

C++ Korea 6th Seminar

C++ 오픈 소스 101

(나도 이제 오픈 소스 프로젝트 할 수 있다!)

Nexon Korea, Microsoft MVP

옥찬호

소개

- 옥찬호 (Chris Ohk)
 - Nexon Korea Game Programmer
 - Microsoft Development Technologies MVP
 - 페이스북 그룹 C++ Korea 대표
 - IT 전문서 집필 및 번역 다수
 - 게임샐러드로 코드 한 줄 없이 게임 만들기 (2013)
 - 유니티 Shader와 Effect 제작 (2014)
 - 2D 게임 프로그래밍 (2014)
 - 러스트 핵심 노트 (2017)
 - 모던 C++ 입문 (2017), C++ 최적화 (2019)



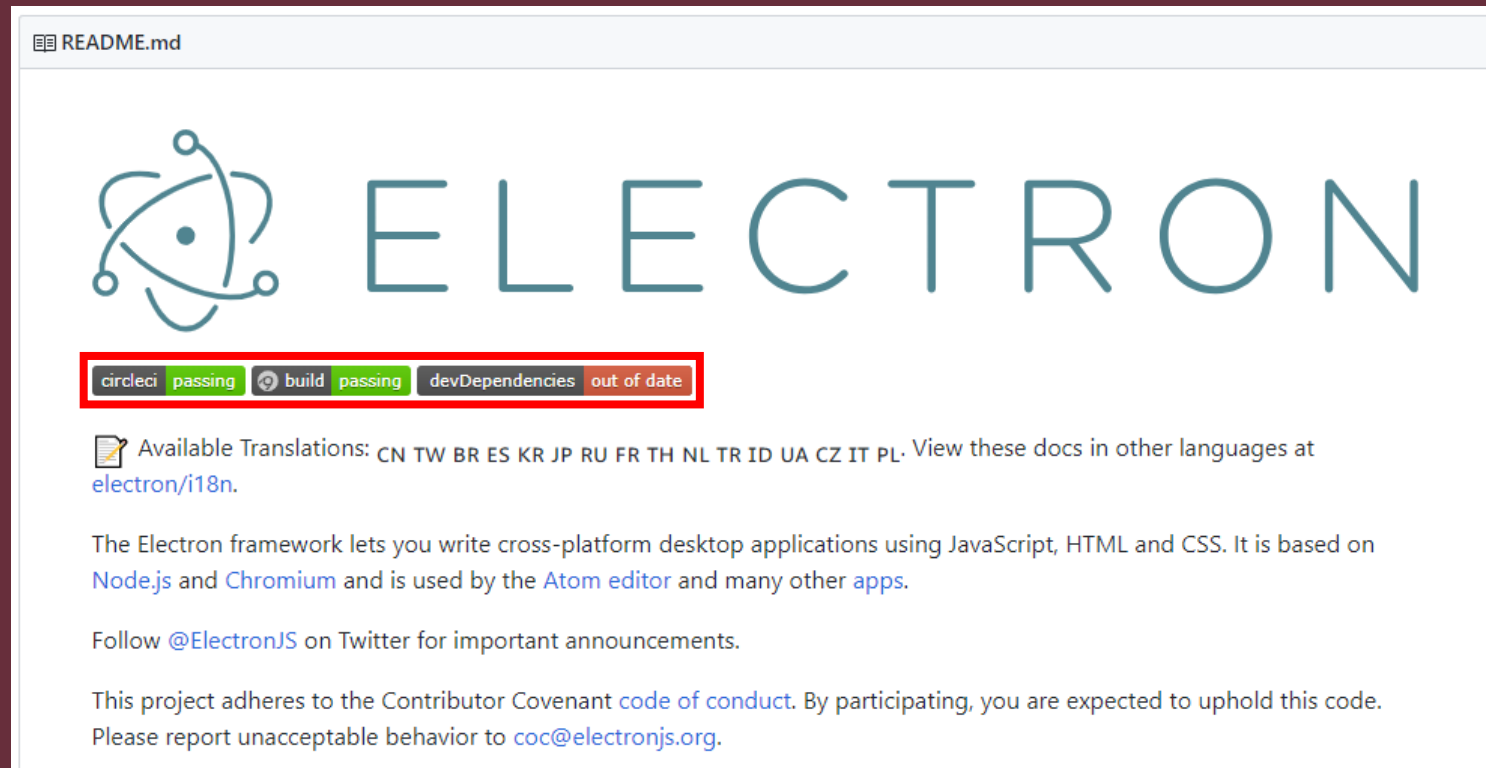
- 들어가며
- C++ 오픈 소스 프로젝트 시작해 보기
 - CMake를 통해 프로젝트 빌드하기
 - clang-format을 통해 코딩 스타일 설정하기
 - 유닛 테스트 프레임워크 설치하기
 - CI를 통해 크로스 플랫폼 빌드 자동화하기
 - 코드 품질 향상을 위한 분석 툴 설치하기
 - 테스트 커버리지 측정을 위한 툴 설치하기
- 마치며

- 세상에는 다양한 오픈 소스 프로젝트들이 있습니다.
그 중에는 C++ 언어를 사용하는 프로젝트도 많습니다.
 - Tensorflow
 - PyTorch
 - OpenCV
 - PhysX
 - Unreal Engine
 - LLVM
 - Electron

- 대규모 오픈 소스 프로젝트들은 유지 보수 관리가 매우 중요합니다.
이 때 이런 프로젝트에서 공통으로 사용하는 것들이 있습니다.



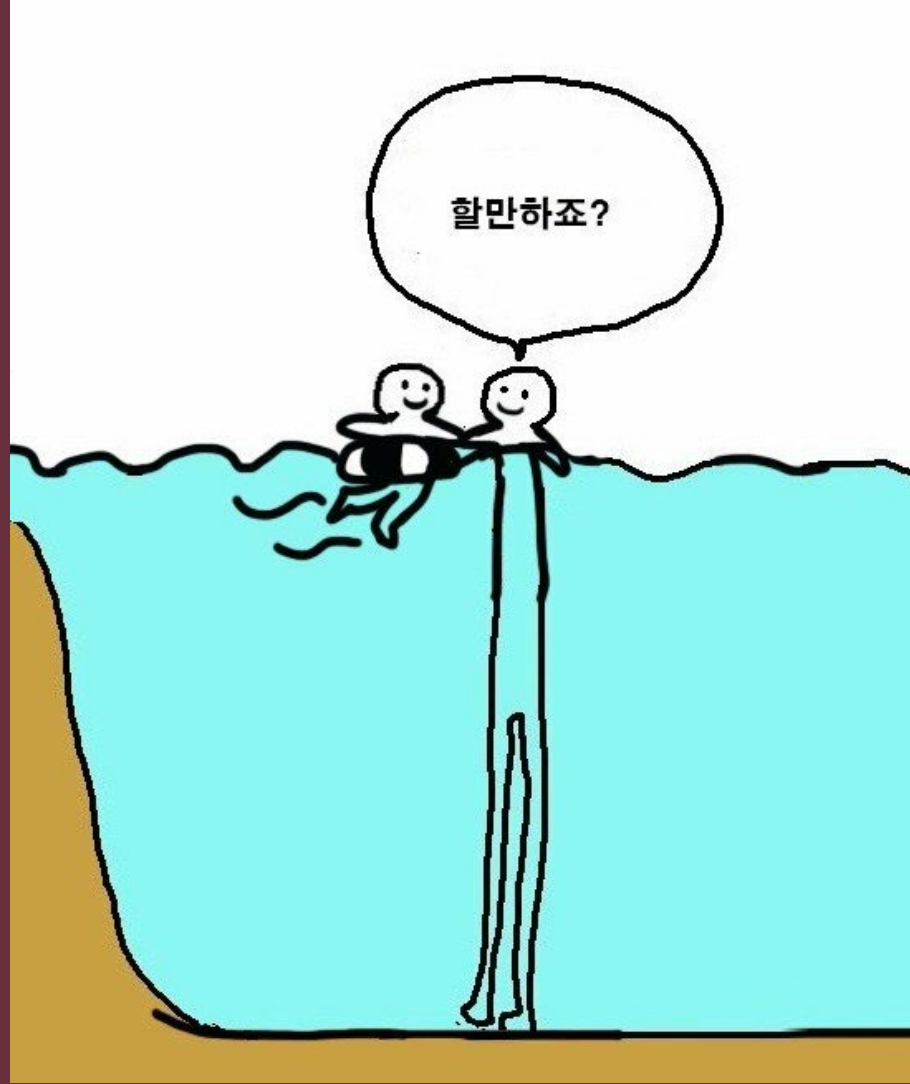
- 대규모 오픈 소스 프로젝트들은 유지 보수 관리가 매우 중요합니다.
이 때 이런 프로젝트에서 공통으로 사용하는 것들이 있습니다.



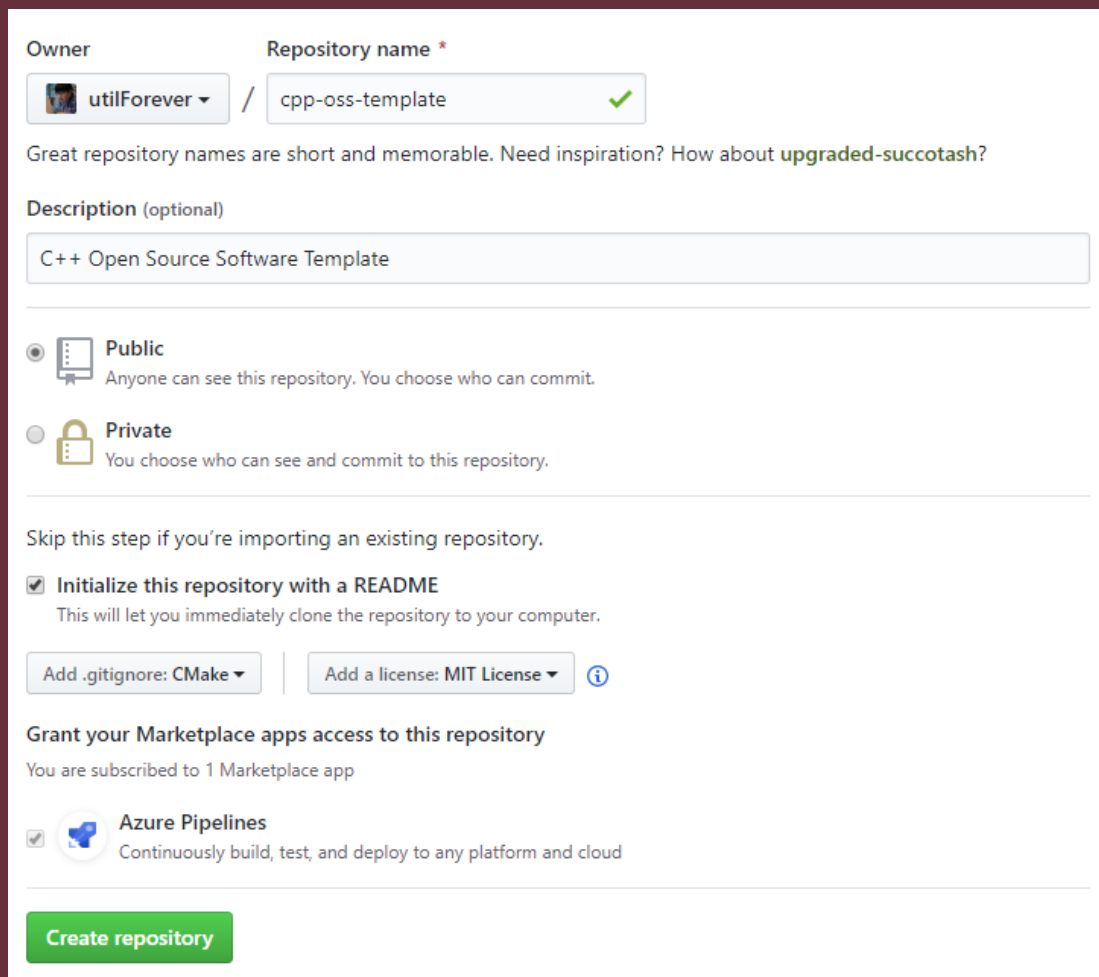
- 이런 배지들은 프로젝트가 잘 관리되고 있음을 의미합니다.
 - 우리 프로젝트는 윈도우/리눅스/맥에서 잘 빌드됩니다.
 - 우리 프로젝트는 코드 품질을 잘 관리하고 있습니다.
 - 우리 프로젝트는 코드의 동작을 잘 테스트하고 있습니다.

- 우리는 다양한 이유로 오픈 소스 또는 사이드 프로젝트를 합니다.
 - 포트폴리오 제작
 - 개인적인 취미
- 기왕이면 우리의 프로젝트도 잘 관리하면 좋지 않을까요?

하지만 막상 시작하려고 하면...



- 먼저 Github에 프로젝트부터 만들어 봅시다.



The screenshot shows the GitHub repository creation interface. At the top, the 'Owner' is set to 'utilForever' and the 'Repository name' is 'cpp-oss-template', which is marked as valid with a green checkmark. Below this, a message suggests great repository names are short and memorable, with a link to 'upgraded-succotash?'. The 'Description (optional)' field contains 'C++ Open Source Software Template'. Under the 'Visibility' section, 'Public' is selected, indicating that anyone can see the repository. The 'Private' option is also visible. A note states to skip this step if importing an existing repository. The 'Initialize this repository with a README' checkbox is checked, with a sub-note that it will clone the repository to the computer. Below this, there are two dropdown menus: 'Add .gitignore: CMake' and 'Add a license: MIT License'. The 'Grant your Marketplace apps access to this repository' section shows that the user is subscribed to 1 Marketplace app, 'Azure Pipelines', which is checked. At the bottom, there is a green 'Create repository' button.

