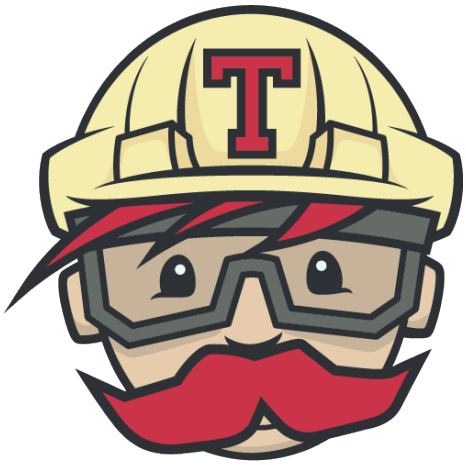
# Travis C++ tutorial

Richèl Bilderbeek

March 13, 2016



## Contents



## 1 Introduction

This is a Travis C++ tutorial, version 0.1

## 1.1 License



Figure 1: Creative Commons license 4.0

## 1.2 Continuous integration

Collaboration can be scary: the other(s)[1] may break the project worked on. The project can be of any type, not only programming, but also collaborative writing.

A good first step ensuring a pleasant experience is to use a version control system. A version control system keeps track of the changes in the project and allows for looking back in the project history when something has been broken.

The next step is to use an online version control repository, which makes the code easily accessible for all contributors. The online version control repository may also offer additional collaborative tools, like a place where to submit bug reports, define project milestones and allowing external people to submit requests, bug reports or patches.

Up until here, it is possible to submit a change that breaks the build.

A continuous integration tools checks what is submitted to the project and possibly rejects it when it does not satisfy the tests and/or requirements of the project. Instead of manually proofreading and/or testing the submission and mailing the contributor his/her addition is rejected is cumbersome at least. A continuous integration tool will do this for you.

Now, if someone changes you project, you can rest assured that his/her submission does not break the project. Enjoy!

## 1.3 Tool used

**git** git is a version control system. It tracks the changes made in the project and allows for viewing the project its history.

**GitHub** GitHub is a site where git repositories are hosted. It gives a git project a website where the files can be viewed. Next to this, there is a project page for issues like bug reports and feature requests.

**Travis CI** Travis CI is a continuous integration (hence the 'CI' in its name) tool that plays well with GitHub. It is activated when someone uploads his/her code to the GitHub.

[1] if not you

**gcov** gcov is a GNU tool to measur the code coverage of (among others) C++ code. It can be actived from a Travis script.

**Codecov** Codecov is a tool to display a gcov code coverage result, that plays well with GitHub. It can be actived from a Travis script.

**gprof** gprof is a GNU tool to profile (among others) C++ code. It can be actived from a Travis script.

### 1.4 Feedback

This tutorial is not intended to be perfect yet. For that, I need help and feedback from the community. All referenced feedback is welcome, as well as any constructive feedback.

## 2 The basic build

The basic build has the following specs:

- Build system: qmake
- C++ compiler: gcc
- C++ version: C++98
- Libraries: STL only
- Code coverage: none
- Source: one single file, main.cpp

First I will show the single file this build is about:

**Algorithm 1** main.cpp

```
#include <iostream>

int main() {
  std::cout << "Hello world\n";
}
```

All the code does is display the text 'Hello world', which is a traditional start for many programming languages. The code is written in C++98. It does not use features from the newer C++ standards. It will not compile under plain C.

This single file is compiled with qmake from the following Qt Creator project file:

**Algorithm 2** travis_qmake_gcc_cpp98.pro

```
TEMPLATE = app
CONFIG += console
CONFIG -= app_bundle qt
SOURCES += main.cpp
QMAKE_CXXFLAGS += -Wall -Wextra -Weffc++ -Werror
```

This Qt Creator project file has a typical setup for a standard console application, except that, in the last line, the warning level is set the the highest level, even making a warning break the build. This forces collaborators to write tidy code.

The bash build script to build this is:

**Algorithm 3** build.sh

```
qmake
make
./travis_qmake_gcc_cpp98
```

This build script calls 'qmake' to create a makefile. Then 'make' is called to compile the makefile. Finally, the created executable 'travis_qmake_gcc_cpp98' is run. There is a potential error in the first and last step: the Qt Creator project file may be incorrect, or the executable will crash, possibly due to a failed test.

Setting up Travis is done by the following .travis.yml[2] file:

**Algorithm 4** .travis.yml

```
language: cpp
compiler: gcc
script: ./build.sh
```

This .travis.yml file has the following elements:

- `language: cpp`

  The main programming language of this project is C++

- `compiler: gcc`

  The C++ code will be compiled by GCC

- `script: ./travis_qmake_gcc_cpp98`

  The script that Travis will run, which is running the generated executable called 'travis_qmake_gcc_cpp98'.

[2] the filename starts with a dot. This means it is a hidden file

# 3 Adding code coverage

In this example, the basic build (chapter 2) is extended by calling gcov and using codecov to show the code coverage.

The basic build has the following specs:

- Build system: qmake
- C++ compiler: gcc
- C++ version: C++98
- Libraries: STL only
- Code coverage: none
- Source: one single file, main.cpp

The single C++ source file used is:

**Algorithm 5** main.cpp

```
#include <iostream>

int main(int argc, char* argv[])
{
  if (argc >= 1) {
    std::cout << argv[0] << '\n';
  }
  else {
    std::cout << "I will never be called\n";
  }
}
```

This file openly contains some dead code, so we expect to observe a code coverage less than 100%.

This single file is compiled with qmake from the following Qt Creator project file:

**Algorithm 6** travis_qmake_gcc_cpp98.pro

```
TEMPLATE = app
CONFIG += console
CONFIG -= app_bundle qt
SOURCES += main.cpp
QMAKE_CXXFLAGS += -Wall -Wextra -Weffc++ -Werror

# gcov
QMAKE_CXXFLAGS += -fprofile-arcs -ftest-coverage
LIBS += -lgcov
```

The Qt Creator project file has two new lines. The first of those adds two compiler flags, which cause the code to be compiled in such a way to gcov can work with it. The second line links the gcov library to the project.

The bash build script to build this, run this and measure the code coverage:

**Algorithm 7** build.sh

```
qmake
make
./travis_qmake_gcc_cpp98_gcov
gcov main.cpp
cat main.cpp.gcov
```

The new step is after having run the executable, where gcov is run on the only source file. The text 'gcov' has generated is then shown using 'cat'.

Setting up Travis is done by the following .travis.yml:

**Algorithm 8** .travis.yml

```
sudo: true
language: cpp
compiler: gcc
before_install: sudo pip install codecov
script: ./build.sh
after_success: codecov
```

This .travis.yml file has some new features:

- sudo: true

  Travis will give super user rights to the script. This will slow the build time, but it is inevitable for the next step

- `before_install: sudo pip install codecov`

  Travis will use pip to install codecov using super user rights

- `after_success: codecov`

  After the script has run successfully, codecov is called