Professional C++ development

using git, GitHub, Travis CI, Boost.Test, gcov and OCLint

http://www.github.com/richelbilderbeek/CppPresentations

What? Why? Mastery?

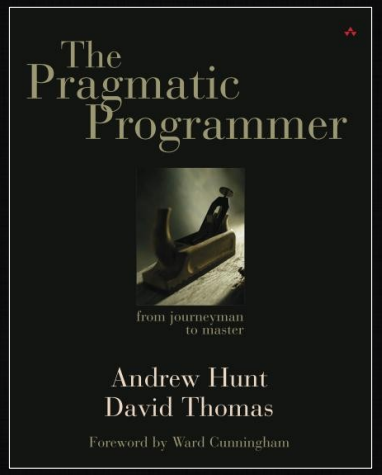
- What: follow all good practices by default
- Why: proven to pay off
- Mastery: set up tools to follow all good practices by default

Setup

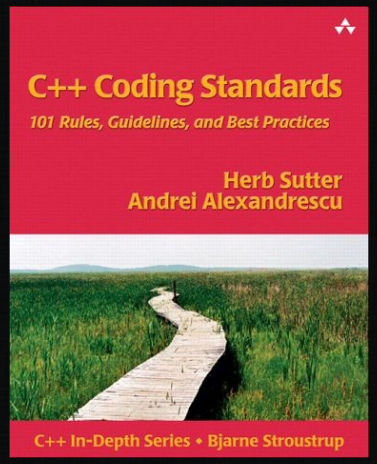
- Version control
- Code hosting
- Continuous integration
- Testing framework
- Code coverage
- Static code analysis