Owner: utilForever / Repository name: cpp-oss-template ✓

Great repository names are short and memorable. Need inspiration? How about [upgraded-succotash?](#)

Description (optional): C++ Open Source Software Template

☒ Public
Anyone can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☒ Initialize this repository with a README
This will let you immediately clone the repository to your computer.

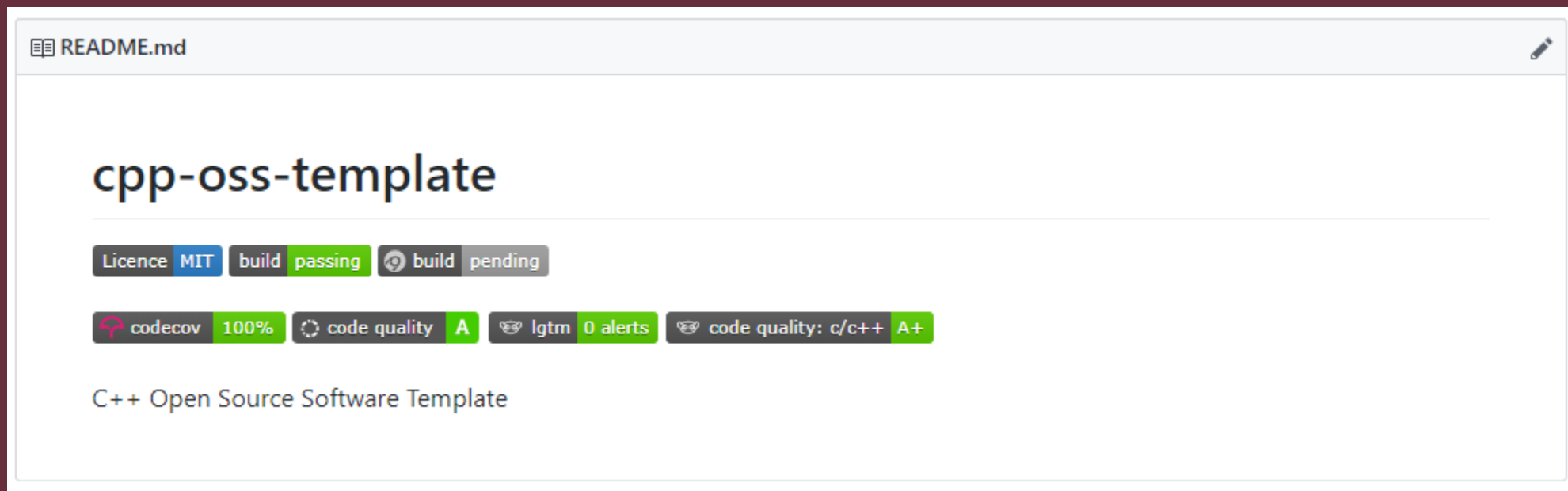
Add .gitignore: CMake | Add a license: MIT License ⓘ

Grant your Marketplace apps access to this repository
You are subscribed to 1 Marketplace app

☒ Azure Pipelines
Continuously build, test, and deploy to any platform and cloud

Create repository

- 우리의 목표



- 우리는 간단한 코드를 통해 정적 라이브러리(.lib)를 생성하는 프로젝트를 만들어보려고 합니다.
- 여러 선택지가 있겠지만 여기서는 CMake를 사용합니다.

- 프로젝트 최상위 경로에 `CMakeLists.txt` 파일을 만듭니다.
(CMake에 대한 자세한 설명은 박동하님의 ‘CMake를 할 때 쯤오오금 도움이 되는 문서 (<https://gist.github.com/luncliff/6e2d4eb7ca29a0afd5b592f72b80cb5c>)’를 참고하시기 바랍니다.)

```
# CMake version
cmake_minimum_required(VERSION 3.8.2 FATAL_ERROR)

# Include cmake modules
list(APPEND CMAKE_MODULE_PATH "${CMAKE_CURRENT_SOURCE_DIR}/CMake")

# Declare project
project(cpp-oss-template)

# Set output directories
set(DEFAULT_CMAKE_LIBRARY_OUTPUT_DIRECTORY ${CMAKE_LIBRARY_OUTPUT_DIRECTORY})
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_BINARY_DIR}/bin)
set(CMAKE_LIBRARY_OUTPUT_DIRECTORY ${PROJECT_BINARY_DIR}/lib)
set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY ${PROJECT_BINARY_DIR}/lib)

# Set enable output of compile commands during generation
set(CMAKE_EXPORT_COMPILE_COMMANDS ON)

# Includes
include_directories(Includes)
```

```
# Compile options
include(CMake/CompileOptions.cmake)

# Build type - Release by default
if(NOT CMAKE_BUILD_TYPE)
    set(CMAKE_BUILD_TYPE Release)
endif()

# Overrides
set(CMAKE_MACOSX_RPATH ON)

# Project modules
add_subdirectory(Sources/cpp-oss-template)
```


- 헤더 파일과 소스 파일을 다음 경로 위치에 만듭니다.
 - 헤더 파일 : Includes/cpp-oss-template/Test.hpp
 - 소스 파일 : Sources/cpp-oss-template/Test.cpp

Test.hpp

```
#ifndef CPP_OSS_TEMPLATE_TEST_HPP
#define CPP_OSS_TEMPLATE_TEST_HPP

int Add(int a, int b);

#endif
```

Test.cpp

```
#include <Test.hpp>

int Add(int a, int b)
{
    return a + b;
}
```

- 라이브러리 파일을 만들기 위한 CMakeLists.txt 파일을 만듭니다.
(Sources/cpp-oss-template 폴더에 생성합니다.)

```
# Target name
set(target cpp-oss-template)

# Define
set(root_dir ${CMAKE_CURRENT_SOURCE_DIR}/../..)

# Includes
include_directories(
    ${CMAKE_CURRENT_SOURCE_DIR}
    ${CMAKE_CURRENT_SOURCE_DIR}/../../Libraries)

# Sources
file(GLOB header_dir
    ${root_dir}/Includes)
file(GLOB_RECURSE headers
    ${header_dir}/*.hpp)
file(GLOB_RECURSE sources
    ${CMAKE_CURRENT_SOURCE_DIR}/*.cpp)

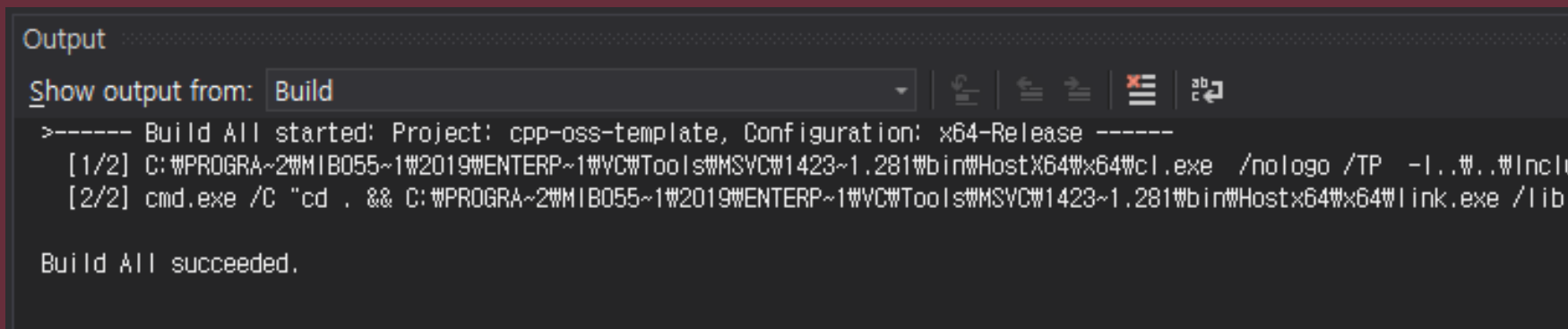
# Build library
add_library(${target} ${sources})
```

```
# Project options
set_target_properties(${target}
    PROPERTIES
    ${DEFAULT_PROJECT_OPTIONS})

# Compile options
target_compile_options(${target}
    PRIVATE
    PUBLIC
    ${DEFAULT_COMPILE_OPTIONS}
    INTERFACE)
target_link_libraries(${target}
    PRIVATE
    PUBLIC
    ${DEFAULT_LINKER_OPTIONS}
    ${DEFAULT_LIBRARIES}
    INTERFACE)

# Install
install(TARGETS ${target} DESTINATION lib)
install(DIRECTORY ${header_dir} DESTINATION include)
```

- 빌드가 잘 되는지 확인합니다.



The screenshot shows a dark-themed IDE window titled "Output". Below the title bar, there is a dropdown menu labeled "Show output from:" with "Build" selected. To the right of the dropdown are several icons: a magnifying glass, a left arrow, a right arrow, a list icon, and a refresh icon. The output text is as follows:

```
>----- Build All started: Project: cpp-oss-template, Configuration: x64-Release -----  
[1/2] C:\#PROGRA~2\MIB055~1\2019\ENTERP~1\VC\Tools\MSVC\1423~1.281\bin\HostX64\x64\cl.exe /nologo /TP -I..#..#Inclu  
[2/2] cmd.exe /C "cd . && C:\#PROGRA~2\MIB055~1\2019\ENTERP~1\VC\Tools\MSVC\1423~1.281\bin\Hostx64\x64\link.exe /lib  
  
Build All succeeded.
```

- 프로그래머들의 영원한 난제

```
if (showHelp) {  
    std::cout << ToString(parser) << '\n';  
    exit(EXIT_SUCCESS);  
}
```

VS

```
if (showHelp)  
{  
    std::cout << ToString(parser) << '\n';  
    exit(EXIT_SUCCESS);  
}
```

- C++에서 코드 스타일에 정답이란 없습니다.
- 사람마다 자기 자신의 코드 스타일을 사용합니다.
 - 들여쓰기를 할 때 탭을 쓸 것이냐 공백 문자를 쓸 것이냐
 - 중괄호는 같은 줄에서 열 것이냐 다음 줄에서 열 것이냐
 - if 문 뒤에 공백 문자를 넣을 것이냐 말 것이냐
- 코드 스타일을 정하는 이유
 - 프로그래머마다 선호하는 코드 스타일이 다릅니다.
 - 따라서 통일된 코드 스타일을 사용하지 않으면 가독성이 떨어집니다.

- 큰 회사들은 나름대로의 코드 스타일 가이드 문서를 정립합니다.
 - Google C++ Style Guide
 - C++ Core Guidelines
 - CMU C++ Coding Standard
 - ROS C++ Style Guide
 - LLVM Coding Standards
- 위 코드 스타일 문서를 참고해서 팀에 적용해도 됩니다.
하지만 우리만의 코드 스타일을 만들고 싶다면 어떻게 해야 할까요?

- C++에서는 clang-format을 사용해 코드 스타일을 지정할 수 있습니다.
(자세한 내용은 <https://clang.llvm.org/docs/ClangFormatStyleOptions.html> 참고)

```
---
Language:      Cpp
# BasedOnStyle: Google
AccessModifierOffset: -3
AlignAfterOpenBracket: Align
AlignConsecutiveAssignments: false
AlignConsecutiveDeclarations: false
AlignEscapedNewlinesLeft: true
AlignOperands:  true
AlignTrailingComments: true
AllowAllParametersOfDeclarationOnNextLine: true
...
```

- C++에서는 clang-format을 사용해 코드 스타일을 지정할 수 있습니다.
(자세한 내용은 <https://clang.llvm.org/docs/ClangFormatStyleOptions.html> 참고)

AfterControlStatement: false

```
if (showHelp) {  
    std::cout << ToString(parser) << '\n';  
    exit(EXIT_SUCCESS);  
}
```


AfterControlStatement: true

```
if (showHelp)  
{  
    std::cout << ToString(parser) << '\n';  
    exit(EXIT_SUCCESS);  
}
```

- 새로 작성하거나 수정한 코드가 성공적으로 빌드되었나요?
빌드만 된다고 해서 끝난 게 아닙니다. 반드시 테스트를 해야 합니다.
- 테스트를 하는 이유
 - 내가 원했던 동작과 실제 동작이 일치하는 지 확인할 수 있습니다.
 - 나중에 코드가 수정되었을 때 나머지 동작에 영향이 없는지를 확인할 수 있습니다.
 - 비정상적인 데이터가 입력되었을 때 어떻게 처리할 지 고려할 수 있습니다.

- C++에서 많이 사용하는 단위 테스트 프레임워크는 다음과 같습니다.
 - Boost.Test
 - googletest
 - Catch2
 - doctest
- 여기서는 doctest를 사용해보도록 하겠습니다.

- doctest : <https://github.com/onqtam/doctest>



C++ testing framework


master branch	Linux/OSX	build passing	Windows	build passing	coverage 96%	coverity passed
dev branch	Linux/OSX	build passing	Windows	build passing	coverage 96%	

doctest is a new C++ testing framework but is by far the fastest both in compile times (by [orders of magnitude](#)) and runtime compared to other feature-rich alternatives. It brings the ability of compiled languages such as [D](#) / [Rust](#) / [Nim](#) to have tests written directly in the production code thanks to a fast, transparent and flexible test runner with a clean interface.

c++ 11/14/17/20 license MIT version 2.3.5 download link cii best practices passing code quality: c/c++ A+ chat on gitter try it online

The framework is and will stay free but needs your support to sustain its development. There are lots of [new features](#) and maintenance to do. If you work for a company using doctest or have the means to do so, please consider financial support. Monthly donations via Patreon and one-offs via PayPal.

