

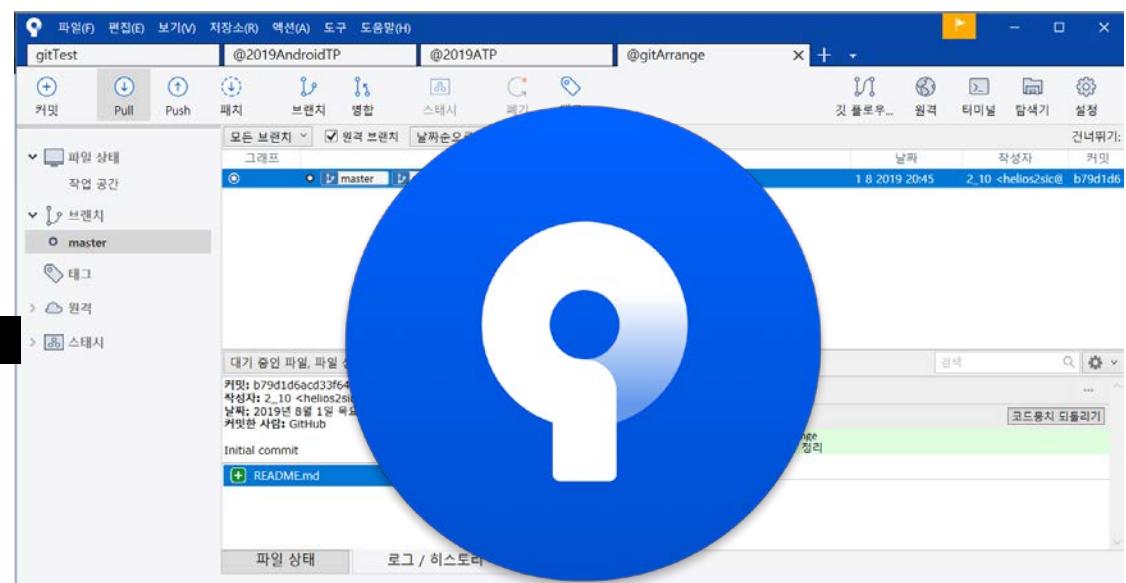
# 깃허브 총정리

Made by 주이식

Inter  
face



# GitHub +



# 1. Fork하기

The screenshot shows a GitHub repository page for the user '2019androidtp' named 'gitArrange'. At the top right, there is a 'Fork' button with a count of '0', which is circled in blue. Below the header, there is a navigation bar with links for 'Code', 'Issues 0', 'Pull requests 0', 'Projects 0', 'Wiki', 'Security', 'Insights', and 'Settings'. The main content area displays basic repository statistics: '1 commit', '1 branch', '0 releases', and '1 contributor'. It also shows a dropdown for 'Branch: master', a 'New pull request' button, and three action buttons: 'Create new file', 'Upload files', 'Find File', and a green 'Clone or download' button. Below these are two commit cards: one from 'jkey20' and another from the repository owner. The repository's README.md file is shown below, containing the text 'gitArrange' and '깃 사용법 정리'.

메인 페이지에서 파란색으로 동그라미 쳐진 곳을 누릅니다.

# 1. Fork하기

Fork gitArrange ×

Where should we fork gitArrange?

**2\_10 jkey20**

 2018-Interface-Programming-Exhibition

 Interface518

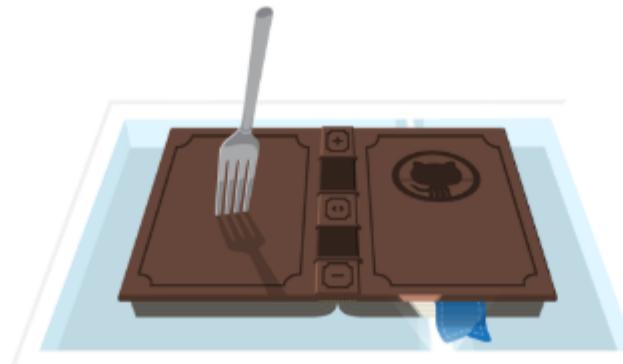
자신의 닉네임을 클릭합니다.

# 1. Fork하기

Forking 2019androidtp/gitArrange

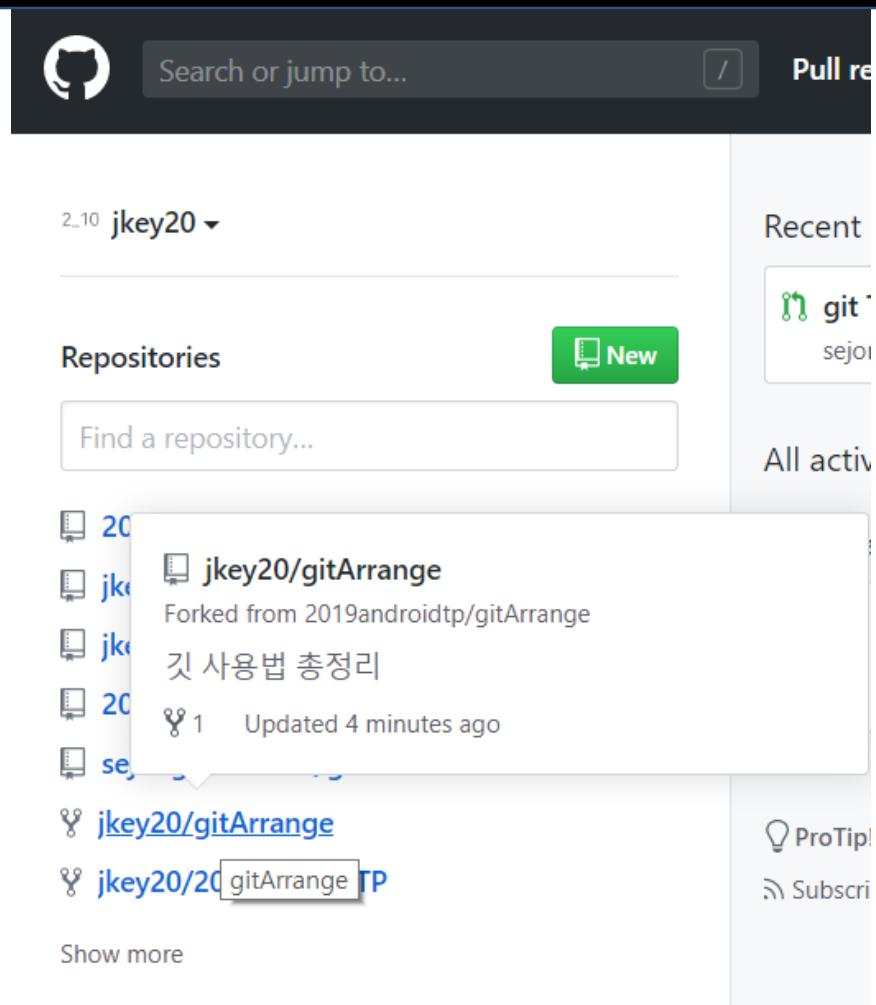
It should only take a few seconds.

