

프로그래밍 역량 강화 전문기관, 민코딩

Baseball KATA



목차

1. Baseball 소개
2. Baseball 프로젝트 준비
3. Baseball TDD 개발

Baseball 소개

Baseball Game 소개

✓ 세 개의 숫자를 맞추는 게임

- 세 자리 숫자를 시도할 때 마다, 정답인지 아닌지 알려주며 정답을 유추할 수 있도록 힌트를 준다.

✓ 힌트를 주는 방식

- Strike / Ball의 개수를 알려준다.
- Ball : 정답과 일치하는 숫자의 개수
- Strike : 정답과 일치하는 숫자이면서, 위치까지 정한 숫자의 개수



Baseball Game 예시

✓ 다음은 정답이 “123” 일 때 예시이다.

- `guess("456")`
→ `solved = false`, `strikes = 0`, `balls = 0`
- `guess("129")`
→ `solved = false`, `strikes = 2`, `balls = 0`
- `guess("240")`
→ `solved = false`, `strikes = 0`, `balls = 1`
- `guess("321")`
→ `solved = false`, `strikes = 1`, `balls = 2`
- `guess("123")`
→ `solved = true`, `strikes = 3`, `balls = 0`



Baseball 프로젝트 준비

초기세팅 & 창배치

- ✓ 왼쪽 화면 : baseball_test.cpp
- ✓ 오른쪽 화면 : baseball.cpp



The screenshot shows a code editor with two open files. The left file, `baseball_test.cpp`, contains the following code:

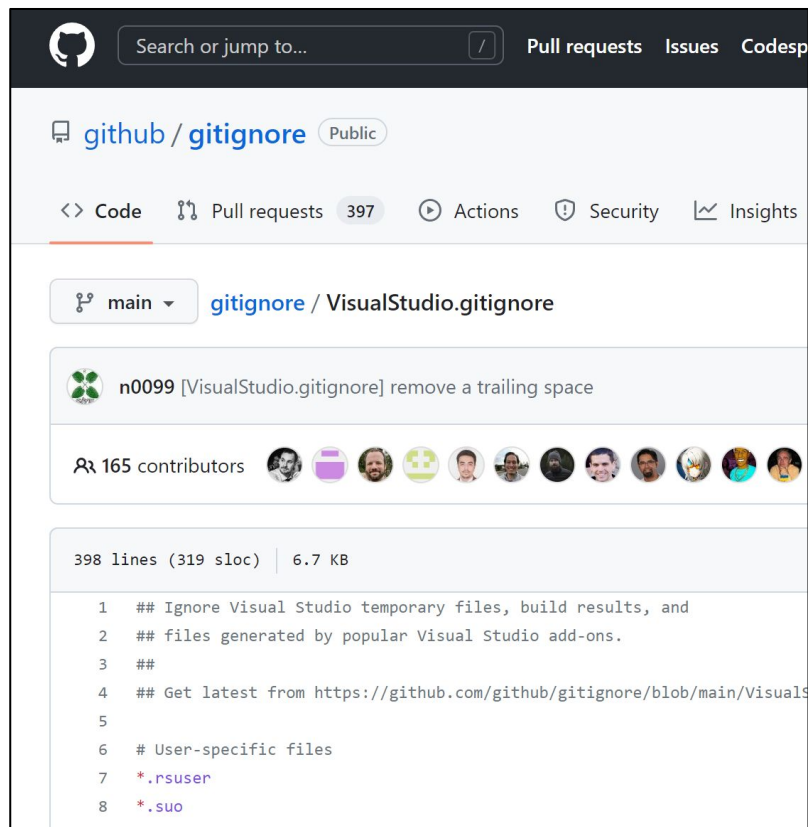
```
1 #include "pch.h"
2 #include "..\Project9\baseball.cpp"
3
4 TEST(BaseballGame, TryGameTest) {
5     EXPECT_EQ(1, 1);
6 }
```

The right file, `baseball.cpp`, contains the following code:

```
1 class Baseball {
2 public:
3
4 };
```

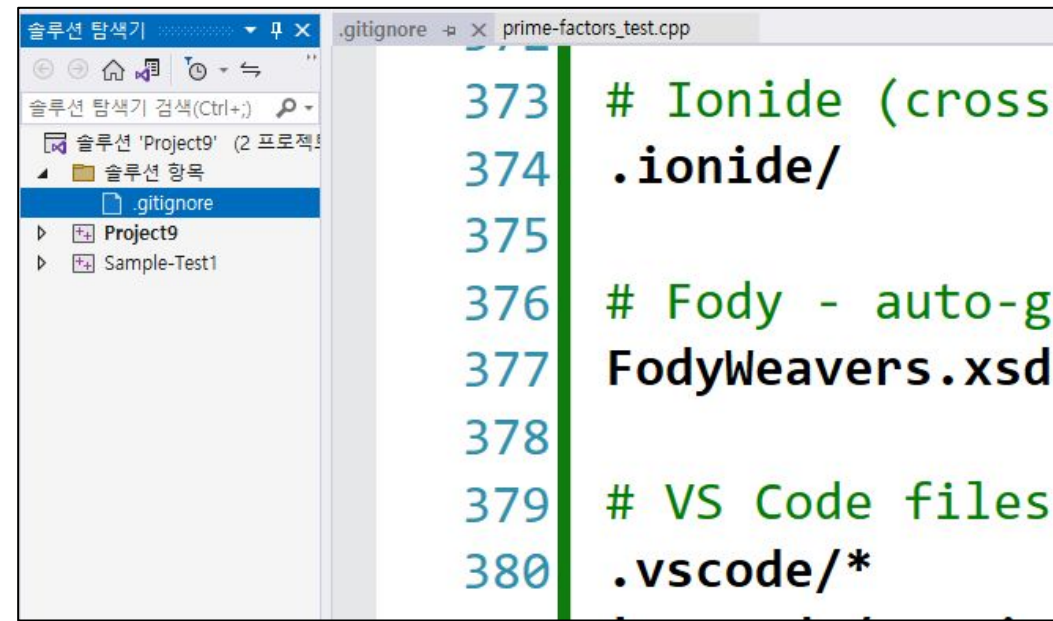
.gitignore 추가

✓ 구글링 : github gitignore 검색 후, visual studio 의 .gitignore



The screenshot shows the GitHub repository page for 'github/gitignore'. The repository is public and has 397 pull requests. The main branch is 'main'. The repository description is 'n0099 [VisualStudio.gitignore] remove a trailing space'. It has 165 contributors. The repository size is 6.7 KB. The code is displayed in a light blue theme with line numbers 1 through 8 visible. The code content is as follows:

```
1 ## Ignore Visual Studio temporary files, build results, and
2 ## files generated by popular Visual Studio add-ons.
3 ##
4 ## Get latest from https://github.com/github/gitignore/blob/main/VisualS
5
6 # User-specific files
7 *.rsuser
8 *.suo
```



The screenshot shows a Visual Studio code editor with a .gitignore file open. The file content is as follows:

```
373 # Ionide (cross
374 .ionide/
375
376 # Fody - auto-g
377 FodyWeavers.xsd
378
379 # VS Code files
380 .vscode/*
```