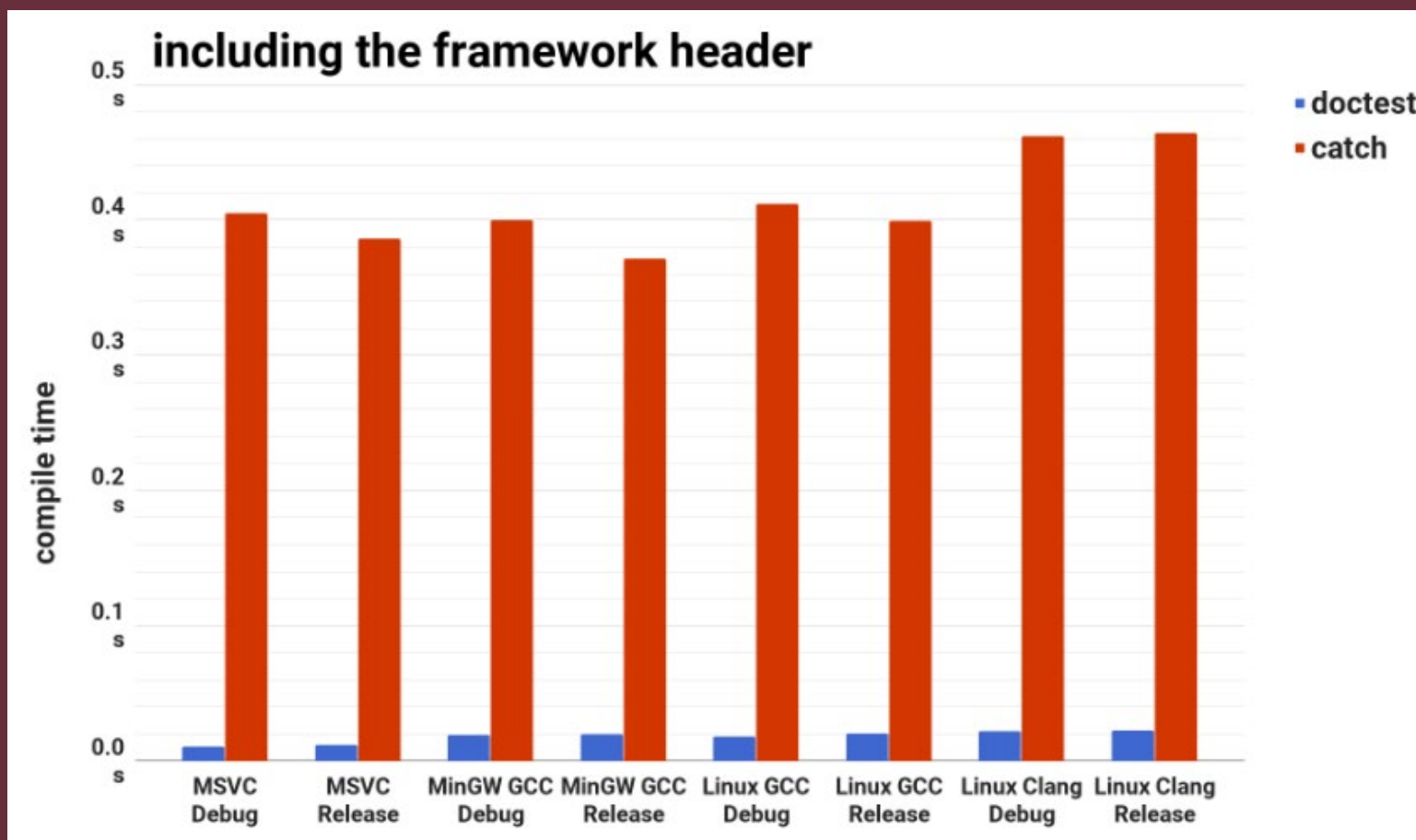
Become my patron on



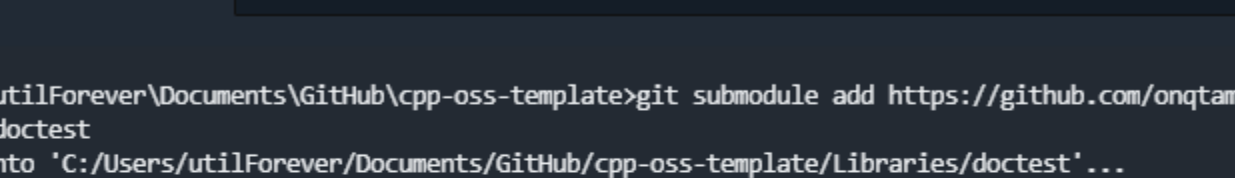
A complete example with a self-registering test that compiles to an executable looks like this:

Donate

- doctest : <https://github.com/onqtam/doctest>



- Github 프로젝트에 doctest 프레임워크를 추가하는 방법
 - `git submodule add <저장소 주소> <저장할 위치>`
`git submodule add https://github.com/onqtam/doctest Libraries/doctest`



The screenshot shows a Windows command prompt window with a dark background. The title bar at the top reads "C:\WINDOWS\SYS...". The command prompt shows the following sequence of commands and output:

```
C:\Users\utilForever\Documents\GitHub\cpp-oss-template>git submodule add https://github.com/onqtam/doctest Libraries\doctest
Cloning into 'C:/Users/utilForever/Documents/GitHub/cpp-oss-template/Libraries/doctest'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 9848 (delta 2), reused 3 (delta 0), pack-reused 9838
Receiving objects: 100% (9848/9848), 5.59 MiB | 3.72 MiB/s, done.
Resolving deltas: 100% (6382/6382), done.
warning: LF will be replaced by CRLF in .gitmodules.
The file will have its original line endings in your working directory.

C:\Users\utilForever\Documents\GitHub\cpp-oss-template>
```

- 유닛 테스트용 프로그램을 만드는 프로젝트를 생성합니다.
(Tests/UnitTests 폴더에 생성합니다.)

```
# Target name
set(target UnitTests)

# Includes
include_directories(${CMAKE_CURRENT_SOURCE_DIR})

# Sources
file(GLOB sources ${CMAKE_CURRENT_SOURCE_DIR}/*.cpp)

# Build executable
add_executable(${target} ${sources})

# Project options
set_target_properties(${target} PROPERTIES ${DEFAULT_PROJECT_OPTIONS})
```


- 유닛 테스트용 프로그램을 만드는 프로젝트를 생성합니다.
(Tests/UnitTests 폴더에 생성합니다.)

```
# Compile options
# GCC and Clang compiler options
if(CMAKE_CXX_COMPILER_ID MATCHES "GNU" OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
    set(DEFAULT_COMPILE_OPTIONS ${DEFAULT_COMPILE_OPTIONS}
        -Wno-unused-variable)
endif()
target_compile_options(${target} PRIVATE ${DEFAULT_COMPILE_OPTIONS})
target_compile_definitions(${target} PRIVATE)

# Link libraries
target_link_libraries(${target} PRIVATE ${DEFAULT_LINKER_OPTIONS}
    cpp-oss-template doctest)
```

- 유닛 테스트 코드를 작성해 봅시다.
(경로 : Tests/UnitTests)

SimpleTest.cpp

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
#include "doctest.h"

#include <cpp-oss-template/Test.hpp>

TEST_CASE("Simple test")
{
    CHECK(Add(2, 3) == 5);
}
```

- 프로젝트 최상위 경로에 있는 CMakeLists.txt에
방금 생성한 UnitTests 프로젝트를 추가합니다.

```
...  
  
# Includes  
include_directories(Includes)  
include_directories(Libraries)  
include_directories(Libraries/doctest/doctest)  
  
...  
  
# Project modules  
add_subdirectory(Libraries/doctest)  
add_subdirectory(Sources/cpp-oss/template)  
add_subdirectory(Tests/UnitTests)
```

- 테스트 코드를 모두 작성했다면, 패스하는지 확인해 봅시다.

```
[doctest] doctest version is "2.3.4"
```

```
[doctest] run with "--help" for options
```

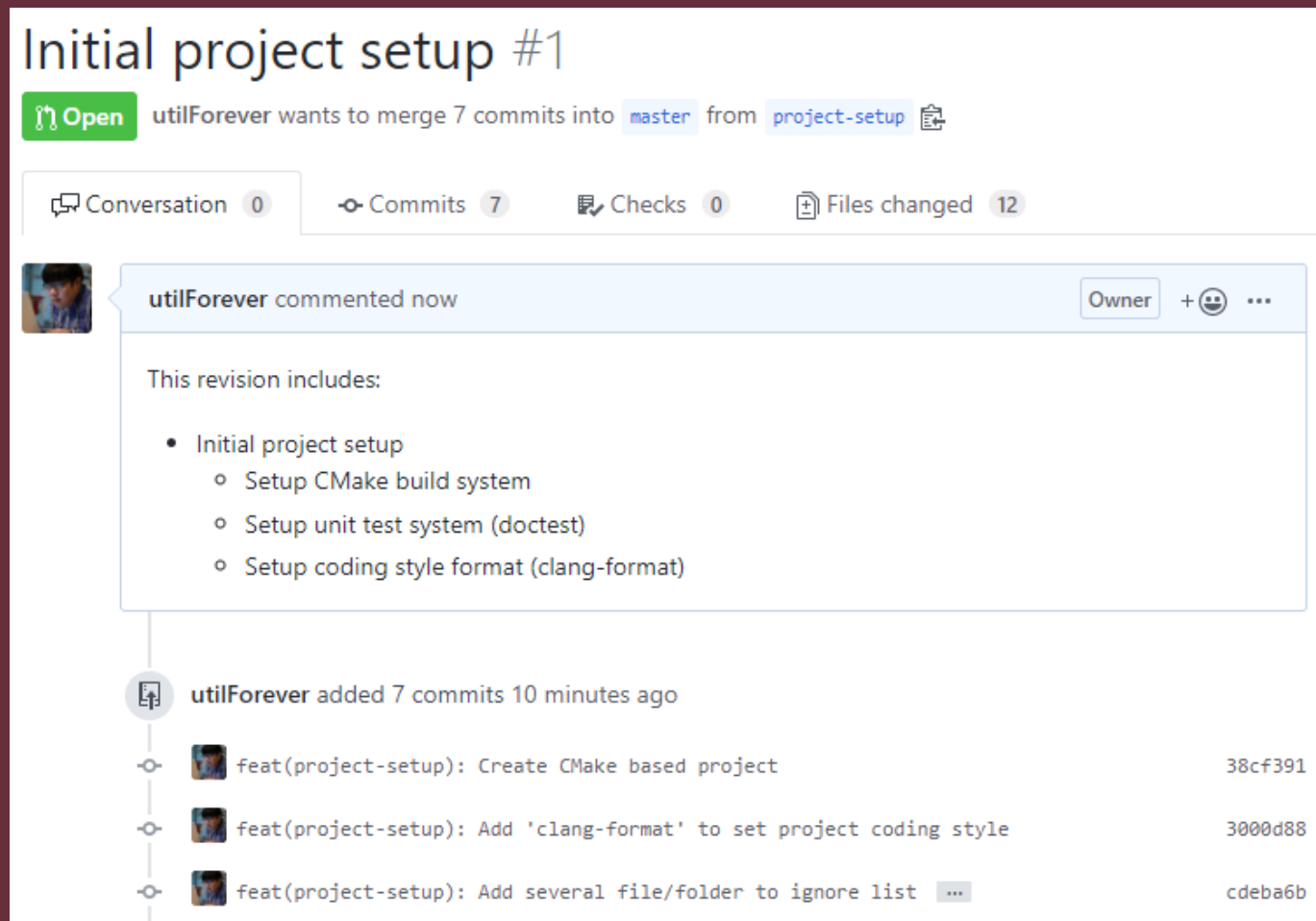
```
=====
```