⟳ Refresh



정상적으로 포크를 했으면 이 창이 나옵니다.

# 2.Clone 만들기



다시 깃허브 메인 페이지로 와서 자신이 포크를 한 페이지를 들어갑니다.  
왼쪽에 두 갈래로 갈라진 모습이 나오는게 자신이 포크를 한 페이지입니다.

# 2.Clone 만들기

jkey20 / gitArrange  
forked from 2019androidtp/gitArrange

Watch 0 Star 0 Fork 1

Code Pull requests 0 Projects 0 Wiki Security Insights Settings

깃 사용법 총정리 Edit

Manage topics

1 commit 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find File Clone or download ▾

This branch is even with 2019androidtp:master.

2..10 jkey20 Initial commit Latest commit b9f068f 5 minutes ago

README.md Initial commit 5 minutes ago

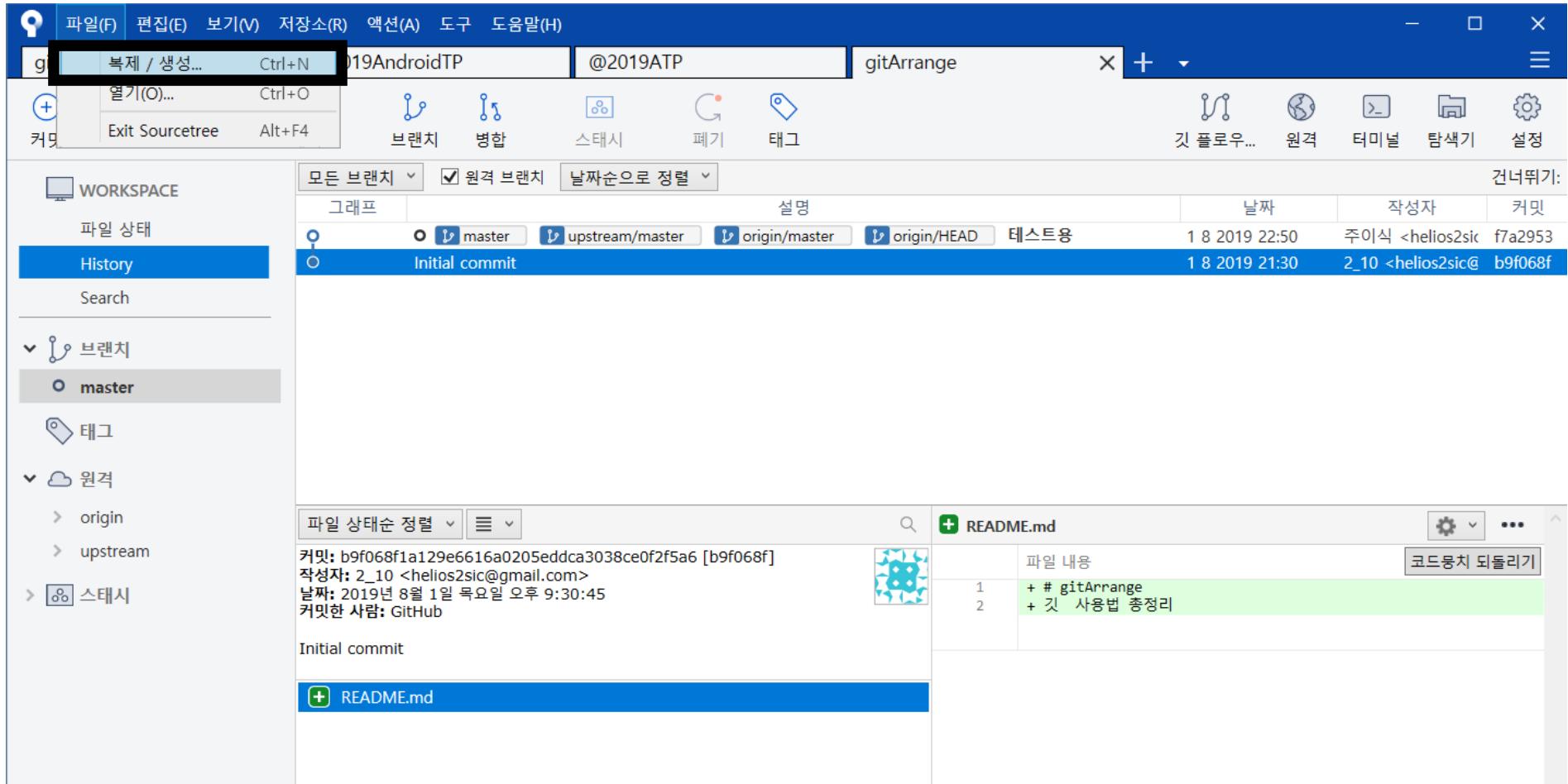
README.md

gitArrange

깃 사용법 총정리

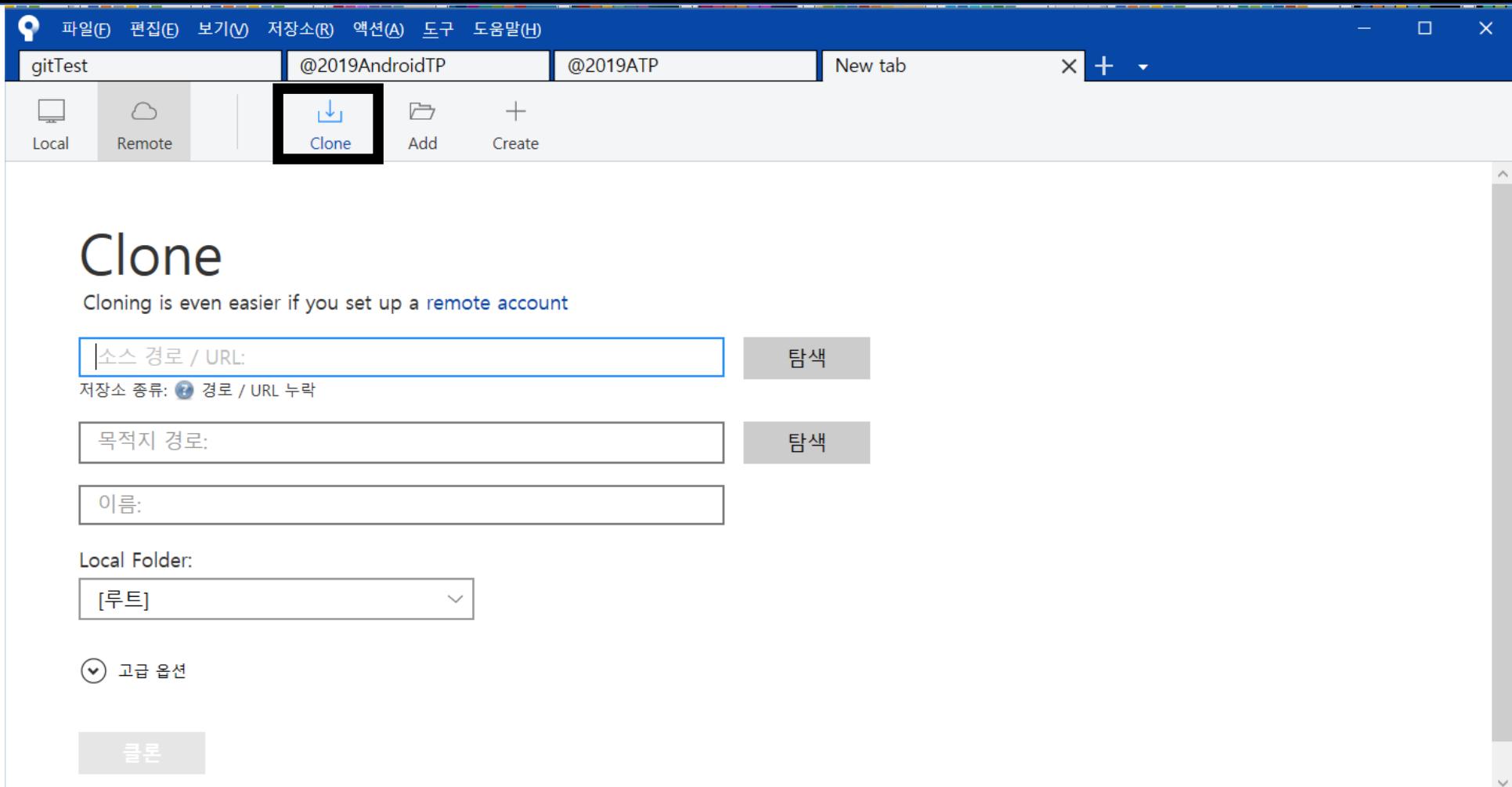
네모 쳐진 부분이 나오면 자신이 포크를 한 페이지에 정상적으로 접속했다는 것입니다.