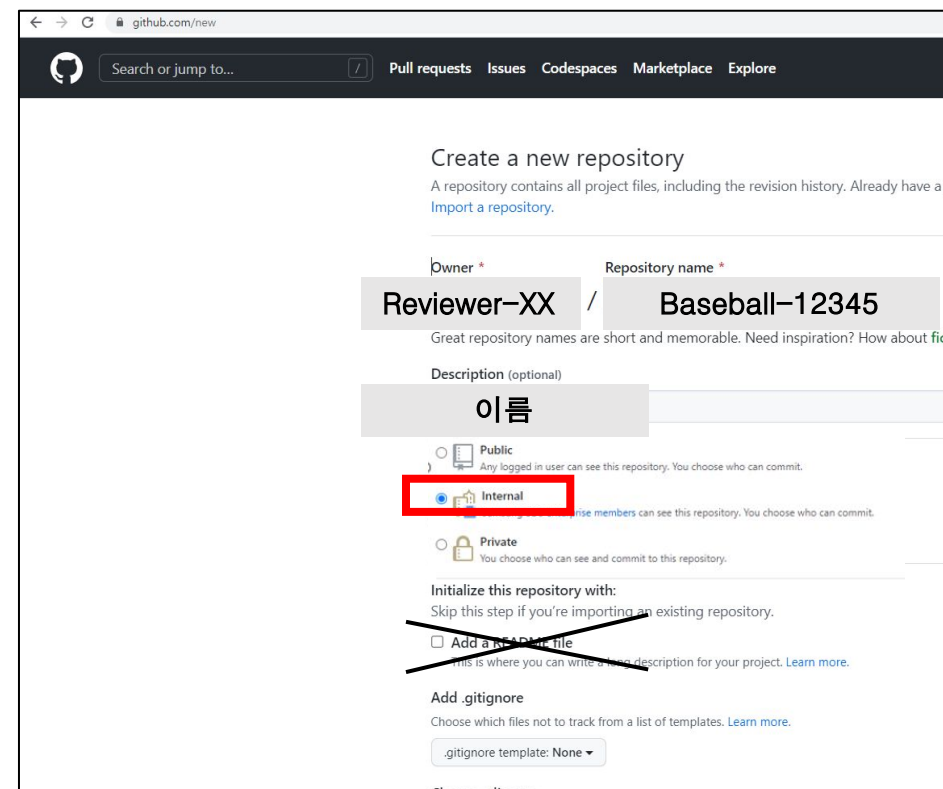

Remote Repository 생성

✓ Github Repository 생성하기

- Repo. name : Baseball-번호
- Description : 이름
- Internal 선택
- README.md 파일 생성 안함

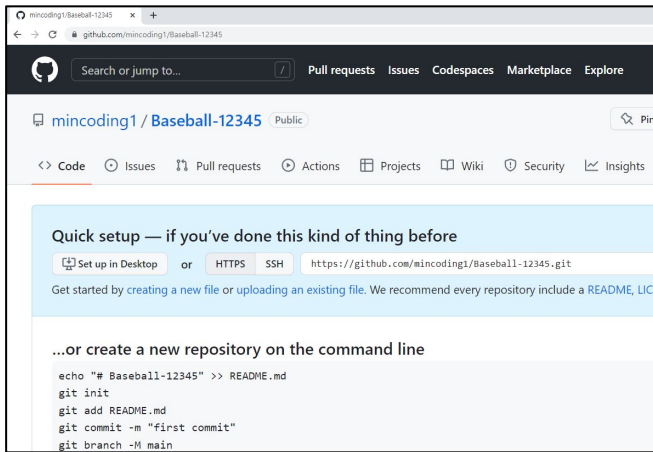
✓ 생성 후 설정

- Setting > Collaborator : 팀원 추가하기

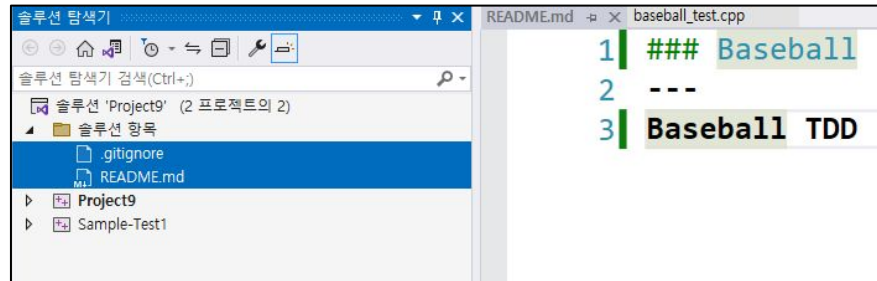


Commit 진행하기

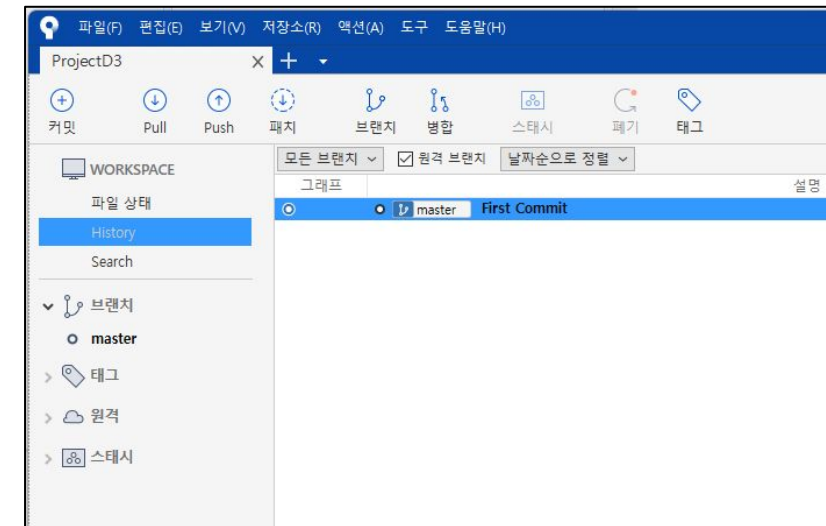
✓ Git GUI 도구를 이용하여 Commit 진행



Repo 생성



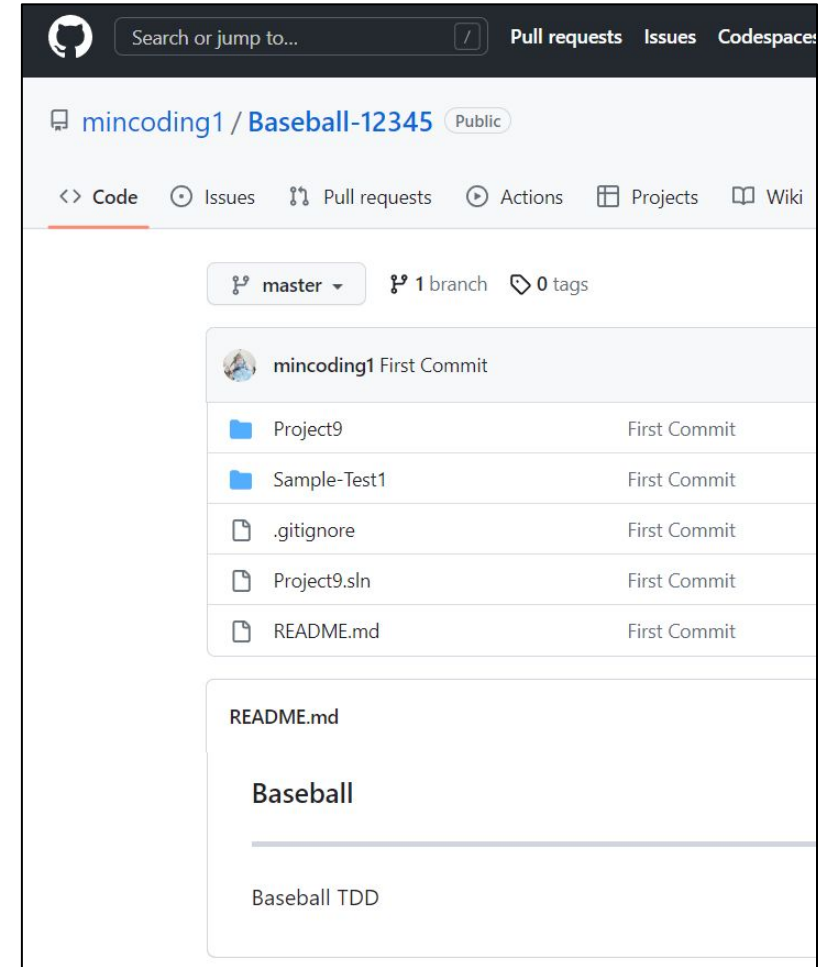
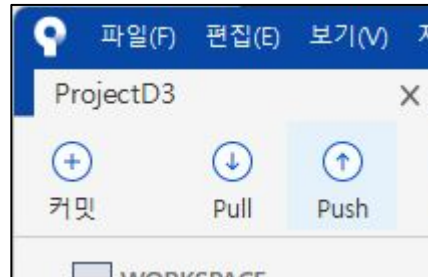
README.md 파일 생성



Commit 진행

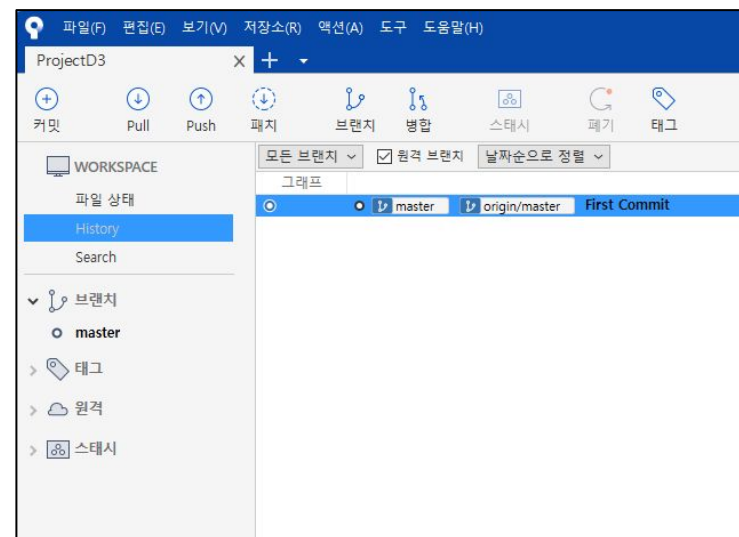
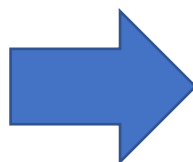
Push 진행