```
[doctest] test cases: 1 | 1 passed | 0 failed | 0 skipped
```

```
[doctest] assertions: 1 | 1 passed | 0 failed |
```

```
[doctest] Status: SUCCESS!
```

- 여기까지 작업한 내용을 Pull Request를 통해 업로드합니다.



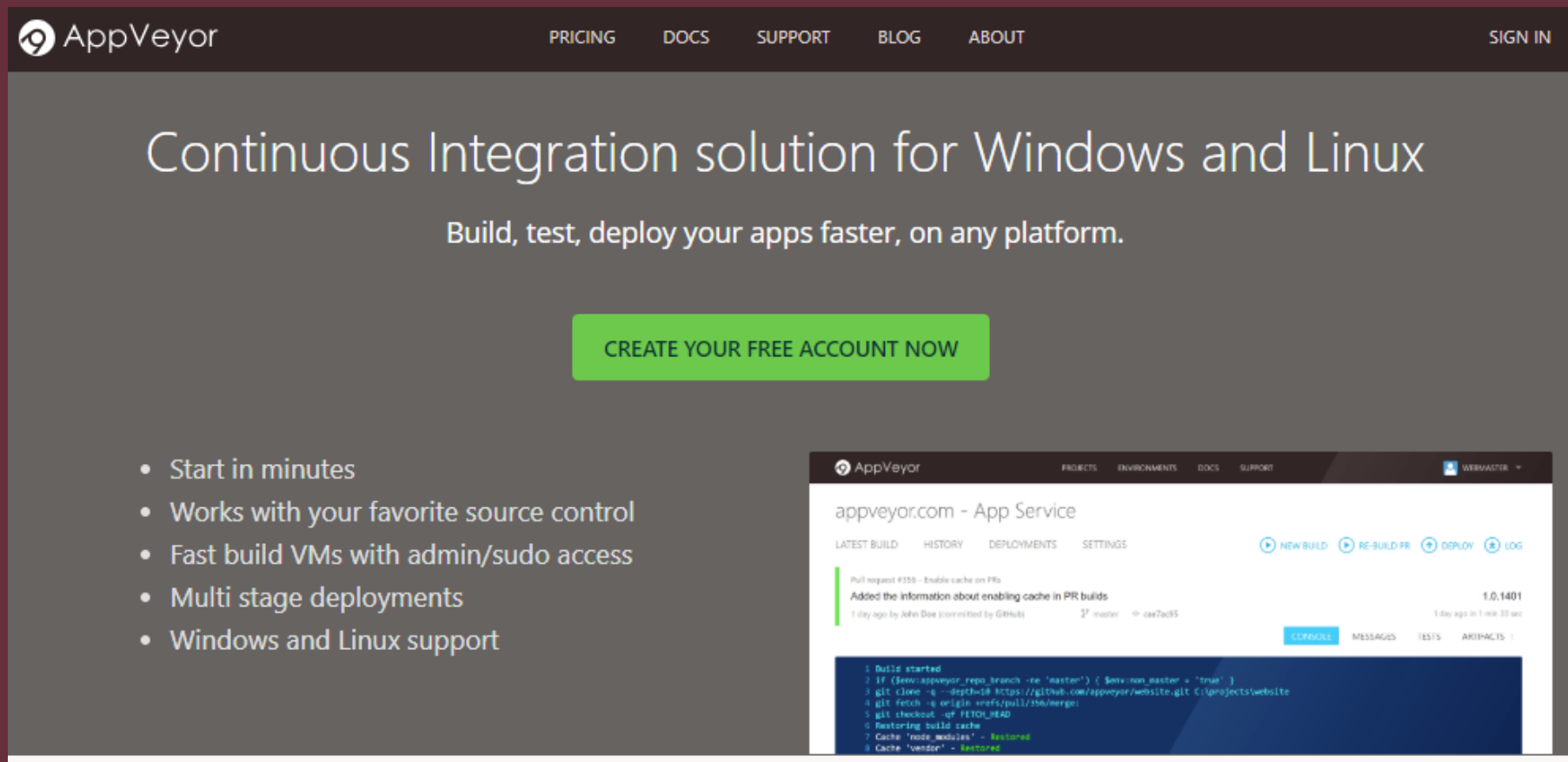
- 지속적인 통합(Continuous Integration)은 이전과 같은 문제점을 해결하기 위해 개발 팀원들이 작성한 코드를 최대한 자주 통합하는 소프트웨어 개발 실천법입니다.
- 지속적인 통합의 장점
 - 자동화를 통해 수시로 통합할 수 있으며, 문제를 조기에 발견하고 조치할 수 있습니다.
 - 코드와 테스트를 개인 환경과 독립적으로 구성할 수 있습니다. 즉, 개발자가 코드를 수정하고 커밋하지 않아 개인 환경에서만 빌드되는 문제를 조기에 수정할 수 있습니다.
 - 다른 개발자가 수정한 내용을 자동으로 빌드하고 통합 테스트를 진행할 수 있습니다

- C++을 지원하는 CI 서비스들은 다음과 같습니다.
 - Travis CI
 - CircleCI
 - AppVeyor
 - Azure Pipelines
- 여기서는 Travis CI와 AppVeyor를 사용해보도록 하겠습니다.

Continuous Integration

C++ Korea
C++ Open Source 101

- AppVeyor : <https://www.appveyor.com/>



The image shows the AppVeyor website and a screenshot of its dashboard. The website header includes the AppVeyor logo, navigation links (PRICING, DOCS, SUPPORT, BLOG, ABOUT), and a SIGN IN button. The main heading is "Continuous Integration solution for Windows and Linux" with the subtext "Build, test, deploy your apps faster, on any platform." A prominent green button says "CREATE YOUR FREE ACCOUNT NOW". Below this, a list of features is provided: "Start in minutes", "Works with your favorite source control", "Fast build VMs with admin/sudo access", "Multi stage deployments", and "Windows and Linux support". The bottom right shows a screenshot of the AppVeyor dashboard for a user named "WEBMASTER". The dashboard displays the "appveyor.com - App Service" page with tabs for LATEST BUILD, HISTORY, DEPLOYMENTS, and SETTINGS. It shows a "Pull request #255 - Enable cache on PRs" with a commit by "John Doe" and a build status of "Succeeded". The build log is visible, showing steps like "Build started", "git clone", "git fetch", "git checkout", "Restoring build cache", "Cache 'node_modules' - Restored", and "Cache 'vendor' - Restored".

AppVeyor

PRICING DOCS SUPPORT BLOG ABOUT SIGN IN

Continuous Integration solution for Windows and Linux

Build, test, deploy your apps faster, on any platform.

[CREATE YOUR FREE ACCOUNT NOW](#)

- Start in minutes
- Works with your favorite source control
- Fast build VMs with admin/sudo access
- Multi stage deployments
- Windows and Linux support

AppVeyor

PROJECTS ENVIRONMENTS DOCS SUPPORT WEBMASTER

appveyor.com - App Service

LATEST BUILD HISTORY DEPLOYMENTS SETTINGS

[NEW BUILD](#) [RE-BUILD PR](#) [DEPLOY](#) [LOG](#)

Pull request #255 - Enable cache on PRs

Added the information about enabling cache in PR builds

1 day ago by John Doe (committed by GitHub) 27 master cae7ac85 1.0.1401 1 day ago in 1 min 33 sec

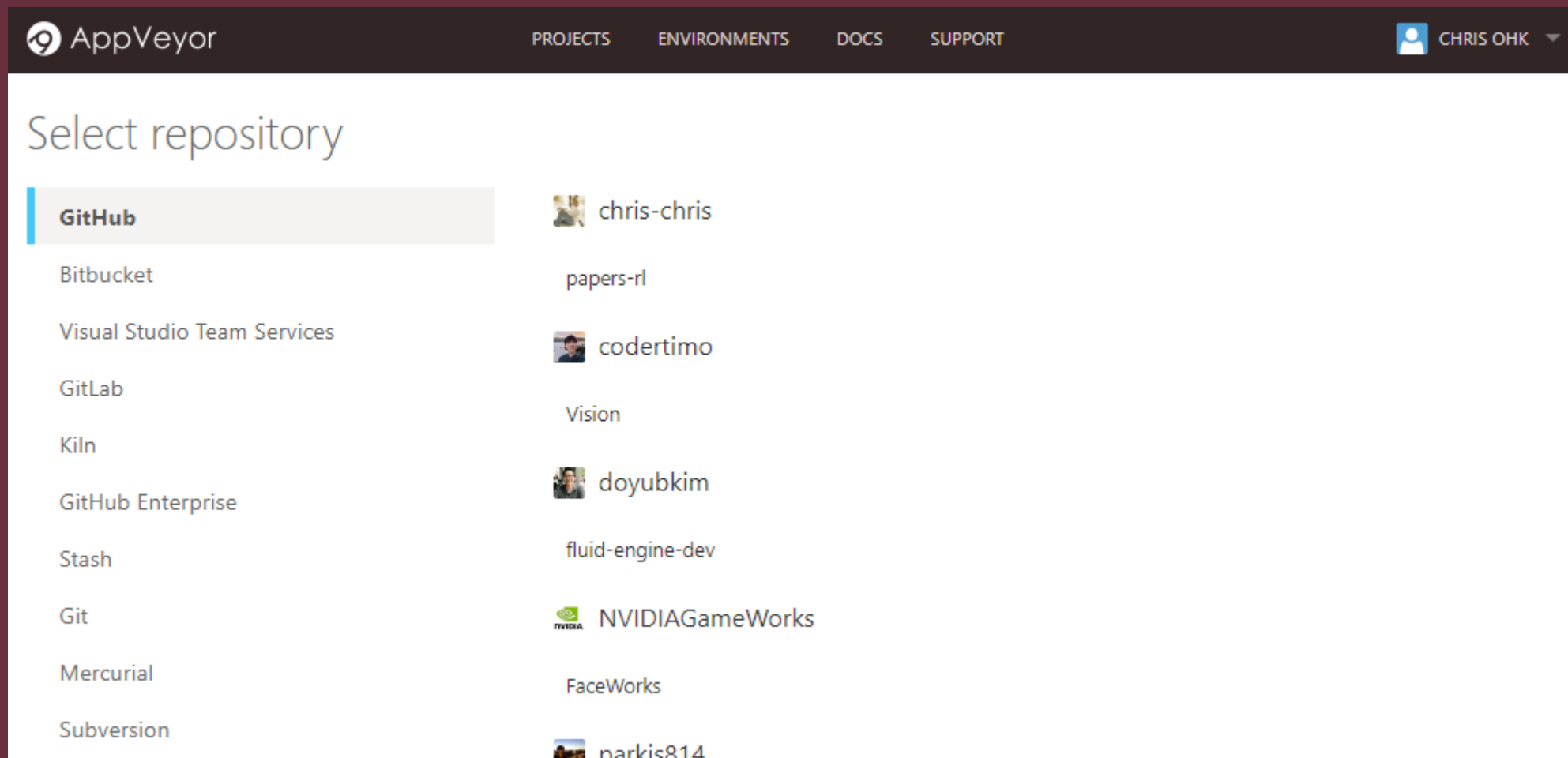
[COMPILE](#) [MESSAGES](#) [TESTS](#) [ARTIFACTS](#)

```
1 Build started
2 if ($env:appveyor_repo_branch -eq 'master') { $env:run_master = 'true' }
3 git clone -q --depth=10 https://github.com/appveyor/website.git C:\projects\website
4 git fetch -q origin refs/pull/255/merge:
5 git checkout -q FETCH_HEAD
6 Restoring build cache
7 Cache 'node_modules' - Restored
8 Cache 'vendor' - Restored
```


Continuous Integration

C++ Korea
C++ Open Source 101

- GitHub 계정으로 로그인한 뒤 새 프로젝트를 추가합니다.



- 최상위 경로에 appveyor.yml 파일을 만들고 다음과 같이 작성합니다.
(자세한 내용은 <https://www.appveyor.com/docs/build-configuration/> 참고)

