

# CIENT Git Tutorial

Git Tutorial

# Git을 배우기 전에

Git을 배우기 전에

# Git?

git:

# Git이란?

- ❖ 가장 인기 있는 버전 관리 시스템이다.
  - 말은 어려워 보이지만 30분이면 기본 기능들은 다 익힌다.
  - 프로그래머라면 익숙해져야 할 Tool이다!
    - 개발자들이 프로그램을 공유할 때 사용하는 소프트웨어이다.
  - 포트폴리오로도 사용할 수 있다!

# Git은 버전 관리 시스템

버전 관리 시스템

버전? Version?

버전 관리 시스템

# 버전(Version)

❖ 의미있는 변화들 - 이고잉

(예)

- 파일을 새로 만들었다 = 내 파일의 첫 번째 버전
- 자소서를 완성했다 = 내 자소서의 완성된 버전
- 새로운 버전인 Unity\_2020을 설치했다

# 파일들의

버전

의미있는 변화를

관리

관리해주는

시스템

소프트웨어

# Git이 왜 필요할까?

Git이 왜 필요할까?



- 직장 상사: 옛날 보고서 들고 와봐!
- 컨퍼를 요청

고객: 옛날 디자인으로 돌아갑시다  
나: 최종본 밖에 없음...





우리를 편하게 해줄 기능들

노티트 디자인 에토 사이트

Git 구경

Git 녹음

# Git을 쓴다면?

- 내가 작업한 것과  
다른 사람이 작업한 것을  
정리해서 볼 수 있다

Fix issue with reliable pipeline where when a packet was dropped and ...  
larus committed on 24 Apr 2019

Commits on Apr 16, 2019

Fix a small error in the readme  
timj-unity committed on 16 Apr 2019

New version of the transport with support for network pipelines  
timj-unity committed on 16 Apr 2019

Commits on Feb 20, 2019

Update workflow-client-server.md  
ArturoNereu committed on 20 Feb 2019

Commits on Oct 25, 2018

Documentation updates  
larus committed on 25 Oct 2018

Commits on Oct 24, 2018

Remove old documentation  
michalbrzozowski committed on 24 Oct 2018

Update workflow-client-server.md  
michalbrzozowski committed on 24 Oct 2018

Update README.md  
asyasmi authored and timj-unity committed on 23 Oct 2018

Update README.md  
michalbrzozowski committed on 24 Oct 2018

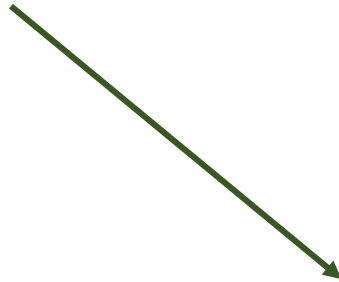
Update the documentation for the unity transport  
timj-unity committed on 24 Oct 2018

Commits on Oct 23, 2018

Initial preview release of the Unity Multiplayer packages and samples  
timj-unity committed on 23 Oct 2018

# Git을 쓴다면?

- 문서의 글자가 갑자기 다 깨진다
- 잘 돌아가던 프로그램이 갑자기 오류가 난다...  
그런데 해결을 못하겠다
- 어느 순간부터 파일 하나가 사라졌다



옛날의 버전으로 복구할 수 있다

# Git을 쓴다면?

- 달라진 부분을 확인할 수 있다
  - 누가 무엇을 수정했는지 보기 쉽다
  - 어느 순간부터 버그가 발생했는지 추적하기 쉽다

▼ 3

Syntax\_Analyzer/Program.cs

47

47

@@ -47,7 +47,8 @@ static void Main(string[] args)

48

48

while (true)

49

49

{

50

-

Develop\_Common.\_Debug(\$"\\nMyCurrentState: {(int)Stack.stack.GetCurrentState()}\\n MySpllterIsLeftTo: {splitter.Le

50

+

Develop\_Common.\_Debug(\$"MyCurrentState: {(int)Stack.stack.GetCurrentState()}\\n MySpllterIsLeftTo: {splitter.Le

51

+

Develop\_Common.\_Debug(\$"Making Decision: {Constants.\_Enum.SLR\_Table[(int)Stack.stack.GetCurrentState()], (int)Vari

51

52

52

53

Parser.Descision(Constants.\_Enum.SLR\_Table[(int)Stack.stack.GetCurrentState()],(int)Variables.parsingInput[splitter

53

54

if (Variables.parsingInput[0] == Constants.\_Enum.CFG\_NonTerminalAndTerminal.CODE && Variables.parsingInput[0] == C

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

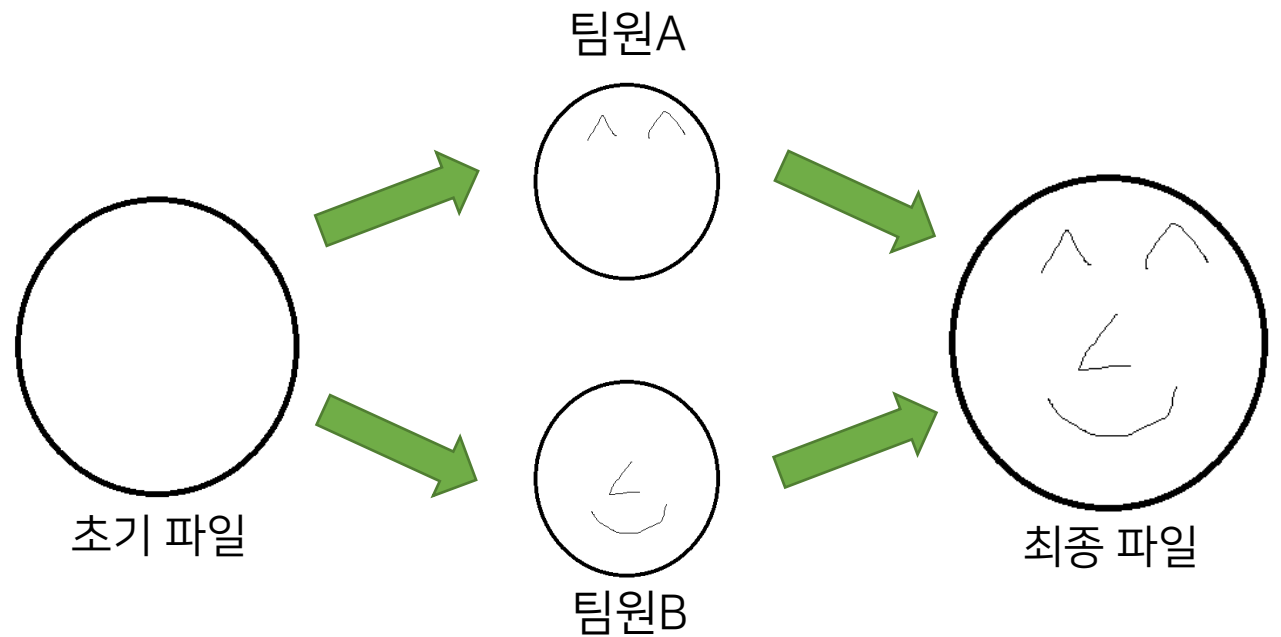
236

237

238

# Git을 쓴다면?

- 협업하기 쉽다
  - 같은 파일을 작업할 때
    - 팀원A. 눈 그리기
    - 팀원B. 코/입 그리기



**이제 Git을 배우자!**

이제 Git을 배우자!



README.md

Azure Pipelines succeeded

## Git - fast, scalable, distributed revision control system

Git is a fast, scalable, distributed revision control system with an unusually rich command set that provides both high-level operations and full access to internals.

Git is an Open Source project covered by the GNU General Public License version 2 (some parts of it are under different licenses compatible with the GPLv2). It was originally written by Linus Torvalds with help of a group of hackers around the net.

```
MINGW64/c:/Users/duryk/Desktop/git-practice
duryk@Juhyeong MINGW64 ~/Desktop
$ mkdir git-practice

duryk@Juhyeong MINGW64 ~/Desktop
$ cd git-practice/

duryk@Juhyeong MINGW64 ~/Desktop/git-practice
$ git init
Initialized empty Git repository in C:/Users/duryk/Desktop/git-practice/.git/

duryk@Juhyeong MINGW64 ~/Desktop/git-practice (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

기본적으로 Git은 풍부한 명령어들로 작동한다  
명령어 환경이 익숙하지 않다면..?

유용어 화음이 허공의 외로움이다...

# 2가지의 인터페이스

CLI(command line interface): git-bash

-> 명령어(command)

---

GUI(graphical user interface): Source Tree

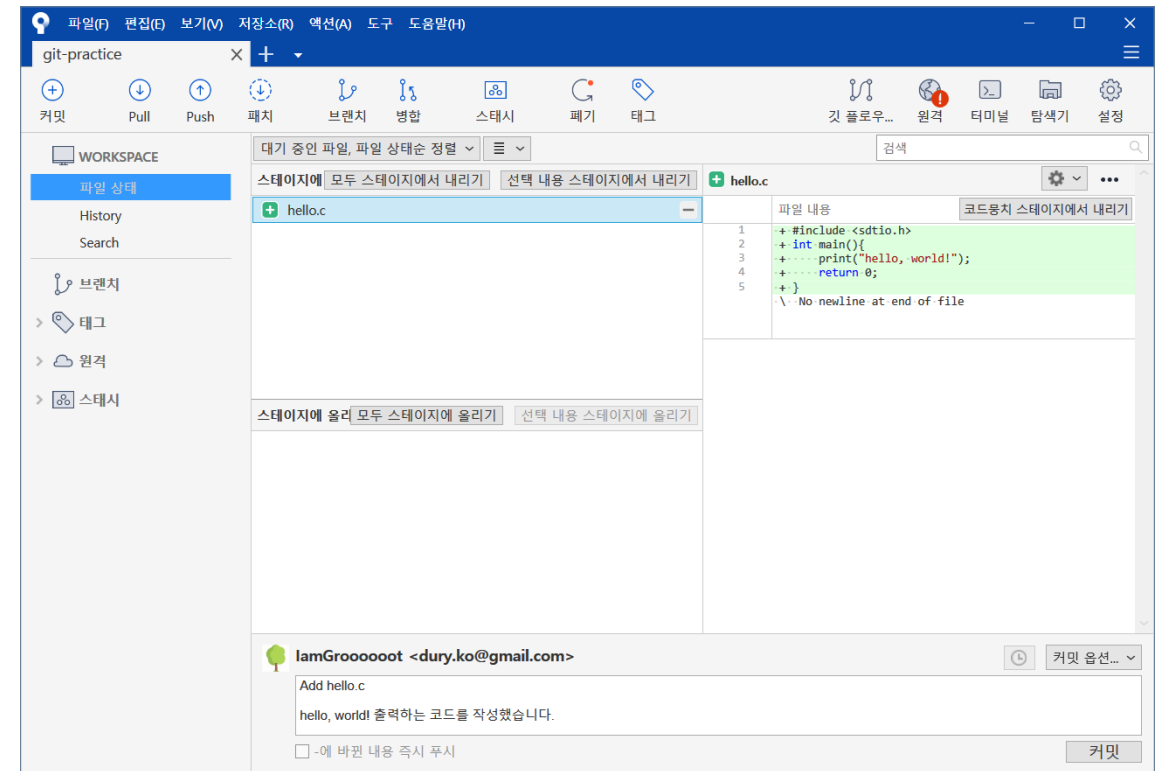
-> 이미지/아이콘 클릭

# CLI vs GUI

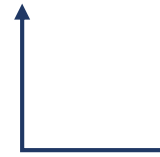
우리가 사용할 프로그램



