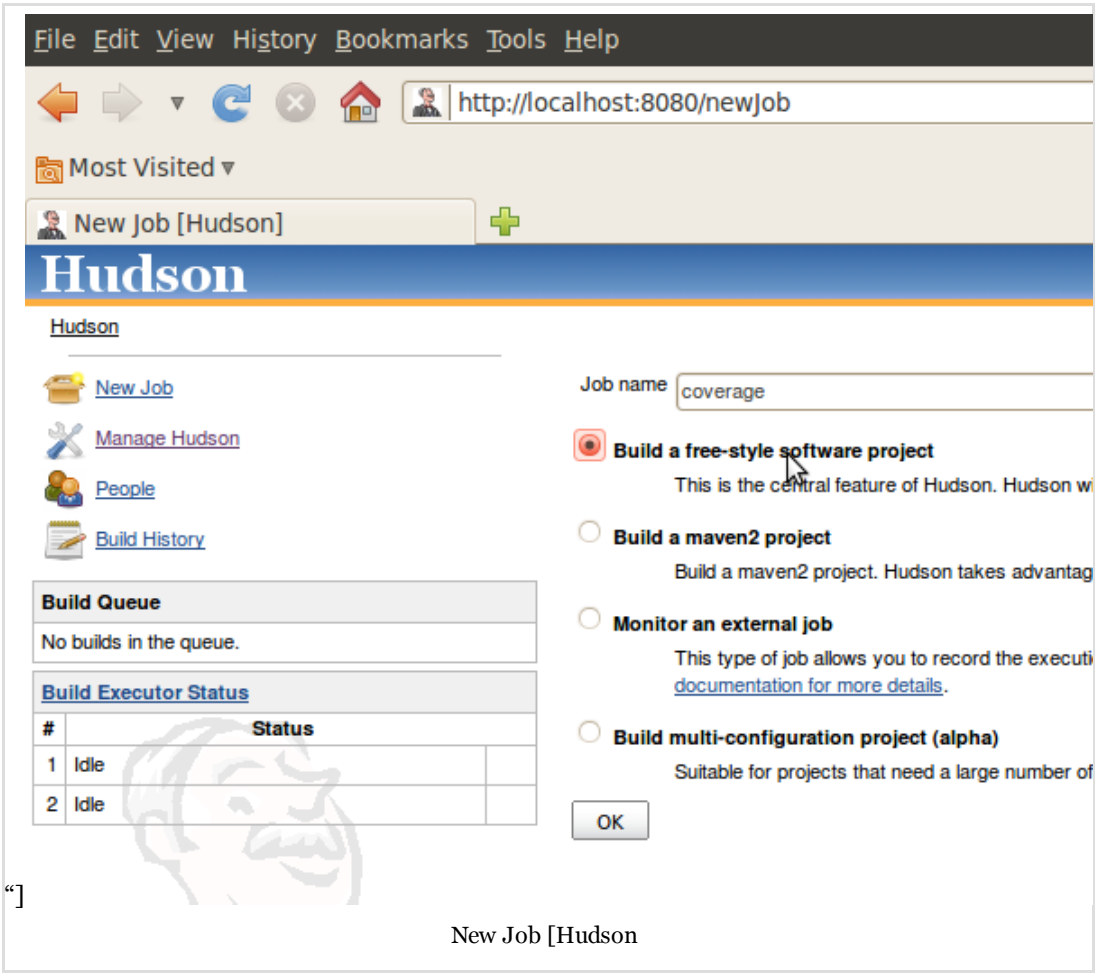
```
25 }
```

Continuous Build

First we must install Hudson on the build machine. Hudson is available for most operating systems and it is pretty easy to install. Once hudson is up and running we will need to install these plugins from our hudson management console:

- Git Plugin: To poll the git repository for your code
- Cobertura Plugin: To gather the test coverage results a publish them with the build