# 2.Clone 만들기



다시 소스트리로 돌아와서 왼쪽 상단에 복제 생성을 누릅니다.

# 2.Clone 만들기



이 화면이 나오면 제대로 하고 있는 것 입니다.  
만약 다르게 나온다면 네모 친 Clone을 누르면 이 화면이 나옵니다.

# 2.Clone 만들기

jkey20 / gitArrange  
forked from 2019androidtp/gitArrange

Watch 0 Star 0 Fork 1

Code Pull requests 0 Projects 0 Wiki Security Insights Settings

깃 사용법 총정리 Edit

Manage topics

1 commit 1 branch 0 releases 1 contributor

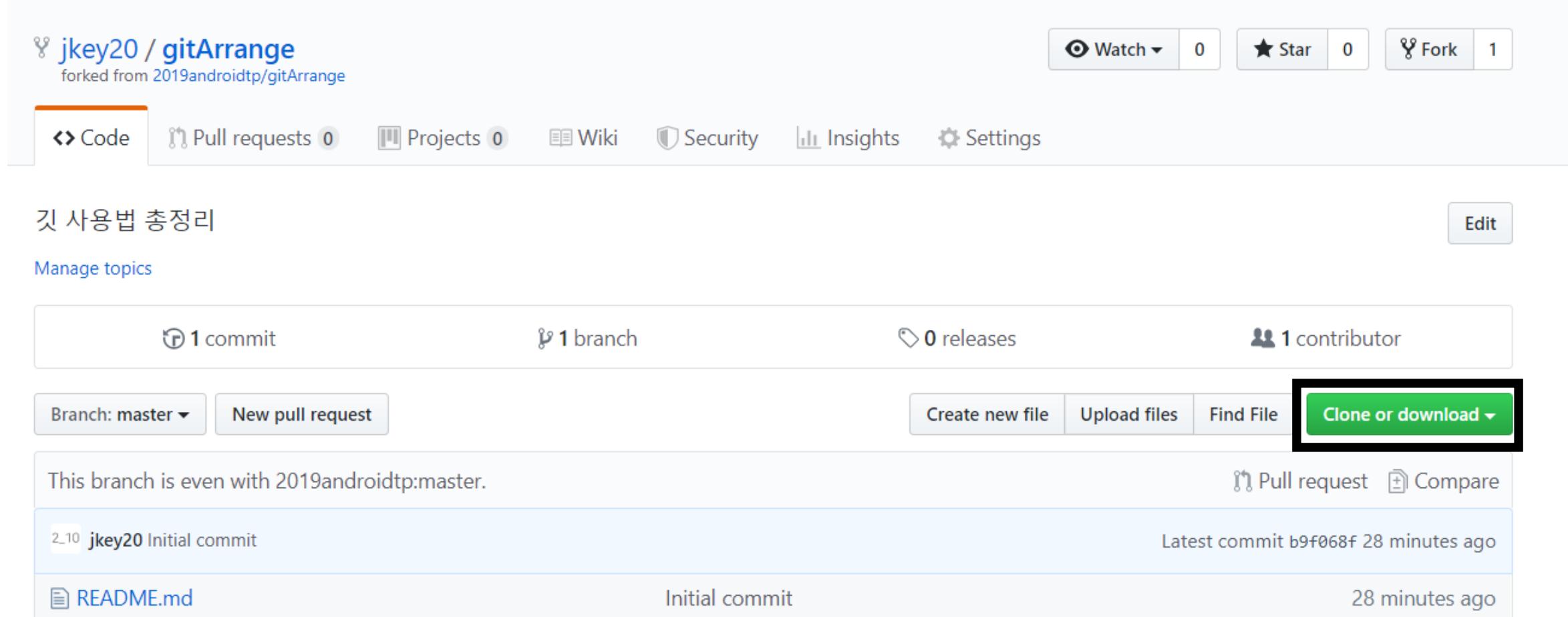
Branch: master New pull request Create new file Upload files Find File Clone or download

This branch is even with 2019androidtp:master.

Pull request Compare

2.10 jkey20 Initial commit Latest commit b9f068f 28 minutes ago

README.md Initial commit 28 minutes ago



다시 포크를 한 페이지에 들어가서 네모 친 버튼을 누릅니다.