Version control

- Keeps a history of code



Tip 23: Always Use Source Code Control

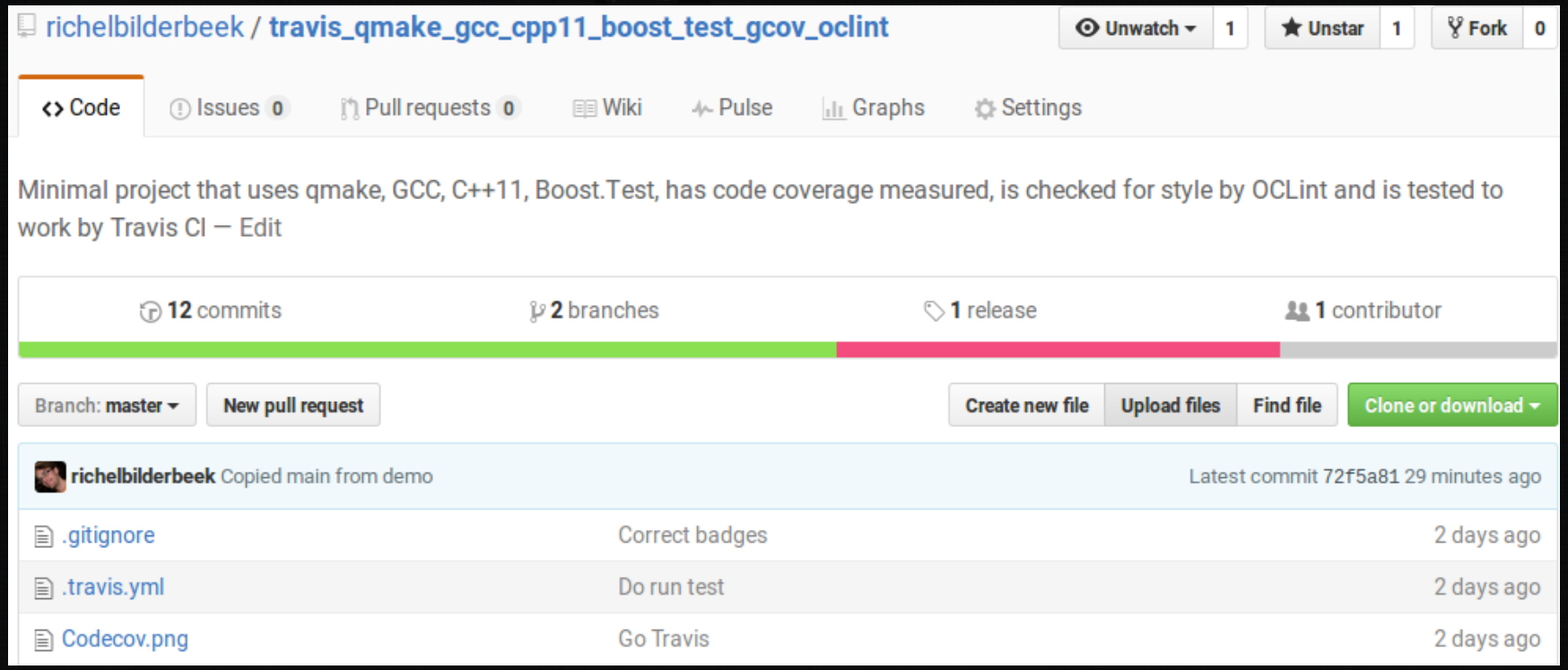


Chapter 3.
Use a version control system

git

Code hosting

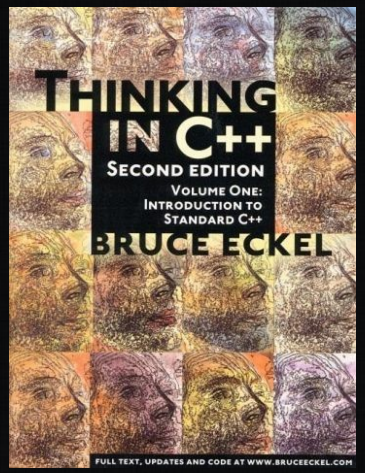
- Host your (version controlled) code
- Using GitHub is good practice [1]



[1] Perez-Riverol, Yasset, et al. "Ten Simple Rules for Taking Advantage of git and GitHub." bioRxiv (2016): 048744.