```
~/git-practice
└─ cd git-practice
└─ echo "My name is Juhyeong."
My name is Juhyeong.
└─ echo "My name is Juhyeong." >> README.md
└─ git add .
└─ git commit -m "Add my name"
[master 92d0f82] Add my name
1 file changed, 1 insertion(+)
create mode 100644 README.md
└─
```



# 우리가 사용할 프로그램은 Source Tree



Git을 편하게 사용하기 위해 만든 GUI 프로그램!

Source Tree

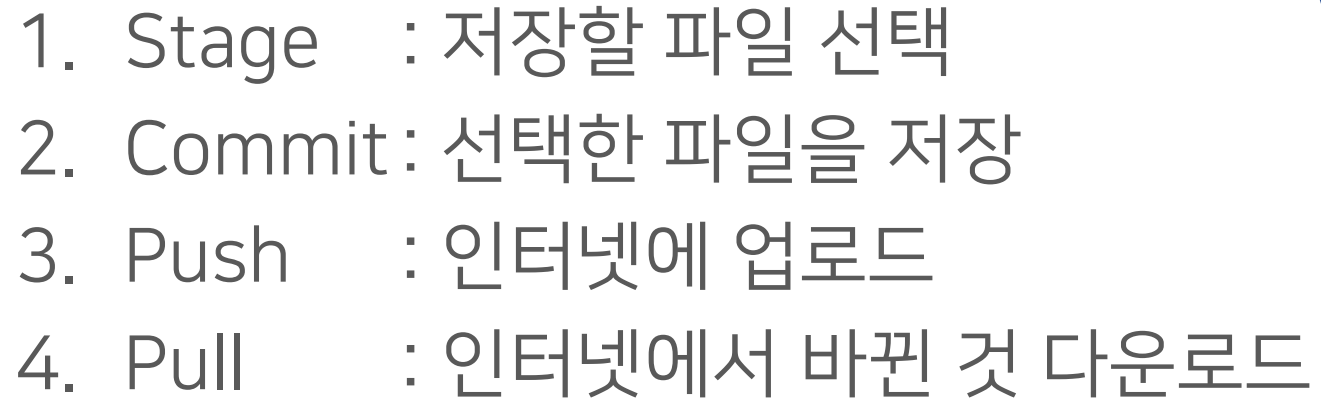
**Source Tree**를 사용해서  
버전 관리를 시작해봅시다

머지 도구를 익혀주세요

# 주로 사용하게 될 것은 4개의 기능

여기에 있는 기능들만 알아가도  
OK

## 0. 저장소 선언: Git으로 관리 시작

- 
1. Stage : 저장할 파일 선택
  2. Commit: 선택한 파일을 저장
  3. Push : 인터넷에 업로드
  4. Pull : 인터넷에서 바뀐 것 다운로드

5. Clone : 인터넷에서 저장소 전체 다운 받기

# + 문제가 발생할 때 사용할 기능

- 실수로 비밀번호를 기록했다
  - Git - Reset
- 실수로 파일을 삭제하고 올렸다
  - Git - Checkout해서 복원
- 오류가 있는 파일을 올렸다
  - Git - Revert or Reset
- 다른 사람이 짰 코드랑 내 코드랑 충돌이 나서 멘붕
  - 원하는 코드 선택해서 합친다 (Git - Conflict 해결)
- 공유한 저장소가 에러가 나면서 다운이 안된다
  - Git - Stash
- Git이 이상한 메시지를 뱉으면서 계속 실패했다고 하는데 나는 알아들을 수가 없다
  - 다 삭제하고 다시 복제하거나 구글링...

**주로 쓸 기능들부터 배워보자**

1. 이 문서의 내용을 살펴보고

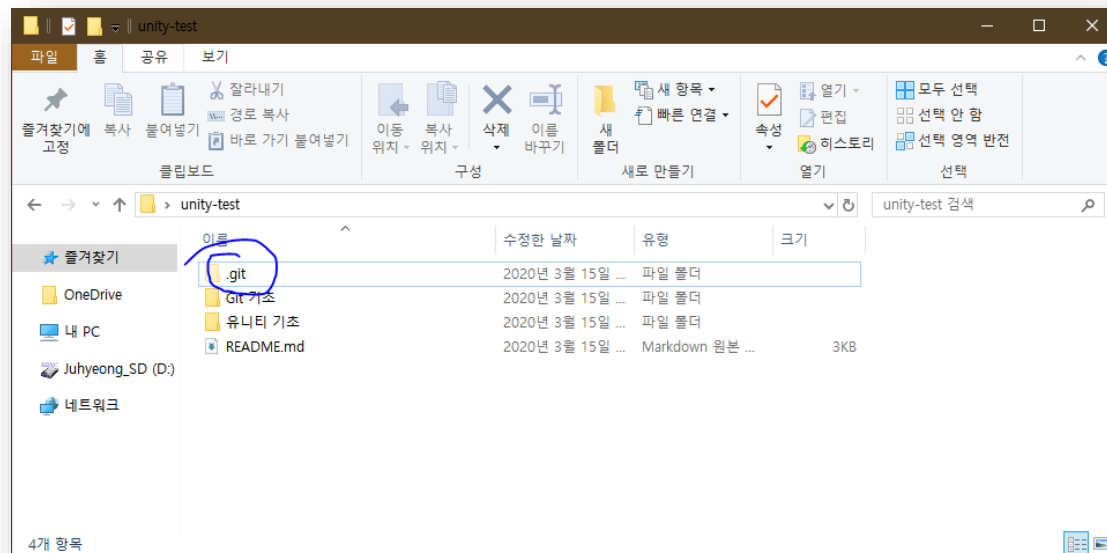


# Git 기본 용어

- 저장소(Repository)
  - 내가 Git으로 관리할 폴더
- 커밋(Commit)
  - Git에서 버전을 기록하는 단위
  - 커밋한다 = Git에 하나의 버전을 기록한다
- 로컬(Local) - 네트워크 없이 접속 가능한 것
  - 로컬 컴퓨터 = 내 컴퓨터
- 원격(Remote) - 네트워크 접속해서 접속해야 하는 것
  - 클라우드(Google Drive같은 것)
  - Git 저장소를 올려서 공유하는 곳
    - (ex) GitHub, Bitbucket

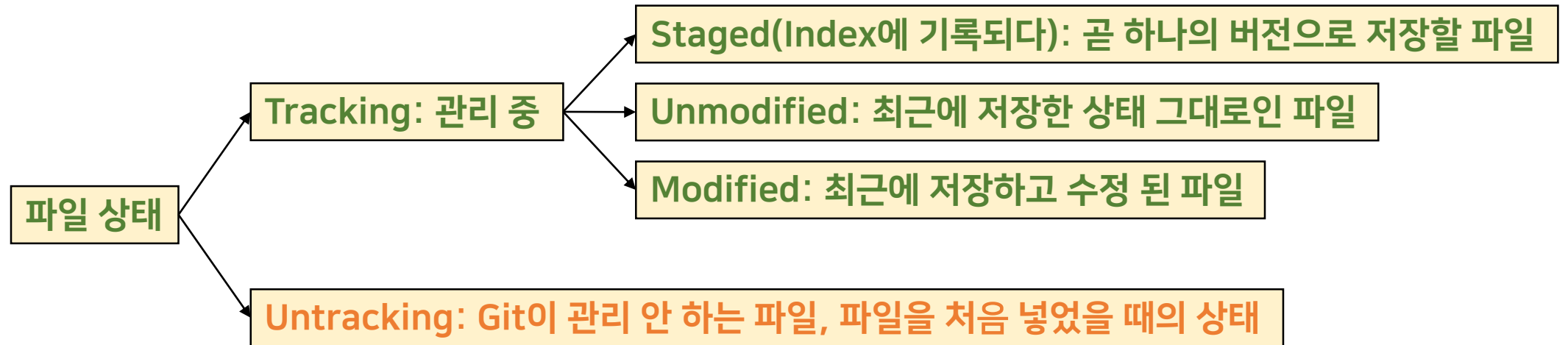
# 0. 저장소 선언하기

- 이 폴더를 Git이라는 툴로 관리하겠다는 것
- .git 숨김 폴더가 생겼는지 확인
  - 이 폴더에 우리의 버전들이 기록된다. 우리는 건들 일이 없다!
  - 이 파일을 삭제하면 우리가 Git으로 저장한 기록들이 다 날라가기 때문에 절대 삭제하면 안 된다.



# Workspace - 파일 상태

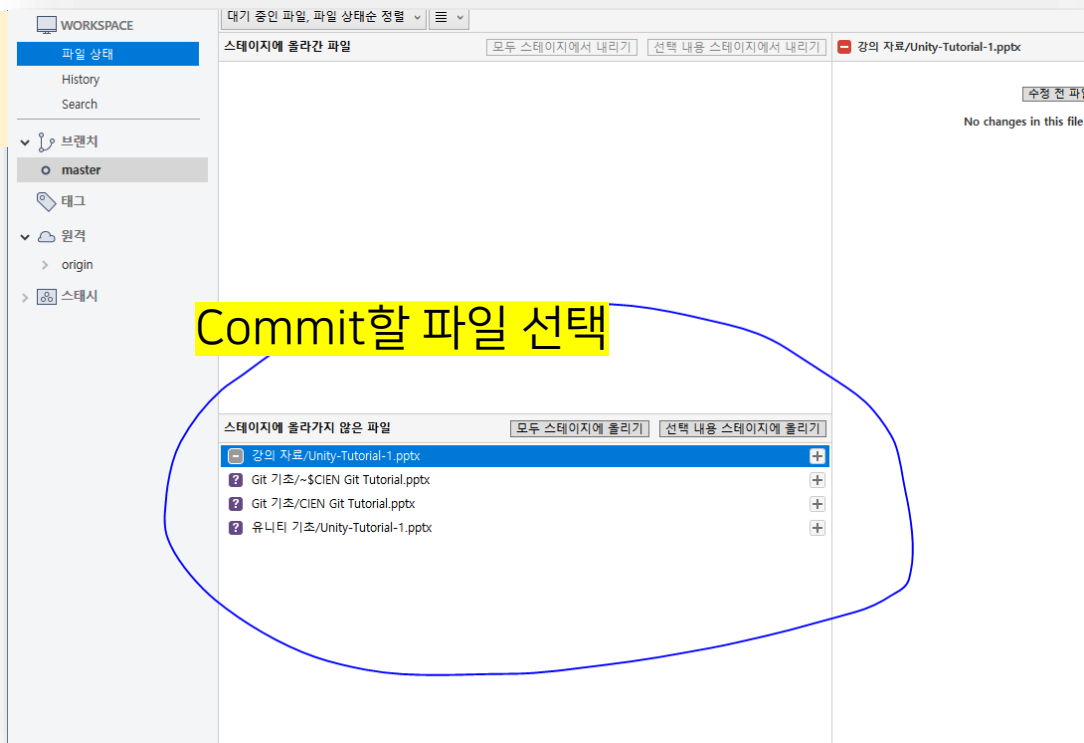
- 파일 상태: 내 파일의 상태를 확인할 수 있는 창
- 새로운 Text파일을 만들고 파일 상태를 확인해보자