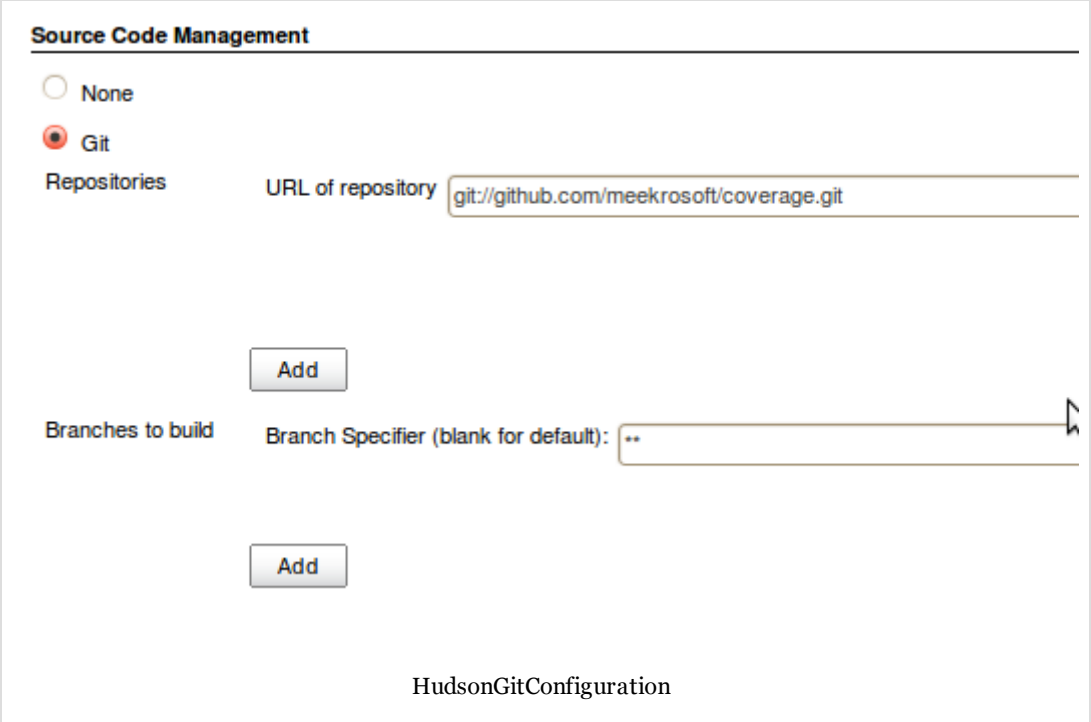
The next step is to create a hudson job for building our project.



“]

New Job [Hudson

Then we need a way for Hudson to get access to the source code. The typical way to do this is to get hudson to poll from your version control system. Using git this is pretty easy.



HudsonGitConfiguration

One thing to remember is to set up the git user account for the hudson server. To do this in Ubuntu you need to issue the following commands:

```
mike@mike-laptop:~$  sudo -s -u hudson
hudson@mike-laptop:~$ cd  /var/lib/hudson/jobs/coverage/workspace
hudson@mike-laptop:/var/lib/hudson/jobs/coverage/workspace$  ls -al
total 20
drwxr-xr-x 5 hudson nogroup 4096 2010-05-31 22:33  .
drwxr-xr-x 4 hudson nogroup 4096 2010-05-31 22:33 ..
drwxr-xr-x  3 hudson nogroup 4096 2010-05-31 22:33 Debug
drwxr-xr-x 8 hudson  nogroup 4096 2010-05-31 22:33 .git
-rw-r--r-- 1 hudson nogroup 0  2010-05-31 22:33 README
drwxr-xr-x 2 hudson nogroup 4096 2010-05-31  22:33 src
hudson@mike-laptop:/var/lib/hudson/jobs/coverage/workspace$
hudson@mike-laptop:/var/lib/hudson/jobs/coverage/workspace$ git  config user.email "some@ema
hudson@mike-laptop:/var/lib/hudson/jobs/coverage/workspace$  git config user.name "hudson"
hudson@mike-laptop:/var/lib/hudson/jobs/coverage/workspace$
```

Generating the coverage statistics with gcov

To get gcc to instrument the generated binary with coverage and profiling (necessary for branch stats) code we must provide the compile with these two additional options: **-fprofile-arcs -ftest-coverage**. And we must link the final executable with **-lgcov**. Now when the coverage test executable is run it will output .gcda and .gcno files with the coverage statistics. These settings are set in the makefiles for the project.