Baseball 작업내용
첫 Push



Github 에서 Branch 생성하기

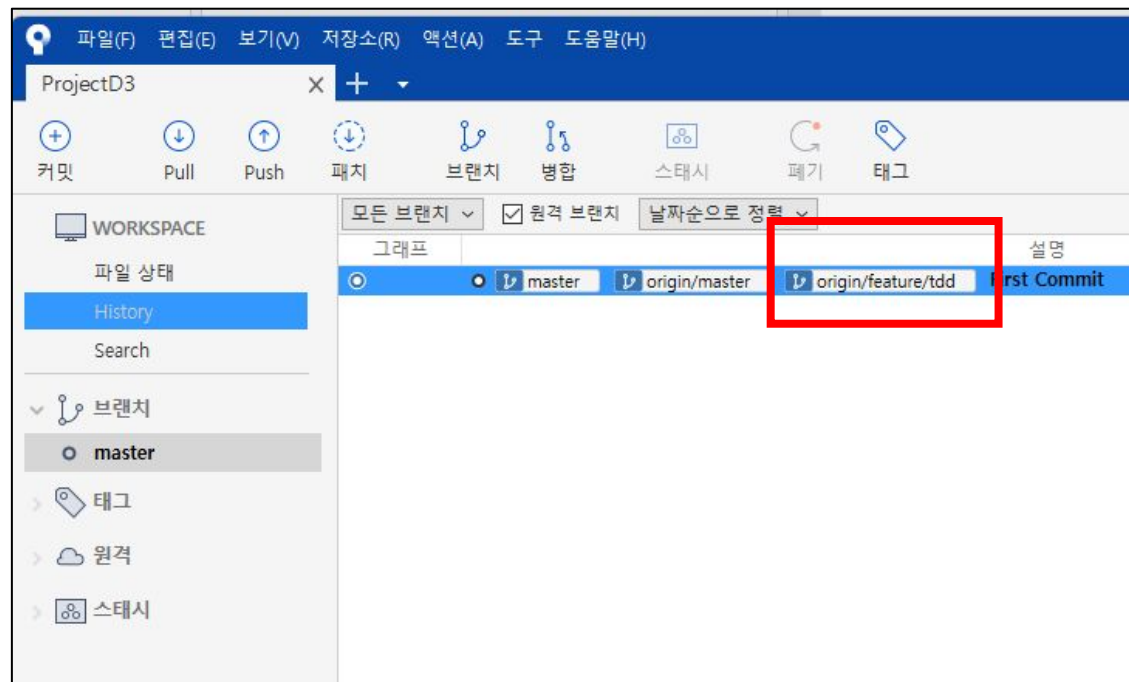
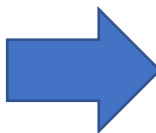
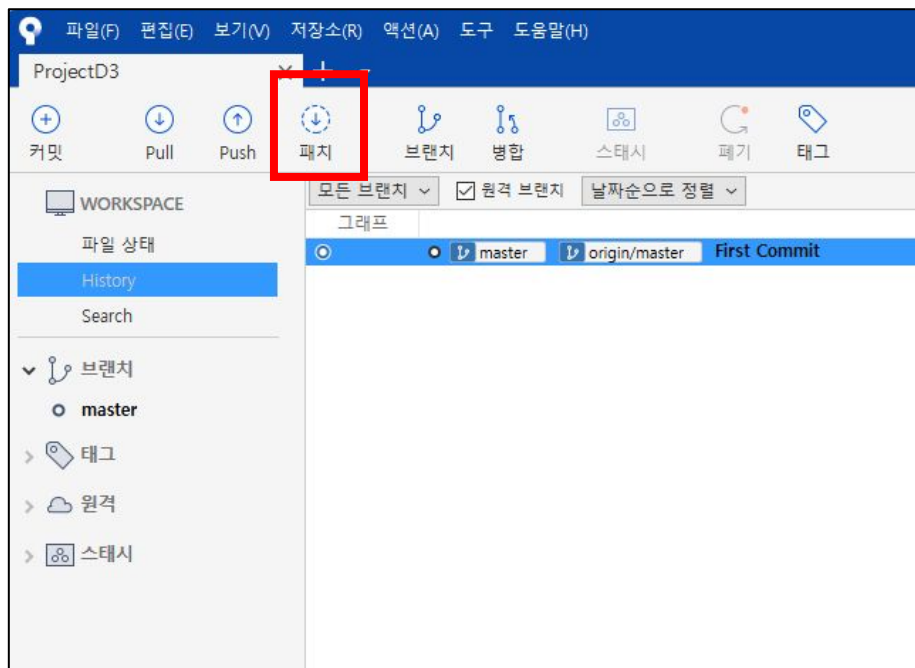
✓ github에서 branch 생성



아직까지 Local 은
Remote 에 새로운 Branch가 생성됨을 모른다!