- Git의 파일 상태 구분법



# 1. Stage에 올리기 - Commit할 파일 선택

- Commit하기 전 단계
- Commit 할 파일들 선택하기

Stage 전  
캡처



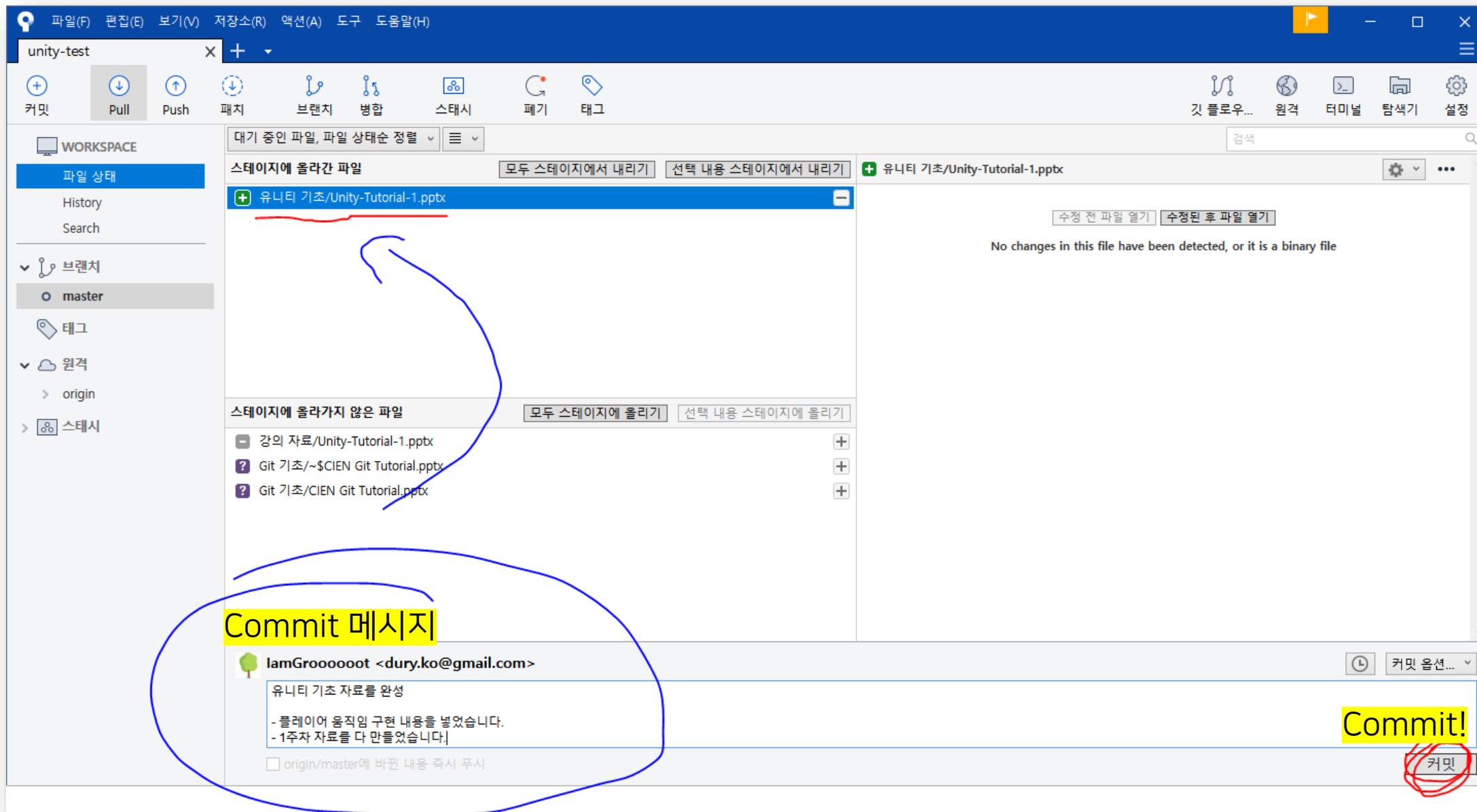
Stage 후  
캡처



## 2. Commit하기 - Git의 실질적인 기록

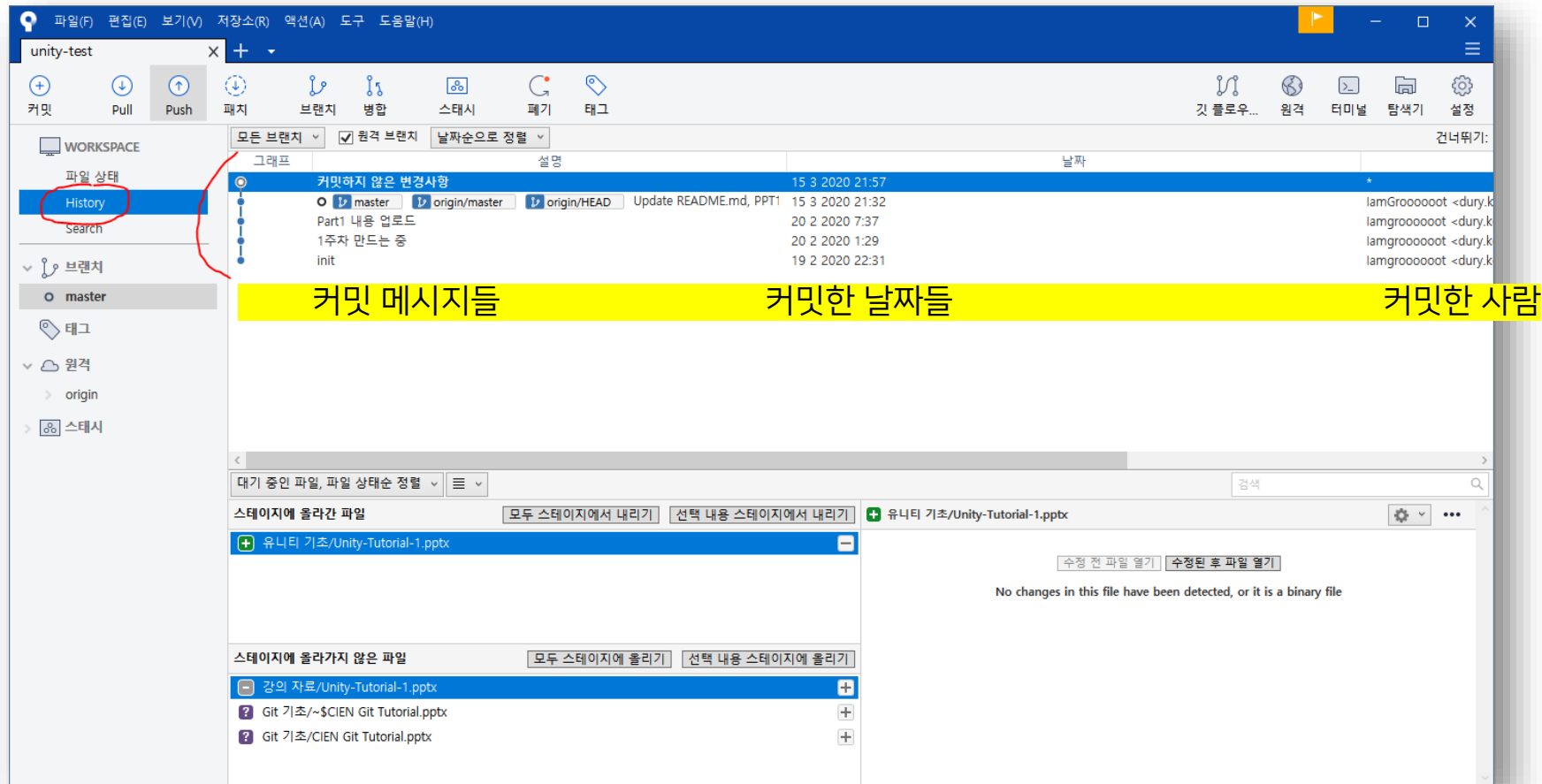
- 커밋한다 = 하나의 버전으로 저장한다
- 무조건 Stage(Index)에 올라간 파일들만 기록된다.
- 어떤 것이 변했는지 알 수 있도록 메시지를 남겨줘야 한다.

## • 메시지를 작성한 후 커밋을 클릭



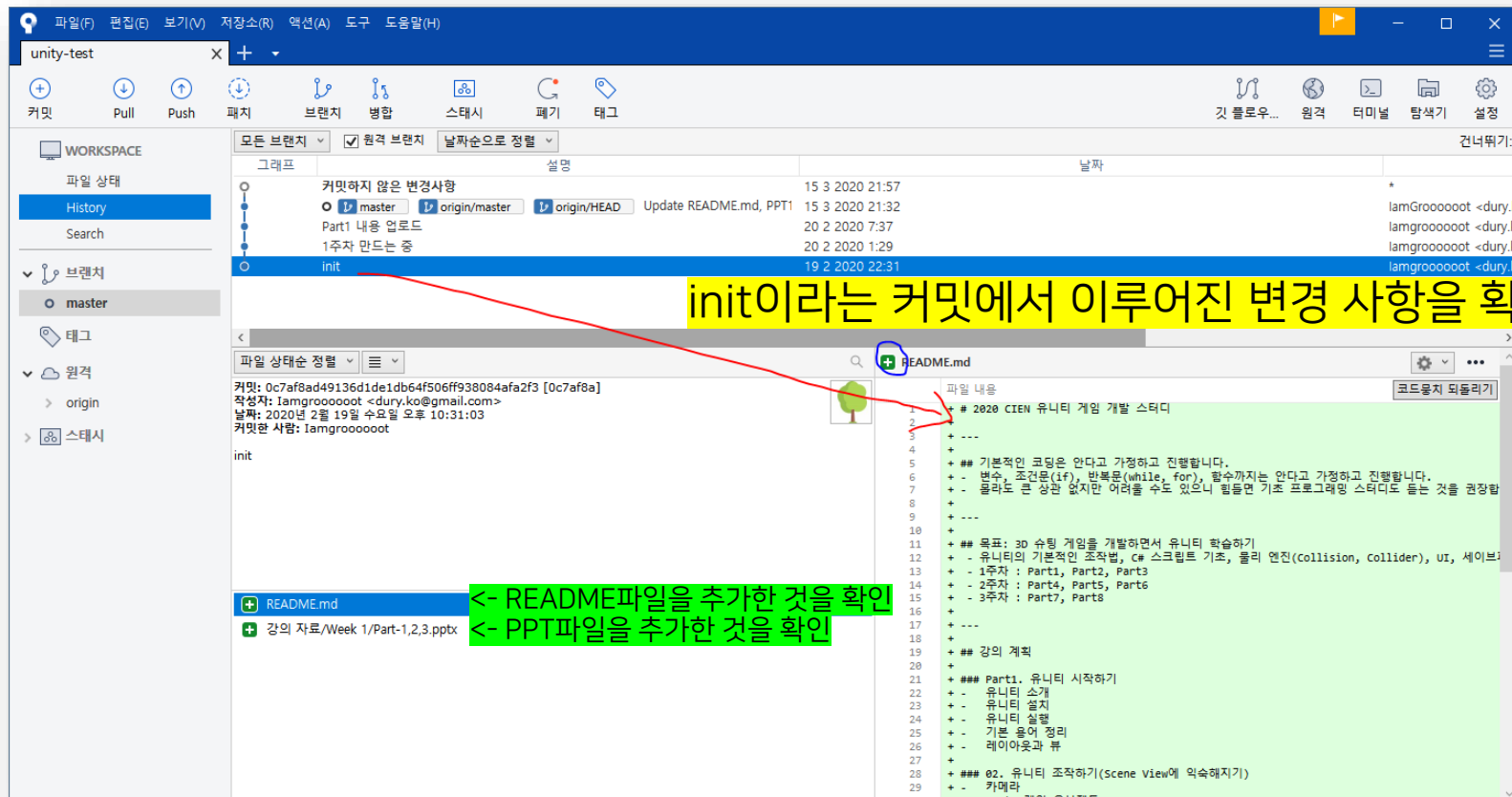
# Workspace – History

- 저장된 commit들의 기록을 확인하는 창



# Workspace – History – Show

- 각각의 커밋에서 어떤 것이 바뀌었는지 확인해보자



<- README파일에 추가한 내용을 확인



**내 컴퓨터에서 만든 저장소를  
공유해보자**

윙크왜곡와

# Git 기본 용어 2

- 푸시(Push)

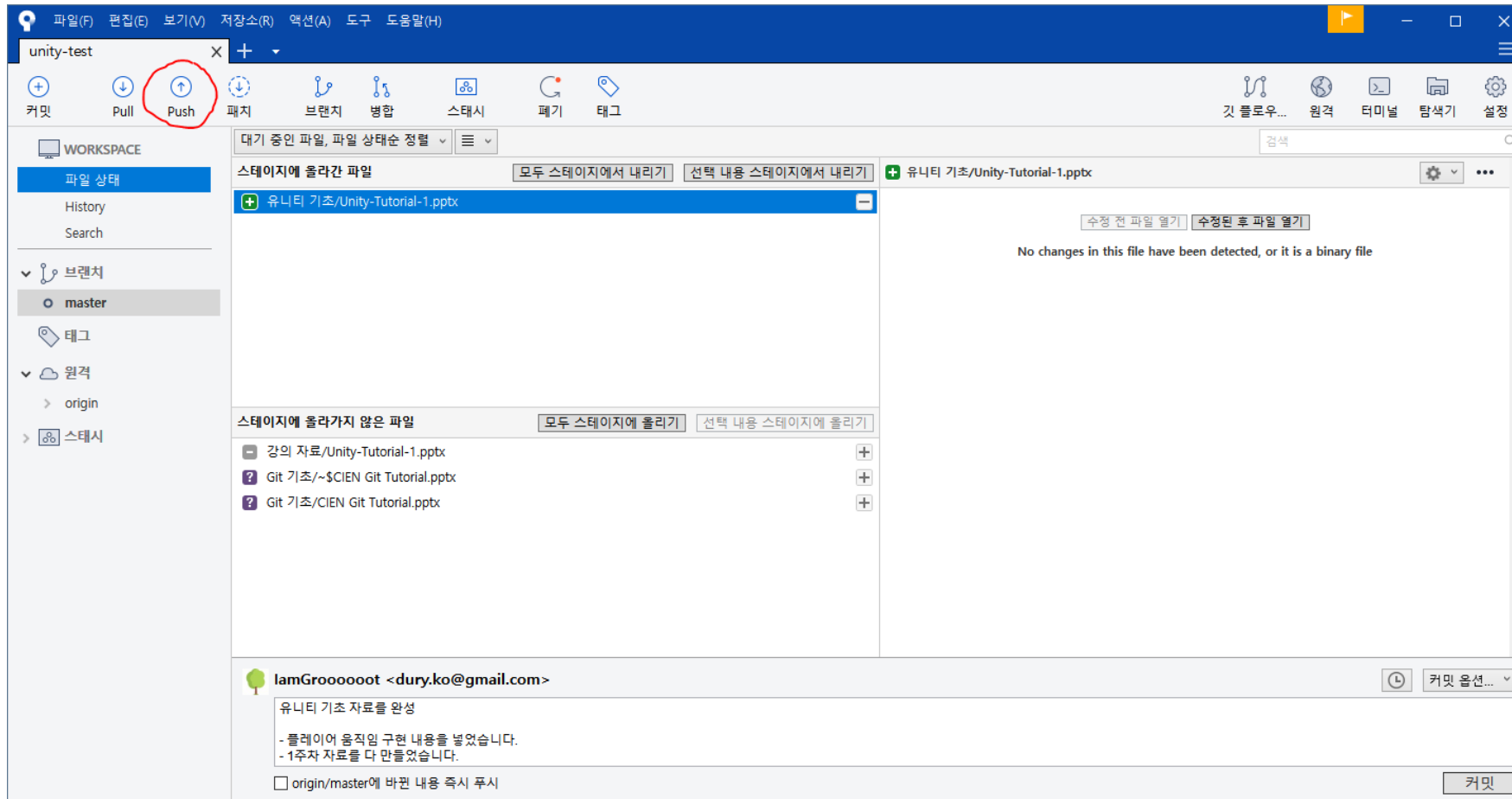
- 로컬에 있는 저장소를 인터넷(Remote)에 올리는 것
- .git 숨김 폴더를 포함해서 나의 모든 파일과 그 기록들을 업로드 한다.

- 복제(Clone)

- 인터넷에 있는(원격) 저장소를 로컬에 다운 받는 것

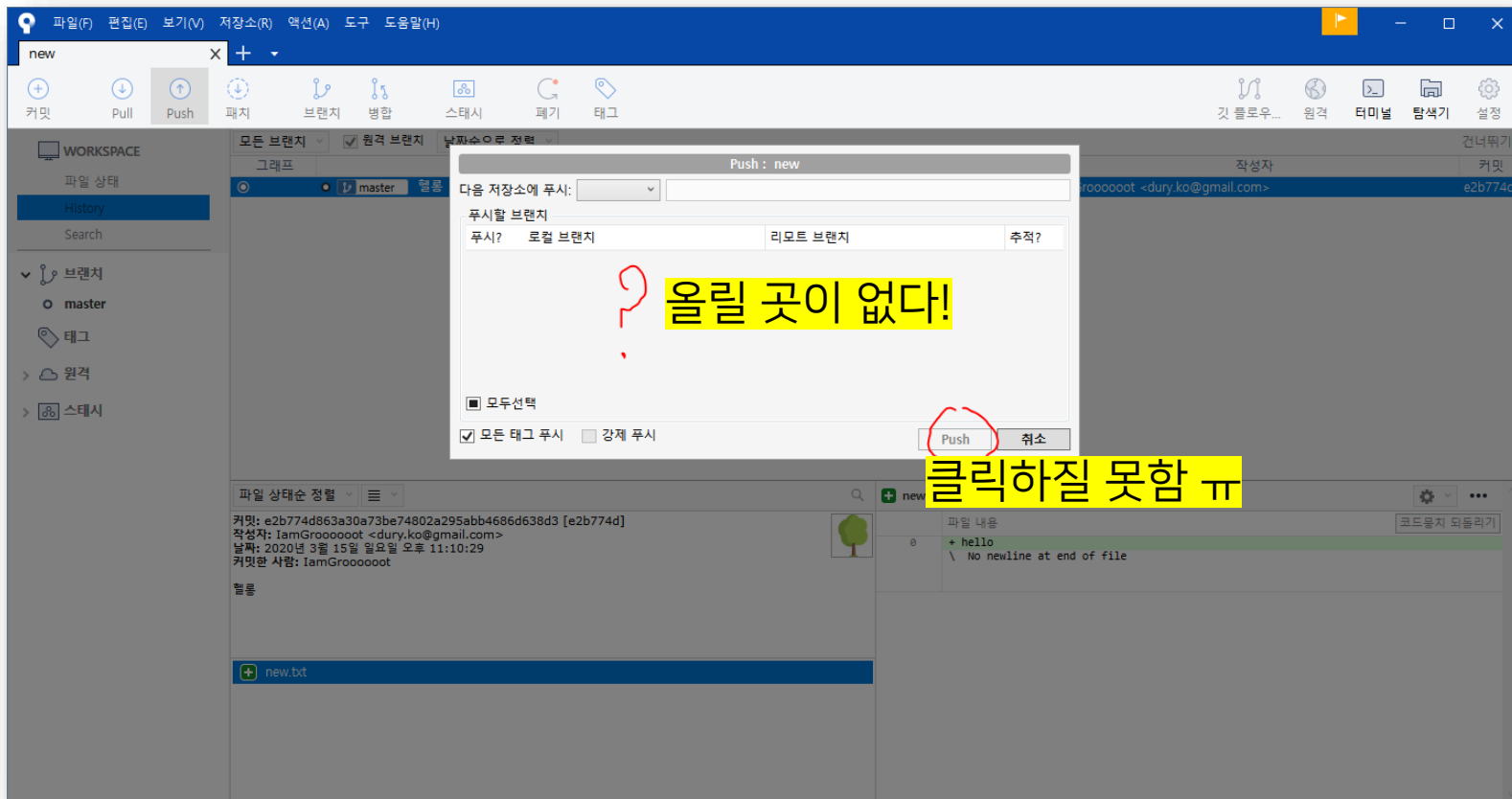
# 3. Push – 인터넷에 저장소 올리기

- Push 버튼 클릭



# 3. 실패

- 올릴 곳이 없어서 Push를 못한다.



### 3. Push – 실패한 이유

당연하다. 어디에 올릴지 알려준 적이 없기 때문  
어디에 올릴지 링크를 연결해주면 된다

어디에?

- 카카오톡? N드라이브? 구글 드라이브?  
    └└, GitHub

GitHub에 Push해서  
공유해보자

유평왜곡사

# GitHub?

- 개발자용 N드라이브/구글 드라이브 같은 곳
- Git 저장소를 올려서 공유하는 사이트
  - 앞으로 협업할 때 사용할 사이트
  - 많은 오픈소스 프로젝트들이 위치하는 곳

## 용어 정리

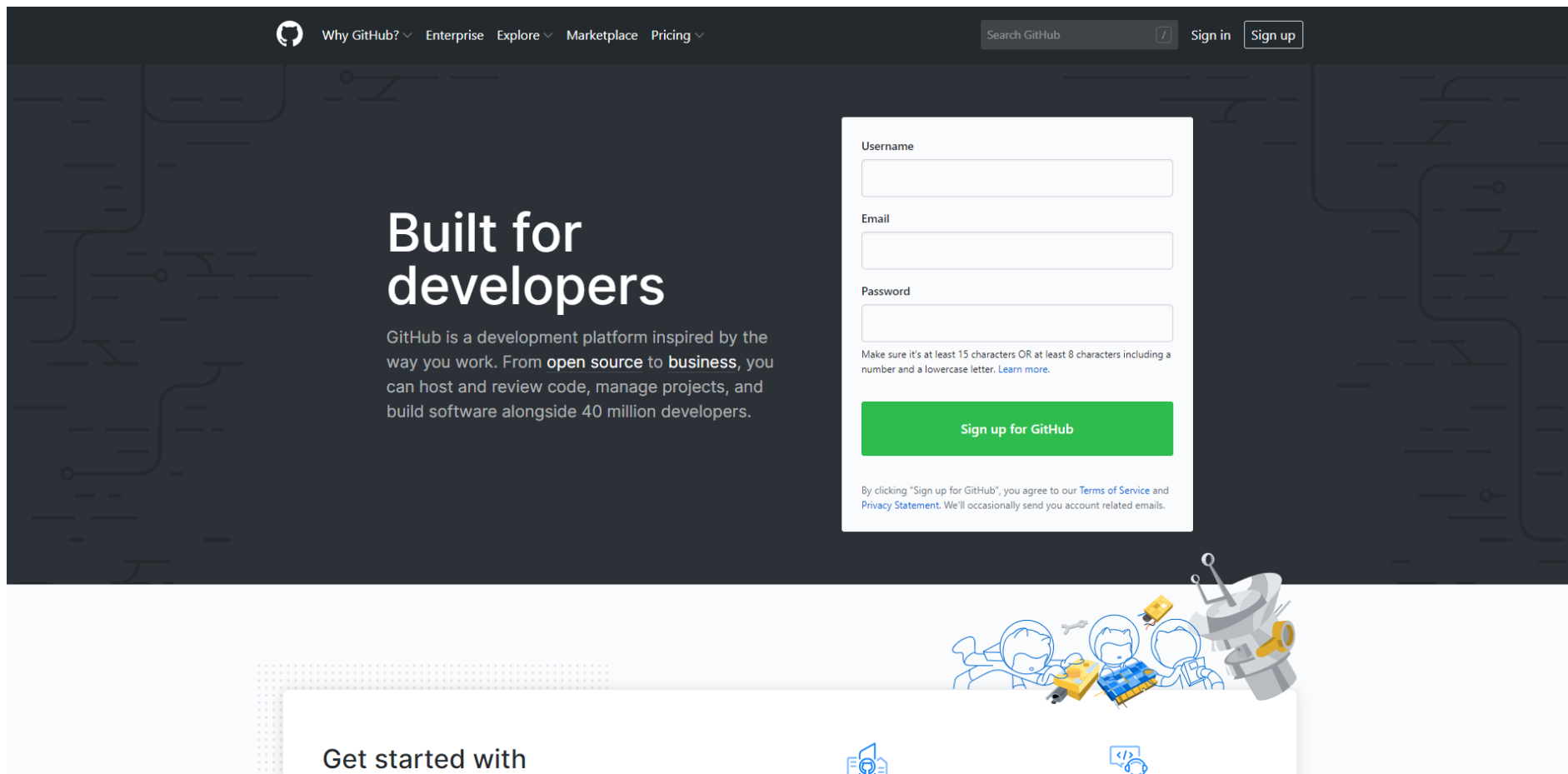
- GitHub에 올린 저장소 = 원격 저장소
- 내 컴퓨터의 저장소 = 로컬 저장소



**GitHub**

# GitHub 가입하기

- <https://github.com/> 여기서 가입 한 후, 이메일 인증

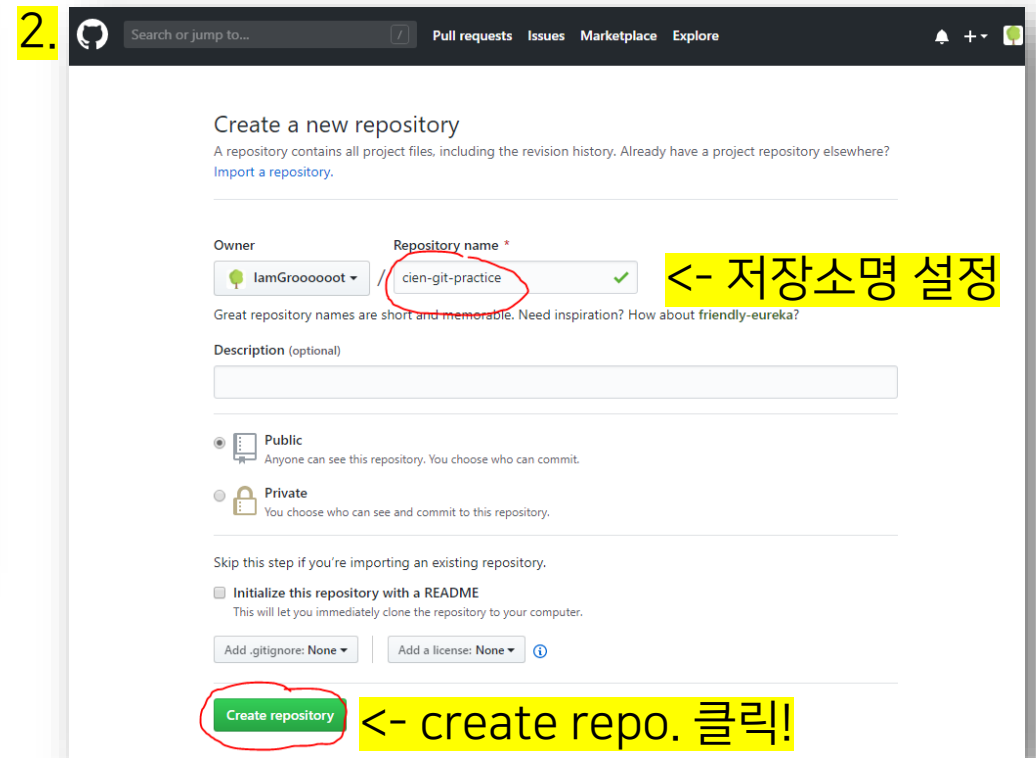
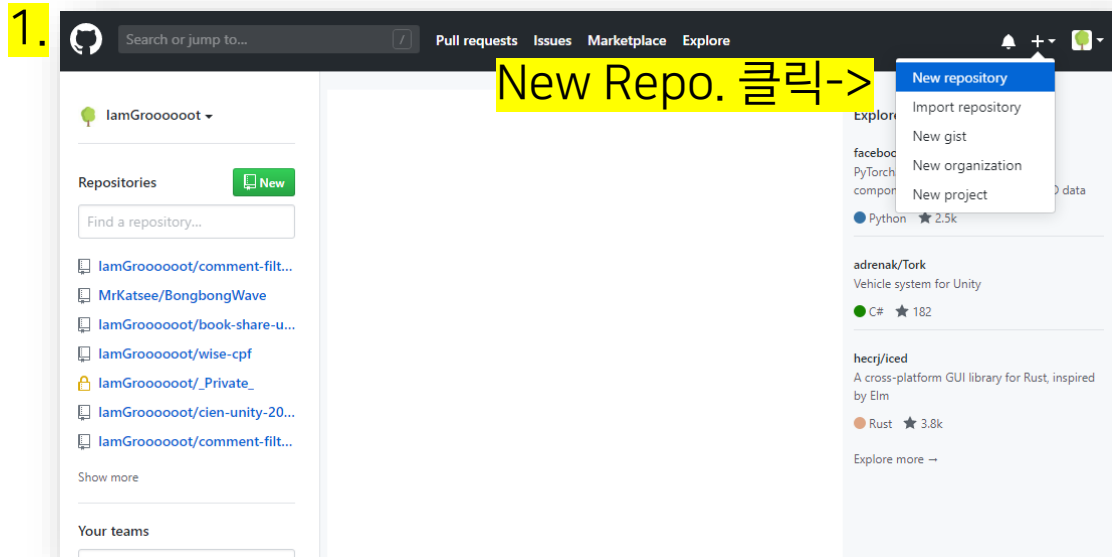




# GitHub에 Push할 곳 만들기

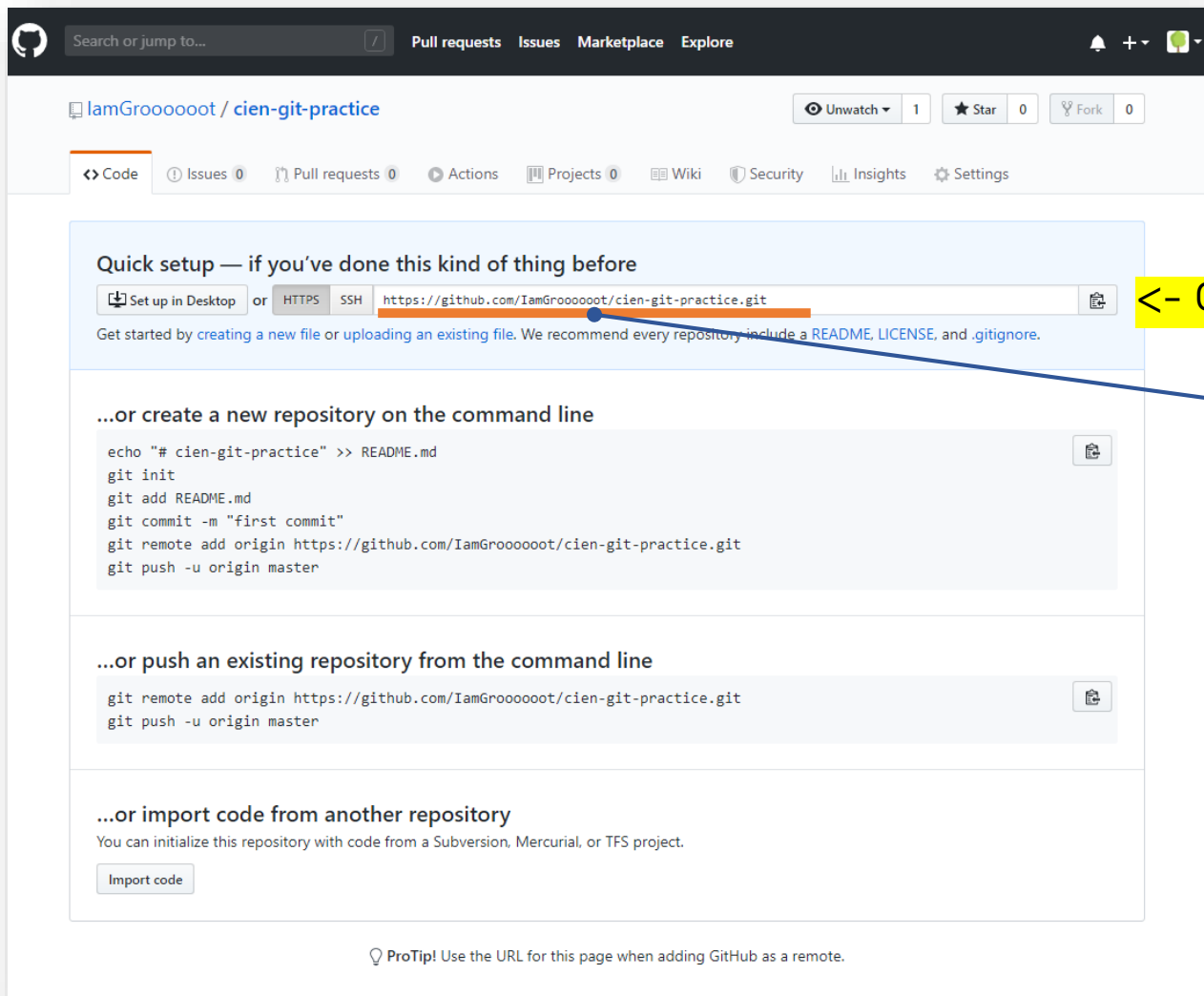
- 우리 컴퓨터에 있는 Git 저장소를 올리기 위해 비어 있는 GitHub 저장소를 만들자
- GitHub 저장소 = 우리의 원격 저장소

# GitHub에서 원격 저장소 만들기



# GitHub에서 원격 저장소 만들기

3.



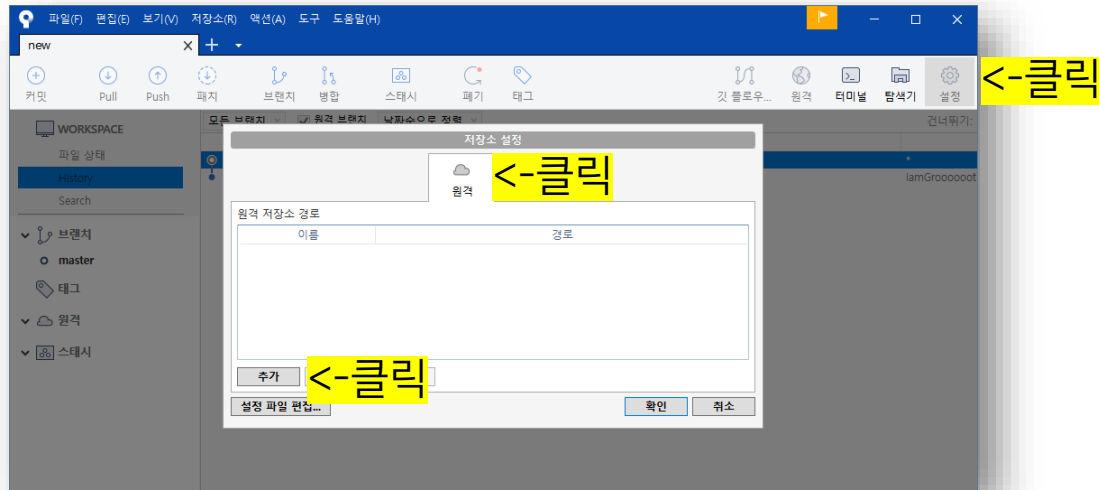
<- 이 링크를 복사하자

