

# Análisis de coberbura usando **gcov** y **lcov**

Departamento de Computación, FCEyN, Universidad de Buenos Aires.

1er Cuatrimestre 2020

# Introducción

- ▶ Esta guía introduce dos herramientas para el análisis de cobertura de un programa.
- ▶ Las dos instrucciones son GCOV y LCOV.
- ▶ Gracias a ellas, se puede generar automáticamente un reporte gráfico de la cobertura.
- ▶ Nosotros nos basaremos en Test Suites generados para llegar a un 100 % de cobertura.
- ▶ Análisis de cobertura con CLION.

# GCOV

El GCOV ya viene incluido en el compilador **gcc** del CLION. Esta herramienta genera archivos adicionales durante la ejecución, incluyendo en el CMakeList las siguientes líneas:

```
# -g: Hace que podamos debuggear el programa.  
# --coverage: Hace que se pueda calcular el cubrimiento de los casos de test.  
set(CMAKE_CXX_FLAGS "-g --coverage")  
# Un temita de gcov que no se lleva tan bien con CMake,  
# y entonces hay que poner esta línea.  
# Para más información, ver https://texus.me/2015/09/06/cmake-and-gcov/.  
set(CMAKE_CXX_OUTPUT_EXTENSION_REPLACE 1)
```

Suponiendo que nuestro programa principal se denomina **main.cpp** durante la compilación son generados dos archivos adicionales: **main.gcno** y **main.gcda**. Se pueden encontrar en el directorio de compilación:  
`cmake-build-debug/CMakeFiles/LaboTesting.dir/`.

# LCOV

La aplicación `lcov` tiene que ser ejecutada desde línea de comando, una vez que fueron generados los `.gcno` y `.gcda`. Como ejemplo supondremos que, desde línea de comando vamos hasta el directorio donde están estos archivos.

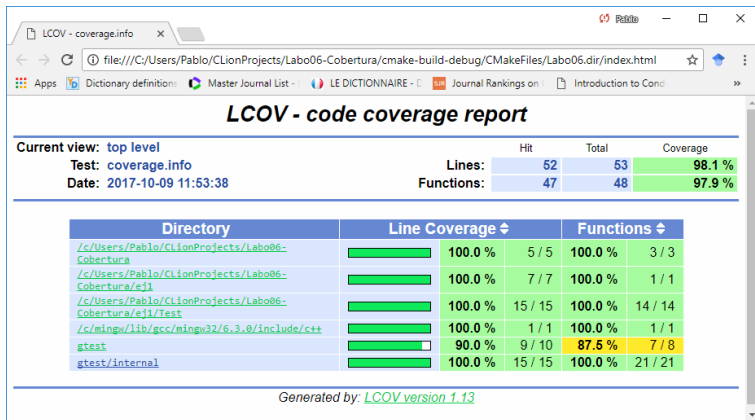
```
cmake-build-debug/CMakeFiles/LaboTesting.dir> lcov --capture  
--directory . --output-file pnegri/labo07/cobertura/coverage.info
```

Esta instrucción de **lcov** crea en el *pnegri/labo07/cobertura/* el archivo *coverage.info*.

El siguiente paso es generar el reporte en formato html con la instrucción **genhtml**. Para ello podemos ubicarnos en el directorio donde fue generado el archivo `coverage.info`, en nuestro caso *pnegri/labo07/cobertura/*:

```
pnegri/labo07/cobertura/> genhtml coverage.info --output-directory .
```

En el directorio *pnegri/labo07/cobertura/*, se genero un reporte en html de la cobertura del programa. Para verlo, simplemente se abre el archivo **index.html** generado.



# LCOV - Instalación

## Linux

- ▶ El LCOV es un proyecto de git que se encuentra en <https://github.com/linux-test-project/lcov>. Para instalarlo podemos usar la guía o correr las siguientes instrucciones:
  - ▶ Chequer si el compilador ya está instalado ejecutando en la terminal: `g++ --version`.
  - ▶ Si está instalado verán versión del compilador.
  - ▶ Si no está instalado ejecutar: `sudo apt install g++`
  - ▶ Chequer si LCOV ya está instalado ejecutando en la terminal: `lcov --version`
  - ▶ Si está instalado verán versión de lcov.
  - ▶ Si no está instalado ejecutar: `sudo apt install lcov`

# LCOV - Instalación

## Windows

- ▶ Se debe crear un entorno unix para correr el LCOV.
  - ▶ El primer paso es instalar MSYS2<sup>1</sup>: adjuntamos un tutorial en el folder el TP.
  - ▶ Otro tutorial<sup>2</sup>. Para instalar paquetes en el MSYS2 se utiliza desde su consola:  
`pacman -S PAQUETE`  
Se deben instalar al menos los siguientes paquetes: git, gcc, make, binutils.