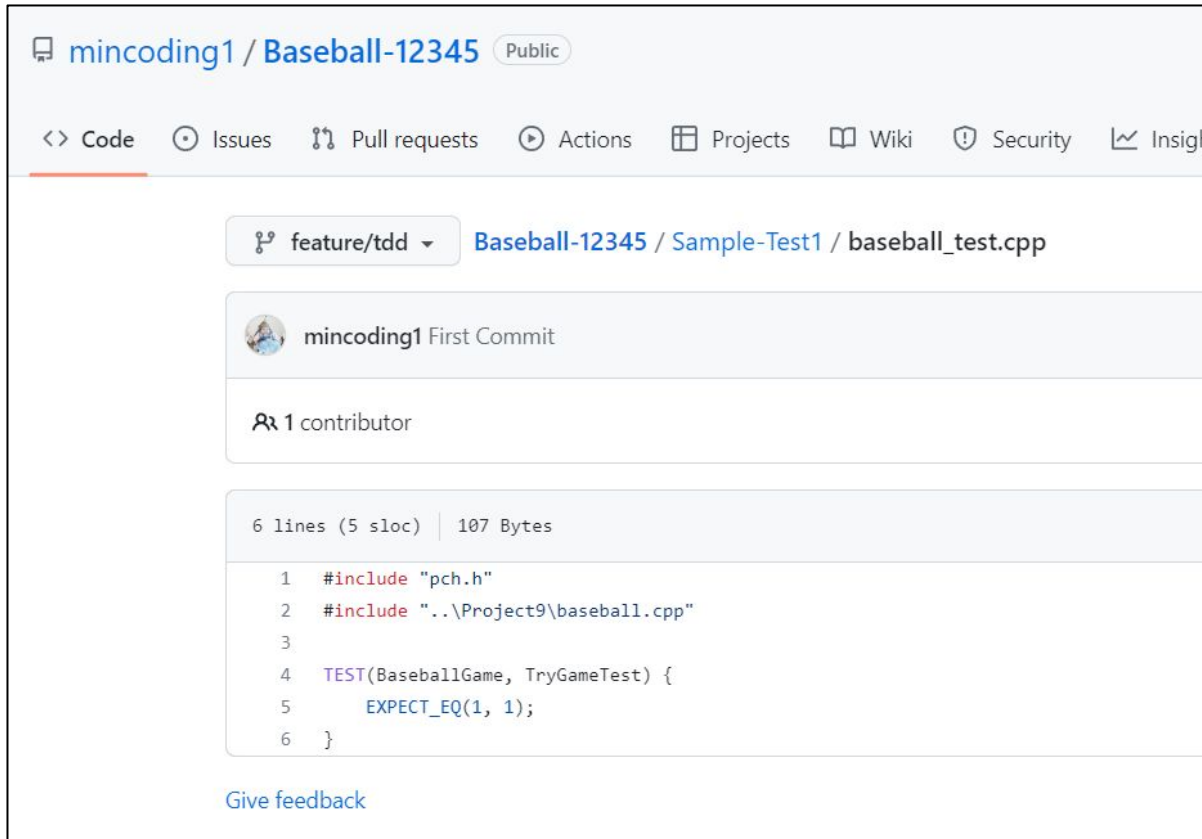
Fetch

✓ Upstream (Github) 으로부터 최신 정보를 가져오는 명령



Github에서 Commit 수행

feature/tdd 에서 Commit 1회 진행



mincoding1 / Baseball-12345 Public

<> Code Issues Pull requests Actions Projects Wiki Security Insights

feature/tdd Baseball-12345 / Sample-Test1 / baseball_test.cpp

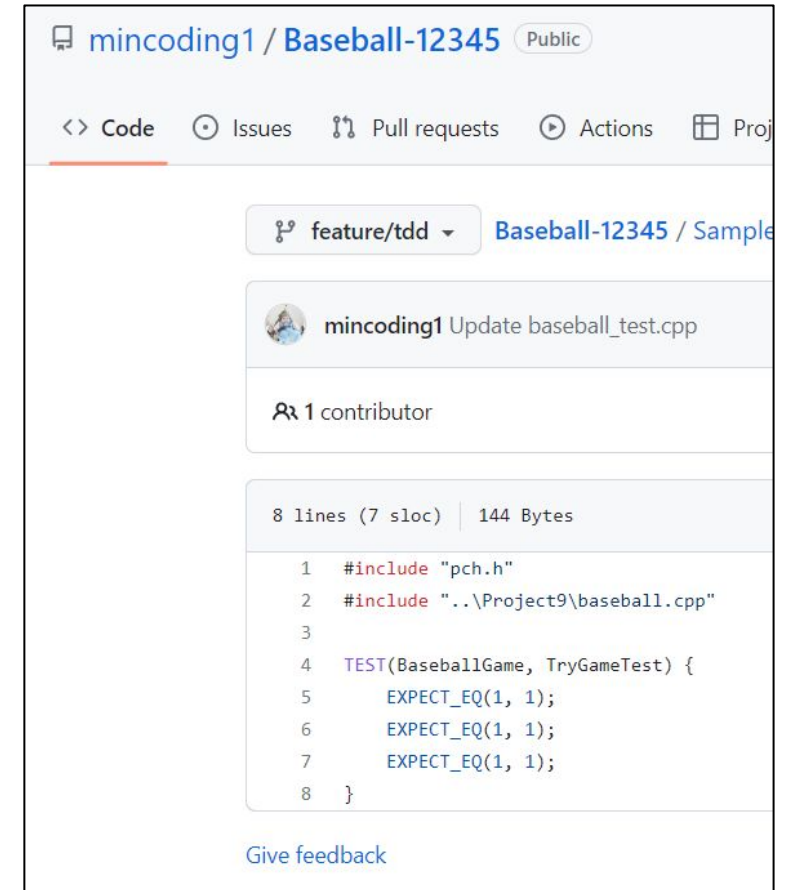
mincoding1 First Commit

1 contributor

6 lines (5 sloc) | 107 Bytes

```
1 #include "pch.h"
2 #include "..\Project9\baseball.cpp"
3
4 TEST(BaseballGame, TryGameTest) {
5     EXPECT_EQ(1, 1);
6 }
```

[Give feedback](#)



mincoding1 / Baseball-12345 Public

<> Code Issues Pull requests Actions Projects

feature/tdd Baseball-12345 / Sample-Test1 / baseball_test.cpp

mincoding1 Update baseball_test.cpp

1 contributor

8 lines (7 sloc) | 144 Bytes

```
1 #include "pch.h"
2 #include "..\Project9\baseball.cpp"
3
4 TEST(BaseballGame, TryGameTest) {
5     EXPECT_EQ(1, 1);
6     EXPECT_EQ(1, 1);
7     EXPECT_EQ(1, 1);
8 }
```

[Give feedback](#)

Fetch & Checkout

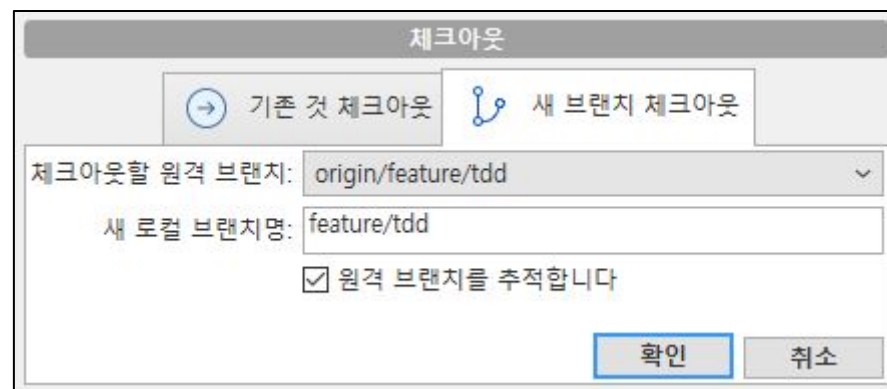
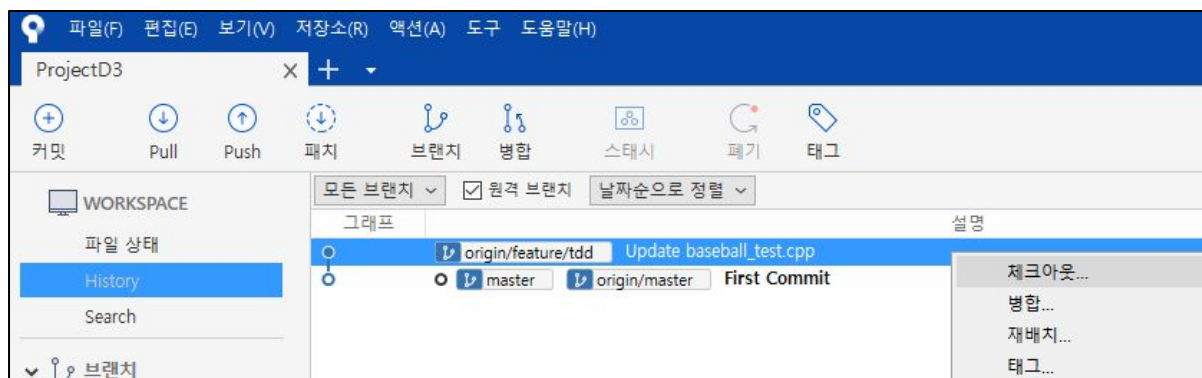
✓ 현재 브랜치 현황