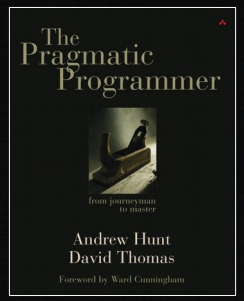
Continuous integration

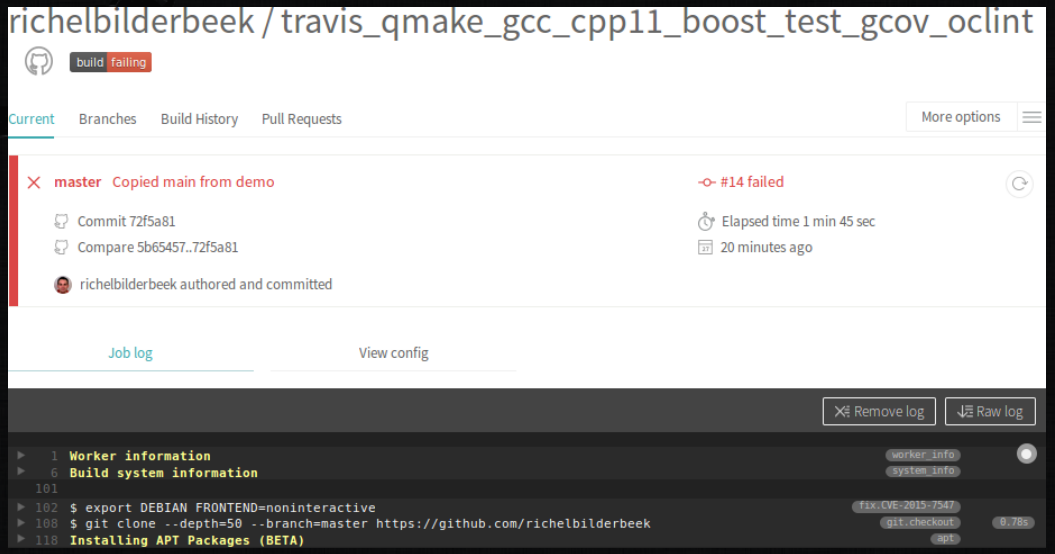
- Run scripts upon pushing new code



Automate the running of your tests through a makefile or similar tool

Tip 61: Don't Use Manual Procedures

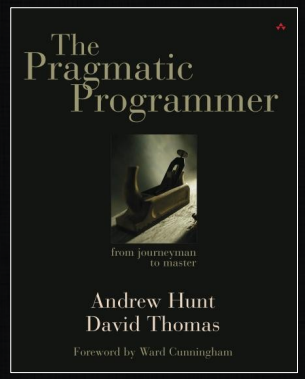




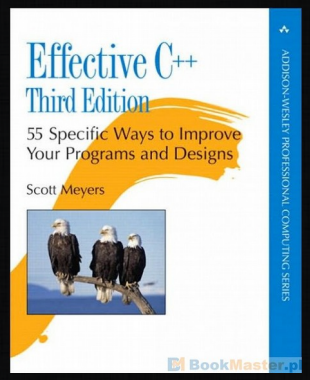
Travis CI

Testing framework

- To test your code



Tip 62: Test Early.
Test Often. Test Automatically

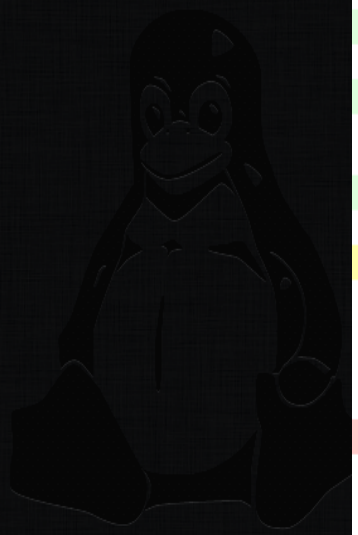


Item 55:
Familiarize yourself with Boost

```
Starting /home/richel/GitHubs/build-travis_qmake_gcc_cpp11_boost_test_gcov_oclint_test-Desktop-Debug/
travis_qmake_gcc_cpp11_boost_test_gcov_oclint_test...
Running 2 test cases...

*** No errors detected
```

Code coverage

- Measures code that is actually used
- Correlates with quality [1]

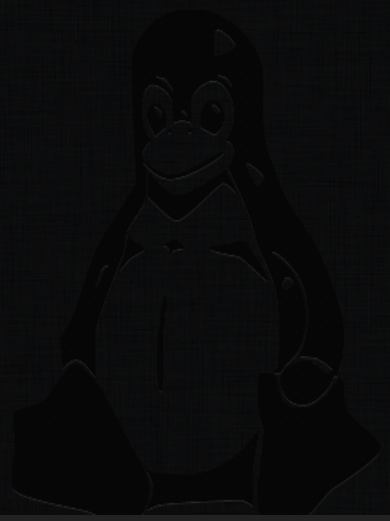


```
#include "my_functions.h"

#include <numeric>
#include <stdexcept>

bool is_odd(const int i) noexcept
{
  return i % 2 == 1;
}

double calc_mean(const std::vector<double>& v)
{
  if (v.empty())
  {
    throw std::invalid_argument(
      "cannot calculate the mean"
      "of an empty vector"
    );
  }
  const double sum{
    std::accumulate(
      std::begin(v),
      std::end(v),
      0.0
    )
  };
  return sum / static_cast<double>(v.size());
}
```

[1] Del Frate, Fabio, et al. "On the correlation between code coverage and software reliability." Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on. IEEE, 1995.