이 주소가 바로 내 원격 저장소의 링크이다!  
이 주소로 추가하면 여기에 Push할 수 있다.  
이 주소를 공유해서 다른 사람들과 협업한다.

Source Tree로  
로컬 저장소와 GitHub 원격 저장소를 연결해보자

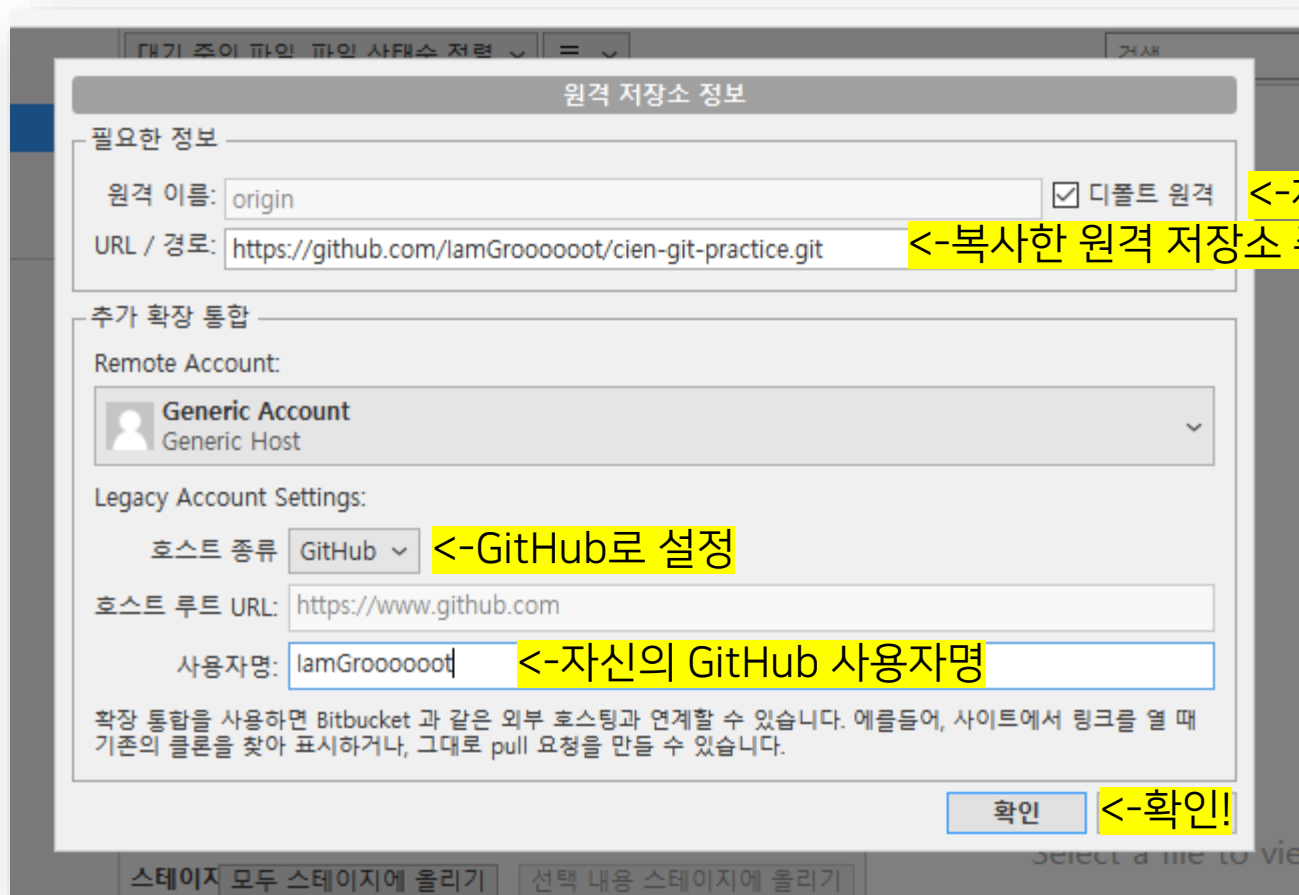
# Source Tree로 저장소에 Remote 추가하기

- 상단의 설정 버튼 > 원격 탭 > 추가 클릭



# Source Tree로 저장소에 Remote 추가하기

- 내 기본(origin) 원격을 방금 만든 GitHub 저장소로 설정해주자!



The dialog box '원격 저장소 정보' (Remote Repository Information) is shown. It has two main sections: '필요한 정보' (Required Information) and '추가 확장 통합' (Additional Extension Integration). In the '필요한 정보' section, '원격 이름' (Remote Name) is 'origin' with a checked '디폴트 원격' (Default Remote) checkbox, and 'URL / 경로' (URL / Path) is 'https://github.com/lamGrooooooot/cien-git-practice.git'. In the '추가 확장 통합' section, 'Remote Account' is 'Generic Account' (Generic Host). Under 'Legacy Account Settings', '호스트 종류' (Host Type) is 'GitHub', '호스트 루트 URL' (Host Root URL) is 'https://www.github.com', and '사용자명' (Username) is 'lamGrooooooot'. A note at the bottom explains that extension integration can connect to external hosting like Bitbucket. Buttons at the bottom are '확인' (Confirm) and '취소' (Cancel).

원격 저장소 정보

필요한 정보

원격 이름: origin ☒ 디폴트 원격

URL / 경로: https://github.com/lamGrooooooot/cien-git-practice.git

추가 확장 통합

Remote Account:

Generic Account  
Generic Host

Legacy Account Settings:

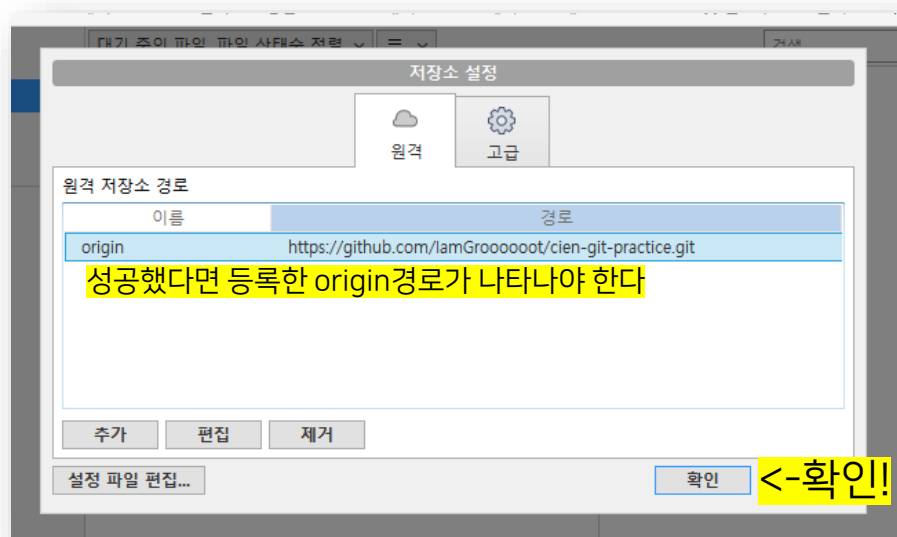
호스트 종류: GitHub

호스트 루트 URL: https://www.github.com

사용자명: lamGrooooooot

확장 통합을 사용하면 Bitbucket 과 같은 외부 호스팅과 연계할 수 있습니다. 예를들어, 사이트에서 링크를 열 때 기존의 클론을 찾아 표시하거나, 그대로 pull 요청을 만들 수 있습니다.

확인