▪ Local

- master

▪ Remote

- origin/master
- origin/feature/tdd



Checkout 결과

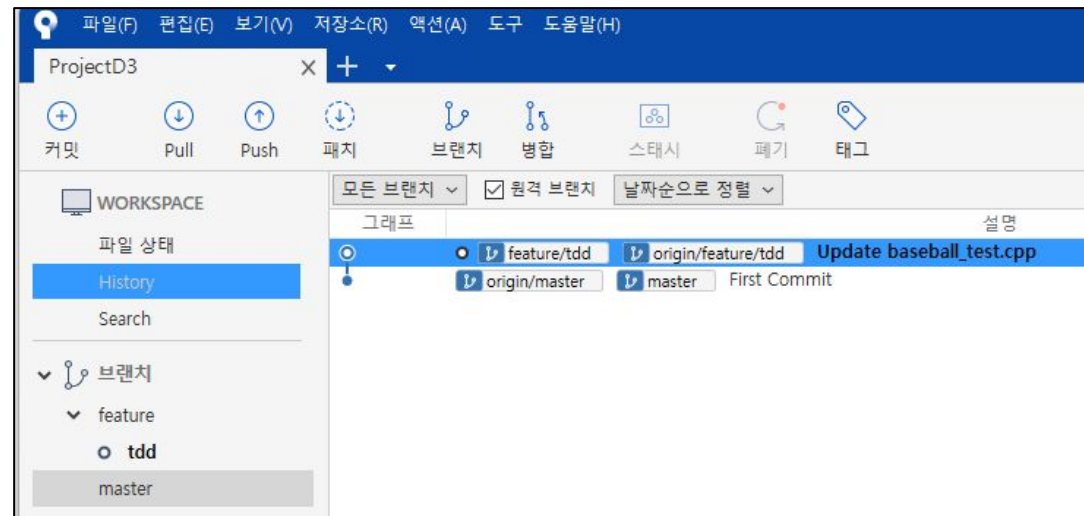
✓ Checkout 이후 브랜치 현황

▪ Local

- master
- **feature/tdd**

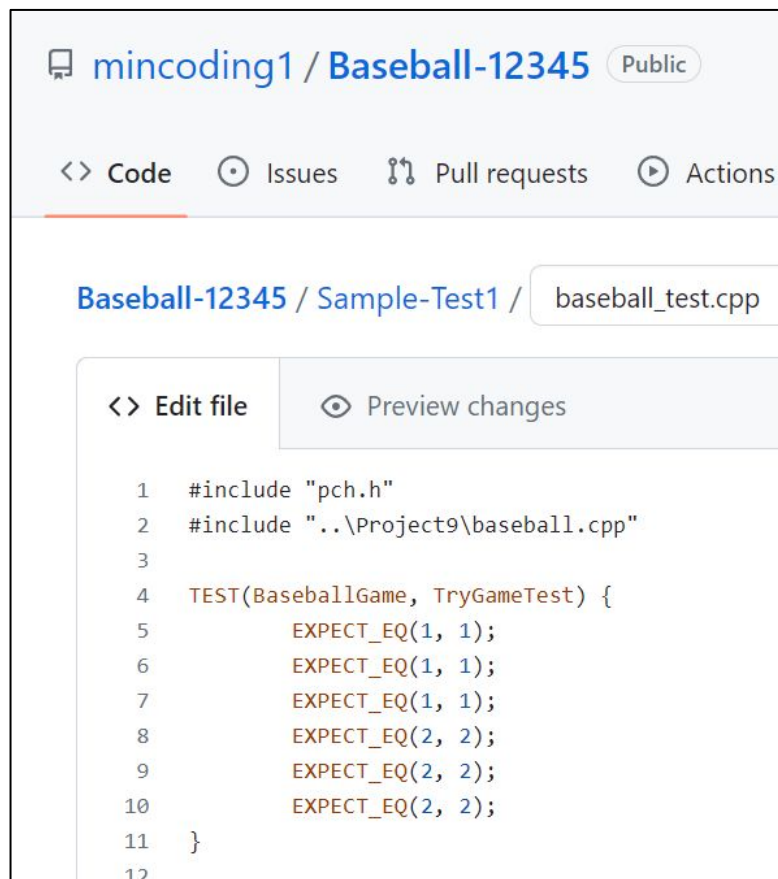
▪ Remote

- origin/master
- **origin/feature/tdd**



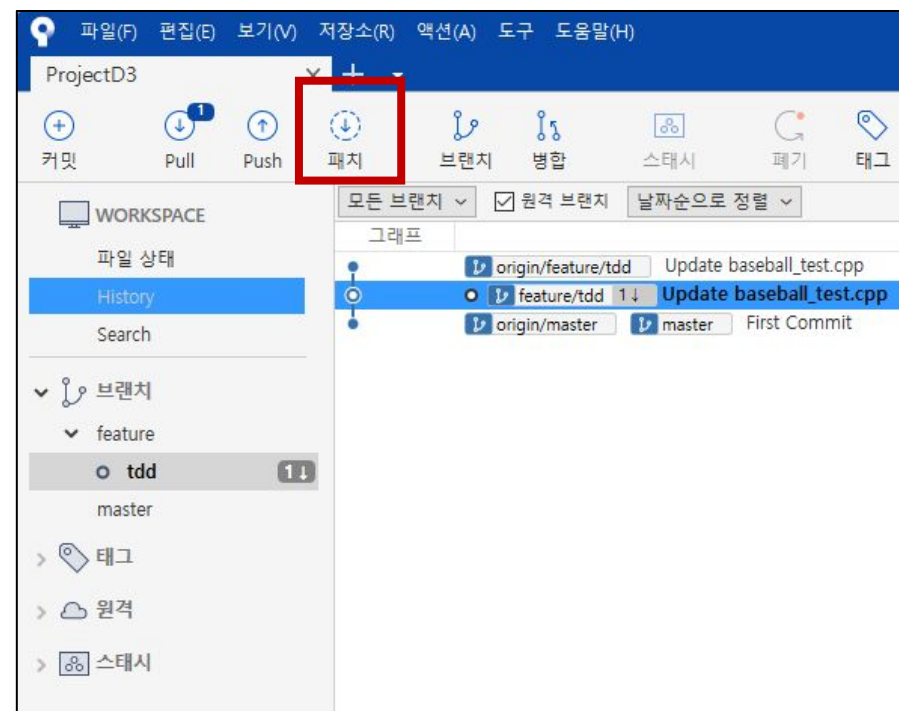
Github 에서 한번 더 Commit 수행

✓ Github 에서 Commit 후, Fetch 로 정보를 가져온다.



The screenshot shows the GitHub web interface for a repository named 'mincoding1 / Baseball-12345'. The repository is public. The 'Code' tab is selected, showing the file 'baseball_test.cpp' in the 'Sample-Test1' directory. The file content is displayed in a code editor with line numbers 1 through 12. The code includes a header file 'pch.h' and a test function 'TEST(BaseballGame, TryGameTest)' with several 'EXPECT_EQ' assertions.

```
1 #include "pch.h"
2 #include "..\Project9\baseball.cpp"
3
4 TEST(BaseballGame, TryGameTest) {
5     EXPECT_EQ(1, 1);
6     EXPECT_EQ(1, 1);
7     EXPECT_EQ(1, 1);
8     EXPECT_EQ(2, 2);
9     EXPECT_EQ(2, 2);
10    EXPECT_EQ(2, 2);
11 }
12
```



Pull 진행

- ✓ origin/feature/tdd 의 신규 변경사항을,
로컬의 feature/tdd 브랜치로 가져온다.

Pull

원격 저장소에서 가져오기: origin
https://github.com/mincoding1/Baseball-12345

가져오기 위한 원격 브랜치 feature/tdd 새로고침

로컬 브랜치에서 가져오기 feature/tdd

옵션

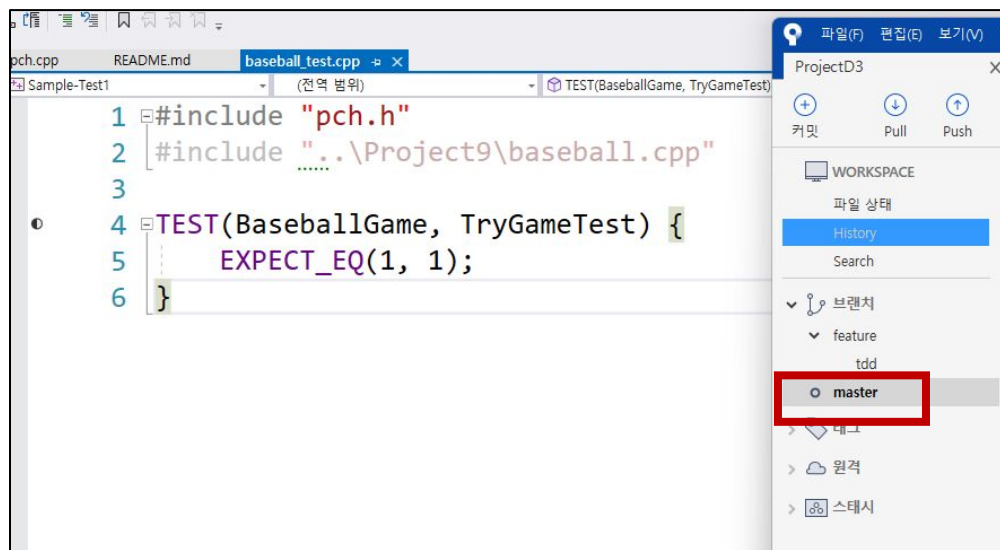
☒ 충돌이 없으면 묶어서 바로 커밋
☐ 병합 커밋에 있는 메시지를 첨부하세요
☐ fast-forward가 가능해도 새 커밋으로 생성
☐ 병합 대신 재배포 (경고 : 무시되지 않은 변경 사항이 있는지 확인하십시오)