# 2.Clone 만들기

The screenshot shows a GitHub repository page for 'jkey20 / gitArrange'. The page includes navigation tabs for Code, Pull requests, Projects, Wiki, Security, Insights, and Settings. Below the tabs, there's a section for '깃 사용법 총정리' (Git Usage Summary) with an 'Edit' button. The summary shows 1 commit, 1 branch, 0 releases, and 1 contributor. A dropdown menu for the branch 'master' is open, showing options like 'New pull request'. On the right, there's a 'Clone or download' button, which is highlighted with a black rectangle. A tooltip for 'Clone with HTTPS' is visible, containing the URL 'https://github.com/jkey20/gitArrange.git' and a copy icon. Other download options shown are 'Open in Desktop' and 'Download ZIP'.

jkey20 / gitArrange  
forked from 2019androidtp/gitArrange

Watch 0 Star 0 Fork 1

Code Pull requests 0 Projects 0 Wiki Security Insights Settings

깃 사용법 총정리 Edit

Manage topics

1 commit 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find File Clone or download

This branch is even with 2019androidtp:master.

2\_10 jkey20 Initial commit

README.md Initial commit

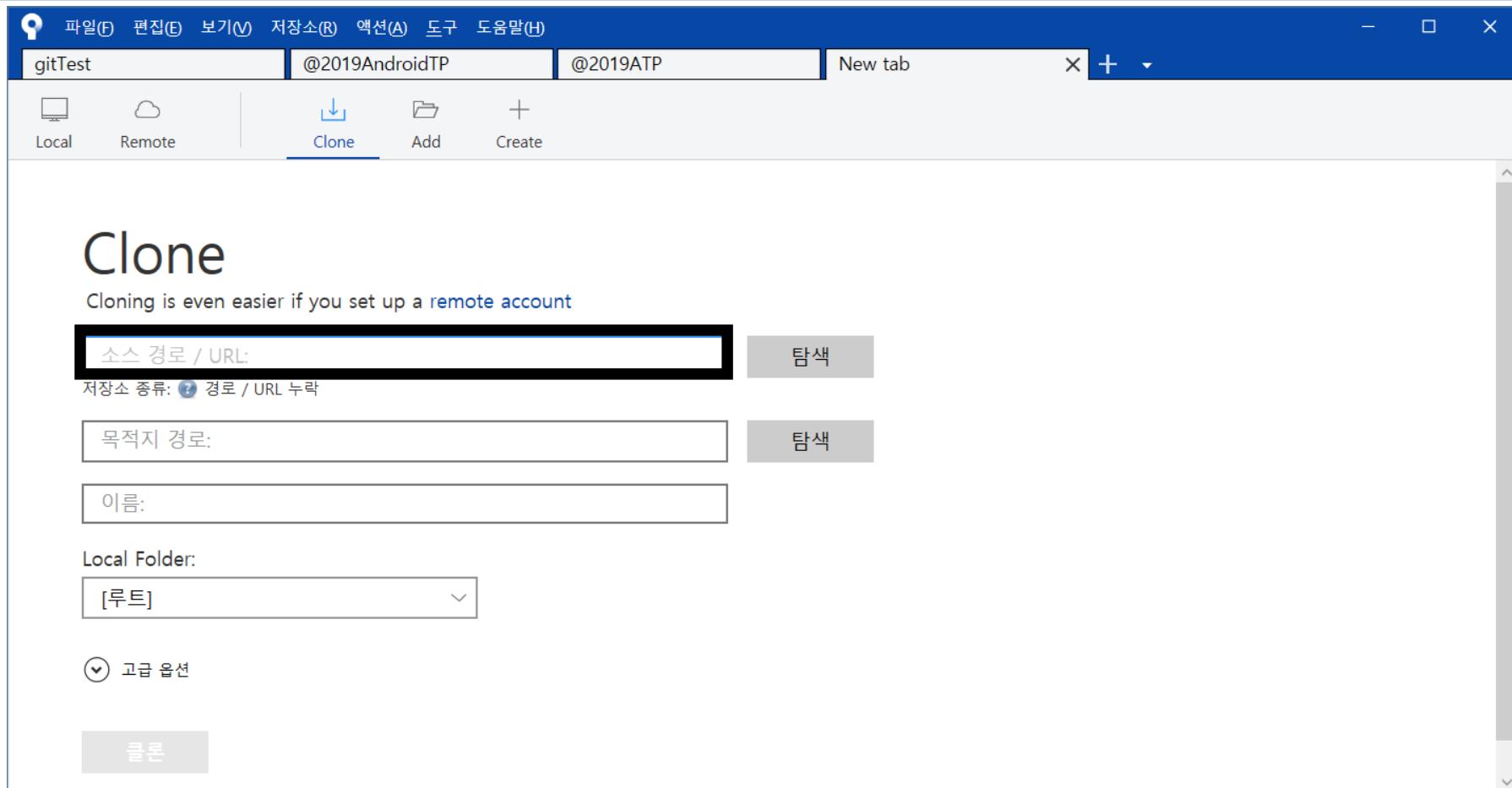
README.md

Clone with HTTPS Use SSH  
Use Git or checkout with SVN using the web URL.  
https://github.com/jkey20/gitArrange.git