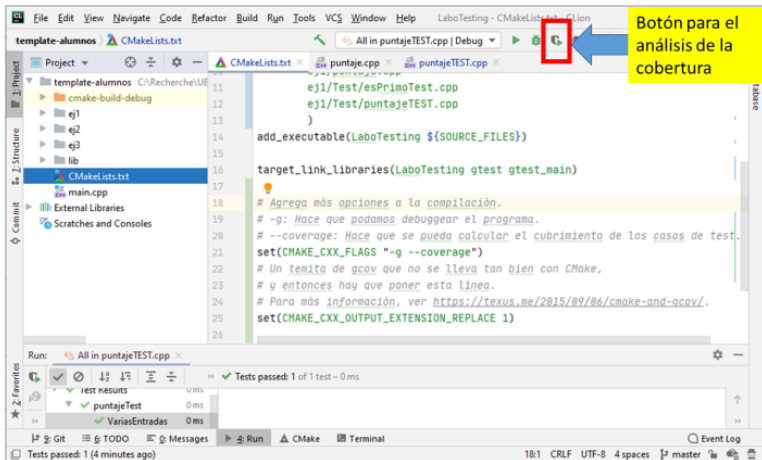
---

<sup>1</sup><https://www.msys2.org/>

<sup>2</sup><https://txt.arboreus.com/2015/05/29/>

# Análisis de cobertura con CLION

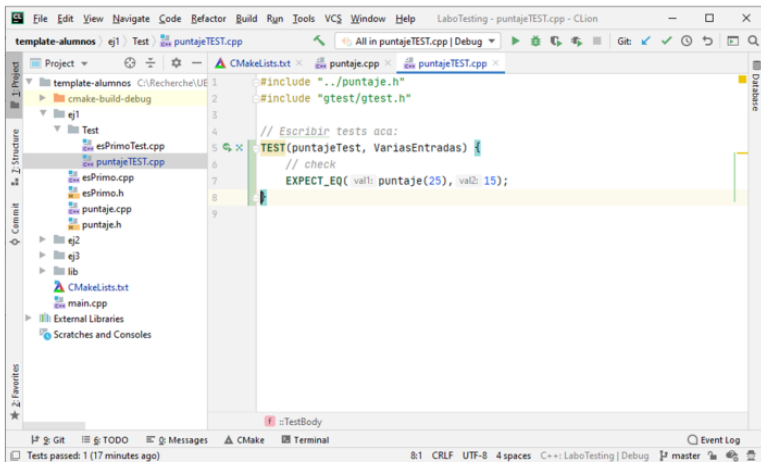
CLION, desde su versión 2019, posee un plugin que permite realizar un análisis de cobertura desde su entorno gráfico. Tomamos como ejemplo el laboratorio 07. Vemos también agregadas las líneas para generar la cobertura en el CMakeList.txt.





# Análisis de cobertura con CLION

Tenemos generado un test para que se ejecute la funcion puntaje, en el archivo puntaje.cpp.



# Análisis de cobertura con CLION

1. Se corre desde este botón y aparece una ventana de cobertura

2. La ventana de los tests sigue como siempre

3. Se muestra porcentajes de cobertura dentro de cada folder. El que nos interesa es el ej1.

Element	Statistics, %
cmake-build-debug	70% files, 44% lines covered
ej1	75% files, 39% lines covered
ej2	
ej3	
lib	60% files, % lines covered
main.cpp	100% lines covered

Run: All in puntajeTEST.cpp

Tests passed: 1 of 1 test - 0 ms

test results

puntajeTest 0 ms

VariasEntradas 0 ms

Tests passed: 1

Tests passed: 1 (moments ago)

3:1 CRLF UTF-8 4 spaces C++: LaboTesting | Debug master

# Análisis de cobertura con CLION

The screenshot displays the CLion IDE interface with the 'puntajeTEST.cpp' file open. The code is annotated with colored markers indicating test coverage status: green for lines executed and red for lines not executed. A green arrow points to line 5, and a red arrow points to line 6. The 'Coverage' window on the right shows a table of coverage statistics for the 'ej1' test run.