```
version: 0.1 ({build})
skip_branch_with_pr: true

image:
- Visual Studio 2017
- Visual Studio 2019

platform:
- x64

matrix:
fast_finish: true # Stop remaining jobs after a job failure

configuration:
- Release
```

clone_folder: C:\cpp-oss-template

install:

- git submodule update --init
- ps: |
 - if ("\$env:APPVEYOR_BUILD_WORKER_IMAGE" -eq "Visual Studio 2017") {
 - \$env:CMAKE_GENERATOR = "Visual Studio 15 2017"
 - } else {
 - \$env:CMAKE_GENERATOR = "Visual Studio 16 2019"
 - }

before_build:

- md C:\cpp-oss-template\build
- cd C:\cpp-oss-template\build
- cmake .. -G "%CMAKE_GENERATOR%" -A x64

build:

project: C:\cpp-oss-template\build\cpp-oss-template.sln
parallel: true
verbosity: normal

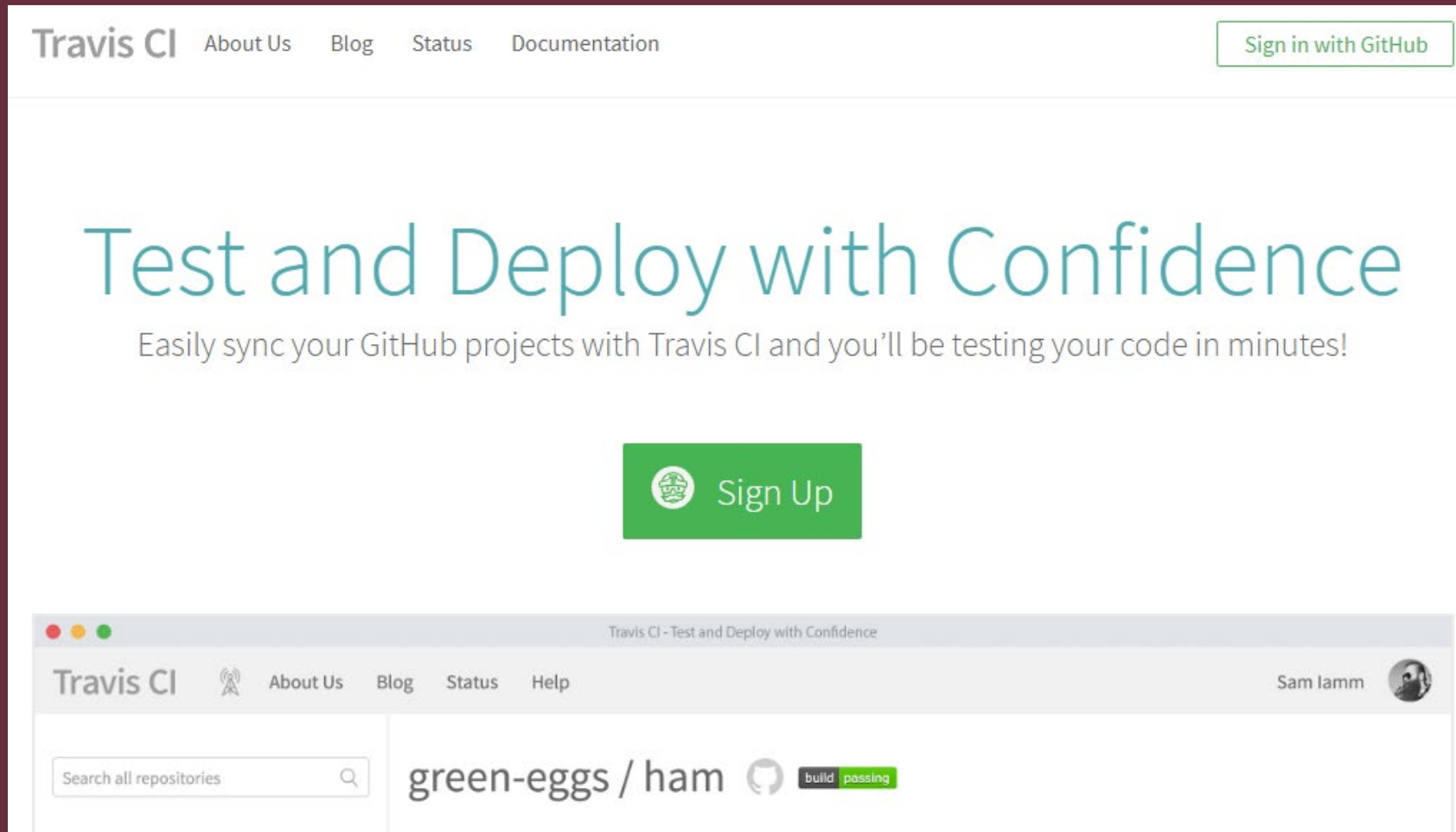
after_build:

- C:\cpp-oss-template\build\bin\Release\UnitTests.exe

Continuous Integration

C++ Korea
C++ Open Source 101

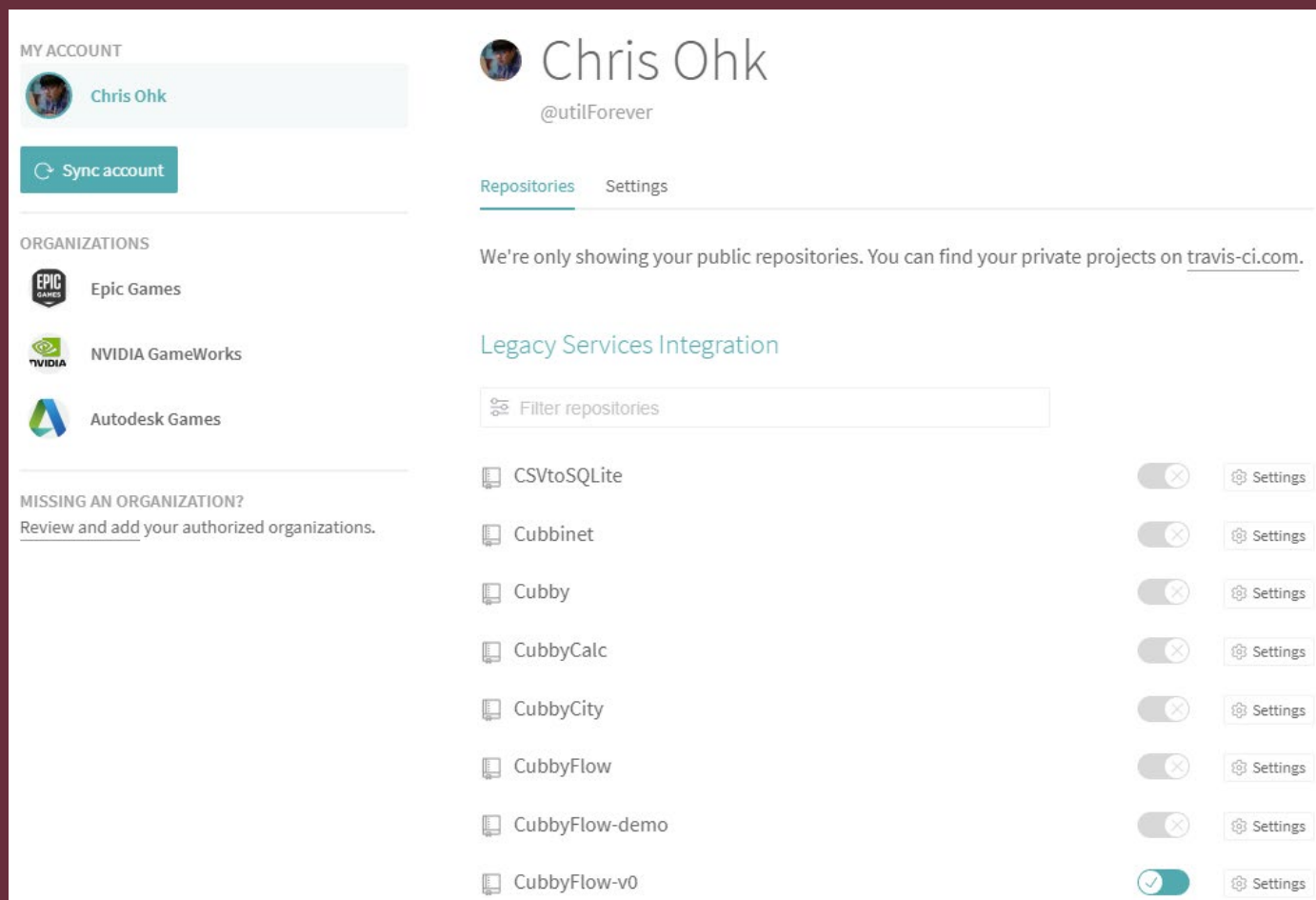
- Travis CI : <https://travis-ci.org/>



Continuous Integration

C++ Korea
C++ Open Source 101

- GitHub 계정으로 로그인한 뒤 새 프로젝트를 추가합니다.



- 최상위 경로에 .travis.yml 파일을 만들고 다음과 같이 작성합니다.
(자세한 내용은 <https://docs.travis-ci.com/user/customizing-the-build/> 참고)

```
language: cpp
matrix:
  include:
    - name: Test Ubuntu 18.04 LTS + gcc
      os: linux
      dist: bionic
      sudo: required
      script: sh Scripts/travis_build.sh
    - name: Test OS X 10.14 + Xcode 10.2 + clang
      os: osx
      osx_image: xcode10.2
      compiler: clang
      script: sh Scripts/travis_build.sh
```

- Scripts 폴더에 travis_build.sh를 만들고 다음 내용을 입력합니다.

travis_build.sh