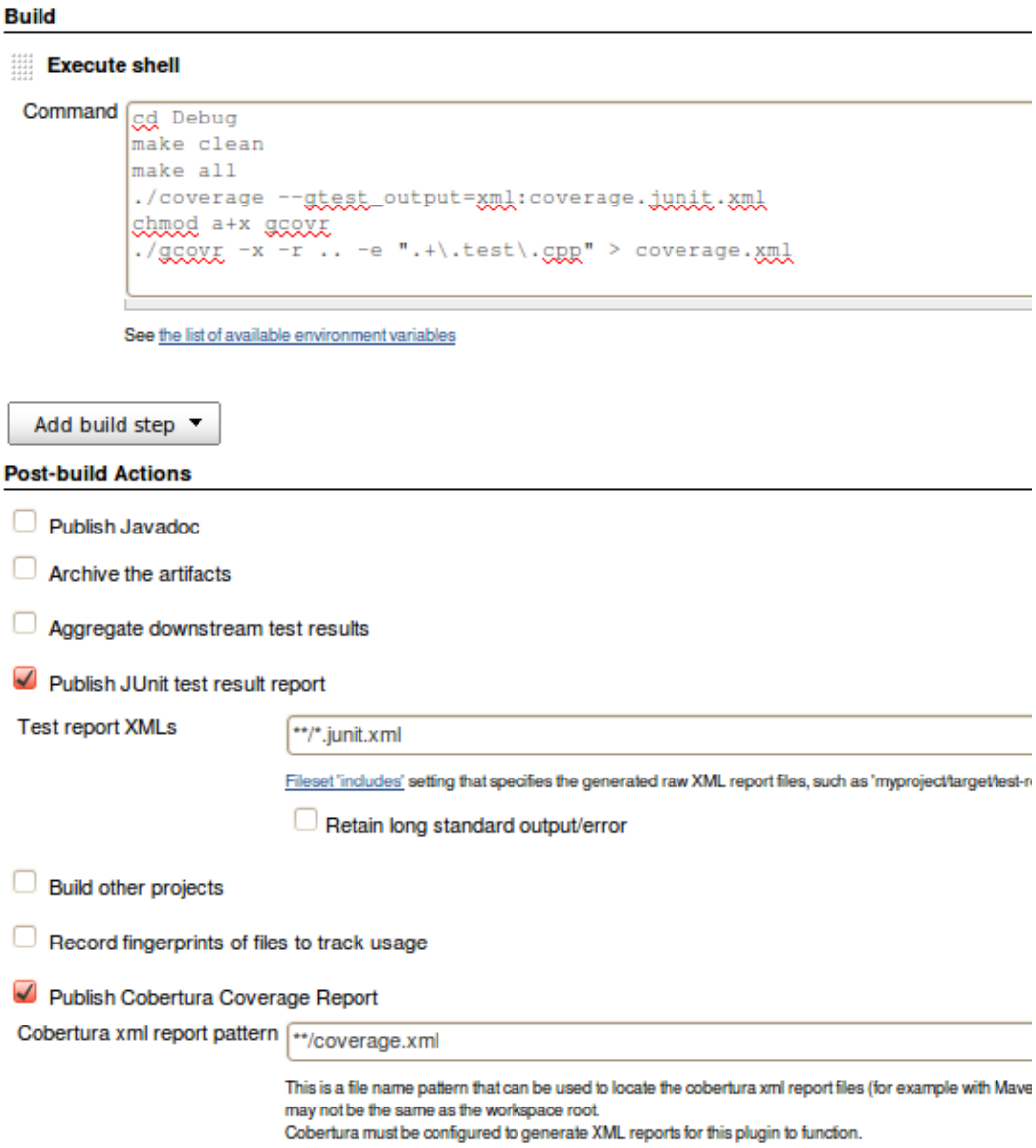
Write some tests

Breaking from TDD conventional wisdom for the purposes of this article, lets write some tests for the production code we already have.

```
1  #include  <gtest/gtest.h>
2  #include "ProgressState.h"
3
4  TEST(ProgressStateTest,  zeroValueOfValidTargetIsZeroPercent)
5  {
6    ProgressState  progress(100);
```

```
7     progress.setValue(0);
8     ASSERT_EQ((unsigned int) 0,  progress.getPercentage());
9   }
10
11  TEST(ProgressStateTest,  negativeValueOfValidTargetIsZeroPercent)
12  {
13    ProgressState  progress(100);
14    progress.setValue(-100);
15    ASSERT_EQ((unsigned int)  0, progress.getPercentage());
16  }
17
18  TEST(ProgressStateTest,  valueEqualTargetIsHundredPercent)
19  {
20    ProgressState progress(200);
21    progress.setValue(200);
22    ASSERT_EQ((unsigned int) 100,  progress.getPercentage());
23  }
```