The dialog box '저장소 설정' (Repository Settings) is shown. It has tabs for '원격' (Remote) and '고급' (Advanced). The '원격' tab is active, showing a table of remote repositories. The table has columns '이름' (Name) and '경로' (Path). The first row is 'origin' with path 'https://github.com/lamGrooooooot/cien-git-practice.git'. A note below the table says '성공했다면 등록된 origin경로가 나타나야 한다' (If successful, the registered origin path should appear). Buttons at the bottom are '추가' (Add), '편집' (Edit), '제거' (Remove), '설정 파일 편집...' (Edit Settings File...), and '확인' (Confirm).

저장소 설정

원격

원격 저장소 경로

이름	경로
origin	https://github.com/lamGrooooooot/cien-git-practice.git

성공했다면 등록된 origin경로가 나타나야 한다

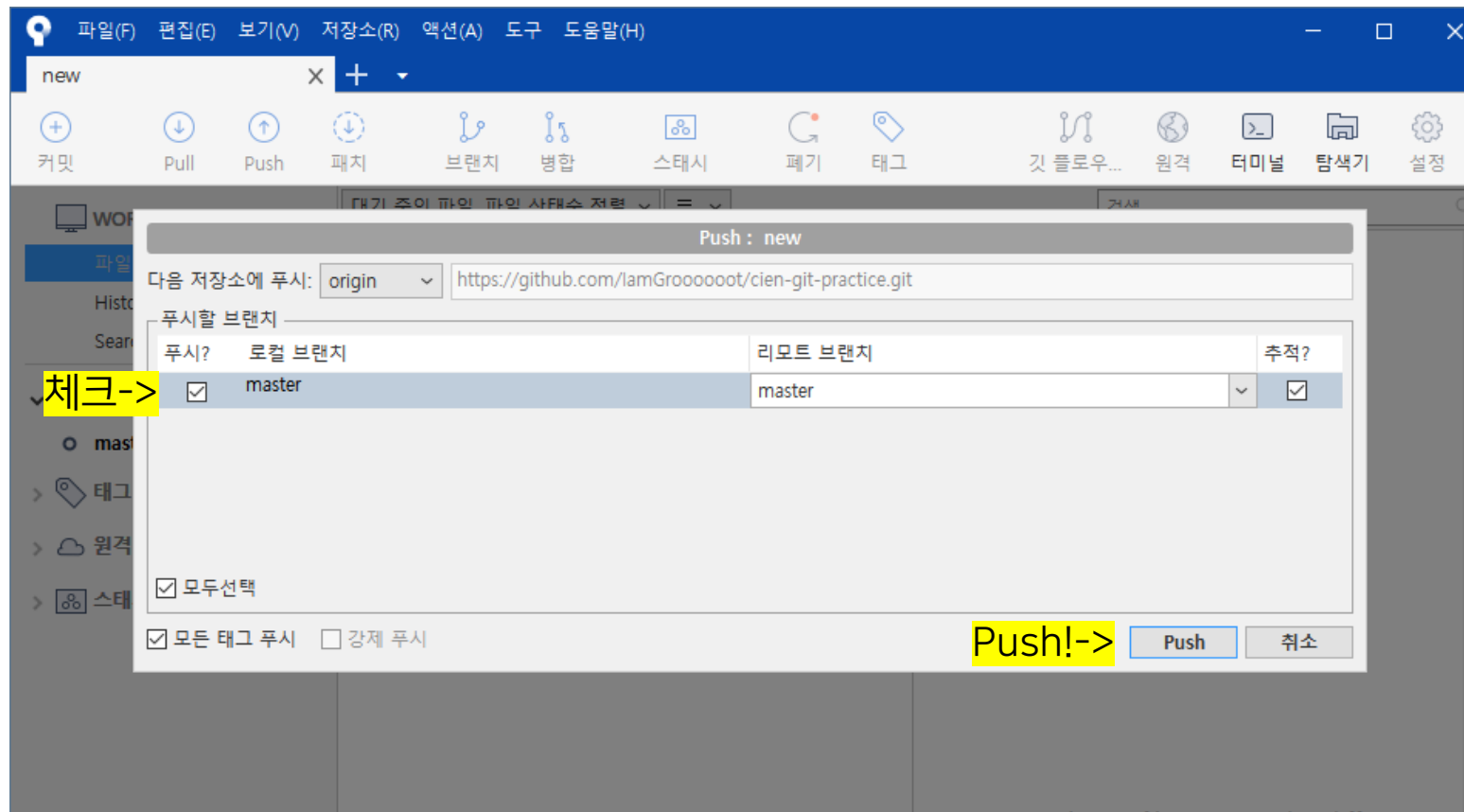
추가 편집 제거

설정 파일 편집...

확인

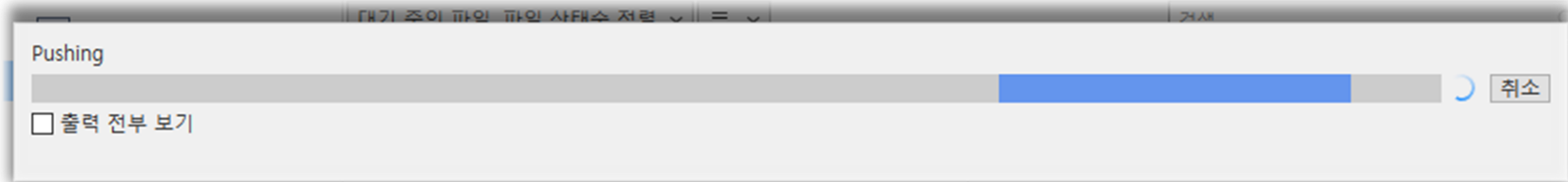
# 3. Push – 재시도

- 다시 Push클릭 후, master(기본 브랜치) 체크

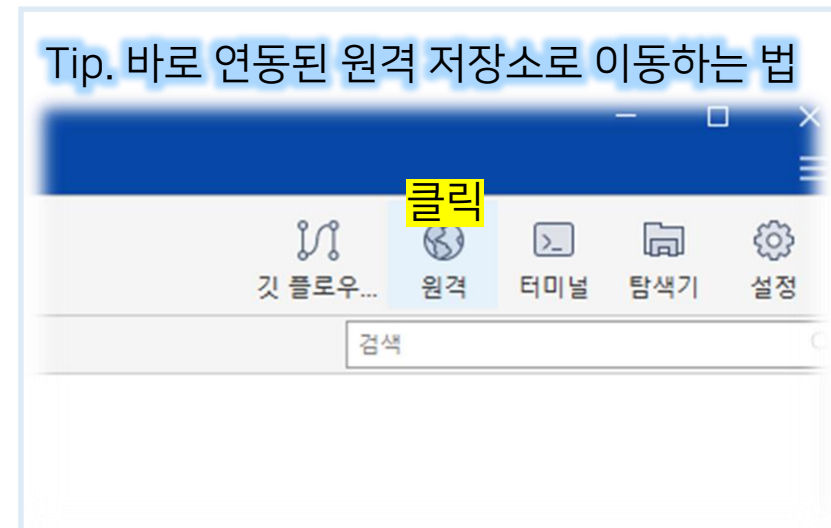
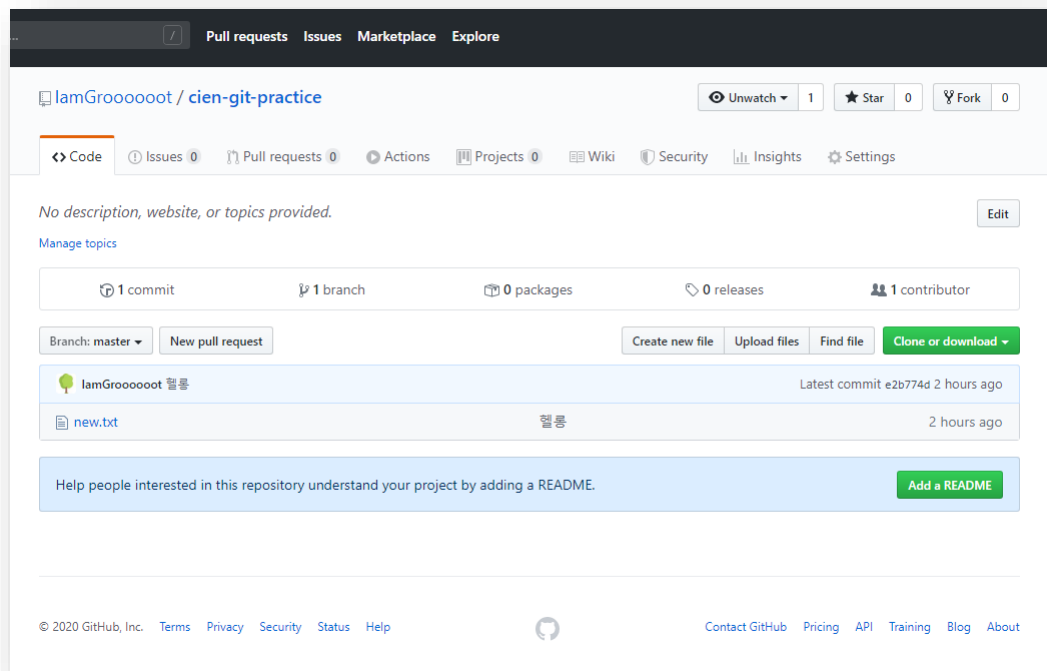


# 3. Push – 이제 된다!

- Push 성공!



- GitHub 원격 저장소에 가서 잘 올라갔는지 확인해보자.





### 3. Push – 매번 연동해야 할까?

- L L
- .git 숨김 폴더에 다 저장되므로 처음 한번만 origin 원격을 설정하면 된다

### 3. Push – 원격 저장소에서 다운로드한 저장소는?

- ㄱ
- 우리가 방금은 로컬에서 저장소를 만들었다  
그렇기 때문에 Git은 원격 저장소의 주소를 몰랐다!
- 원격 저장소(GitHub)에서 내 컴퓨터로 저장소를 복제하면 Git이 원격 저장소의 주소를 파악하고 오기 때문에 바로 Push 할 수 있다!

그래서 처음부터 GitHub에서 만든 저장소 다운 받아서 쓰면 덜 귀찮다.  
이것을 복제(Clone)라고 부른다.

# 원격 저장소를 복제해보자

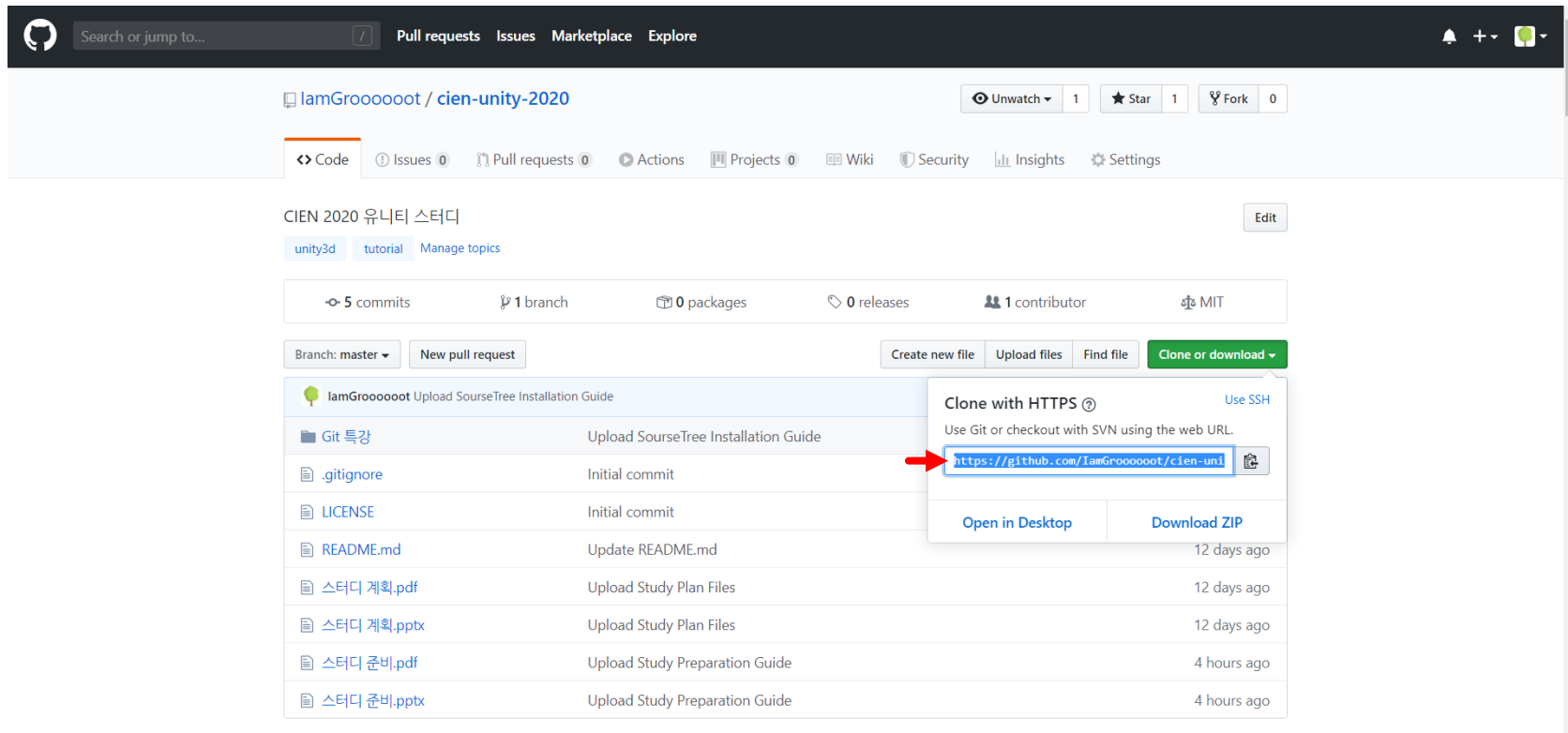
GitHub 저장소를 로컬에 복제해보자

이것이 바로 이 문서의 주제입니다

이것이 바로 이 문서의 주제입니다

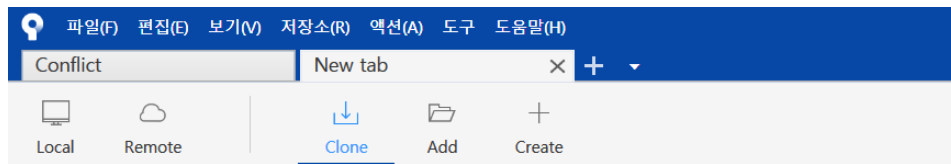
# 복제하고 싶은 GitHub 저장소에 들어가기

Clone or Download 클릭 후 HTTPS 링크 복사



# 복제(Clone)하기

- 새로운 탭에서 복제(Clone)를 누른 후 링크를 붙여 넣는다.
  - 붙여넣어도 클론 버튼이 비활성화되어 있다면 한번 클릭하면 활성화 됨



## Clone

Cloning is even easier if you set up a [remote account](#)

이 칸에 ctrl+v

탐색

저장소 종류: Git 저장소입니다

탐색

Local Folder:

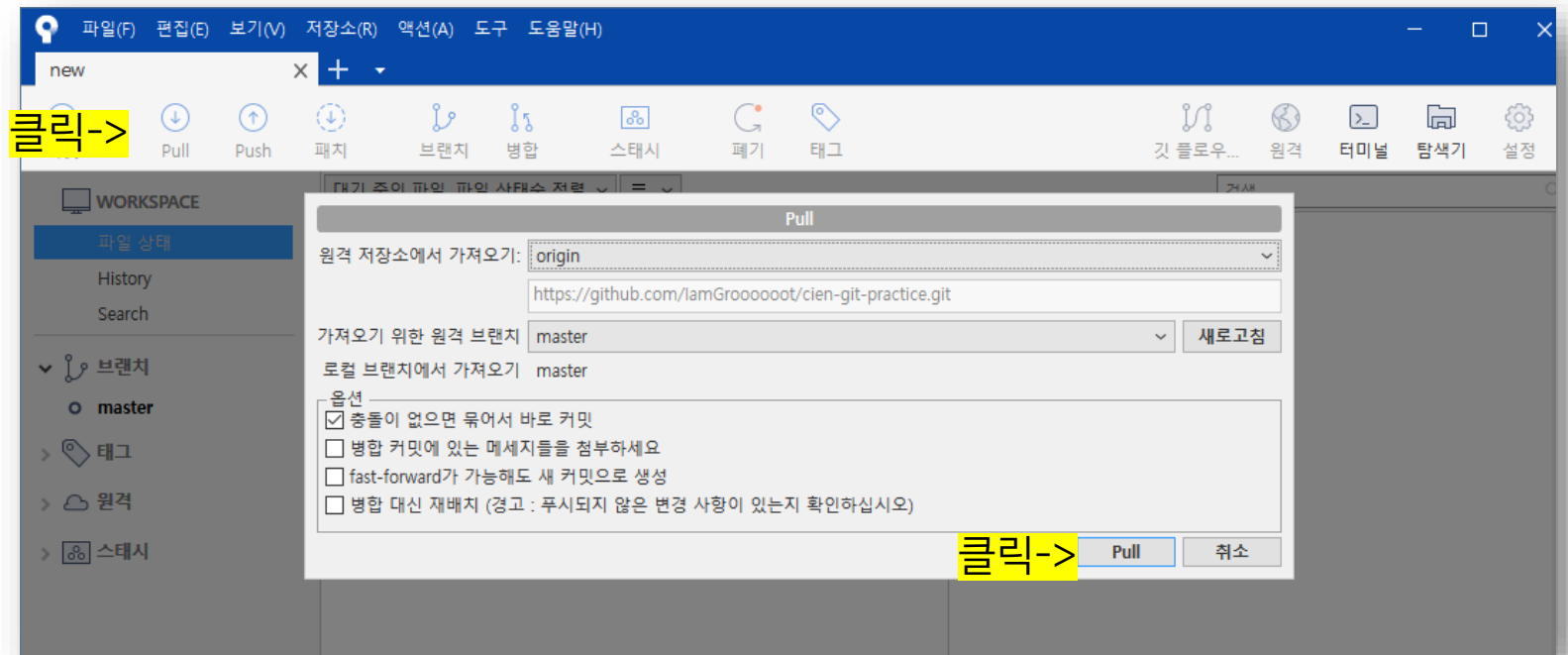
> 고급 옵션

클론

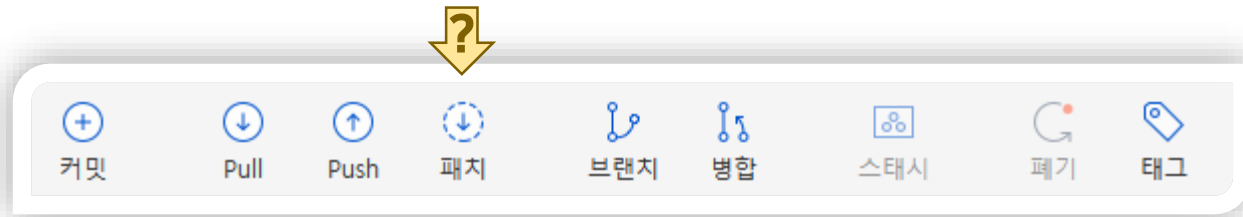
클론을 눌러서 복제!

# 4. Pull - 인터넷에서 새로 올라온 커밋 가져오기

- 다른 동료가 Push를 했다면 내 로컬의 Git 저장소는 알려주기 전까지 모른다
- 변경된 사항은 원격 저장소로부터 다운 받아야 알 수 있다
- 그것이 바로 Pull이다
  - Pull 버튼 클릭



# Fetch?



- Pull은 변경 사항을 다운 받고 내 로컬 저장소에 적용까지 한다
  - 다운하고 자동으로 Merge까지 해준다 (뒤에서 함)
- Fetch는 다운 받기만 하고 적용은 안 한다
  - 실제 파일은 변하지 않는다
- 요약
  - Pull = Fetch(다운만 함) + Merge(합침)
  - Fetch하고 Merge하는 것을 권장

# 발생할 수 있는 문제 해결

Tip

5.9.5 L YD 도네 에드

ub



# \* Commit 취소하기 (주의)

두 가지 방법이 있다

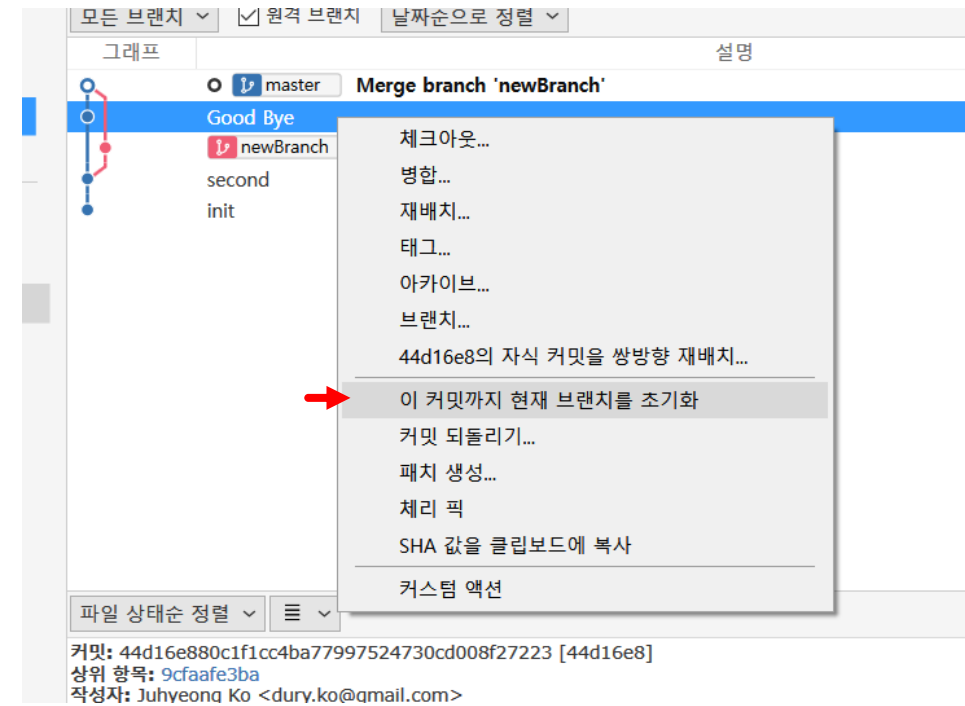
1. Revert
2. Reset

주의: 절대로 Push한 커밋은 수정하면 안 된다

무조건 해야 한다면 팀원들에게 모두 알리고 Force Push한다

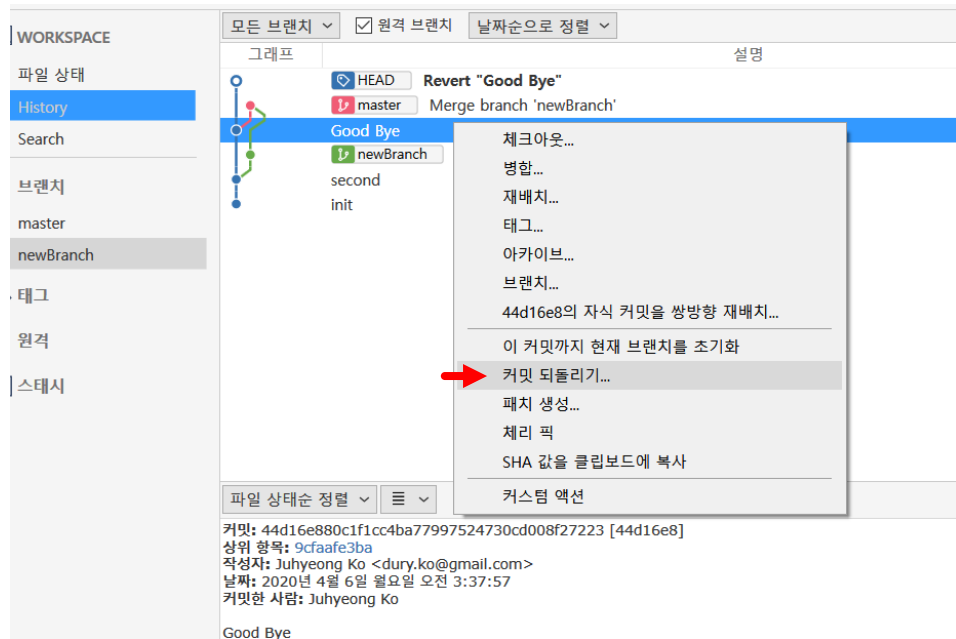
# \* Reset (초기화)

- Soft: HEAD만 옮긴다
  - `git reset --soft [커밋]`
- Hard: [커밋] 이후로 완전히 지운다
  - 제일 강력: HEAD, WD, Index 싹 다 지움
  - `git reset --hard [커밋]`
- Mixed(기본): HEAD도 옮기고 Index도 바꾼다
  - 기본값
  - `git reset --mixed [커밋]`



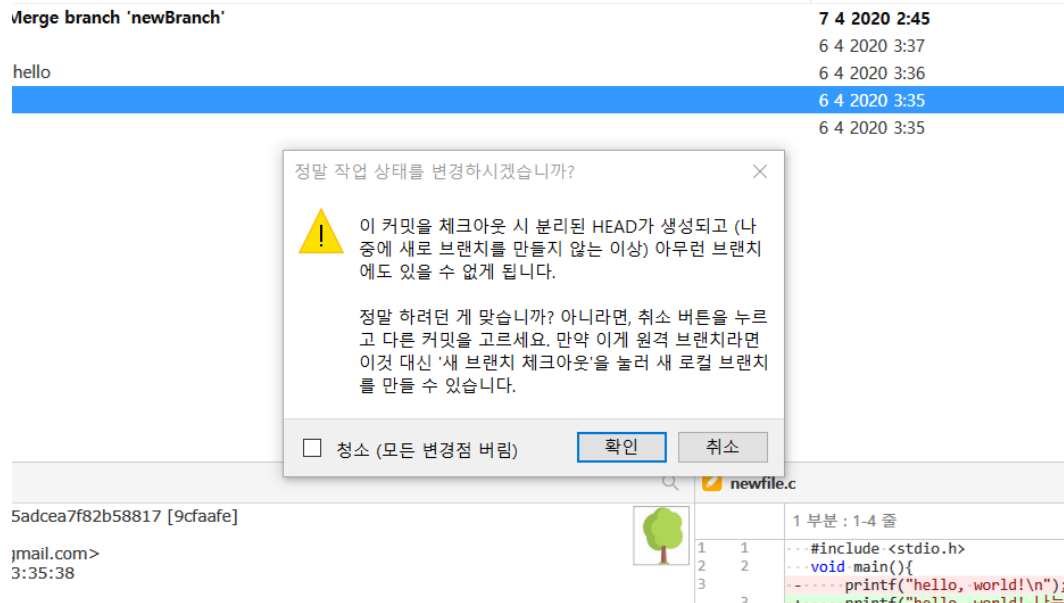
# \* Revert (되돌리기)

- 파일들을 이전 [커밋]인 상태로 되돌린 새로운 커밋을 만든다
- 이전 커밋을 수정하지 않기 때문에 안전하고 Revert 후 Push 가능
- git revert [커밋]
  - 바로 커밋해주는게 싫다면 git revert -n [커밋]



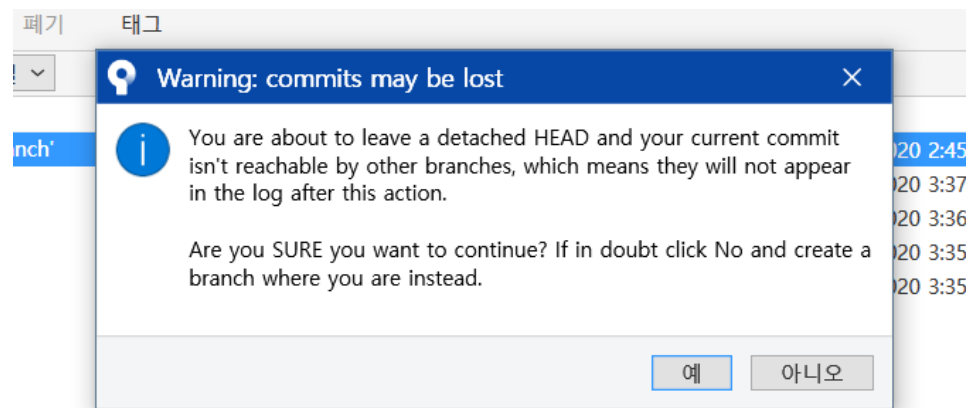
# \* Checkout Commit - 예전 커밋 구경하기

- Commit에는 Commit을 구별하기 위한 SHA값이 있다
- 그 값으로 예전 커밋을 가져와서 확인할 수 있다
  - 소스 트리는 더블 클릭하면 알아서 해준다
- **Detached HEAD 상태가 되는 것을 주의해야 한다**



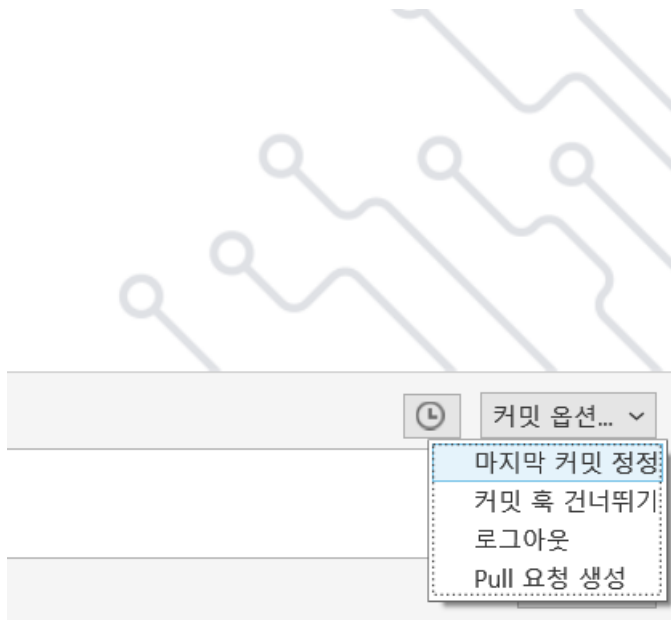
# \* Detached Head 상태

- HEAD가 분리됐을 때
  - HEAD가 커밋을 가리키므로 브랜치에 커밋 할 수 없음 = 일시적인 커밋만 가능
- 브랜치가 가리키는 최신 커밋을 더블 클릭함으로써 해결 가능



# \* 이전 Commit 메시지 수정

- `git commit --amend` or `git reset --soft`
- (커밋 버튼 위) "커밋 옵션..."에서 "마지막 커밋 정정" 클릭



# 작업 환경 나누기

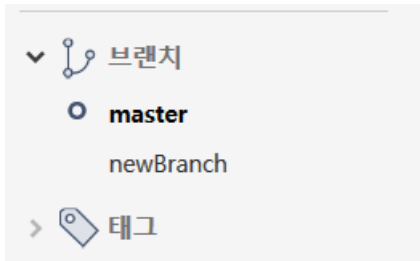