5. En verde por donde pasó al correr el test.

6. En rojo por donde NO pasó al correr el test.

4. Entramos dentro del folder ej1, donde vemos la cobertura de los archivos cpp. Nos interesa puntaje.cpp. No esta cubierto 100%

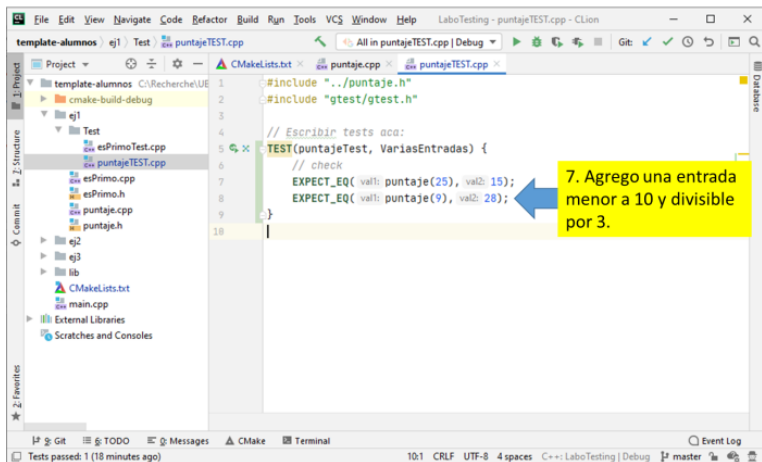
Element	Statistics, %
Test	100% files, 38% lines covered
esPrimo.cpp	0% lines covered
esPrimo.h	0% lines covered
puntaje.cpp	75% lines covered
puntaje.h	0% lines covered

Run: All in puntajeTEST.cpp  
Tests passed: 1 of 1 test - 0 ms

Test Results  
puntajeTest 0 ms  
VariasEntradas 0 ms

Tests passed: 1 (9 minutes ago)

# Análisis de cobertura con CLION



# Análisis de cobertura con CLION

9. Todas las líneas ahora están en verde, indicando que al correr los 2 tests, se cubrieron.

8. Ahora si tengo 100% de cobertura en puntaje.cpp.

The screenshot shows the CLion IDE interface with the following components:

- Editor:** Displays the source code for `puntaje.cpp`. The code is as follows:

```
1 int puntaje(int b) {  
2     int res;  
3     if (b < 10) {  
4         res = 2 * b;  
5     } else {  
6         res = b;  
7     }  
8     if (b % 3 == 0) {  
9         res = res + 10;  
10    } else {  
11        res = res - 10;  
12    }  
13    return res;  
14 }  
15
```

  
The code is annotated with green vertical bars on the left, indicating 100% line coverage. A green arrow points from the text box '9. Todas las líneas ahora están en verde...' to these green bars.
- Project Explorer:** Shows the project structure with files and their coverage status:
  - `esPrimo.cpp`: 0% lines covered
  - `esPrimo.h`: 0% lines covered
  - `puntaje.cpp`: 100% lines covered
  - `puntaje.h`: 0% lines covered
  - `lib`: 60% files, 43% lines covered
  - `main.cpp`: 100% lines covered
- Coverage Window:** Displays a table of coverage statistics for the selected file, `puntaje.cpp`.

Element	Statistics, %
Test	100% files, 42% lines cov...
esPrimo.cpp	0% lines covered
esPrimo.h	0% lines covered
puntaje.cpp	100% lines covered
puntaje.h	0% lines covered

A blue arrow points from the text box '8. Ahora si tengo 100% de cobertura en puntaje.cpp.' to the row for `puntaje.cpp` in this table.
- Run Window:** Shows the execution results of the tests.
  - Tests passed: 1 of 1 test - 0ms
  - Testing started at 8:33 PM ...
  - Test Results: `puntajeTest` (0ms)