Open in Desktop Download ZIP

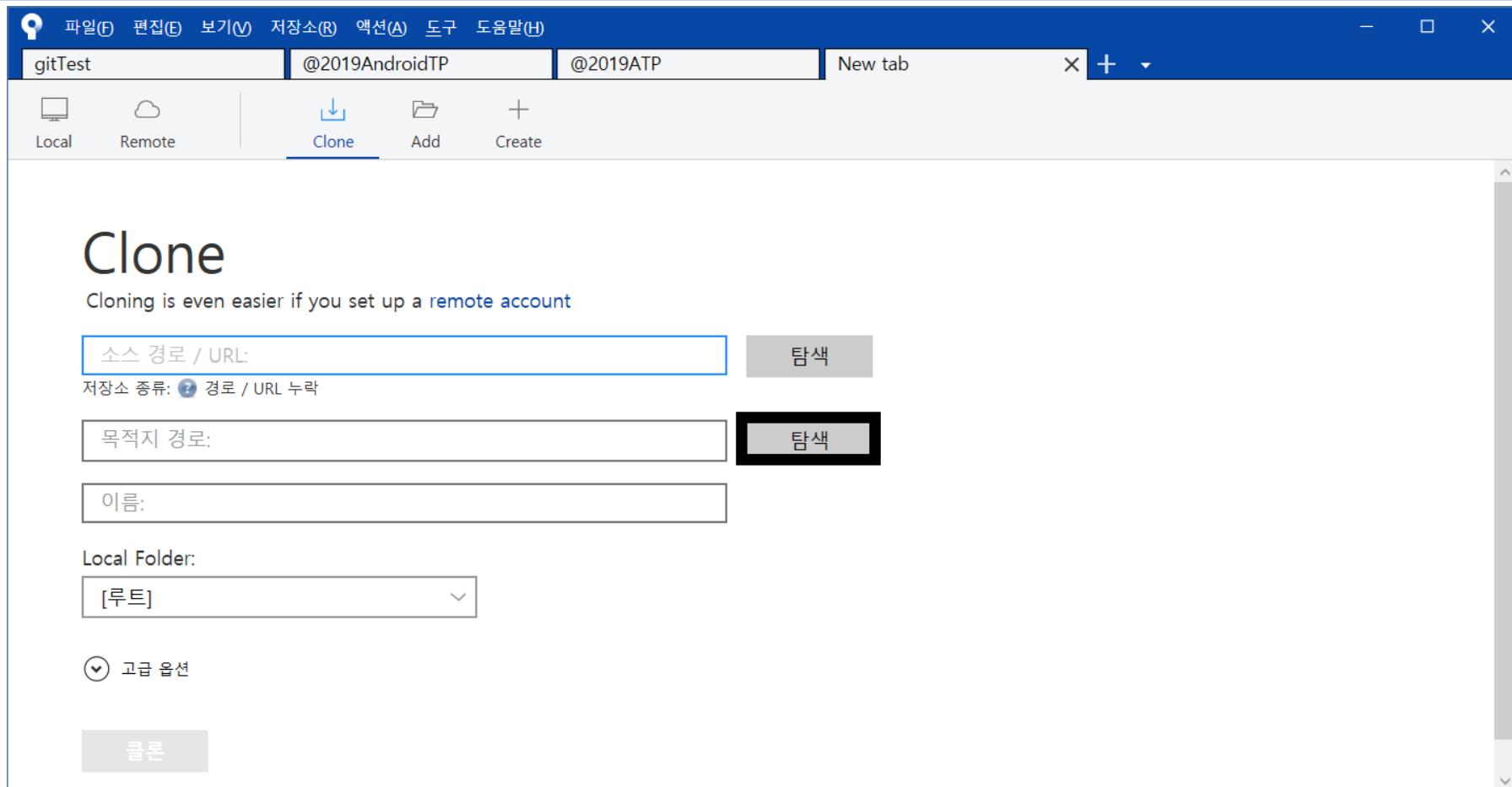
다음과 같이 창이 나오면 네모 친 버튼을 누릅니다.

# 2.Clone 만들기



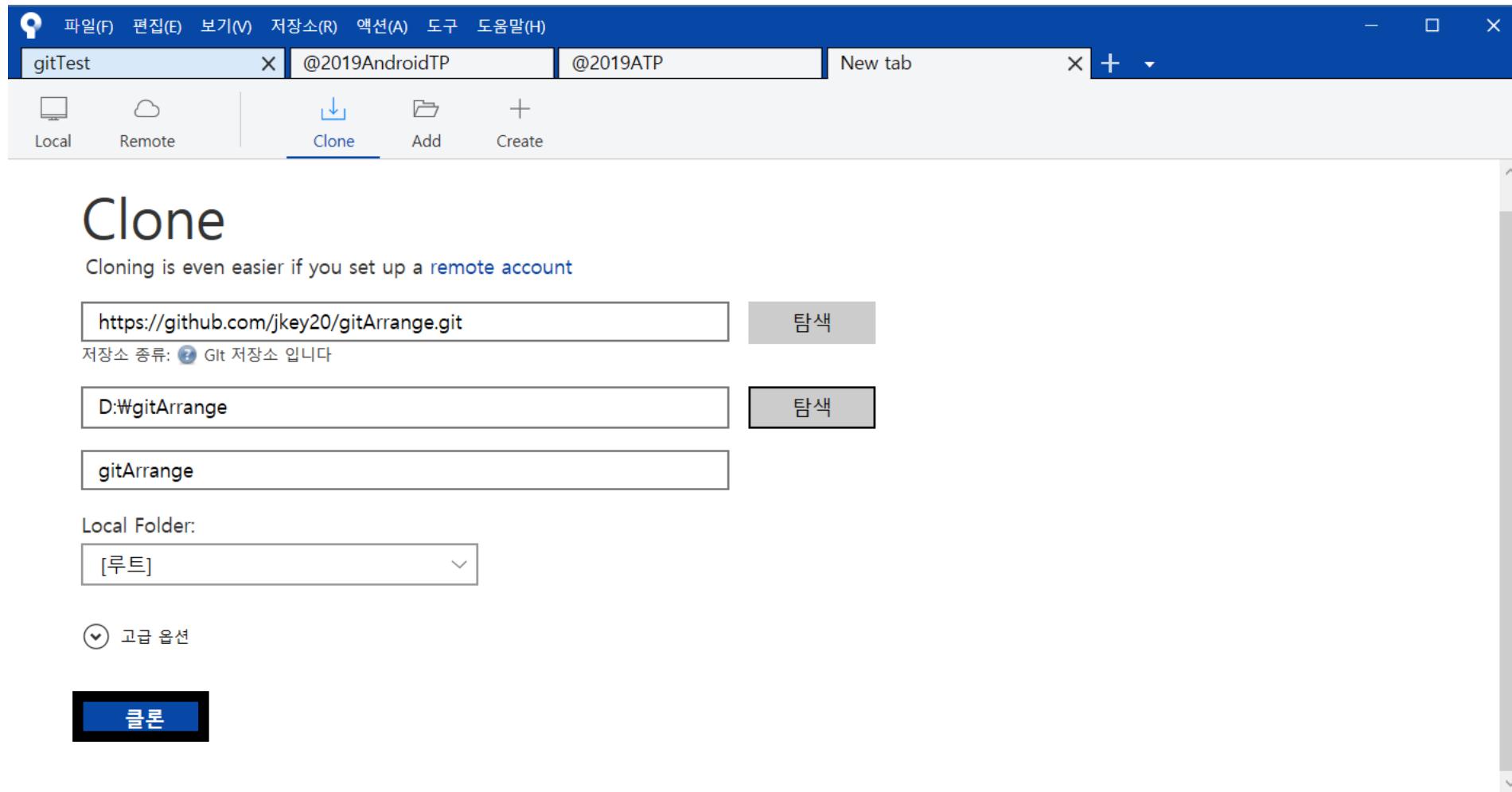
다시 소스트리로 돌아가서 네모친 부분에 **ctrl + v** 를 누릅니다.