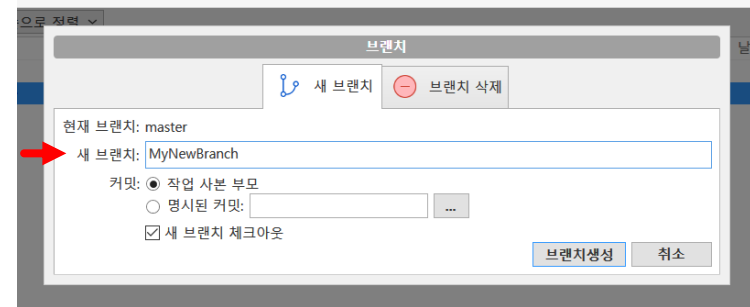
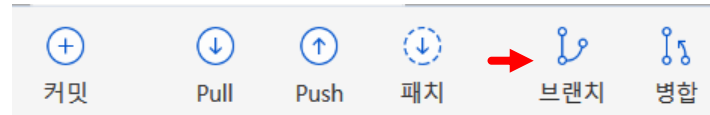
Branch

작업 환경 나누기

DL9UCU

# 브랜치(Branch)

- 브랜치 생성 클릭
- 새 브랜치 이름 설정
- 옆에 브랜치 목록에서 원하는 브랜치를 클릭해서 이동할 수 있다



- master 브랜치는 처음에 자동 생성되는 기본 브랜치이다



# 활용할 때

- Bug Fix
- 새로운 기능
- 내 컴퓨터에서만 테스트할 코드
- ...

# 작업 합치기

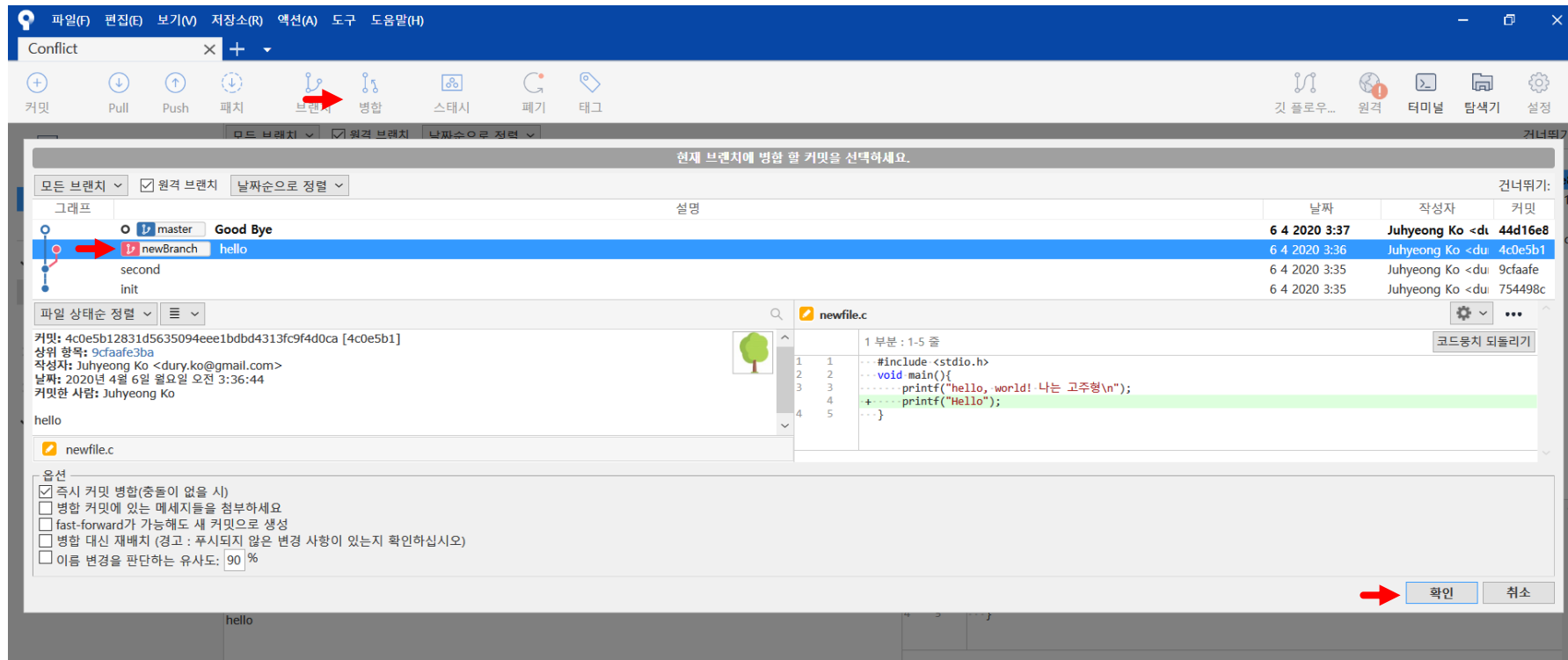
Merge

작업 합치기

merge

# 병합(Merge)

- 병합 클릭 -> 현재 병합할 브랜치 선택
- **주의!** 병합할 브랜치랑 병합될 브랜치를 절대 혼동하지 말자

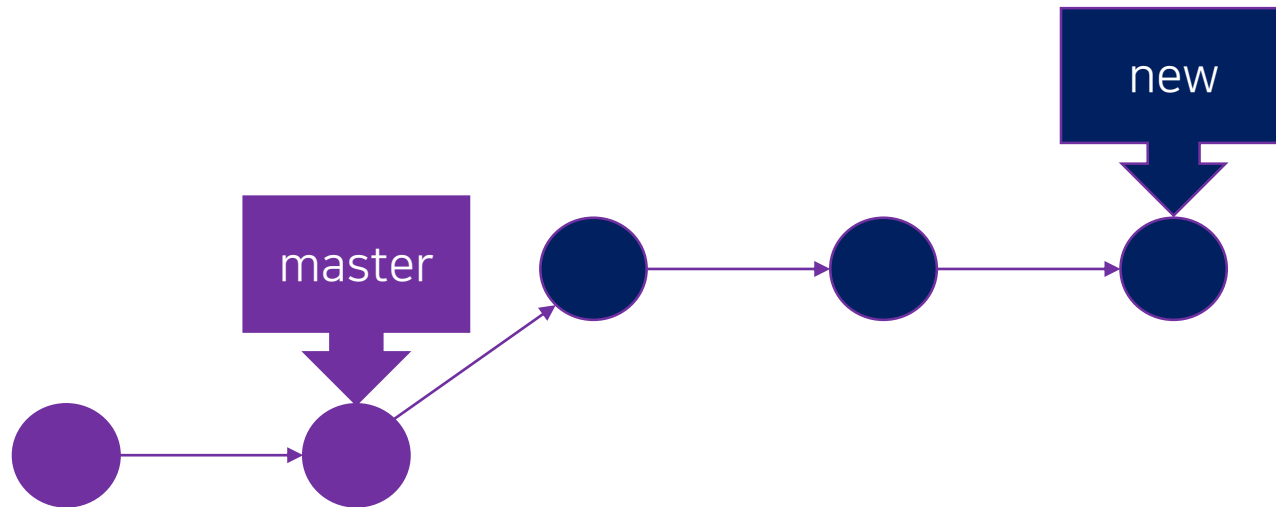


# 두 종류의 Merge

1. Fast Forward Merge
2. True Merge (3-Way Merge)

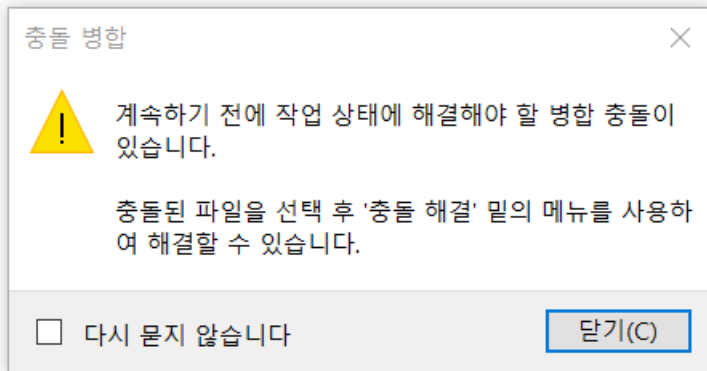
# 1. Fast Forward Merge

- Merge긴 Merge인데 뭔가 인정해주고 싶지 않은 가짜 Merge
- 커밋이 한 라인에 있을 때 합쳐 줌
- 충돌하지 않음



## 2. True Merge (3-way merge)

- 진정한 Merge
- Base, 브랜치1, 브랜치2  
=> 이렇게 3가지를 비교하면서 동작하여 3-Way Merge라고 흔히 부른다
- 충돌(Conflict)가 발생할 수 있다
  - 충돌은 Git이 알아서 코드들을 합쳐주지 못 할 때 발생한다.

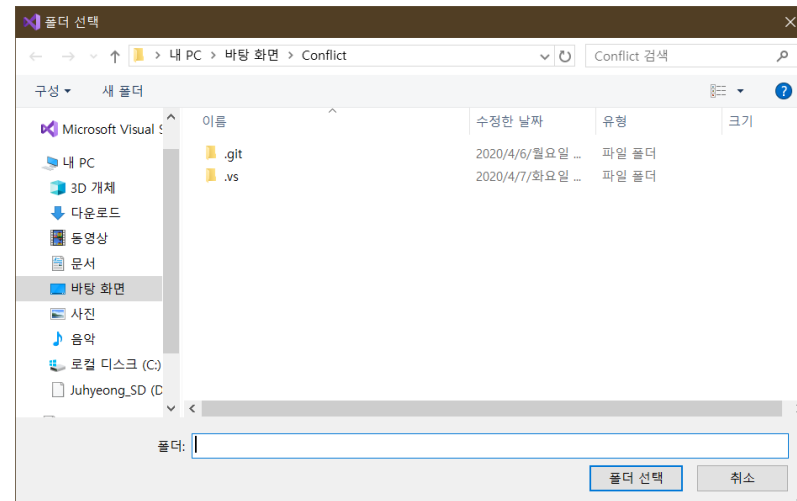
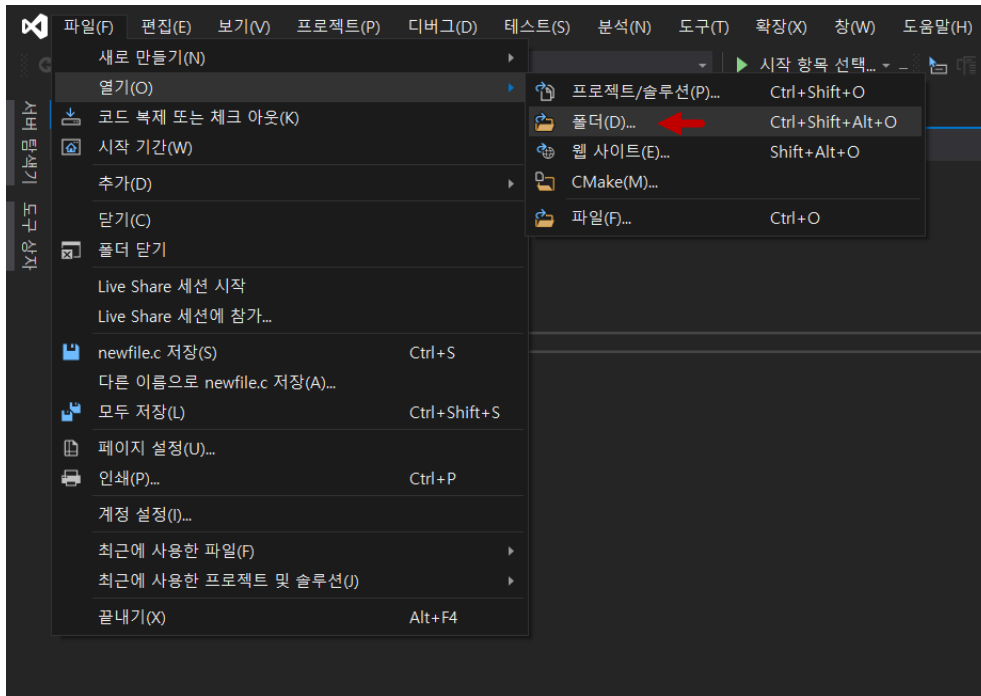


## 2. 충돌(Conflict) 해결법

- Source Tree의 충돌(Conflict) 해결 기능은 별로이다...
- 그래서 Visual Studio에 내장된 Git 기능으로 해결해보자
  - Visual Studio도 개발 프로그램이어서 기본적으로 Git이 내장되어 있다

# 2. Visual Studio로 Conflict 해결하기

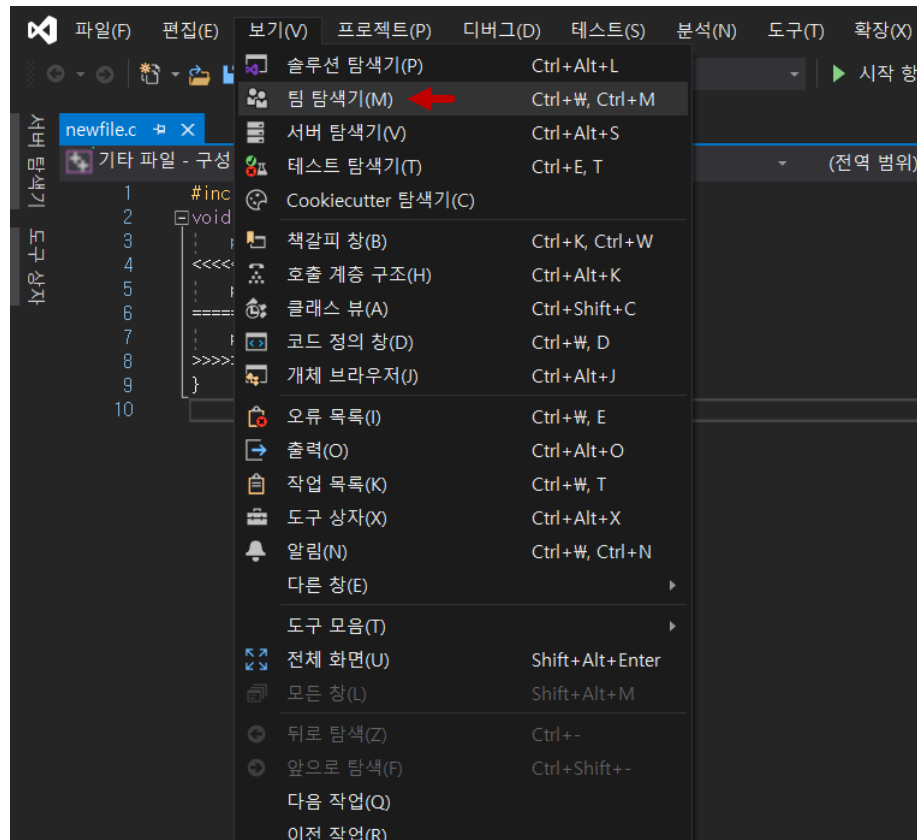
## 1. Conflict가 발생한 Git 저장소 열기



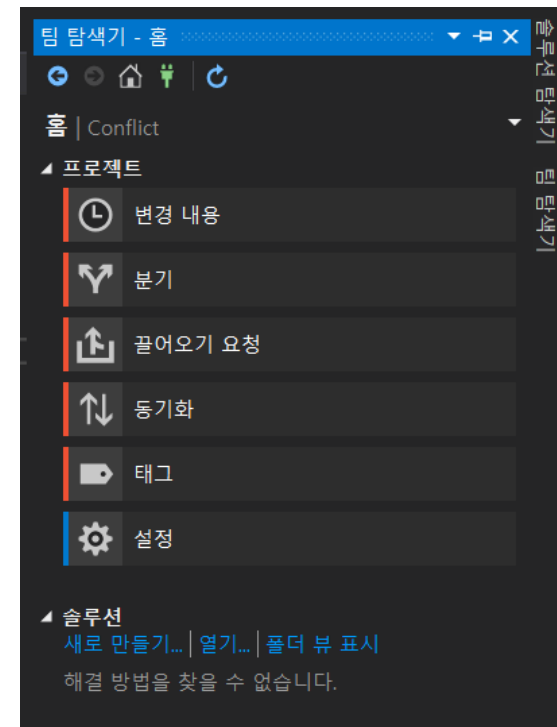


# 2. Visual Studio로 Conflict 해결하기

## 2. 보기 메뉴에서 팀 탐색기 창을 연다

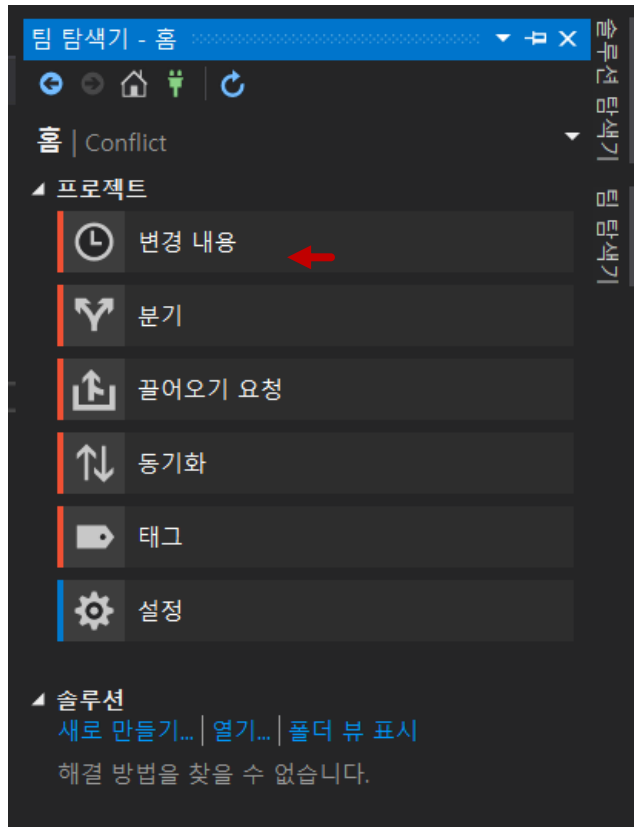


이 창을 열어야 한다

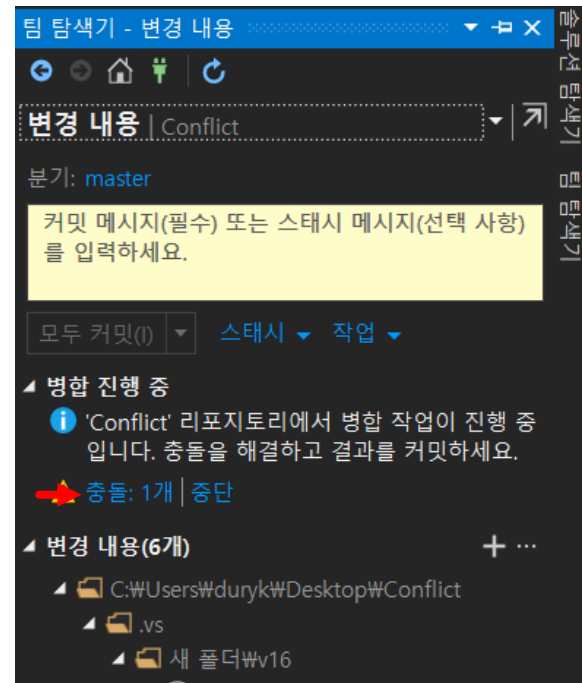


## 2. Visual Studio로 Conflict 해결하기

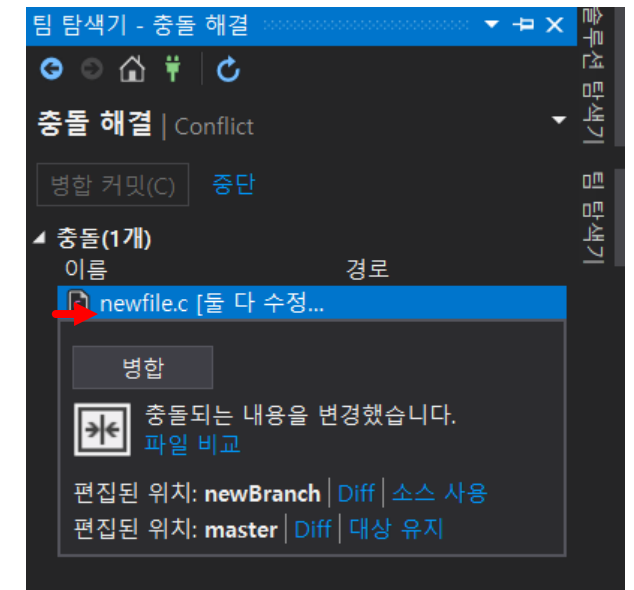
3. 변경 내용을 클릭. 병합 작업이 진행 중이라고 떠야 함.



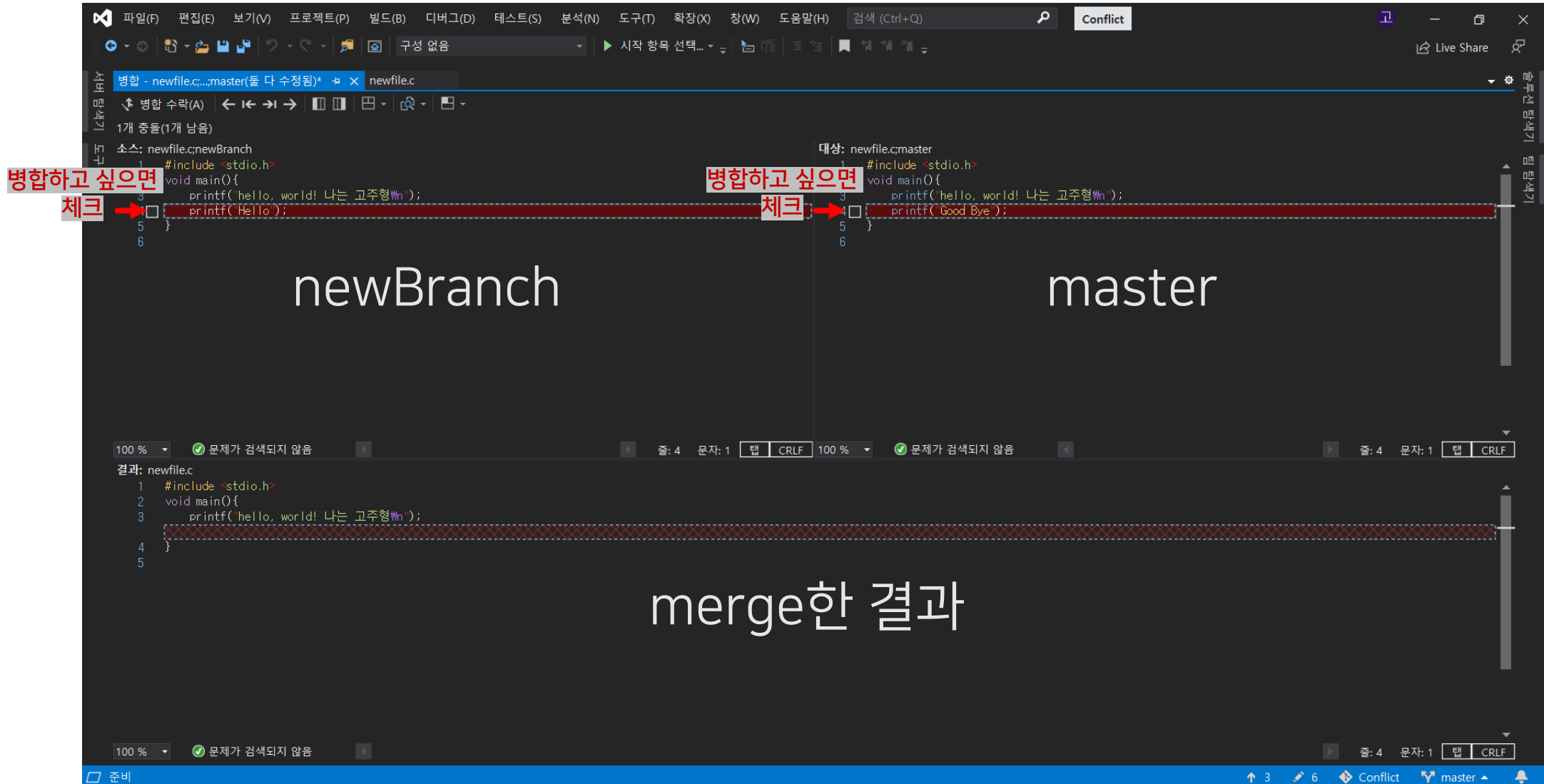
4. 충돌 1개 클릭



5. 병합 클릭

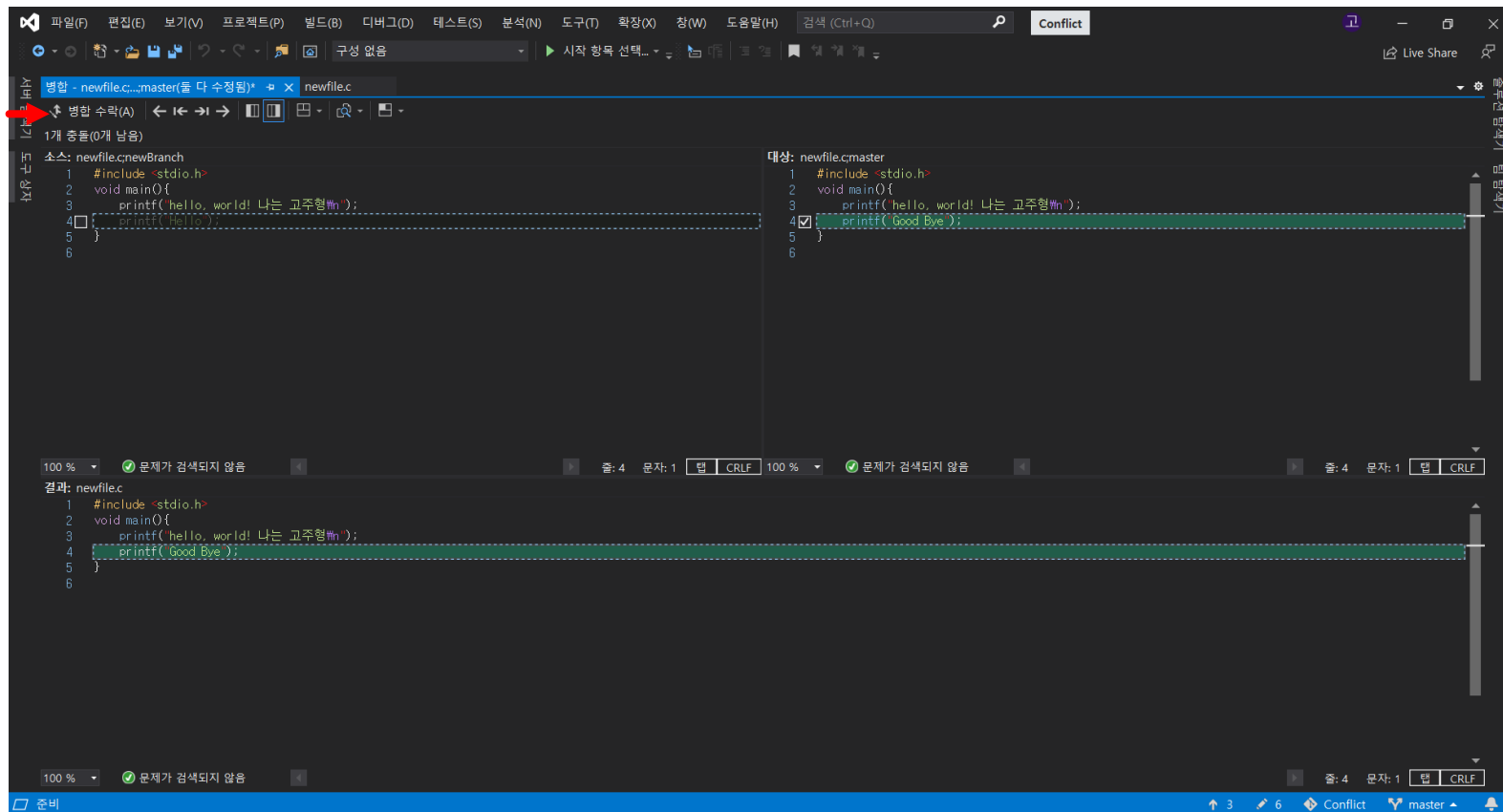