```
#!/usr/bin/env bash

set -e

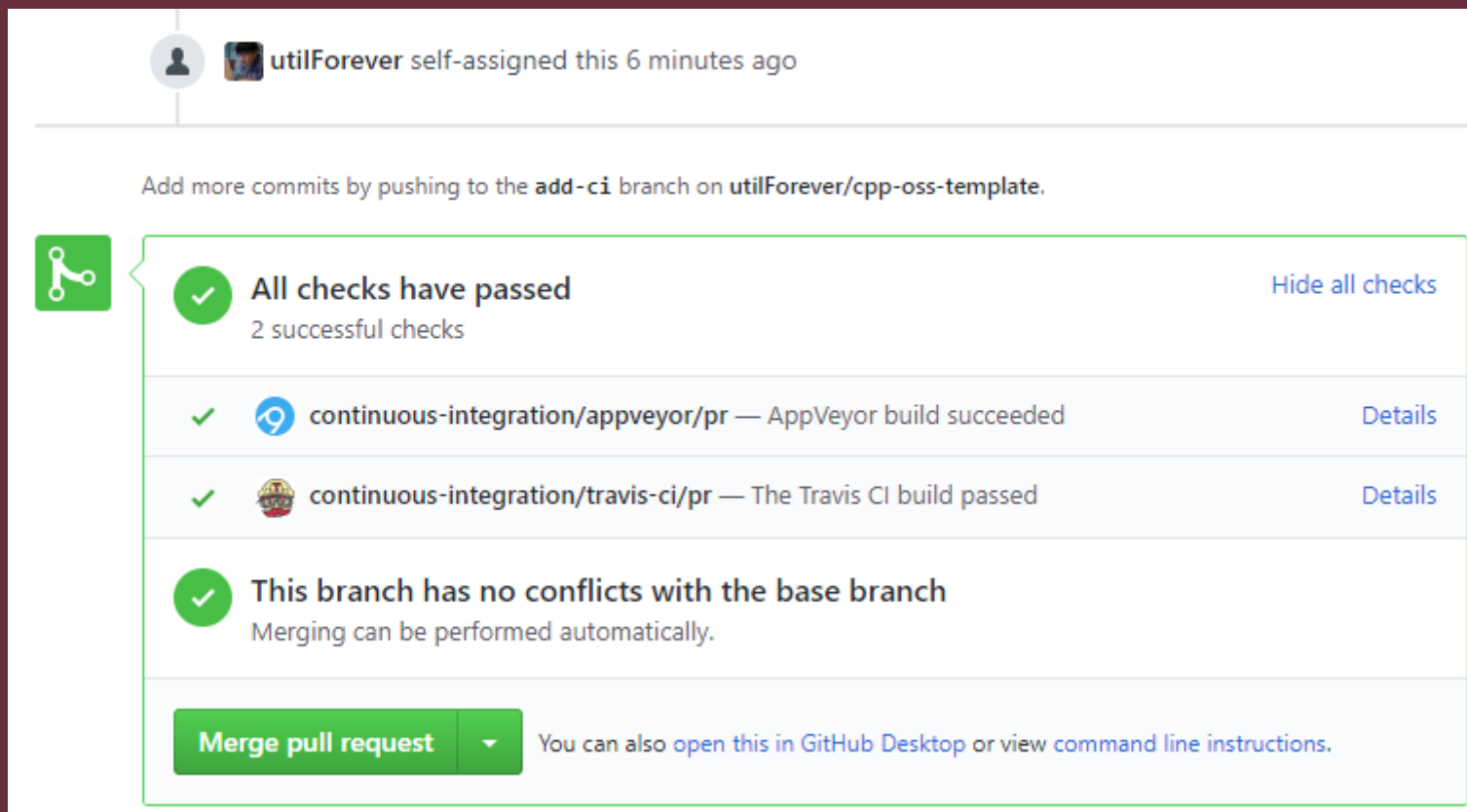
export NUM_JOBS=1

mkdir build
cd build
cmake ..
make
bin/UnitTests
```

PR을 통한 업로드

C++ Korea
C++ Open Source 101


- 이제 코드를 커밋하거나 PR를 할 때마다 프로젝트를 자동 빌드합니다.





The screenshot shows a GitHub pull request interface. At the top, a user 'utilForever' is noted as having self-assigned the task 6 minutes ago. Below this, a message instructs the user to add more commits by pushing to the 'add-ci' branch on the 'utilForever/cpp-oss-template' repository. The main section displays a green box with a 'Merge pull request' button and a dropdown arrow. To the right of the button, it says 'You can also [open this in GitHub Desktop](#) or [view command line instructions](#).' Above the button, three checks are listed, all with green checkmarks: 'All checks have passed' (2 successful checks), 'continuous-integration/appveyor/pr — AppVeyor build succeeded', and 'continuous-integration/travis-ci/pr — The Travis CI build passed'. Each check has a 'Details' link. A 'Hide all checks' link is also present at the top right of the checks section.

utilForever self-assigned this 6 minutes ago

Add more commits by pushing to the **add-ci** branch on **utilForever/cpp-oss-template**.

 Merge pull request ▾ You can also [open this in GitHub Desktop](#) or [view command line instructions](#).

All checks have passed [Hide all checks](#)
2 successful checks

-  [continuous-integration/appveyor/pr](#) — AppVeyor build succeeded [Details](#)
-  [continuous-integration/travis-ci/pr](#) — The Travis CI build passed [Details](#)

This branch has no conflicts with the base branch
Merging can be performed automatically.

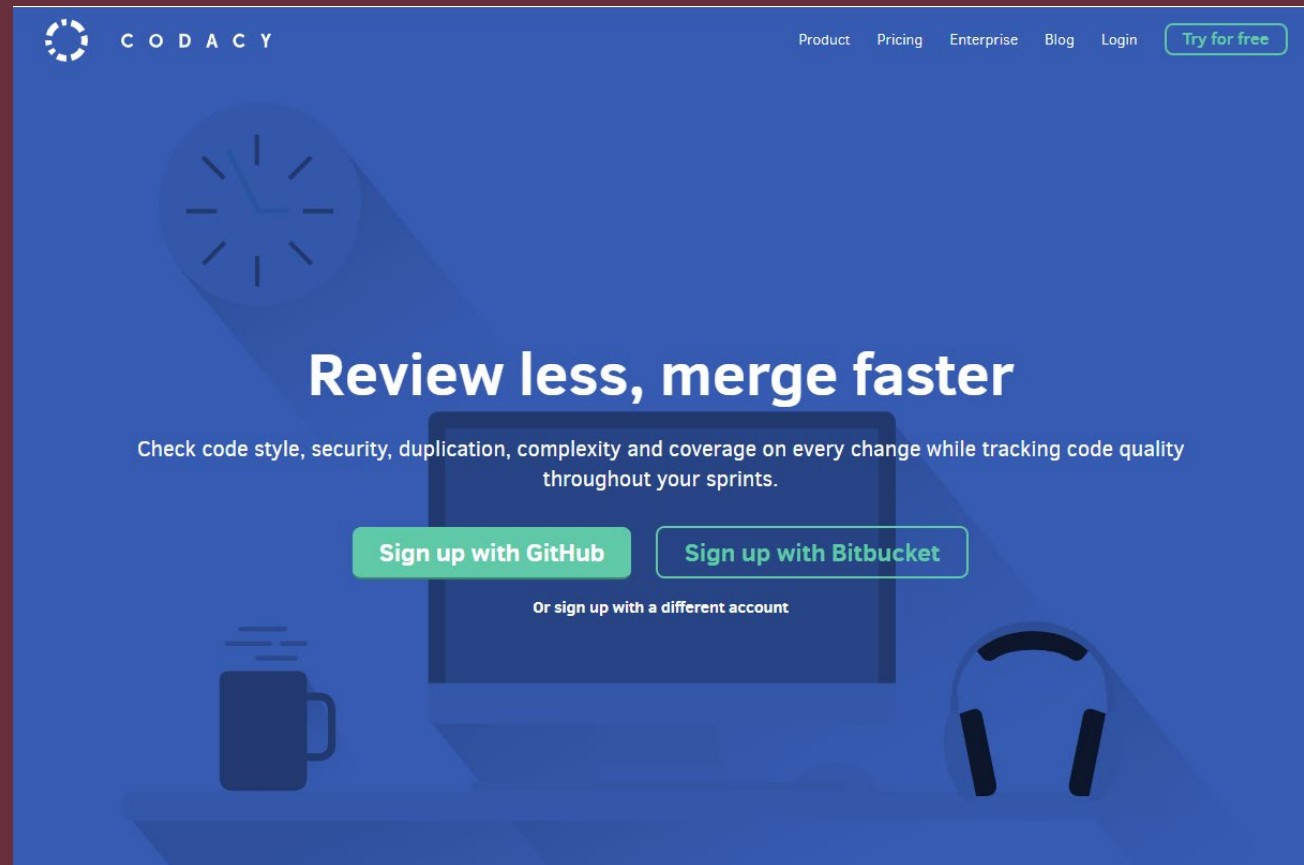
- 코드 리뷰를 하더라도 세부 사항을 모두 보기에는 한계가 있습니다.
 - 보통 크래시가 날 수 있는 부분이나 NULL 포인터 체크 위주로 살펴봅니다.
 - 새로운 기능을 구현하는 경우 보통 담당자가 책임지는 경우가 많습니다.
(모든 내용을 이해하면서 코드를 살펴보기에는 시간이 너무 많이 듭니다.)
- 내가 짠 코드는 잘 짠 코드일까요?
 - 대학생이던 시절에도, 지금도 항상 하고 있는 생각입니다.
 - 누군가 확인해줬으면 좋겠는데 부탁할 사람이 없습니다.
 - 그렇다면 어떻게 해야 할까요?

- C++로 작성한 코드의 품질을 측정해주는 좋은 툴들이 있습니다.
 - Codacy
 - CodeFactor
 - LGTM
- 여기서는 Codacy와 LGTM을 사용해보도록 하겠습니다.

Code Quality

C++ Korea
C++ Open Source 101

- Codacy : <https://www.codacy.com/>

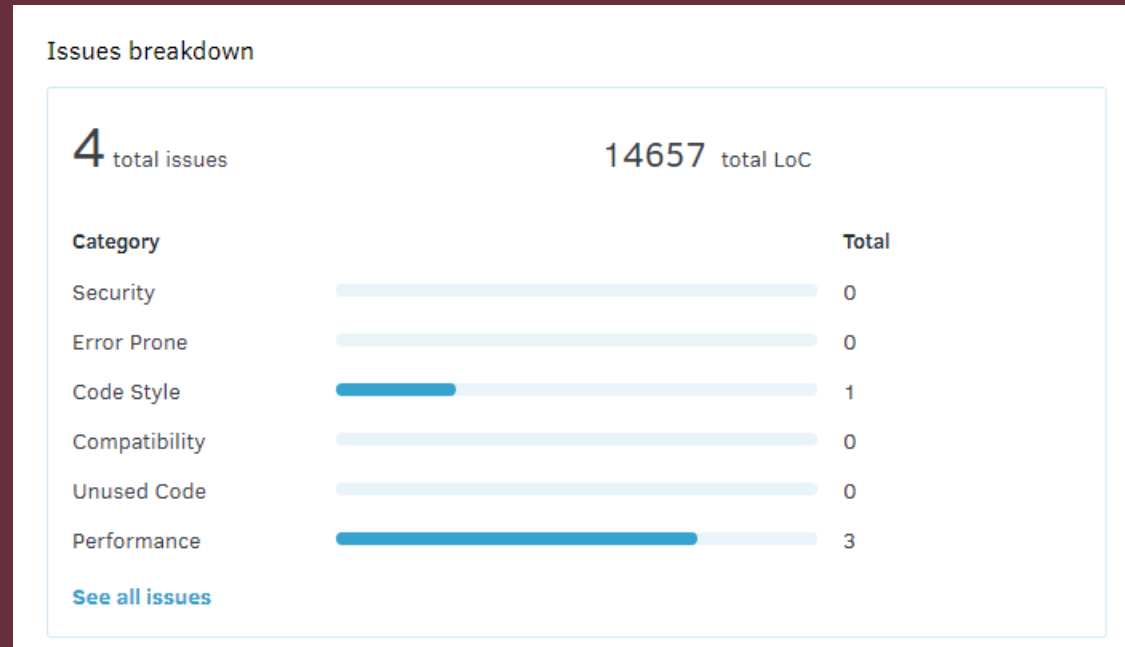


- Github 계정으로 로그인한 뒤 프로젝트를 추가하면 빌드를 진행합니다.
(설정 파일을 따로 추가할 필요가 없다는 게 장점!)
빌드가 되면 코드 품질을 측정해 등급과 이슈 개수를 보여줍니다.

Add project

PROJECT	LAST COMMIT	ISSUES
<div><div>A</div><div>CubbyFlow-v0</div><div>GitHub / Public</div></div>	<div>Chris Ohk 12 days ago</div> <div>Merge pull request #56 from sjvs/patch-1</div>	<div><div>0 NEW</div><div>0 FIXED</div><div>85 TOTAL</div></div>
<div><div>A</div><div>Hearthstonepp</div><div>GitHub / Public</div></div>	<div>Chris Ohk 12 days ago</div> <div>Merge pull request #139 from sjvs/patch-1</div>	<div><div>0 NEW</div><div>0 FIXED</div><div>4 TOTAL</div></div>

- 프로젝트를 클릭해서 들어가면 이슈를 몇 가지 항목으로 분류합니다.
(보안, 코드 스타일, 호환성, 사용하지 않는 코드, 성능 등)



- 이슈를 하나씩 살펴보고 싶다면 각 항목을 클릭하면 됩니다.
이제 내 코드에 어떤 성능 문제가 있는지 확인할 수 있습니다.

The screenshot displays a web-based code quality tool interface. At the top, it shows 'Current Issues' with a dropdown menu set to 'master'. Below this is a filter bar with several dropdown menus: 'Language' (set to 'All'), 'Category' (set to 'Performance'), 'Level' (set to 'All'), 'Pattern' (set to 'All'), and 'Author' (set to 'All'). A close button 'x' is also present. The main content area is titled 'Programs/Console/Console.cpp' and lists three issues, each with a description and a corresponding code snippet:

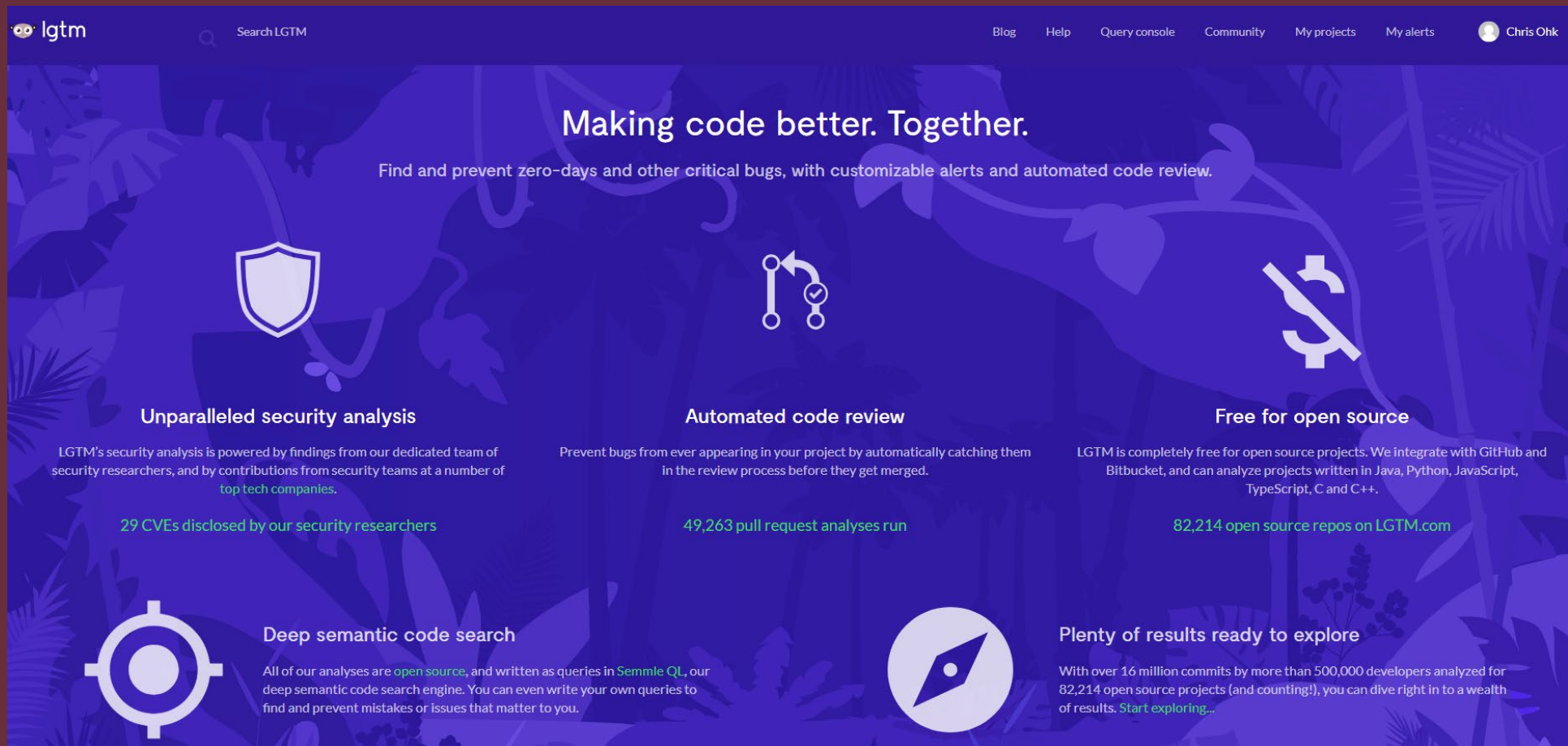
- Function parameter 'questionStr' should be passed by reference.**
459 `size_t Console::InputMenuNum(std::string questionStr, size_t menuSize)`
- Function parameter 'sentence' should be passed by reference.**
478 `bool Console::InputYesNo(std::string sentence) const`
- Function parameter 'commandStr' should be passed by reference.**
495 `std::string commandStr) const`

At the bottom, there are navigation links: '< Previous' and 'Next >'.

Code Quality

C++ Korea
C++ Open Source 101

- LGTM : <https://lgtm.com/>



The screenshot shows the LGTM website homepage with a dark blue background and a tropical leaf pattern. The navigation bar at the top includes the LGTM logo, a search bar, and links for Blog, Help, Query console, Community, My projects, My alerts, and a user profile for Chris Ohk. The main heading is "Making code better. Together." followed by the tagline "Find and prevent zero-days and other critical bugs, with customizable alerts and automated code review." Below this, there are six feature cards arranged in a 2x3 grid. Each card has an icon, a title, a description, and a statistic.

Feature	Statistic
Unparalleled security analysis	29 CVEs disclosed by our security researchers
Automated code review	49,263 pull request analyses run
Free for open source	82,214 open source repos on LGTM.com
Deep semantic code search	-
Plenty of results ready to explore	-

Code Quality

C++ Korea
C++ Open Source 101

- Github 계정으로 로그인한 뒤 프로젝트를 추가하면 빌드를 진행합니다.
이 때 빌드를 위해 별도의 설정 파일을 추가해야 합니다.
(별도의 설정 파일 없이 빌드가 되는 경우도 있습니다.)

11 Projects	<div>Add project</div> <div>GitHub or Bitbucket project URL...</div>			<div>Add</div>		
CppKorea/CppTemplateStudy	<div>±0 alerts</div>	<div>±0 lines</div>	<div>1 Alerts</div>	<div>12 Contributors</div>	<div>127 Lines of code</div>	<div>×</div>
doyubkim/fluid-engine-dev	<div>±0 alerts</div>	<div>-4 lines</div>	<div>114 Alerts</div>	<div>5 Contributors</div>	<div>65.9k Lines of code</div>	<div>×</div>
utilForever/CubbyFlow	<div>±0 alerts</div>	<div>±0 lines</div>	<div>2 Alerts</div>	<div>4 Contributors</div>	<div>1.6k Lines of code</div>	<div>×</div>
utilForever/CubbyFlow-v0	<div>±0 alerts</div>	<div>±0 lines</div>	<div>83 Alerts</div>	<div>3 Contributors</div>	<div>61k Lines of code</div>	<div>×</div>

- 최상위 경로에 .lgTM.yml 파일을 만들고 다음과 같이 작성합니다.
(자세한 내용은 <https://lgTM.com/help/lgtm/lgtm.yml-configuration-file> 참고)

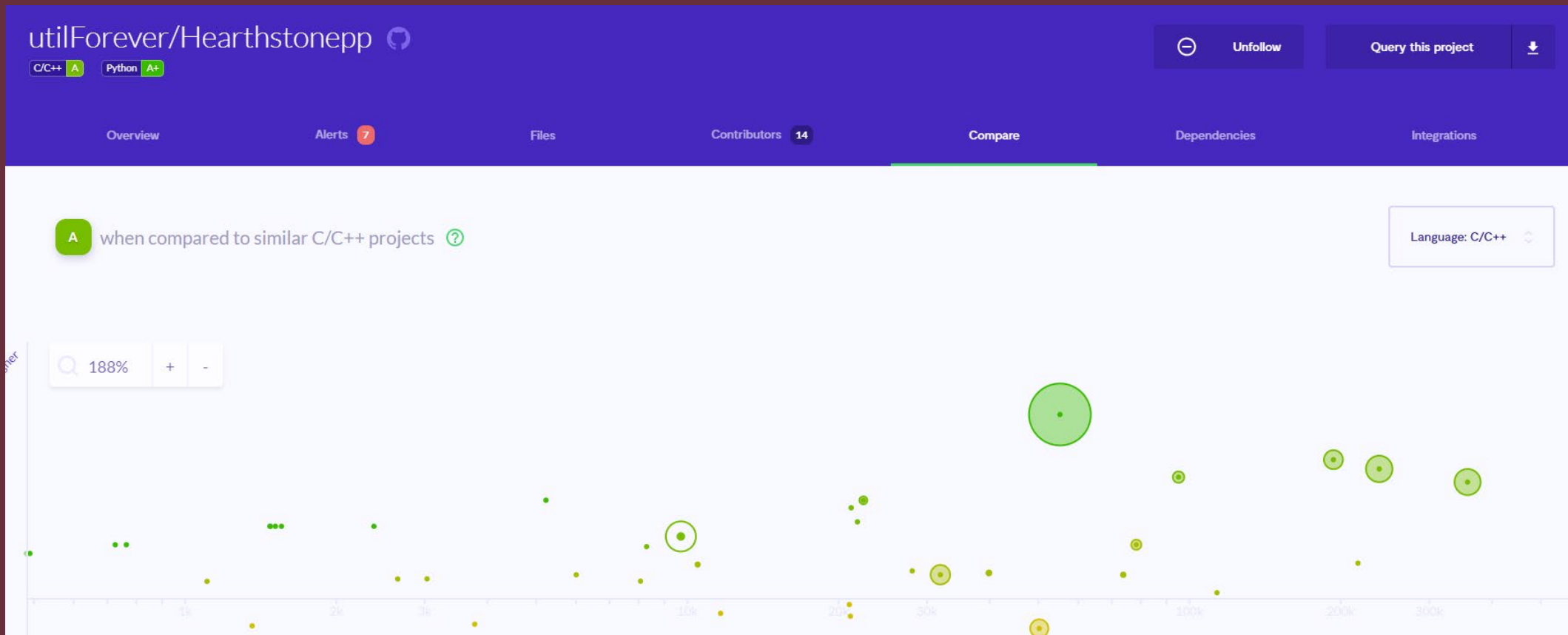
```
extraction:
  cpp:
    configure:
      command:
        - mkdir _lgtm_build_dir
        - cd _lgtm_build_dir
        - cmake ..
    index:
      build_command:
        - cd _lgtm_build_dir
        - make