# 2.Clone 만들기



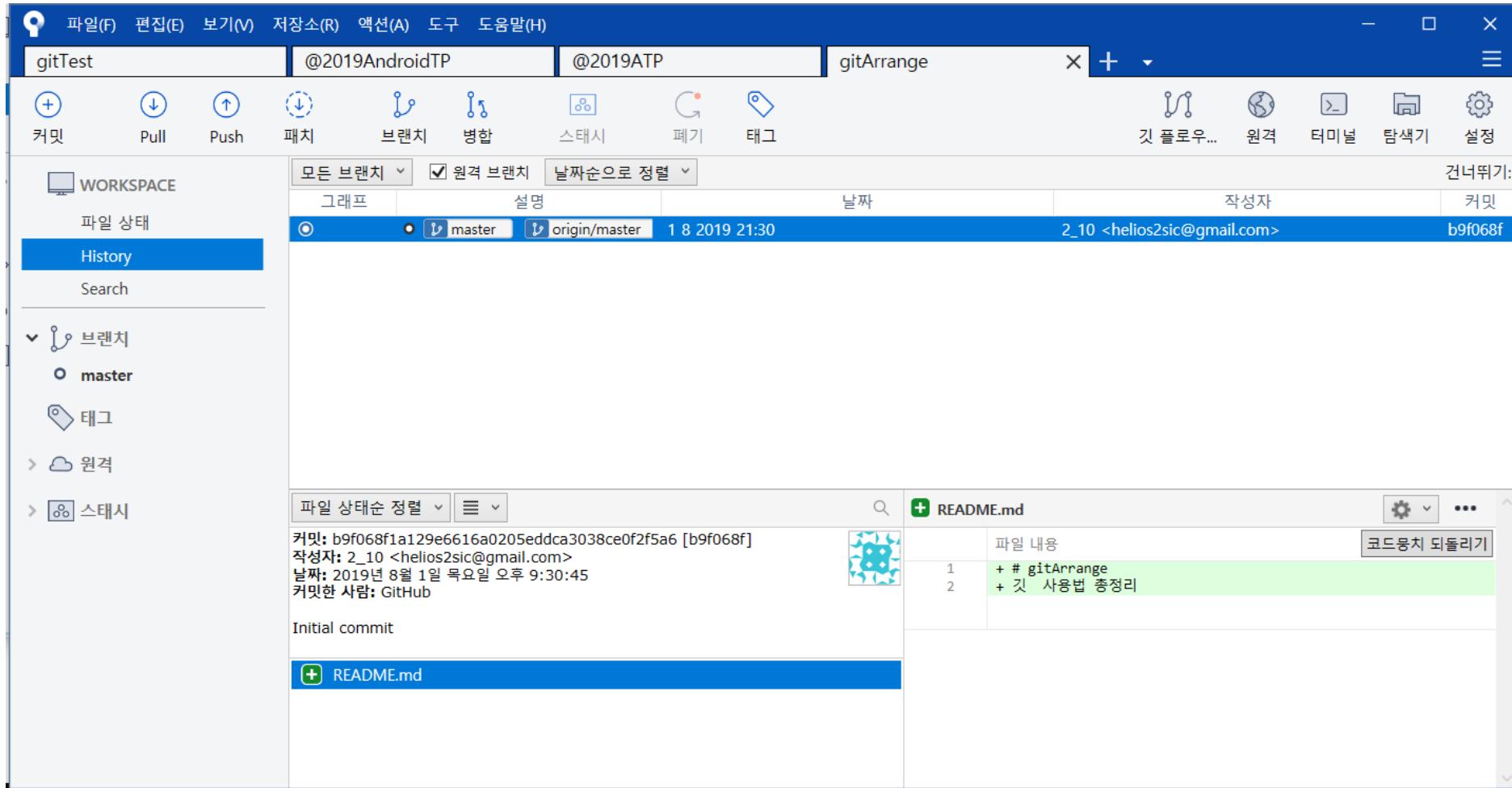
그 다음 탐색을 눌러 자신이 저장할 파일들의 위치를 지정해 줍니다.

# 2.Clone 만들기



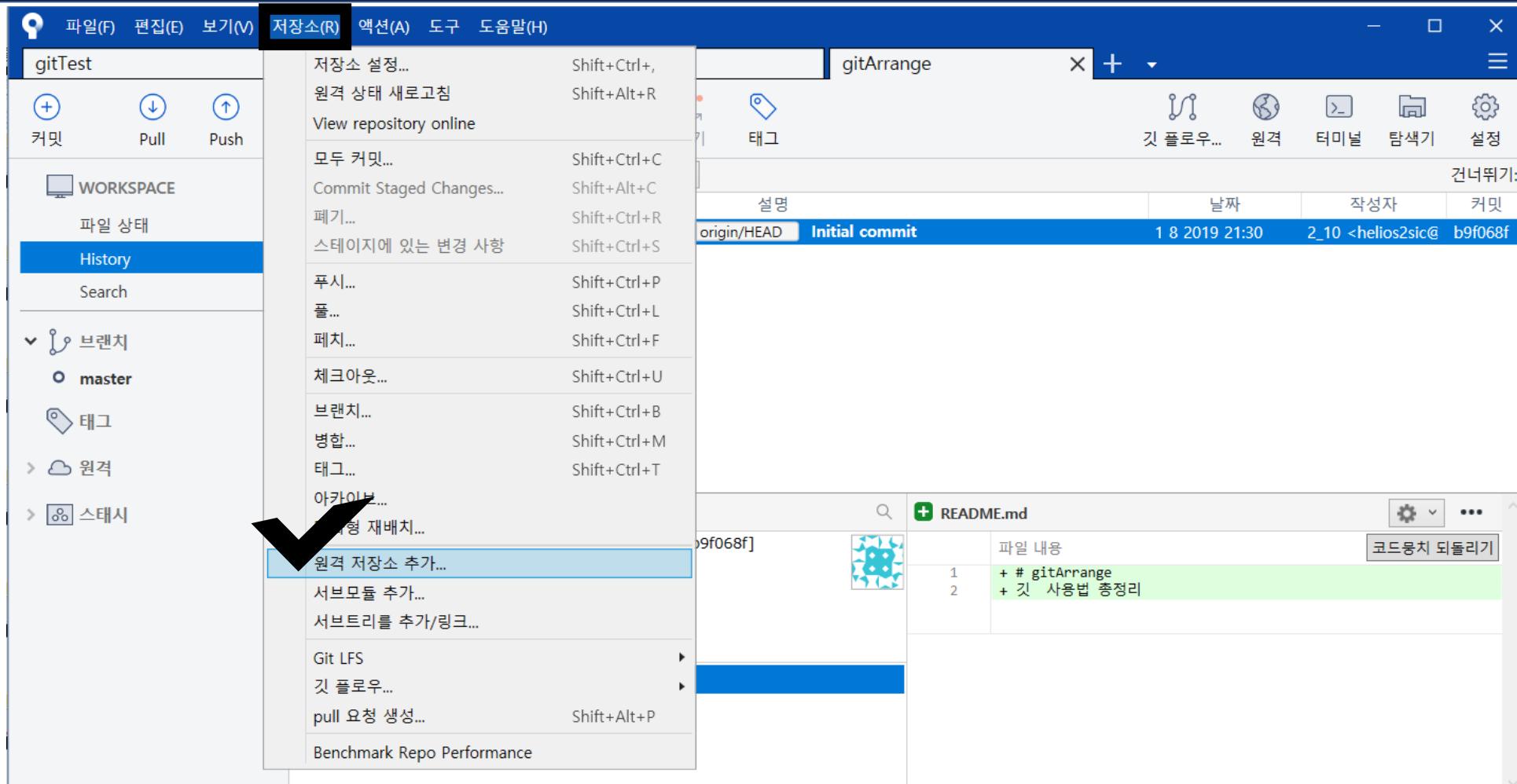
다 입력을 했으면 클론을 눌러줍니다.