Pull

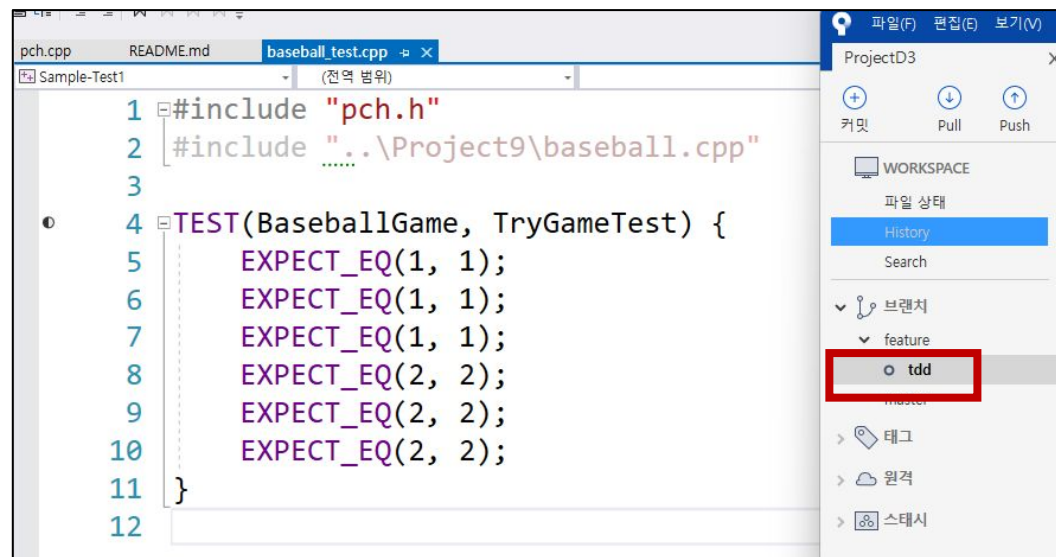
취소

Checkout 테스트

- ✓ 브랜치를 더블클릭시,
소스코드가 변경되는지 확인



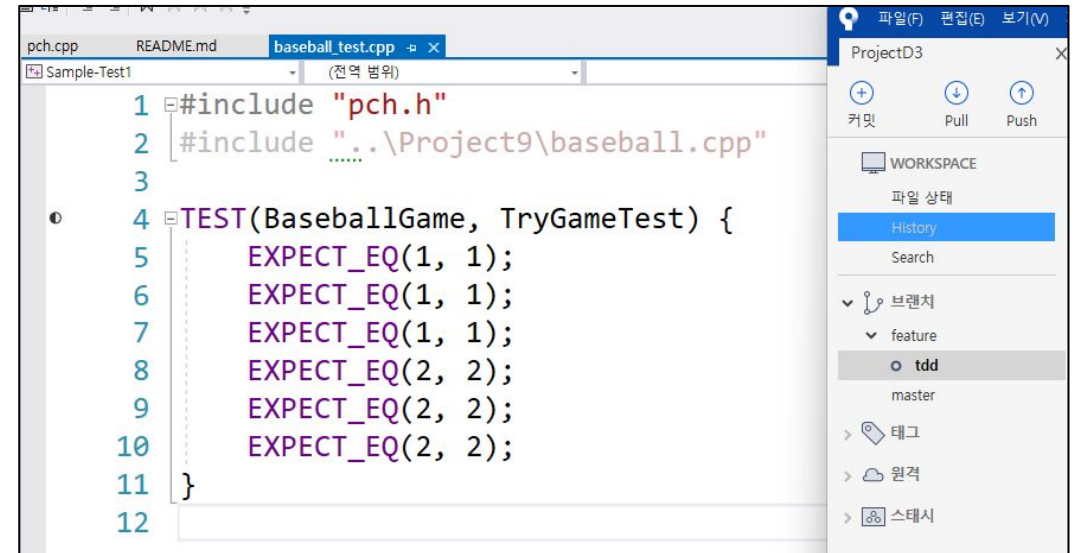
체크아웃 후, Visual Studio 쪽으로 마우스 클릭



체크아웃 후, Visual Studio 쪽으로 마우스 클릭

feature/tdd 브랜치에서 개발 시작

- ✓ 해당 Branch에서 개발 후,
Push 진행
- ✓ 그리고
Github에서 PR 요청 후 Merge 진행



The screenshot shows a code editor with a file named `baseball_test.cpp` open. The code contains a test function `TEST(BaseballGame, TryGameTest)` with five `EXPECT_EQ` assertions. The sidebar on the right shows the project structure for `ProjectD3`, with the `tdd` branch selected under the `feature` directory.

```
1 #include "pch.h"
2 #include "..\Project9\baseball.cpp"
3
4 TEST(BaseballGame, TryGameTest) {
5     EXPECT_EQ(1, 1);
6     EXPECT_EQ(1, 1);
7     EXPECT_EQ(1, 1);
8     EXPECT_EQ(2, 2);
9     EXPECT_EQ(2, 2);
10    EXPECT_EQ(2, 2);
11 }
12
```

ProjectD3

커밋 Pull Push

WORKSPACE

파일 상태

History

Search

브랜치

- feature
 - tdd
- master

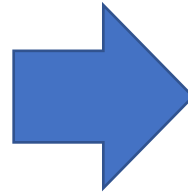
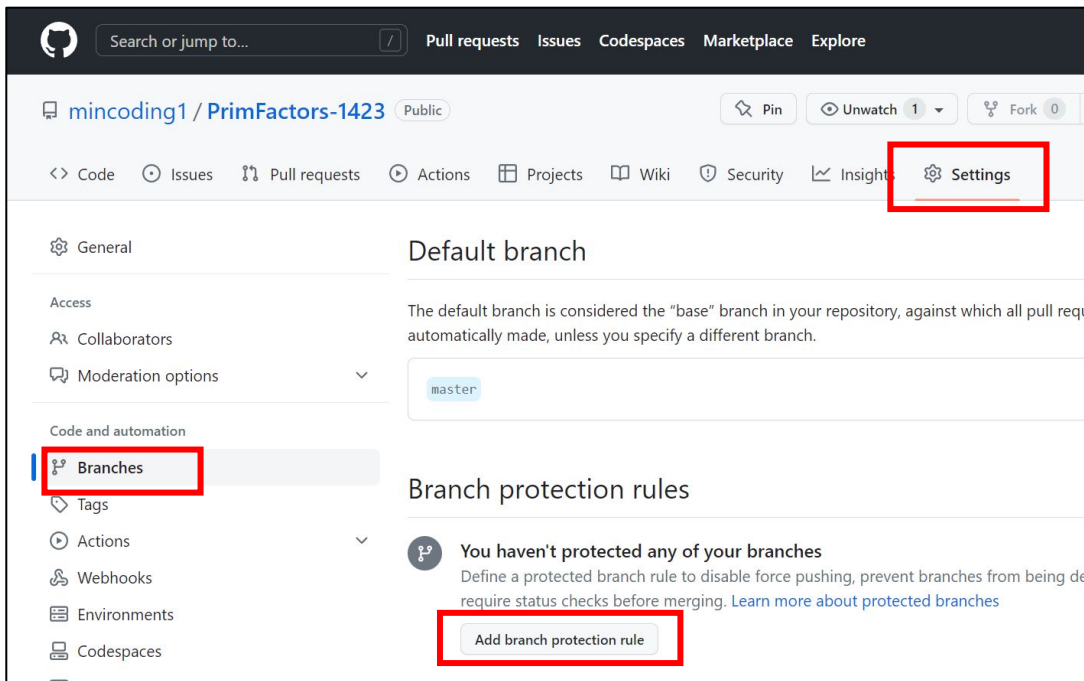
태그

원격

스태시

Github 설정하기

- ✓ Branch Protection Rule
 - Approve 1인 이상 필수



Branch name pattern *

master

Protect matching branches

- ☒ **Require a pull request before merging**
When enabled, all commits must be made to a non-protected branch into a branch that matches this rule.
- ☒ **Require approvals**
When enabled, pull requests targeting a matching branch require a specified number of approvals before they can be merged.

Required number of approvals before merging: 1 ▼

Create

실습 과정 구성

- ✓ 본 실습 과정은 총 8개의 TDD Cycle Step 이 있다.
- ✓ 각 Step 은 다음과 같이 나누어져 있다.

1. Red
2. Green
3. Refactor

프로젝트 제출 방법

✓ Commit 타이밍

1. Red 단계에서는 Commit 하지 않는다.
2. Green 모두 수행 후, Unit Test Pass 시 **Commit** 을 한다.
3. Refactor 모두 수행 후, Unit Test 를 수행하며, Pass 시 **Commit**을 한다.

✓ Commit 메시지 헤더 Format

- Red 단계 : Commit 하지 않는다. (동작되는 코드만 Commit 해야 하기 때문)
- Green 단계 : **[feature] page 번호**
- Refactor 단계 : **[refactoring] page 번호**

✓ Push 하여 최종 제출하기

- 8 단계 까지 완료 이후, Push 1회만 진행

Baseball TDD 개발

Baseball Game 실습 (01 / 04)

✓ Red : ToDo 작성

- guess("12")를 넣었을때, 세 글자가 아니므로 Exception 이 발생해야만 한다.

```
TEST(BaseballGame, ThrowExceptionWhenInputLengthIsUnmached) {  
    Baseball game;  
    EXPECT_THROW(game.guess(string("12")), length_error);  
}
```

메서드 생성하기

- ✓ guess를 클릭하고, Alt + Enter를 누른다.
- ✓ 메서드를 생성한다.

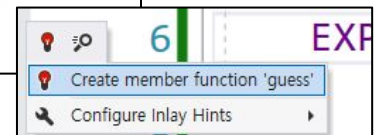
```
#include <stdexcept>

using namespace std;
class Baseball {
    ...
};
```

baseball.cpp 에 수기로 입력한다.

```
TEST(BaseballGame, ThrowException)
{
    Baseball game;
    EXPECT_THROW(game.guess("string"), ...);
}
```

guess 에 커서를 두고
Alt + Enter 누른다.



Pass 될 정도로 구현하기

- ✓ Exception을 추가하고, Test 결과를 확인한다.
 - Green 단계 완료, Commit 을 수행한다.

```
#include <stdexcept>

using namespace std;
class Baseball {
public:
    void guess(const string& string) {
        throw length_error("Must be three letters.");
    }
};
```

▲ ✓ BaseballGame (1 test)	Success
✓ ThrowExceptionWhenInputLengthIsUnmached	Success

코드를 더 범용적으로 리팩토링

- ✓ length가 세 글자가 아닐때만 Exception 발생
 - Refactor 단계 완료, Commit 을 수행한다.

```
class Baseball {  
public:  
    void guess(const string& string) {  
        if (string.length() != 3) {  
            throw length_error("Must be three letters.");  
        }  
    }  
};
```

▲ ✓ BaseballGame (1 test)	Success
✓ ThrowExceptionWhenInputLengthIsUnmached	Success

Baseball Game 실습 (02 / 04)

✓ Red : ToDo 작성

- guess("12s")를 넣었을때, 숫자가 아닌 다른 문자가 있으므로 Exception 이 발생해야만 한다.

```
TEST(BaseballGame, ThrowExceptionWhenInvalidChar) {  
    Baseball game;  
    EXPECT_THROW(game.guess(string("12s")), invalid_argument);  
}
```

BaseballGame (2 tests)	Failed: 1 test failed
✓ ThrowExceptionWhenInputLengthIsUnmached	Success
✗ ThrowExceptionWhenInvalidChar	Failed

Pass 될 정도로 구현하기

- ✓ 0 ~ 9 문자가 아니면, Exception 발생
 - Green 단계 완료, Commit 을 수행한다.

```
class Baseball {  
public:  
    void guess(const string& string) {  
        if (string.length() != 3) {  
            throw length_error("Must be three letters.");  
        }  
  
        for (char ch : string) {  
            if (ch < '0' || ch > '9') {  
                throw invalid_argument("Mus be number");  
            }  
        }  
    }  
};
```