path_classifiers:
  library:
    - Libraries
```

Code Quality

C++ Korea
C++ Open Source 101

- 이제 LGTM이 빌드를 하고 나서 프로젝트의 코드 품질을 분석합니다. 분석이 끝나면 등급이 나오고 프로젝트의 상대적 위치를 보여줍니다.



- 파일별로 어느 부분이 어떤 문제를 갖고 있는지를 확인할 수 있습니다.

Resource not released in destructor ▾

Source root/Sources/.../BasicTasks/DrawTask.cpp

```
11 1-96
97 {
98     case +CardType::MINION:
99         m_entity = new Minion(card);
100         break;
101     case +CardType::WEAPON:
102         m_entity = new Weapon(card);
103         break;
104     default:
105         m_entity = new Entity(card);
106 }
107 }
```

Resource m_entity is acquired by class DrawCardTask but not released anywhere in this class.

Resource m_entity is acquired by class DrawCardTask but not released anywhere in this class.

Resource m_entity is acquired by class DrawCardTask but not released anywhere in this class.

11 108-128

- 작성한 코드 중에서 얼마나 많은 부분을 단위 테스트하고 있을까요?
테스트를 진행할 때 코드가 얼마나 실행되었는지를 나타내는 지표를
코드 커버리지 (Code Coverage)라고 합니다.

```
int foo(int x, int y)
{
    int z = 0;
    if ((x > 0) && (y > 0))
    {
        z = x;
    }
    return z;
}
```

foo(1, 1)을 호출할 때와
foo(0, 1)을 호출할 때 다른 점이 무엇일까요?

- C++의 코드 커버리지를 검사하는 툴들이 있습니다.
 - CppUnit
 - lcov/gcov
- 여기서는 lcov를 사용해보도록 하겠습니다.

Test Coverage

C++ Korea
C++ Open Source 101

- lcov : <http://ltp.sourceforge.net/coverage/lcov.php>

LCOV - the LTP GCOV extension

About LCOV

LCOV is a graphical front-end for GCC's coverage testing tool [gcov](#). It collects gcov data for multiple source files and creates HTML pages containing the source code annotated with coverage information. It also adds overview pages for easy navigation within the file structure. LCOV supports statement, function and branch coverage measurement.

Example output

HTML output of a small example project.

LCOV - code coverage report						
Current view: top level		Hit		Total		Coverage
Test: Basic example (view description)		Lines: 20		22		90.9 %
Date: 2016-12-20 14:12:28		Functions: 3		3		100.0 %
Legend: Not exec. ■ Not covered ■ Covered (90.9 %)		Branches: 8		10		80.0 %
Directory	Line Coverage %	Functions %	Branches %			
example	<div><div></div></div> 90.0 % 9 / 10	100.0 % 1 / 1	75.0 % 3 / 4			
example/method	<div><div></div></div> 91.7 % 11 / 12	100.0 % 2 / 2	83.3 % 5 / 6			
Generated by: LCOV version 1.12						

Screenshot 1: Overview page

LCOV - code coverage report						
Current view: top level - example/methods - Basic.c (source / functions)		Hit	Total	Coverage		
Test: Basic example (view description)		Lines: 8	8	100.0 %		
Date: 2016-12-20 14:12:28		Functions: 1	1	100.0 %		
Legend: Lines ■ hit ■ not hit (Branches ■ taken ■ not taken ■ not executed)		Branches: 4	4	100.0 %		
Branch data	Line data	Source code				
1	1	/* methods/Basic.c */				
2	2	/* Calculate the sum of a given range of integer numbers. */				
3	3	/* This particular method of implementation works by way of brute force, i.e. it iterates over the entire range while adding the numbers up finally */				
4	4	/* Get the total sum, as a positive side effect, we're able to easily detect overflow, i.e. situations in which the sum would exceed the capacity of an integer variable. */				
5	5	/* */				
6	6	/* */				
7	7	/* */				
8	8	/* */				
9	9	/* */				
10	10	/* */				
11	11	/* */				
12	12	/* */				
13	13	/* */				
14	14	/* */				
15	15	/* */				
16	16	/* */				
17	17	/* */				
18	18	/* */				
19	19	/* */				
20	20	/* */				
21	21	/* */				
22	22	/* */				
23	23	/* */				
24	24	/* */				
25	25	/* */				
26	26	/* */				
27	27	/* */				
28	28	/* */				
29	29	/* */				
30	30	/* */				
31	31	/* */				
32	32	/* */				
33	33	/* */				
34	34	/* */				
35	35	/* */				
36	36	/* */				
37	37	/* */				
38	38	/* */				
39	39	/* */				
40	40	/* */				
41	41	/* */				
42	42	/* */				
43	43	/* */				
44	44	/* */				
45	45	/* */				
46	46	/* */				
47	47	/* */				
48	48	/* */				
49	49	/* */				
50	50	/* */				
51	51	/* */				
52	52	/* */				
53	53	/* */				
54	54	/* */				
55	55	/* */				
56	56	/* */				
57	57	/* */				
58	58	/* */				
59	59	/* */				
60	60	/* */				
61	61	/* */				
62	62	/* */				
63	63	/* */				
64	64	/* */				
65	65	/* */				
66	66	/* */				
67	67	/* */				
68	68	/* */				
69	69	/* */				
70	70	/* */				
71	71	/* */				
72	72	/* */				
73	73	/* */				
74	74	/* */				
75	75	/* */				
76	76	/* */				
77	77	/* */				
78	78	/* */				
79	79	/* */				
80	80	/* */				
81	81	/* */				
82	82	/* */				
83	83	/* */				
84	84	/* */				
85	85	/* */				
86	86	/* */				
87	87	/* */				
88	88	/* */				
89	89	/* */				
90	90	/* */				
91	91	/* */				
92	92	/* */				
93	93	/* */				
94	94	/* */				
95	95	/* */				
96	96	/* */				
97	97	/* */				
98	98	/* */				
99	99	/* */				
100	100	/* */				
101	101	/* */				
102	102	/* */				
103	103	/* */				
104	104	/* */				
105	105	/* */				
106	106	/* */				
107	107	/* */				
108	108	/* */				
109	109	/* */				
110	110	/* */				
111	111	/* */				
112	112	/* */				
113	113	/* */				
114	114	/* */				
115	115	/* */				
116	116	/* */				
117	117	/* */				
118	118	/* */				
119	119	/* */				
120	120	/* */				
121	121	/* */				
122	122	/* */				
123	123	/* */				
124	124	/* */				
125	125	/* */				
126	126	/* */				
127	127	/* */				
128	128	/* */				
129	129	/* */				
130	130	/* */				
131	131	/* */				
132	132	/* */				
133	133	/* */				
134	134	/* */				
135	135	/* */				
136	136	/* */				
137	137	/* */				
138	138	/* */				
139	139	/* */				
140	140	/* */				
141	141	/* */				
142	142	/* */				
143	143	/* */				
144	144	/* */				
145	145	/* */				
146	146	/* */				
147	147	/* */				
148	148	/* */				
149	149	/* */				
150	150	/* */				
151	151	/* */				
152	152	/* */				
153	153	/* */				
154	154	/* */				
155	155	/* */				
156	156	/* */				

- **lcov 사용 방법**