Static code analysis

- Checks code beyond the compiler



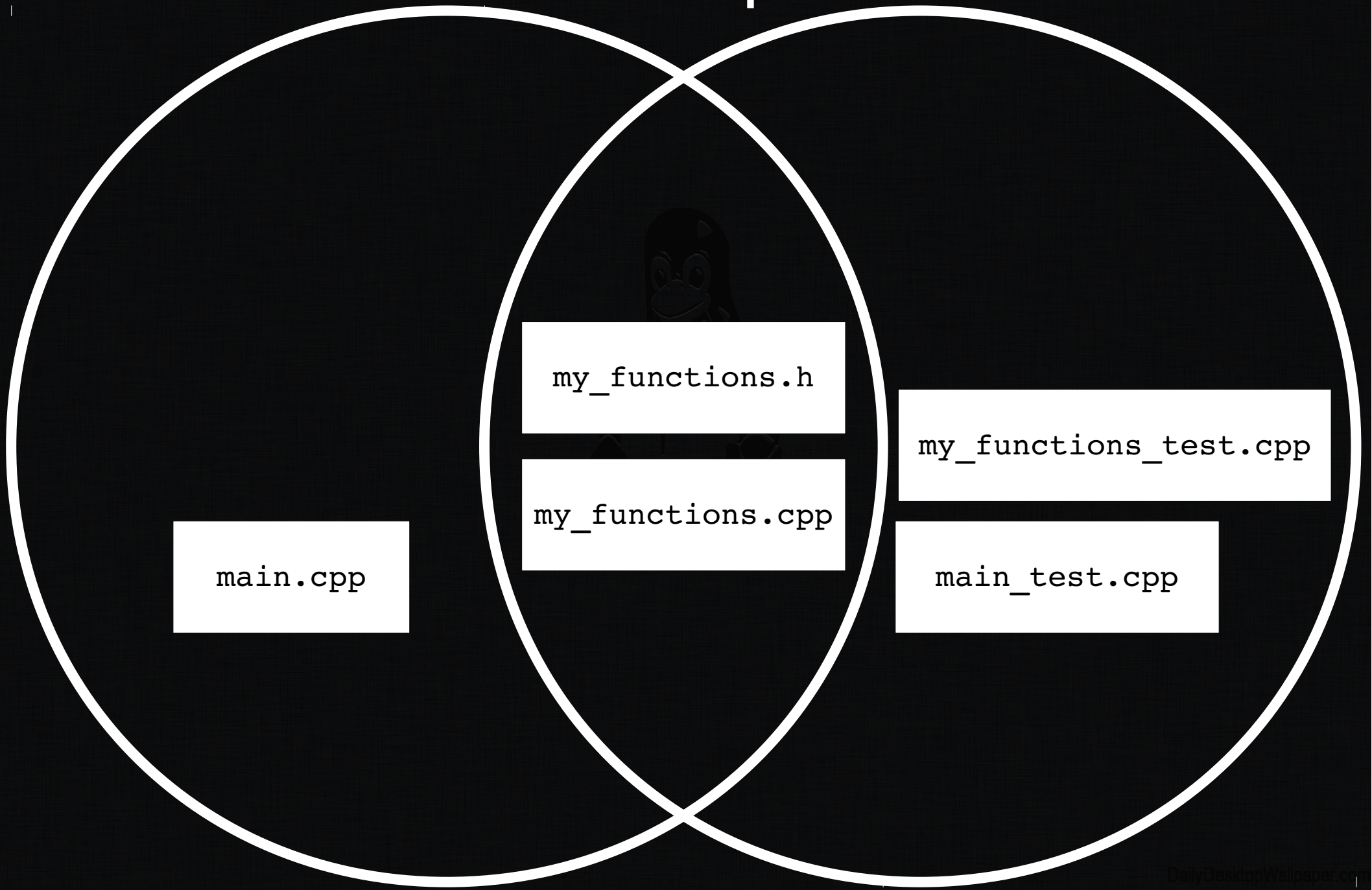
```
OCLint Report

Summary: TotalFiles=3 FilesWithViolations=1 P1=0 P2=1 P3=0

/home/travis/build/richelbilderbeek/travis_qmake_gcc_cpp11_boost_test_gcov_oclint/my_functions.cpp:8:10: broken oddness check [basic|P2]

[OCLint (http://oclint.org) v0.10.3]
OCLint: OK
OCLint: Fail
```