✓ BaseballGame (2 tests)	Success
✓ ThrowExceptionWhenInputLengthIsUnmached	Success
✓ ThrowExceptionWhenInvalidChar	Success

Refactoring

- ✓ 중복 코드를 제거하기 위한 리팩토링 사전 세팅
 - try ~ catch 문으로 변경

```
TEST(BaseballGame, ThrowExceptionWhenInvalidChar) {  
    Baseball game;  
    EXPECT_THROW(game.guess(string("12s")), invalid_argument);  
}
```



```
TEST(BaseballGame, ThrowExceptionWhenInvalidChar) {  
    Baseball game;  
    try {  
        game.guess(string("12s"));  
        FAIL();  
    }  
    catch (exception e) {  
        //PASS  
    }  
}
```

Test Fixture 도입

- ✓ 중복코드를 제거하기 위한 Test Fixture를 추가한다.
 - guess 수행 후 Exception 이 발생했는지 확인하는
assertIllegalArgument 전역 메서드 생성

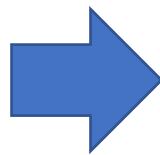
```
class BaseballFixture : public testing::Test {  
public:  
    Baseball game;  
    void assertIllegalArgument(string guessNumber) {  
        //game.guess() 수행 후, Exception이 발생해야 PASS 이다.  
    }  
};
```


Test Fixture 완성하기

- ✓ 기존 Try ~ Catch문을 복사하여, 해당 메서드를 완성한다.

```
class BaseballFixture : public testing::Test {
public:
    Baseball game;
    void assertIllegalArgument(string guessNumber) {
        //game.guess() 수행 후, Exception이 발생해야 PASS 이다.
    }
};
```

```
TEST(BaseballGame, ThrowExceptionWhenInvalidChar) {
    Baseball game;
    try {
        game.guess(string("12s"));
        FAIL();
    }
    catch (exception e) {
        //PASS
    }
}
```



```
class BaseballFixture : public testing::Test {
public:
    Baseball game;
    void assertIllegalArgument(string guessNumber) {
        try {
            game.guess(guessNumber);
            FAIL();
        }
        catch (exception e) {
            //PASS
        }
    }
};
```

Test Fixture 적용하기

✓ 모든 테스트 케이스에, Test Fixture를 적용한다.

```
TEST(BaseballGame, ThrowExceptionWhenInputLengthIsUnmached) {  
    Baseball game;  
    EXPECT_THROW(game.guess(string("12")), length_error);  
}  
  
TEST(BaseballGame, ThrowExceptionWhenInvalidChar) {  
    Baseball game;  
    try {  
        game.guess(string("12s"));  
        FAIL();  
    }  
    catch (exception e) {  
        //PASS  
    }  
}
```



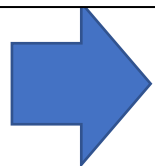
```
TEST_F(BaseballFixture, ThrowExceptionWhenInputLengthIsUnmached) {  
    assertIllegalArgument("12");  
}  
  
TEST_F(BaseballFixture, ThrowExceptionWhenInvalidChar) {  
    assertIllegalArgument("12s");  
}
```

✓ BaseballFixture (2 tests)	Success
✓ ThrowExceptionWhenInputLengthIsUnmached	Success
✓ ThrowExceptionWhenInvalidChar	Success

Test Fixture 함수, 하나로 만들기

- ✓ 2 개의 Test 함수를
1 개의 Test 함수로 합친다.

```
TEST_F(BaseballFixture, ThrowExceptionWhenInputLengthIsUnmached) {  
    assertIllegalArgument("12");  
}  
  
TEST_F(BaseballFixture, ThrowExceptionWhenInvalidChar) {  
    assertIllegalArgument("12s");  
}
```



```
TEST_F(BaseballFixture, ThrowExceptionWhenInvalidCase) {  
    assertIllegalArgument("12");  
    assertIllegalArgument("12s");  
}
```

✓ BaseballFixture (1 test)	Success
✓ ThrowExceptionWhenInvalidCase	Success

지금까지 완성된 코드

```
#include "pch.h"
#include "../Project13/baseball.cpp"

class BaseballFixture : public testing::Test {
public:
    Baseball game;
    void assertIllegalArgument(string guessNumber) {
        try {
            game.guess(guessNumber);
            FAIL();
        }
        catch (exception e) {
            //PASS
        }
    }
};

TEST_F(BaseballFixture, ThrowExceptionWhenInvalidCase) {
    assertIllegalArgument("12");
    assertIllegalArgument("12s");
}
```

baseball_test.cpp

```
#include <stdexcept>
using namespace std;

class Baseball {
public:
    void guess(const string& string) {
        if (string.length() != 3) {
            throw length_error("Must be three letters.");
        }

        for (char ch : string) {
            if (ch < '0' || ch > '9') {
                throw invalid_argument("Mus be number");
            }
        }
    }
};
```

baseball.cpp

Parameter 이름 변경하기

- ✓ “string” □ “guessNumber” 로 이름 변경
 - Refactor 단계 완료, Commit 을 수행한다.

```
#include <stdexcept>
using namespace std;

class Baseball {
public:
    void guess(const string& guessNumber) {
        if (guessNumber.length() != 3) {
            throw length_error("Must be three letters.");
        }

        for (char ch : guessNumber) {
            if (ch < '0' || ch > '9') {
                throw invalid_argument("Mus be number");
            }
        }
    }
};
```

▲ ✓ BaseballFixture (1 test)	Success
✓ ThrowExceptionWhenInvalidCase	Success

Baseball Game 실습 (03 / 04)

✓ Red : ToDo 작성

- guess("121") 에서, 중복된 숫자가 존재하므로 Exception 이 발생해야만 한다.

```
TEST_F(BaseballFixture, ThrowExceptionWhenInvalidCase) {  
    assertIllegalArgument("12");  
    assertIllegalArgument("12s");  
    assertIllegalArgument("121");  
}
```

BaseballFixture (1 test)	Failed: 1 test failed
ThrowExceptionWhenInvalidCase	Failed

Pass 될 정도로 구현하기

- ✓ 같은 숫자가 존재한다면, Exception 발생
 - Green 단계 완료, Commit 을 수행한다.

```
class Baseball {
public:
    void guess(const string& guessNumber) {
        if (guessNumber.length() != 3) {
            throw length_error("Must be three letters.");
        }

        for (char ch : guessNumber) {
            if (ch < '0' || ch > '9') {
                throw invalid_argument("Mus be number");
            }
        }


        if (guessNumber[0] == guessNumber[1]
            || guessNumber[0] == guessNumber[2]
            || guessNumber[1] == guessNumber[2]) {
            throw invalid_argument("Must not have the same number");
        }
    }
};
```

✓ BaseballFixture (1 test)	Success
✓ ThrowExceptionWhenInvalidCase	Success

메서드 추출하기

- ✓ 수식의 내용을 설명하도록, 메서드 추상화하기

```
if (guessNumber[0] == guessNumber[1]
    || guessNumber[0] == guessNumber[2]
    || guessNumber[1] == guessNumber[2]) {
    throw invalid_argument("Must not have the same number");
}
```

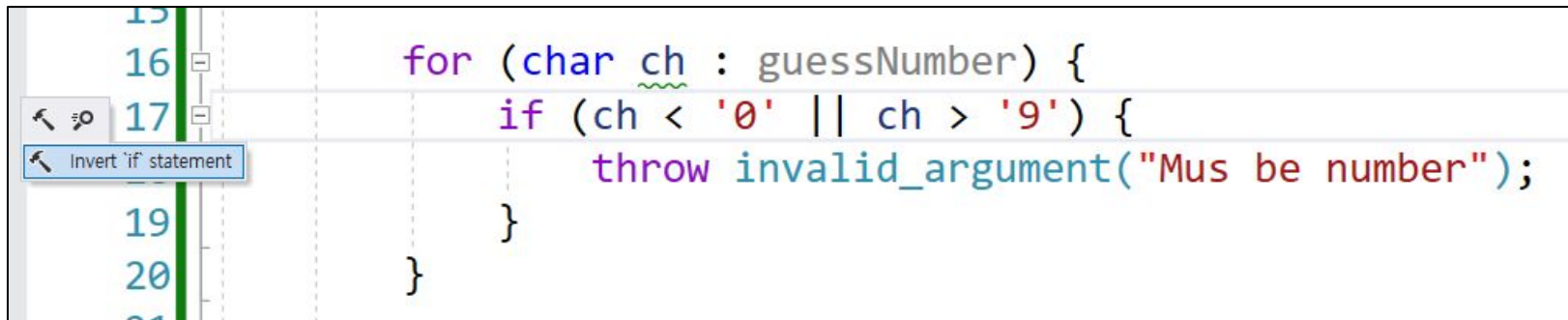


```
if (isDuplicatedNumber(guessNumber)) {
    throw invalid_argument("Must not have the same number");
}
```