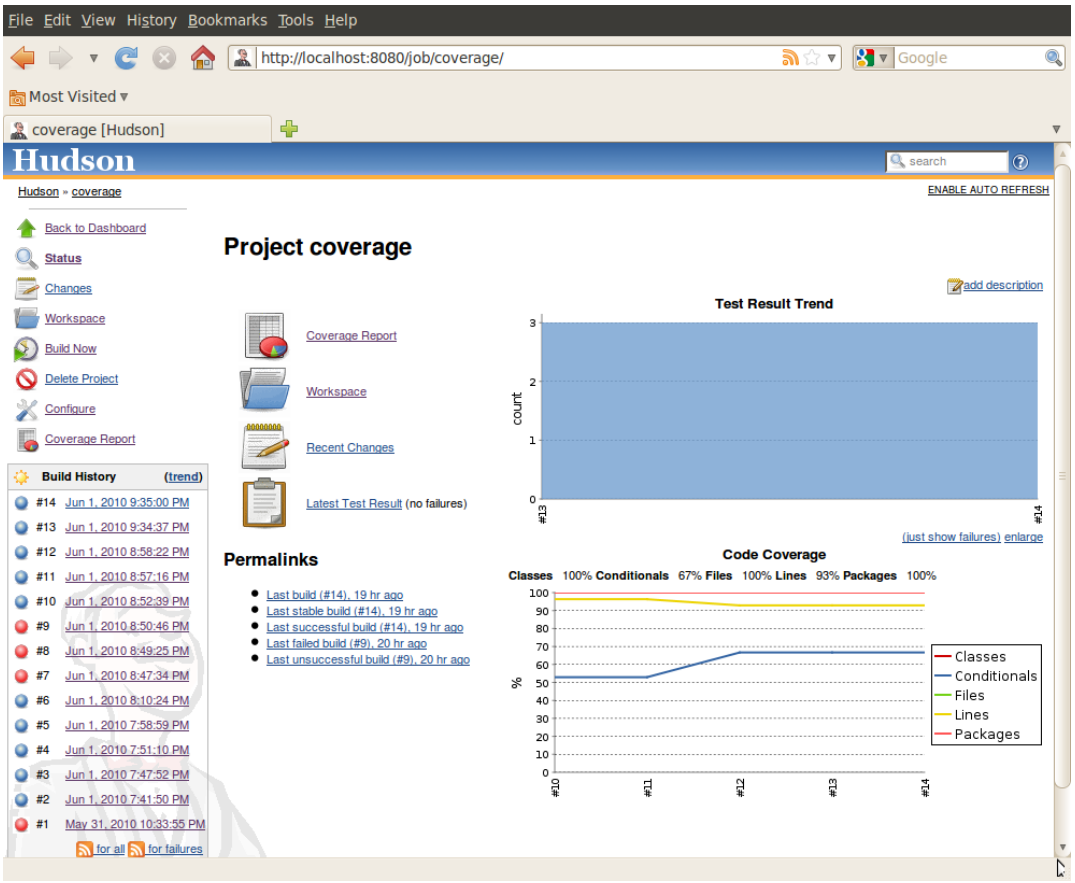
This looks like a pretty good start, right? Now lets run the tests and collect the statistics in Hudson. To do this we need to add a build step to our Hudson job.