```
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Debug
make -j 8
lcov -c -i -d Tests/UnitTests -o base.info
bin/UnitTests
lcov -c -d Tests/UnitTests -o test.info
lcov -a base.info -a test.info -o coverage.info
lcov -r coverage.info '/usr/*' -o coverage.info
lcov -r coverage.info '*/Libraries/*' -o coverage.info
lcov -r coverage.info '*/Tests/*' -o coverage.info
lcov -l coverage.info
genhtml coverage.info -o out
```

Test Coverage

C++ Korea
C++ Open Source 101

- 페이지를 확인해 보면 폴더별로 커버리지 결과를 볼 수 있습니다.


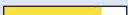








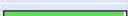









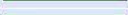
LCOV - code coverage report

Current view: top level

Test: Total.info

Date: 2018-07-21 02:05:38

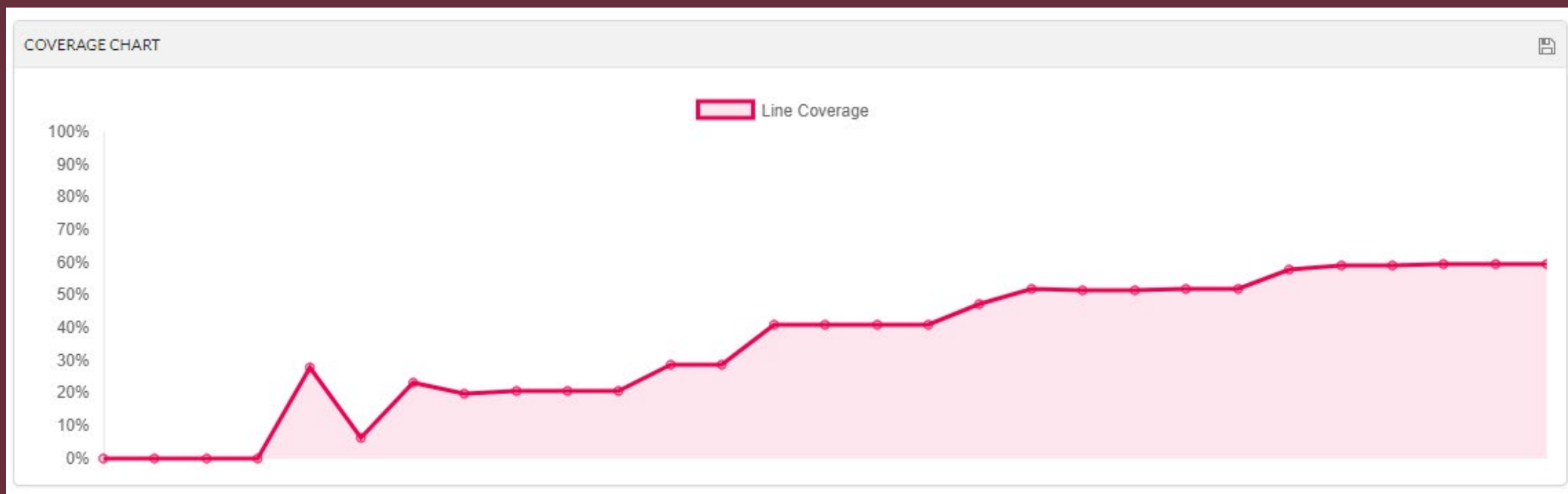
	Hit	Total	Coverage
Lines:	6606	6905	95.7 %
Functions:	3316	3539	93.7 %

Directory	Line Coverage ↕	Functions ↕
Array	 100.0 % 794 / 794	99.6 % 547 / 549
BoundingBox	 79.6 % 218 / 274	78.2 % 43 / 55
Collider	 100.0 % 8 / 8	100.0 % 12 / 12
Emitter	 100.0 % 8 / 8	71.4 % 10 / 14
FDM	 100.0 % 26 / 26	100.0 % 16 / 16
Field	 100.0 % 4 / 4	50.0 % 4 / 8
Geometry	 88.0 % 733 / 833	72.7 % 178 / 245
Grid	 100.0 % 20 / 20	80.0 % 40 / 50
LevelSet	 100.0 % 8 / 8	100.0 % 2 / 2
Math	 91.3 % 596 / 653	100.0 % 97 / 97
Matrix	 98.2 % 2326 / 2368	99.8 % 983 / 985
Point	 100.0 % 379 / 379	98.8 % 159 / 161
QueryEngine	 86.9 % 73 / 84	68.2 % 30 / 44
Ray	 100.0 % 22 / 22	100.0 % 8 / 8
Searcher	 100.0 % 8 / 8	62.5 % 10 / 16
Solver/FDM	 83.3 % 20 / 24	55.6 % 20 / 36
Solver/Grid	 100.0 % 12 / 12	50.0 % 6 / 12
Solver/LevelSet	 100.0 % 6 / 6	50.0 % 6 / 12
Surface	 100.0 % 19 / 19	72.7 % 16 / 22
Utils	 95.6 % 259 / 271	89.2 % 390 / 437
Vector	 98.4 % 1067 / 1084	97.5 % 739 / 758

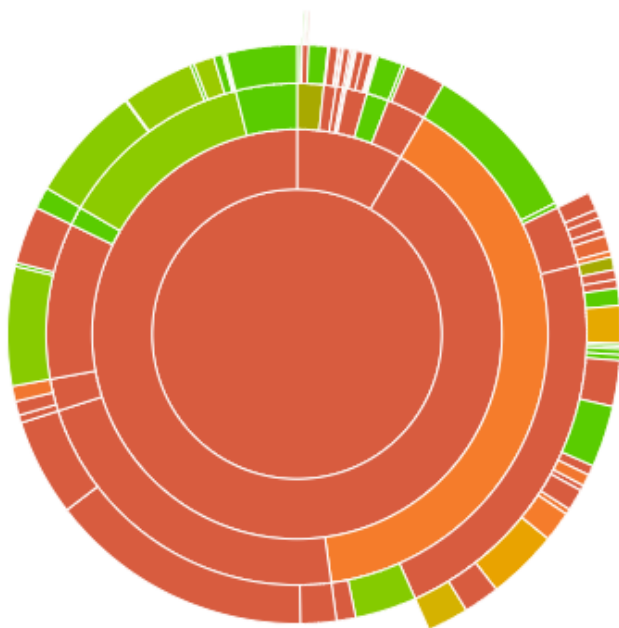
Test Coverage

C++ Korea
C++ Open Source 101

- 시각화된 결과를 보고 싶다면 Codecov와 같은 툴과 연동하면 됩니다.
- Codecov : <https://codecov.io/>



COVERAGE SUNBURST



ALL RECENT COMMITS

[Update HowToAddTask - Fix some words2](#)[Browse Report](#)**revsic** 5 days ago documentation 7f3e55a CI Passed[Update HowToAddTask - Fix some words](#)[Browse Report](#)**revsic** 5 days ago documentation d860eca CI Passed[Merge branch 'documentation' of https://github.com/utilForever/Hearthstonepp into documentation](#)[Browse Report](#)**revsic** 6 days ago documentation ae767ec CI Passed[Upload HowToAddTask - Document for task implementaiton](#)[Browse Report](#)**revsic** 7 days ago documentation 0bb4763 CI Passed[Merge remote-tracking branch 'origin/documentation' into documentation](#)[Browse Report](#)**utilForever** 11 days ago documentation beb52db CI Passed[Merge pull request #139 from sjvs/patch-1](#)[Browse Report](#)**utilForever** 13 days ago master 38e5bf5 CI Passed[View all recent commits](#)

Files



Coverage

Includes

118

47

0

71

39.83%

Sources

1,281

792

0

489

61.82%

Project Totals (77 files)

1,399

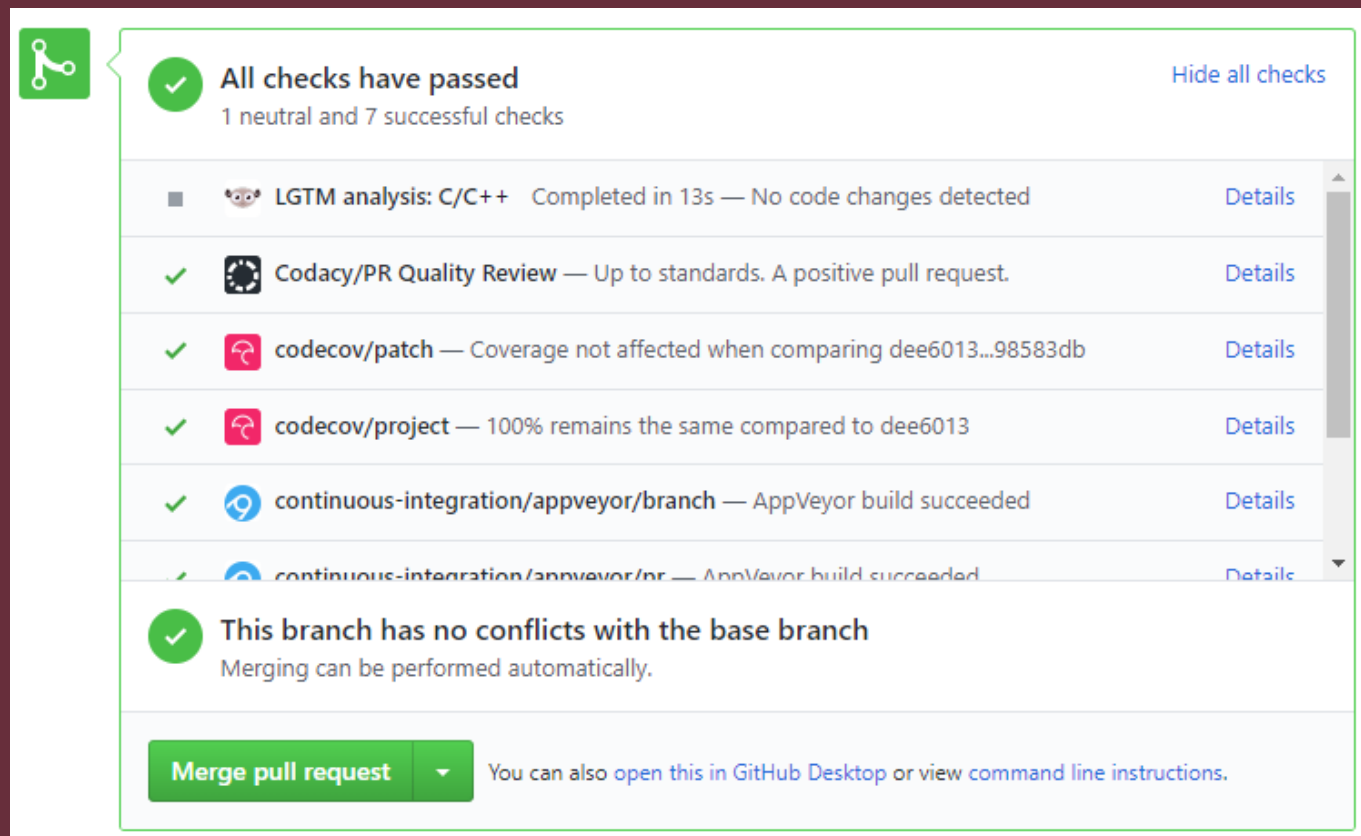
839

0







560

59.97%

- 이제 코드를 커밋하거나 PR를 할 때마다 코드 품질을 체크하고 테스트 커버리지를 측정해 줍니다.



The screenshot shows a GitHub Actions workflow status for a pull request. At the top, a green checkmark icon and the text "All checks have passed" are displayed, along with "1 neutral and 7 successful checks" and a "Hide all checks" link. Below this, a list of checks is shown, each with a status icon, a checkmark, and a "Details" link:



-  LGTM analysis: C/C++ Completed in 13s — No code changes detected [Details](#)
-  Codacy/PR Quality Review — Up to standards. A positive pull request. [Details](#)
-  codecov/patch — Coverage not affected when comparing dee6013...98583db [Details](#)
-  codecov/project — 100% remains the same compared to dee6013 [Details](#)
-  continuous-integration/appveyor/branch — AppVeyor build succeeded [Details](#)
-  continuous-integration/appveyor/pr — AppVeyor build succeeded [Details](#)

Below the list, a green checkmark icon and the text "This branch has no conflicts with the base branch" are displayed, along with "Merging can be performed automatically." At the bottom, a green button labeled "Merge pull request" is shown, followed by the text "You can also [open this in GitHub Desktop](#) or view [command line instructions](#)."

- CI/코드 품질/테스트 커버리지 툴 사이트의 설정 메뉴를 보면 배지를 추가하는 방법이 나와 있습니다.

Project badges

Using shields.io, you can embed the following images in your `README` files or other project pages. Choose the badge to display and format you need, then copy the markup into your page. The badges will be displayed and updated automatically.



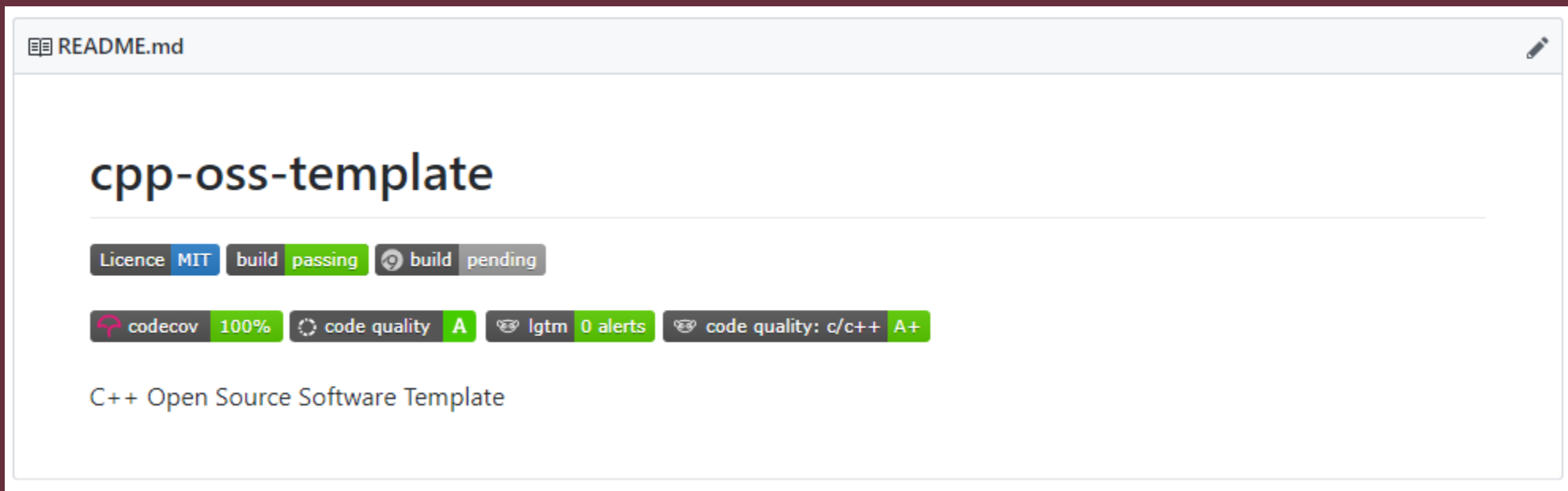
Markdown

HTML

```
[![Total alerts](https://img.shields.io/lgtm/alerts/g/utilForever/cpp-oss-template.svg?logo=lgtm&logoWidth=18)](https://lgtm.com/projects/g/utilForever/cpp-oss-template/alerts/)
```

완성!

C++ Korea
C++ Open Source 101



- 이제 여러분도 C++ 오픈 소스 프로젝트를 시작해 볼 수 있습니다.
 - CMake를 통해 크로스 플랫폼 빌드를 할 수 있습니다.
 - 나만의 코드 스타일을 지정할 수 있습니다.
 - 테스트 프레임워크를 통해 코드의 동작을 테스트할 수 있습니다.
 - CI를 통해 여러 플랫폼에서 정상 빌드됨을 확인할 수 있습니다.
 - 코드 품질 툴을 통해 보안 및 성능 이슈를 관리할 수 있습니다.
 - 테스트 커버리지 툴을 통해 테스트 코드를 관리할 수 있습니다.
- 발표에서 다루지 못한 툴들도 많습니다.
이 부분은 저장소에 따로 추가해 놓을 예정이니 참고 부탁드립니다.

감사합니다

utilForever@gmail.com

<http://github.com/utilForever>