OCLint

This presentation

- Shows two functions and their tests
- Shows the detection of errors in those functions

Setup



main.cpp

```
#include "my_functions.h"
#include <iostream>

int main() {
  std::cout
    << is_odd(42) << '\n'
    << calc_mean( { 41.0, 42.0,43.0 } )
    << '\n';
}
```

my_functions.h

```
#ifndef MY_FUNCTIONS_H
#define MY_FUNCTIONS_H

#include <vector>

///Calculate the mean.
///Will throw if the input is empty
double calc_mean(const std::vector<double>& v);

///Determine if a number is odd
bool is_odd(const int i) noexcept;

#endif // MY_FUNCTIONS_H
```

my_functions.cpp 1/2

```
#include "my_functions.h"

#include <numeric>
#include <stdexcept>

bool is_odd(const int i) noexcept
{
  return i % 2 == 1;
}
```

my_functions.cpp 2/2

```
double calc_mean(const std::vector<double>& v) {
  if (v.empty()) {
    throw std::invalid_argument(
      "cannot calculate the mean"
      "of an empty vector"
    );
  }
  const double sum{
    std::accumulate(
      std::begin(v), std::end(v), 0.0
    )
  };
  return sum / static_cast<double>(v.size());
}
```

main_test.cpp

```
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE my_functions_test_module
#include <boost/test/unit_test.hpp>

//No main needed, BOOST_TEST_DYN_LINK creates it
```

my_functions_test.cpp 1/2

```
#include <boost/test/unit_test.hpp>

#include "my_functions.h"

BOOST_AUTO_TEST_CASE(test_is_odd)
{
  BOOST_CHECK(!is_odd(0));
  BOOST_CHECK( is_odd(1));
}
```

my_functions_test.cpp 1/2

```
BOOST_AUTO_TEST_CASE(test_calc_mean)
{
  const double measured{
    calc_mean( {1.0, 2.0, 3.0} )
  };
  const double expected{2.0};
  BOOST_CHECK_EQUAL(measured, expected);
}
```

Output

```
Running 2 test cases...


*** No errors detected
```

Detection

build	failing
codecov	85%

Detection

```
BOOST_AUTO_TEST_CASE(
  test_calc_mean_needs_nonempty_vector
)
{
  std::vector<double> empty;
  BOOST_CHECK_THROW(
    calc_mean(empty),
    std::invalid_argument
  );
}
```

```
#include "my_functions.h"

#include <numeric>
#include <stdexcept>

bool is_odd(const int i) noexcept
{
  return i % 2 == 1;
}

double calc_mean(const std::vector<double>& v)
{
  if (v.empty())
  {
    throw std::invalid_argument(
      "cannot calculate the mean"
      "of an empty vector"
    );
  }
  const double sum{
    std::accumulate(
      std::begin(v),
      std::end(v),
      0.0
    )
  };
  return sum / static_cast<double>(v.size());
}
```