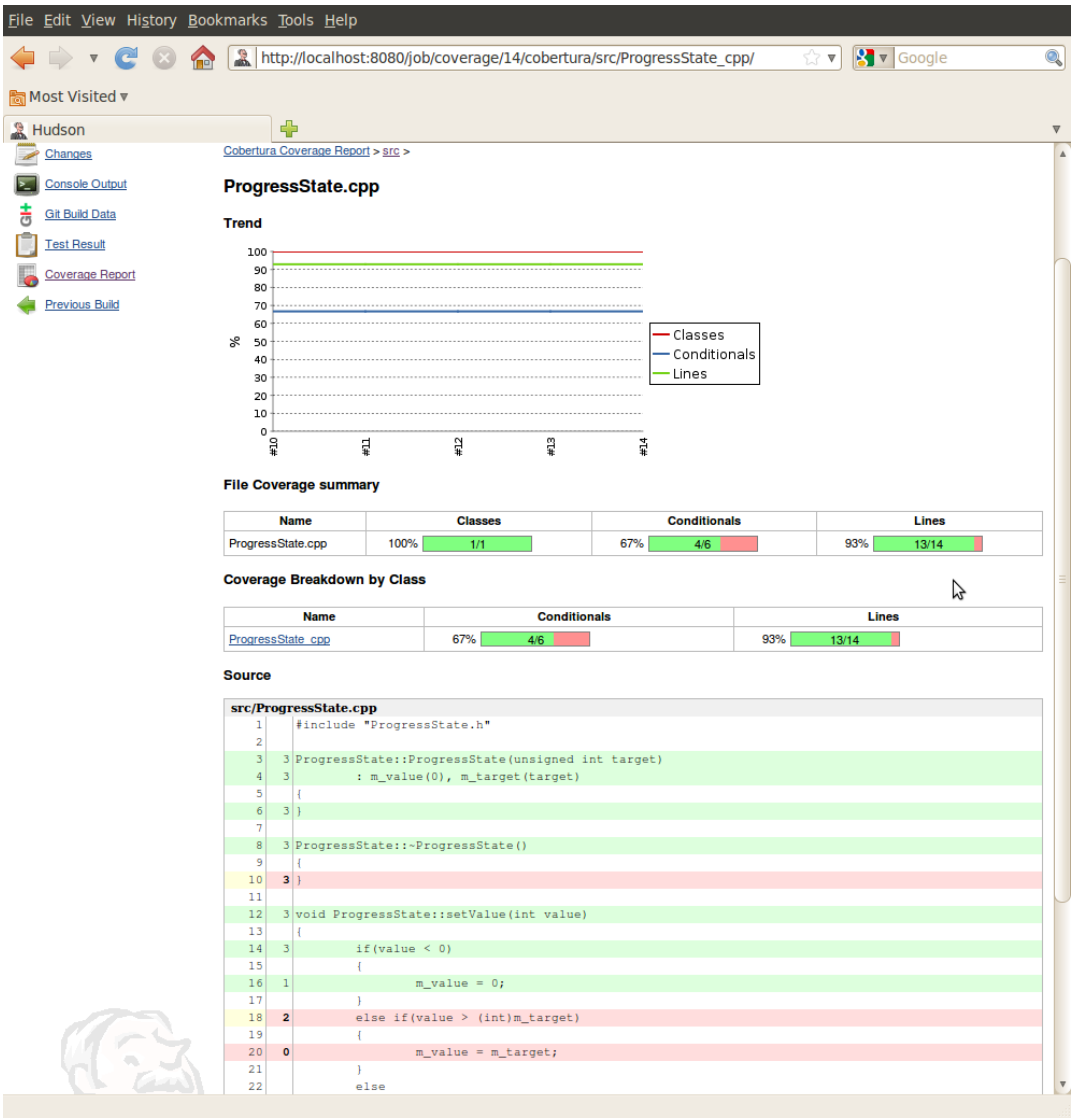
The first three lines of the command simply execute the build. The command on line 4 executes the binary test application we have built, and outputs the test result summary to a junit format XML file.

The final two commands are where the magic is. This executes the gcovr script, a handy python script that converts the gcov output to a Cobertura-style XML file.

Then we have to tell hudson to search the build workspace for the junit and coverage xml files as a post-build action. Now when we run the build we get nice overview charts trending out unit test results and code coverage.



Then when we drill down into the specific source file we can see quite clearly that we have missed a test scenario.



Links

Hudson Continuous Integration Server:

http://hudson-ci.org/

All the source code and makefiles are available publicly on my github account:

http://github.com/meekrosoft/coverage

William E. Hart's Blog post on gcovr:

http://wehart.blogspot.com/2009/07/summarizing-gcov-coverage-statistics.html

- BEYOND: Two Souls and Tribeca http://bit.ly/18vBc4n

-
-
-
- http://bit.ly/18vBc4n

29 Responses to *Continuous Code Coverage with gcc, googletest, and Hudson*

j2

August 11, 2010 at 3:18 pm

Works great! Thanks a lot!

Easily adapted this to work within our SCons build.

Reply

Scott

January 18, 2011 at 6:21 pm

Thanks for posting this, it worked the first time!
I'm using cppunit, but the rest of my setup is identical.

Reply

meekrosoft

January 19, 2011 at 7:43 am

Great, I'm glad it worked well for you both!