## 2. Visual Studio로 Conflict 해결하기



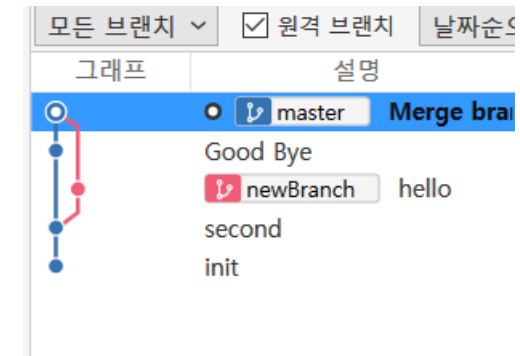
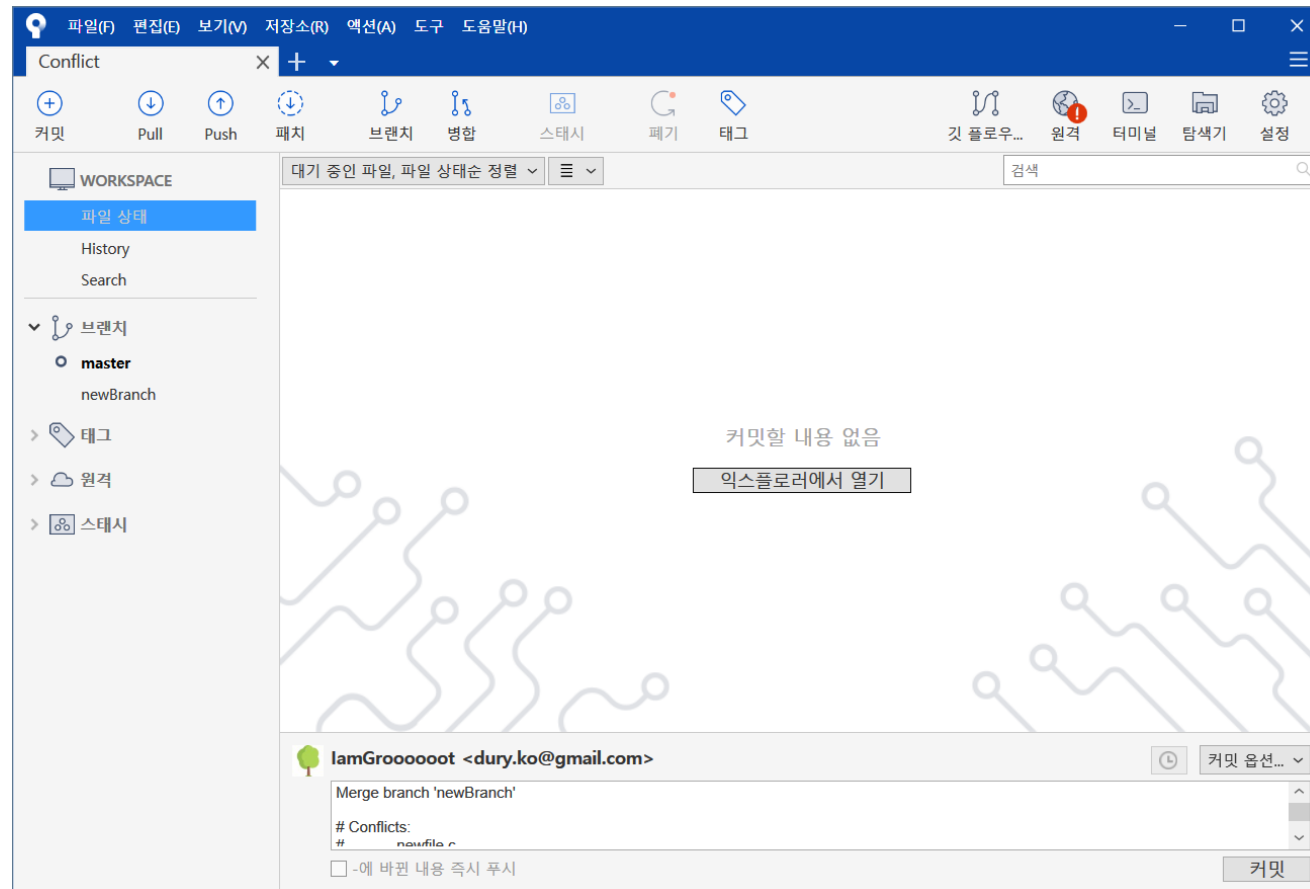
## 2. Visual Studio로 Conflict 해결하기

6. 병합하고 싶은 코드를 선택하고 "병합 수락" 클릭해서 병합을 마무리



## 2. Visual Studio로 Conflict 해결하기

### 7. Source Tree로 돌아와서 병합한 파일을 커밋해서 Conflict 해결



# 흔히 발생하는 문제들

Stash로 해결해보자

등이 트윙이드 도넛

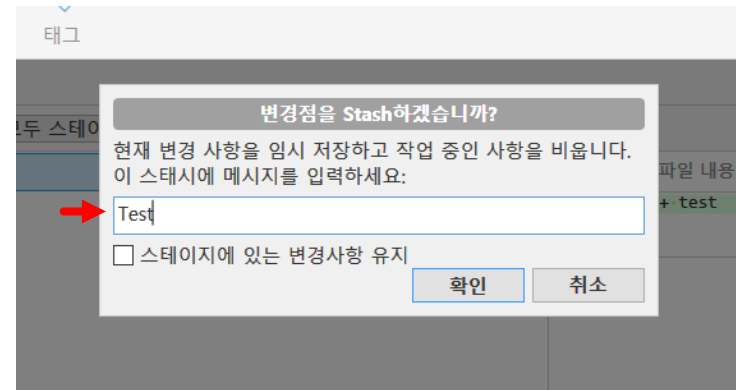
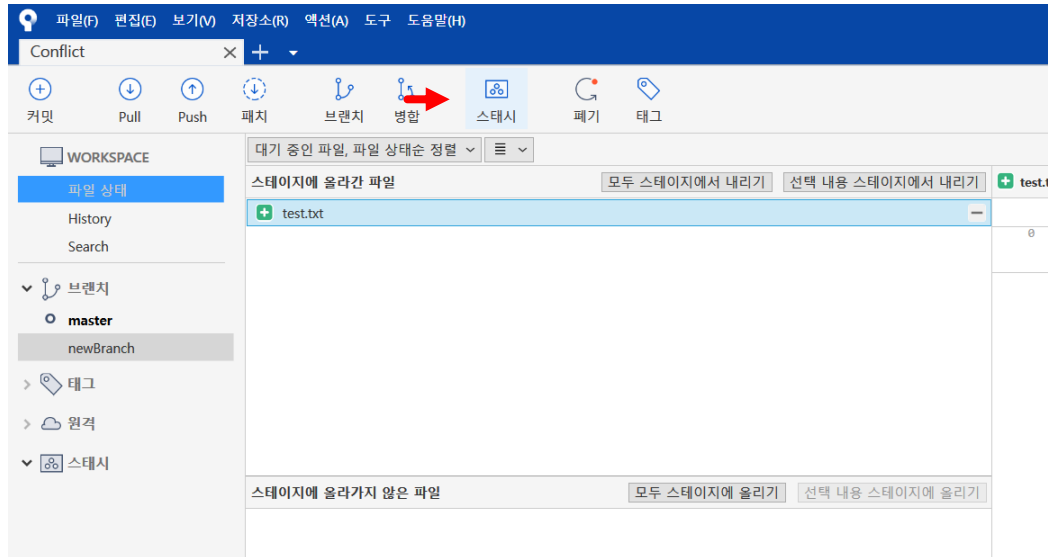
2192112 에드워드

# Stash - 임시 인덱스 저장소

- Stage Area(Index)에 있는 파일을 잠시 저장해 둘 수 있는 저장소이다
- Stack이다
  - Push, Pop
  - LIFO(Last in first Out)

# Stash 사용법 - 저장

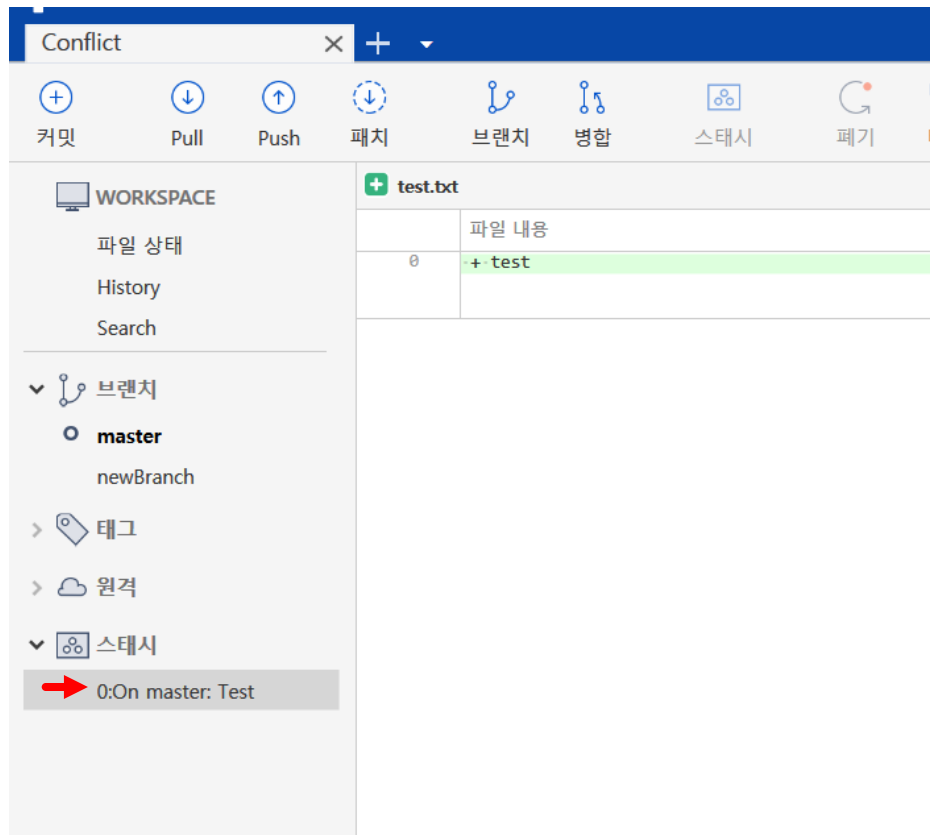
- 스테시 누르고 저장 메시지 남기기





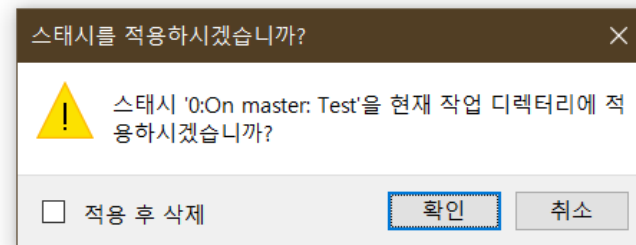
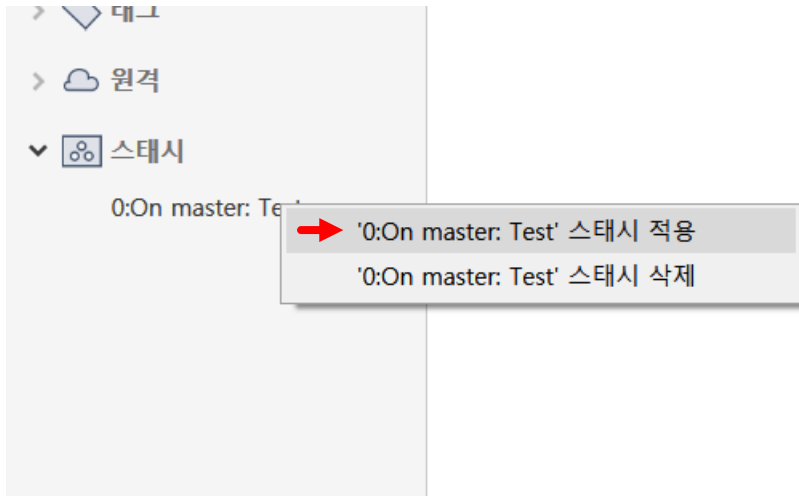
# Stash 사용법 - 저장한 것 확인

- 왼쪽 스테시 메뉴에서 확인



# Stash 사용법 - 적용(불러오기)

- 적용하고 싶은 스테시 왼쪽 클릭 후 스테시 적용
- 적용하고 삭제하고 싶은 경우 "적용 후 삭제" 체크 (= POP)



# Pull할 때 가끔 발생하는 오류

- Error: Your Local Changes to the following files would be overwritten by merge.  
Please commit your file or stash them before you can merge.
- 내가 만료된 옛날의 저장소에서 Pull하려고 한 경우

# Stash를 사용한 간단한 해결법

1. 현재 Index를 Stash에 저장
2. Pull한다
3. Stash 적용
4. Commit
5. Push

# 다른 브랜치에서 작업한 것 가져오기

- Feature 브랜치에 커밋해야 할 것을 Master 브랜치에서 작업한 경우
- Stash로 어떻게 해결할까? (정답은 뒤에)

# 커밋 다른 브랜치로 옮기기

(커밋한 경우) Mixed Reset

1. Stash에 저장
2. 커밋할 브랜치로 이동
3. Stash 적용
4. 커밋

# .gitignore로 쓸모 없는 파일은 무시하기

- .gitignore는 Git이 인식하는 특수한 파일이다
  - 안에 적힌 폴더/파일들은 마치 없는 취급을 한다 -> 내 파일 상태가 깨끗해 진다
- Unity .gitignore를 올바르게 설정하고 GitHub 저장소에 Push해보자
  - GitHub 저장소를 만들 때 설정하면 편하다.
  - 원하는 .gitignore 검색해서 넣어줘도 괜찮다.
    - (주의) 이미 Track된 파일들은 나중에 .gitignore파일을 추가해도 무시되지 않는다.
    - 이땐 index를 초기화해주면 해결된다. 루트 디렉토리에서 `git rm -r --cached`.

**감사합니다**