Detection

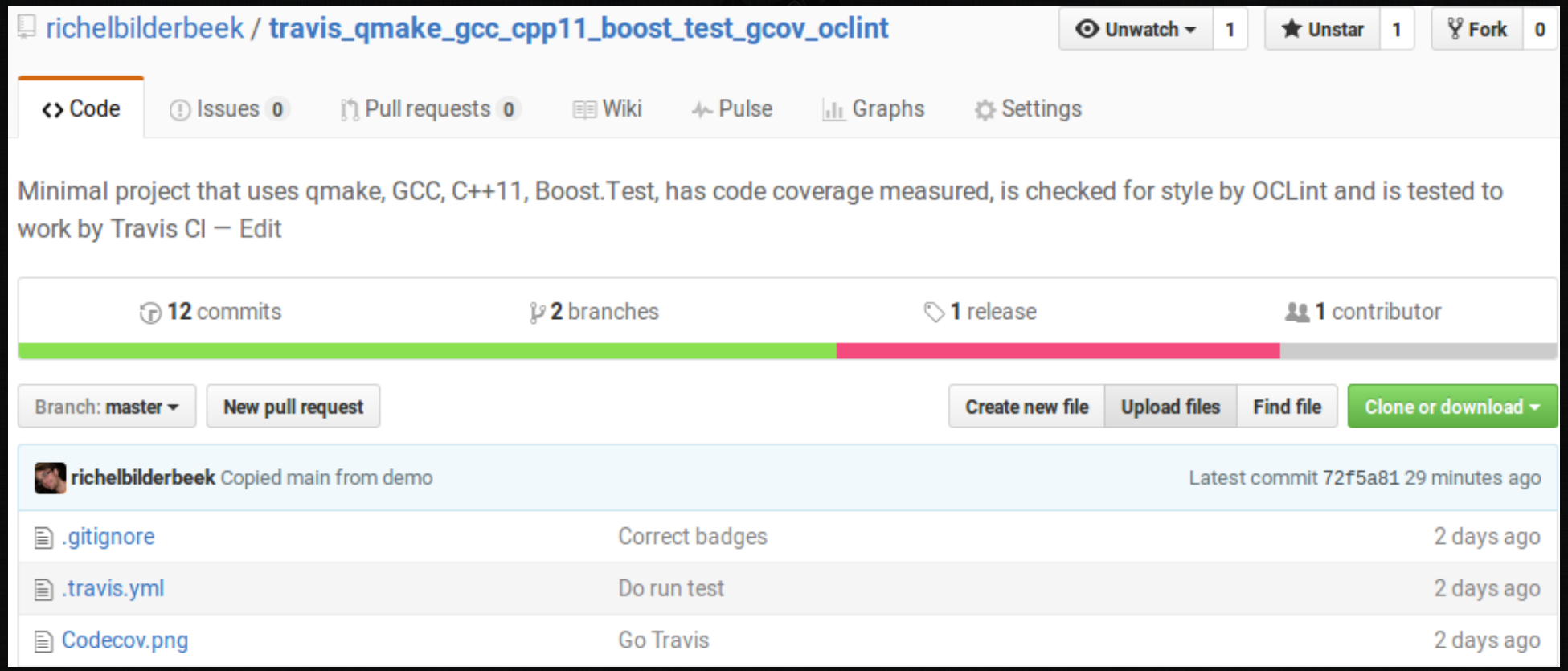
```
bool is_odd(const int i) noexcept
{
  return i % 2 == 1;
}
```

```
OCLint Report

Summary: TotalFiles=3 FilesWithViolations=1 P1=0 P2=1 P3=0

/home/travis/build/richelbilderbeek/travis_qmake_gcc_cpp11_boost_test_gcov_oclint/my_functions.cpp:8:10: broken oddness check [basic|P2]

[OCLint (http://oclint.org) v0.10.3]
OCLint: OK
OCLint: Fail
```

```
bool is_odd(const int i) noexcept
{
  return i % 2 != 0;
}
```

Replicating this setup

- Download (not clone) the files from https://github.com/richelbilderbeek/travis_qmake_gcc_cpp11_boost_test_gcov_oclint



My experiences

- Due to this setup, I have
  - Responded to bugs faster build passing
  - Found and removed dead code codecov 100%
  - Reduced complexity of working code

```
The command "./do_oclint.sh" exited with 1.
```

  - Learned new things

Conclusion

- This setup was known to be useful beforehand
- This setup is easy to replicate
- I think this setup is useful for
  - C++ newbies
  - C++ pros

Discussion

- Creating this setup from scratch is tricky
- Other tools are just as fine

- Need a related C++ setup?
  https://github.com/richelbilderbeek/travis_cpp_tutorial
- Need a related R setup?
  https://github.com/richelbilderbeek/travis_r_tutorial