# 2. Clone 만들기



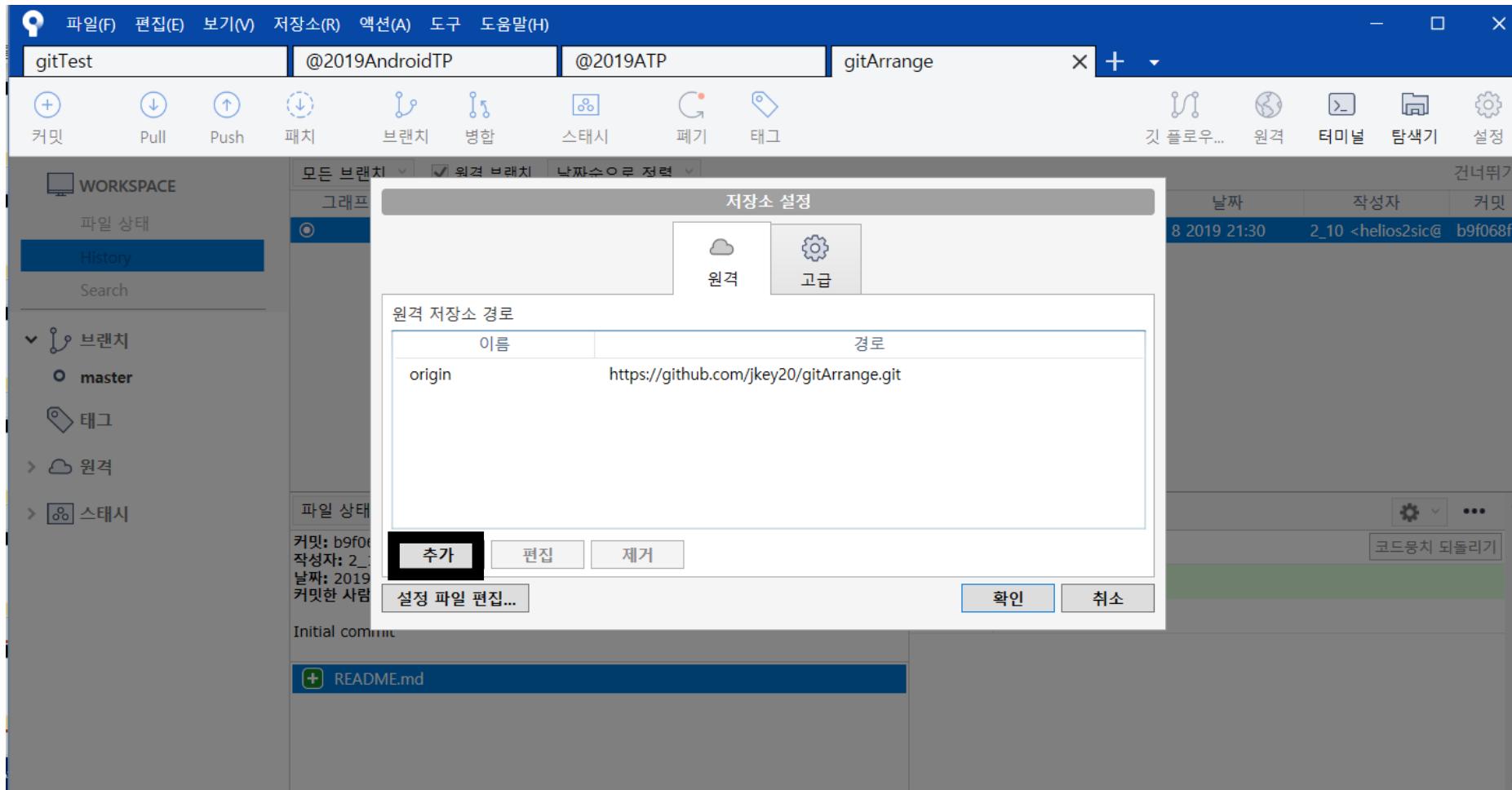
정상적으로 클론을 만들면 다음과 같이 나옵니다.

# 3.Upstream 설정



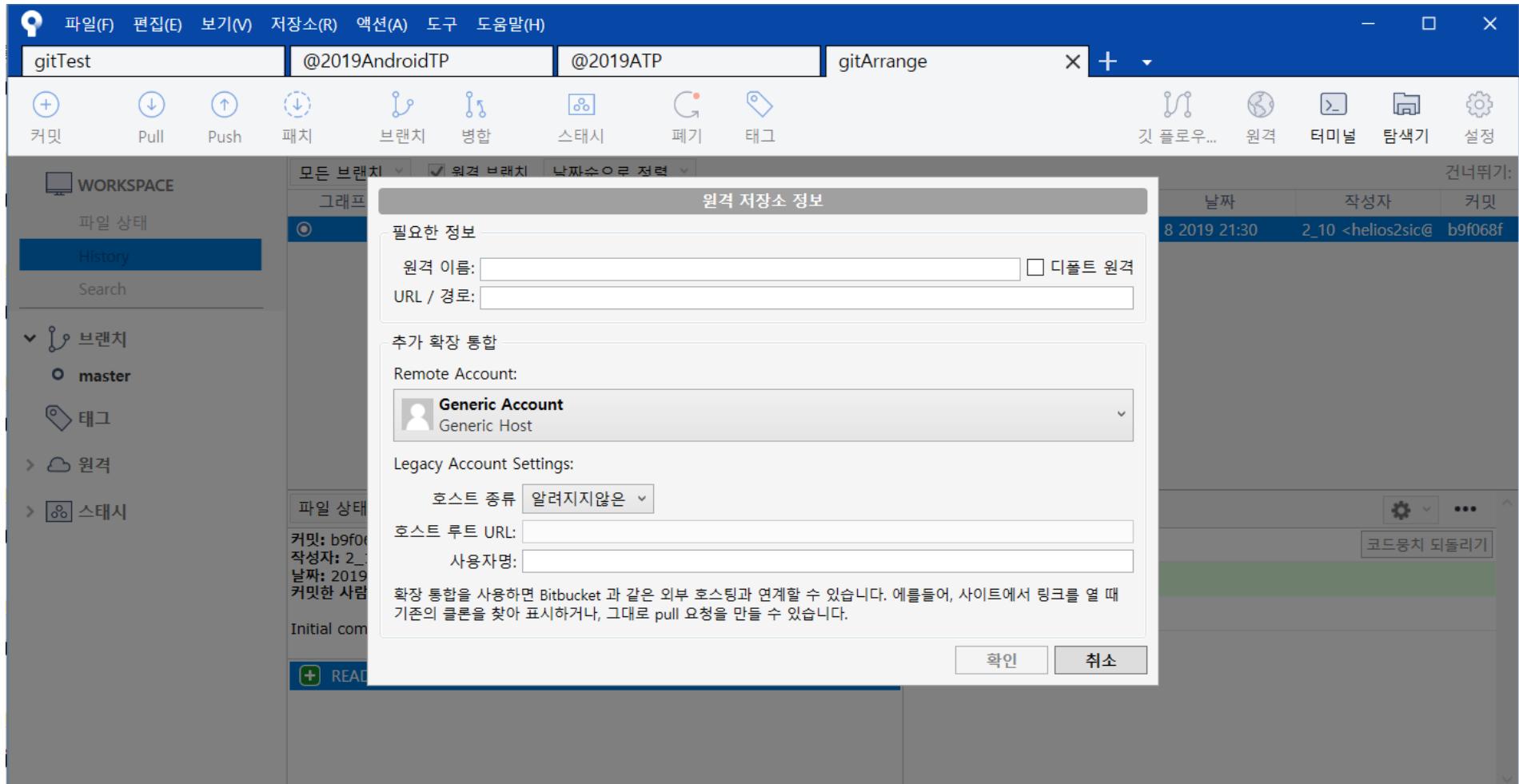
그 다음은 우리가 파일들을 공유하고 저장하기 위한 원격 저장소를 설정해주어야 합니다.  
왼쪽 상단에 저장소를 누르고 원격 저장소 추가를 누릅니다.