Reply

Jose A.

May 18, 2011 at 7:23 am

I tried it but I can't get any information with gcovr. Dou you have some suggestion ?

Reply

meekrosoft

May 18, 2011 at 7:49 am

Jose, what output do you get when you run gcovr? Can you give the command you entered in the shell?

Reply

Jose A.

May 18, 2011 at 8:47 am

The command is ' gcovr -r . -v '
and I get the follow output:
Scanning directory . for gcda files...
Found 6 files
Running gcov: 'gcov /home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest/ftest.gcda –branch-counts –branch-probabilities –preserve-paths –object-directory /home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest' in '/home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest'
Running gcov: 'gcov /home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest/fractiontest.gcda –branch-counts –branch-probabilities –preserve-paths –object-directory /home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest' in '/home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest'
Running gcov: 'gcov /home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest/Fraction.gcda –branch-counts –branch-probabilities –preserve-paths –object-directory /home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest' in '/home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest'
Running gcov: 'gcov /home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest/linux/ftest.gcda –branch-counts –branch-probabilities –preserve-paths –object-directory /home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest/linux' in

‘/home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest/linux’
Running gcov: ‘gcov /home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest/linux/fractiontest.gcda –branch-counts –branch-probabilities –preserve-paths –object-directory /home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest/linux’ in ‘/home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest/linux’
Running gcov: ‘gcov /home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest/linux/Fraction.gcda –branch-counts –branch-probabilities –preserve-paths –object-directory /home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest/linux’ in ‘/home/jnieto/.jenkins/jobs/FractionTest/workspace/view_fraction_jenkins/Unix-vob/src/PrjFracTest/linux’
Gathered coveraged data for 0 files
————————————————————————————
File Lines Exec Cover Missing
————————————————————————————
————————————————————————————
TOTAL 0 0 –%
————————————————————————————

Reply

Jose A.

May 19, 2011 at 7:38 am

I solved the problem, It was a problem with the Operating System language. gcovr doesn’t work in Spanish.
One question, I cannot get to source code of file, Do you know what it could be the problem ?
thanks.

Reply

meekrosoft

May 23, 2011 at 7:51 pm

Sorry Jose, I don’t know what that could be. If you find find out the solution please let me know. Good luck!

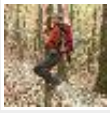
Reply

Jose A.

May 27, 2011 at 9:10 am

OK. Problem solve. It was a problem about the relative paths. Cobertura didn’t find the path to source files.
Thank you.

Corey Downing (@coreydowning)



September 12, 2011 at 9:47 pm

After some issues I managed to get this working with my C++/Qt project but the code coverage results don't seem particularly useful because it doesn't think I have coverage on lines that allocate memory on the heap using new or new[]. Is there a way to obtain more useful coverage information for my application using gcov or another open source tool that I can integrate with Hudson like this?

Reply

meekrosoft

September 13, 2011 at 6:57 am

Mmm, that's interesting. I'm not really sure what that could be, but it might be related to this issue with dynamic and non-dynamic destruction:
http://stackoverflow.com/questions/7199360/what-is-the-branch-in-the-destructor-reported-by-gcov

Let me know if you find what the cause is, I find this interesting.

Reply

Pingback: Diverse läsning | La Vie est Belle

Roman

February 2, 2012 at 1:50 pm

Thank you! It works great with QTestLib too.

Reply

Nick Thompson

March 3, 2012 at 9:09 am

We are mandated to use Bullseye for code coverage rather than gcov,
I know that it can output results in csv, xml or html but does any one have any information using it with husdon/jenkins?

Reply

meekrosoft

March 3, 2012 at 2:49 pm

I'm not sure about Bullseye as they don't seem to display any technical specs on their product website. If the coverage results come in csv or xml it should be pretty simple to create a translator to coburtura format. Since there is an html output there is also the simple option of publishing this using the HTML publisher plugin:
https://wiki.jenkins-ci.org/display/JENKINS/HTML+Publisher+Plugin

Reply

Nick Thompson

March 3, 2012 at 3:05 pm

Cheers for the reply. We have the report on the page type approach as a back up we would like to utilise the output in a more integrated manner. I haven't played around with coburtura yet, or even seen the expected input file format yet; but yes I think thats the easiest approach if there is not a dedicated Bullseye plugin.