✓ BaseballFixture (1 test)	Success
✓ ThrowExceptionWhenInvalidCase	Success

Invert if statement

- ✓ if 에 커서를 두고, Alt + Enter를 누른다.
- ✓ Invert If Statement로, 더 가독성이 좋은 if문으로 변경한다.



```
15  
16 for (char ch : guessNumber) {  
17     if (ch < '0' || ch > '9') {  
18         throw invalid_argument("Mus be number");  
19     }  
20 }
```



```
for (char ch : guessNumber) {  
    if (ch >= '0' && ch <= '9') continue;  
    throw invalid_argument("Mus be number");  
}
```

✓ BaseballFixture (1 test)	Success
✓ ThrowExceptionWhenInvalidCase	Success

메서드 추출하기

- ✓ guess 함수의 모든 내용을 블록 잡고, 메서드 추상화하기
 - Refactor 단계 완료, Commit 을 수행한다.

```
void guess(const string& guessNumber) {  
    if (guessNumber.length() != 3) {  
        throw length_error("Must be three letters.");  
    }  
  
    for (char ch : guessNumber) {  
        if (ch >= '0' && ch <= '9') continue;  
        throw invalid_argument("Must be number");  
    }  
  
    if (isDuplicatedNumber(guessNumber)) {  
        throw invalid_argument("Must not have the same number");  
    }  
}
```



```
void guess(const string& guessNumber) {  
    assertIllegalArgument(guessNumber);  
}
```

▲ ✓ BaseballFixture (1 test)	Success
✓ ThrowExceptionWhenInvalidCase	Success

Baseball Game 실습 (04 / 04)

✓ 지금까지 구현한 내용

- Invalid 한 Parameter가 입력되었을 때 처리하는 코드

✓ 이제부터 구현해야 할 내용

- 정상 Parameter가 입력 되었을 때, 동작하는 코드 구현

ToDo 작성하기

- ✓ 123이 정답일 때,
123을 시도하면 Solved : True, 3 Strikes, 3 Ball 값이 나와야 한다.

```
TEST_F(BaseballFixture, ReturnSolvedResultIfMatchedNumber) {  
    Baseball game("123");  
    GuessResult result = game.guess("123");  
  
    EXPECT_TRUE(result.solved);  
    EXPECT_EQ(3, result.strikes);  
    EXPECT_EQ(0, result.balls);  
}
```

결과 데이터 struct 추가

결과 데이터를 한 곳에 모은
struct를 추가한다.

```
#include <stdexcept>
using namespace std;

struct GuessResult {
    bool solved;
    int strikes;
    int balls;
};

class Baseball {
public:
    bool isDuplicatedNumber(const
        return guessNumber[0] =
            || guessNumber[0] =
            || guessNumber[1] =
    }
}
```

리턴 값 하드코딩

- ✓ ToDo 에 Check되도록
일단 무조건 정답 결과를 return 하도록 코드를 추가한다.

```
void guess(const string& guessNumber) {  
    assertIllegalArgument(guessNumber);  
}
```



```
GuessResult guess(const string& guessNumber) {  
    assertIllegalArgument(guessNumber);  
    return { true, 3, 0 };  
}
```

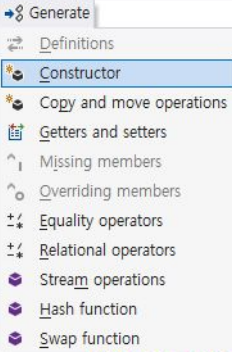

생성자 추가하기

- ✓ question : 문제의 정답을 저장할 필드 추가.
- ✓ private 필드 추가 후, 생성자 추가하기

```
GuessResult guess(const string& question, const int guessNumber[]) {  
    assertIllegalArgument(guessNumber, 3);  
    return { true, 3, 0 };  
}  
  
private:  
    string question;  
};
```

baseball.cpp 하단에,
private 코드 추가하기

```
class Baseball {  
public:  
  
private:  
    string question;  
};
```



Generate
Definitions
Constructor
Copy and move operations
Getters and setters
Missing members
Overriding members
Equality operators
Relational operators
Stream operations
Hash function
Swap function

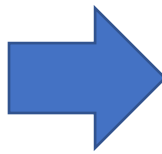
Baseball 클래스
생성자 추가 (Alt + Insert)

```
class Baseball {  
public:  
    explicit Baseball(const string& question)  
        : question(question) {}  
  
    bool isDuplicatedNumber(const string& guessNumber)  
    {  
        return guessNumber[0] == guessNumber[1]  
            || guessNumber[0] == guessNumber[2]  
            || guessNumber[1] == guessNumber[2];  
    }  
};
```


Fixture 코드 수정하기

- ✓ Unit Test 코드 > Test Fixture에서 객체 생성시 생성자 호출하도록 코드 수정.

```
class BaseballFixture : public testing
public:
    Baseball game;
    void assertIllegalArgument(string
        try {
            game.guess(guessNumber);
            FAIL();
        }
        catch (exception e) {
            //PASS
        }
    }
};
```



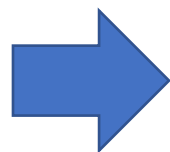
```
class BaseballFixture : public testing:
public:
    Baseball game{ "123" };
    void assertIllegalArgument(string g
        try {
            game.guess(guessNumber);
            FAIL();
        }
        catch (exception e) {
            //PASS
        }
    }
};
```

Fixture 에서는 game("123") 초기화 대신,
Uniform 초기화 { } 를 써야한다.

객체 생성 코드 삭제

- ✓ Fixture 에서 인스턴스를 생성하므로,
Unit Test 에서 객체 생성 코드를 삭제한다.

```
TEST_F(BaseballFixture, ReturnSolvedResultIfMatchedNumber) {  
    Baseball game("123");  
    GuessResult result = game.guess("123");  
  
    EXPECT_TRUE(result.solved);  
    EXPECT_EQ(3, result.strikes);  
    EXPECT_EQ(0, result.balls);  
}
```



```
TEST_F(BaseballFixture, ReturnSolvedResultIfMatchedNumber) {  
    GuessResult result = game.guess("123");  
  
    EXPECT_TRUE(result.solved);  
    EXPECT_EQ(3, result.strikes);  
    EXPECT_EQ(0, result.balls);  
}
```

▲ ✓ BaseballFixture (2 tests)	Success
✓ ReturnSolvedResultIfMatchedNumber	Success
✓ ThrowExceptionWhenInvalidCase	Success

범용적인 코드로 리팩토링

- ✓ 정답을 맞추었을 때,
{true, 3, 0} 값을 리턴하도록 리팩토링

```
GuessResult guess(const string& guessNumber) {  
    assertIllegalArgument(guessNumber);  
    return { true, 3, 0 };  
}
```



```
GuessResult guess(const string& guessNumber) {  
    assertIllegalArgument(guessNumber);  
    if (guessNumber == question) {  
        return { true, 3, 0 };  
    }  
  
    return {false, 0, 0};  
}
```

지금까지 완성된 소스코드

```
#include "pch.h"
#include "../Project13/baseball.cpp"

class BaseballFixture : public testing::Test {
public:
    Baseball game{ "123" };
    void assertIllegalArgument(string guessNumber) {
        try {
            game.guess(guessNumber);
            FAIL();
        }
        catch (exception e) {
            //PASS
        }
    }
};

TEST_F(BaseballFixture, ThrowExceptionWhenInvalidCase) {
    assertIllegalArgument("12");
    assertIllegalArgument("12s");
    assertIllegalArgument("121");
}

TEST_F(BaseballFixture, ReturnSolvedResultIfMatchedNumber) {
    GuessResult result = game.guess("123");

    EXPECT_TRUE(result.solved);
    EXPECT_EQ(3, result.strikes);
    EXPECT_EQ(0, result.balls);
}
```

baseball_test.cpp

```
#include <stdexcept>
using namespace std;

struct GuessResult {
    bool solved;
    int strikes;
    int balls;
};

class Baseball {
public:
    explicit Baseball(const string& question)
        : question(question) {}

    bool isDuplicatedNumber(const string& guessNumber) {
        return guessNumber[0] == guessNumber[1]
            || guessNumber[0] == guessNumber[2]
            || guessNumber[1] == guessNumber[2];
    }

    void assertIllegalArgument(const string& guessNumber) {
        if (guessNumber.length() != 3) {
            throw length_error("Must be three letters.");
        }

        for (char ch : guessNumber) {
            if (ch >= '0' && ch <= '9') continue;
            throw invalid_argument("Mus be number");
        }

        if (isDuplicatedNumber(guessNumber)) {
            throw invalid_argument("Must not have the same number");
        }
    }

    GuessResult guess(const string& guessNumber) {
        assertIllegalArgument(guessNumber);
        if (guessNumber == question) {
            return { true, 3, 0 };
        }

        return { false, 0, 0 };
    }

private:
    string question;
};
```

baseball.cpp

[도전] TDD 구현하기

1. 2 Strikes, 0 Ball 테스트 코드 추가 후,
Strikes를 구하는 코드 개발하기
2. 1 Stirkes, 2 Ball 테스트 코드 추가 후,
Balls을 구하는 코드 구현