Thomas richard

March 6, 2012 at 9:16 am

Very useful. However I had to add a few things to make it work using out of source compilation (with cmake)

First of all when having a project on unix you have to insure that your project workspace does not contain spaces (there is an advanced options in my version of jenkins to setup the project workspace "./workspace/projectname" works fine – or you can just avoid spaces in the project name).

You don't need to set gcovr to be executable every-time, if it's in your /usr/bin directory and already executable like me it works fine.

So the project is set up as follow:
jenkins_proj_root/project/src are the sources for project
jenkins_proj_root/project/unit_tests are the unit tests sources
jenkins_proj_root/build/debug is the out of source binary directory generated by cmake (here for debug – I also have release)

To run gcovr, add a shell execution:
PROJ_ROOT=`pwd` # to get project root – hence no spaces in workspace
cd build/debug/project_unit_tests_dir # go to unit test bin dir
./unit_tests –gtest_output=xml:../unit_tests.xml # run unit tests
gcovr -x -e "unit_tests/*" –root=${PROJ_ROOT}/project/ CMakeFiles/project_unit_tests.dir > coverage.xml

-x for xml output
-e to exclude code from coverage (here the unit_tests)
–root to specify where are the files (it had to be a global path to be available in Jenkins – hence the pwd)
running from the CMake files dir where are the gcov files.

Reply

Khurram

June 19, 2012 at 7:07 pm

Very nice!! Can we get code coverage using g++ as well?

Reply

meekrosoft

June 20, 2012 at 12:14 am

Yes, this is working with g++

Reply

Steve

July 9, 2012 at 4:20 pm

Just a note for a problem that plagued me with Jenkins. I found that I had to use -r to specify the full path for the pwd – and it had to have a trailing "/" – until I discovered this, Cobertura (frustratingly) failed to display annotated source.

Reply

meekrosoft

July 10, 2012 at 7:28 pm

Thanks for the tip!

Reply

Cathy

July 10, 2012 at 8:39 am

I'm currently trying to use Cobertura and LTP GCOV (LCOV) in Hudson to generate results from one set of source codes. The biggest issue I've seen is that the results are very different depending on what tool was used.

Does anyone have an idea why the data/output is different and what would be better tool to use with Hudson?

Reply

meekrosoft

July 10, 2012 at 7:29 pm

Sorry Cathy, I have now experience with LTP GCOV. Please do drop me a line if you figure this out.

Reply

naushad

October 8, 2012 at 12:27 pm

The cobertura does not take care of mocked functions and its report assumes that mocked function should also be

covered in gtest. Is there a way, we can exclude mocked functions in generating the code coverage report.

Reply

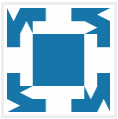
meekrosoft

October 16, 2012 at 4:10 am

This is not a Coburtura thing, but more of a gcov thing. If you really want to ignore coverage for test data you will need to filter it out manually I think.

Reply

zero



November 12, 2012 at 9:02 am

hi , i have launched hudson and create coverage project followed by your blog on ubuntu . I can build the project successfully . But no code coverage .(hudson gcovr ubuntu shell)

Reply

Jack

May 6, 2013 at 11:35 pm

I like this setup, but am having one problem. I can't find a way to show zero coverage stats for files that are NOT executed as part of the tests. These source files will have associated .gcno files, but because they are not ran no .gcda files are produced and and gcovr doesn't show them in the results.

Has anyone found a way to get around this (like perhaps creating a pre-processing step that generates zero initalized .gcda files?) I've tried using touch to create such files but they're not accepted as valid by gcov.

Reply

meekrosoft

May 9, 2013 at 1:13 pm

Ah, that is a very good point. I don't know the answer to your question, but if you find out please let me know.

Reply

Jack

May 9, 2013 at 4:20 pm

I couldn't find a way to easily initialize empty gcda files. I ended up hacking the gcovr script and adding an "–estimate-uncovered-files" option. When enabled gcovr looks at each of the files with gcno but no gcda, estimates line counts (making simple attempts to avoid blank and comment lines), and then adds the data to the internal covdata dict before printing output. It's only about a dozen lines…

The code indicates that at some point an effort was made to process stand alone gcno files, but perhaps wasn't finished?

Unfortunately my code is not generic enough to submit as a patch for gcovr (assumes certain extensions for source/headers, certain conventions for comments etc.), but it works for my purposes.