# 3.Upstream 설정



다음과 같은 창이 뜨면 추가를 누릅니다.

# 3.Upstream 설정



다음과 같은 창이 뜨면 잘 하고 있는 겁니다.

# 3.Upstream 설정

The screenshot shows a GitHub repository page for 'gitArrange' owned by '2019androidtp'. The repository has 1 commit, 1 branch, 0 releases, and 1 contributor. The 'Clone or download' button is highlighted with a green box and labeled '1'. A dropdown menu is open, showing the 'Clone with HTTPS' URL: <https://github.com/2019androidtp/gitArra>, which is also highlighted with a black box and labeled '2'.

깃 사용법 총정리

Manage topics

1 commit 1 branch 0 releases 1 contributor

Branch: master New pull request

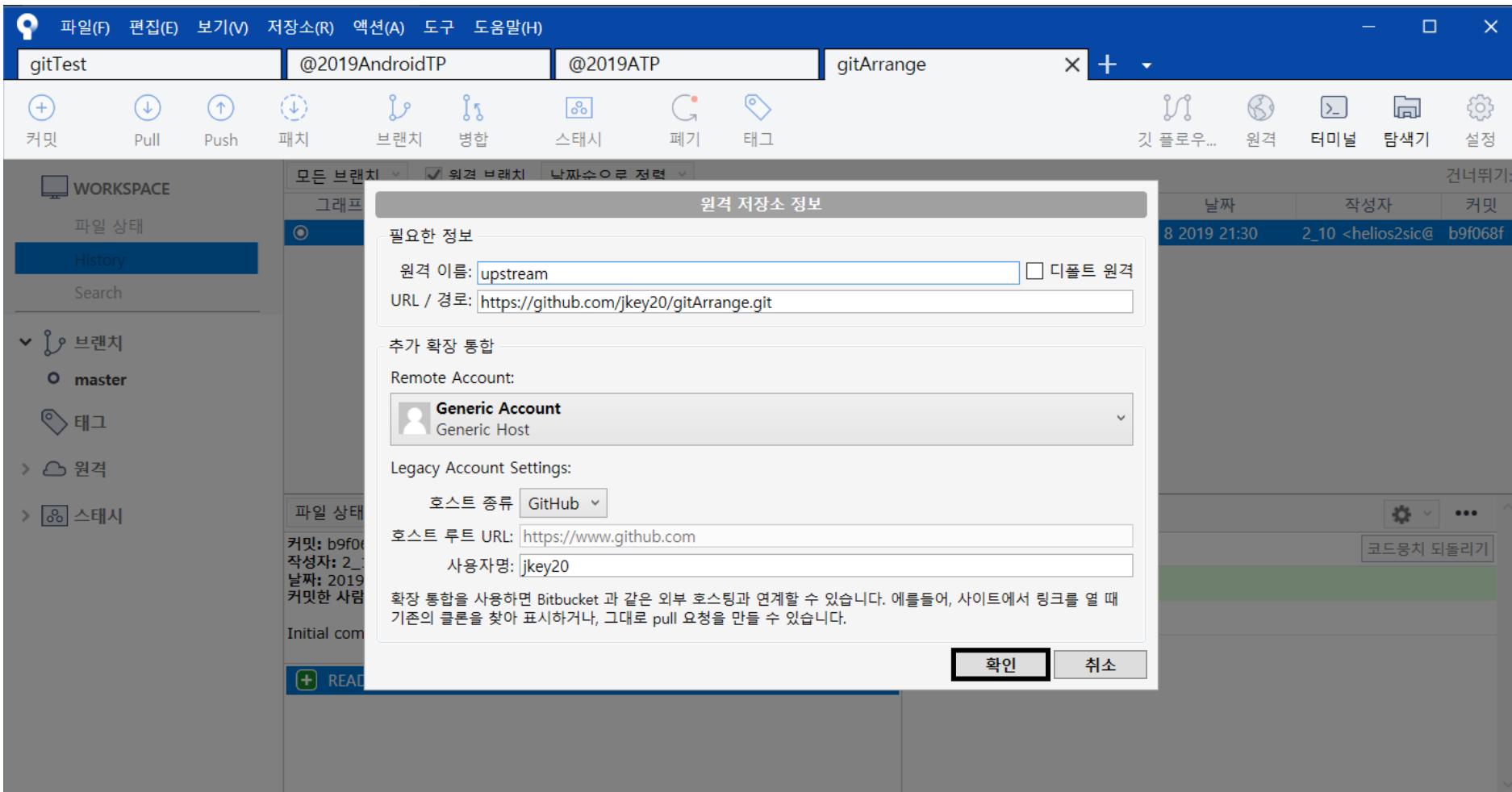
Clone with HTTPS Use SSH

https://github.com/2019androidtp/gitArra

Open in Desktop Download ZIP

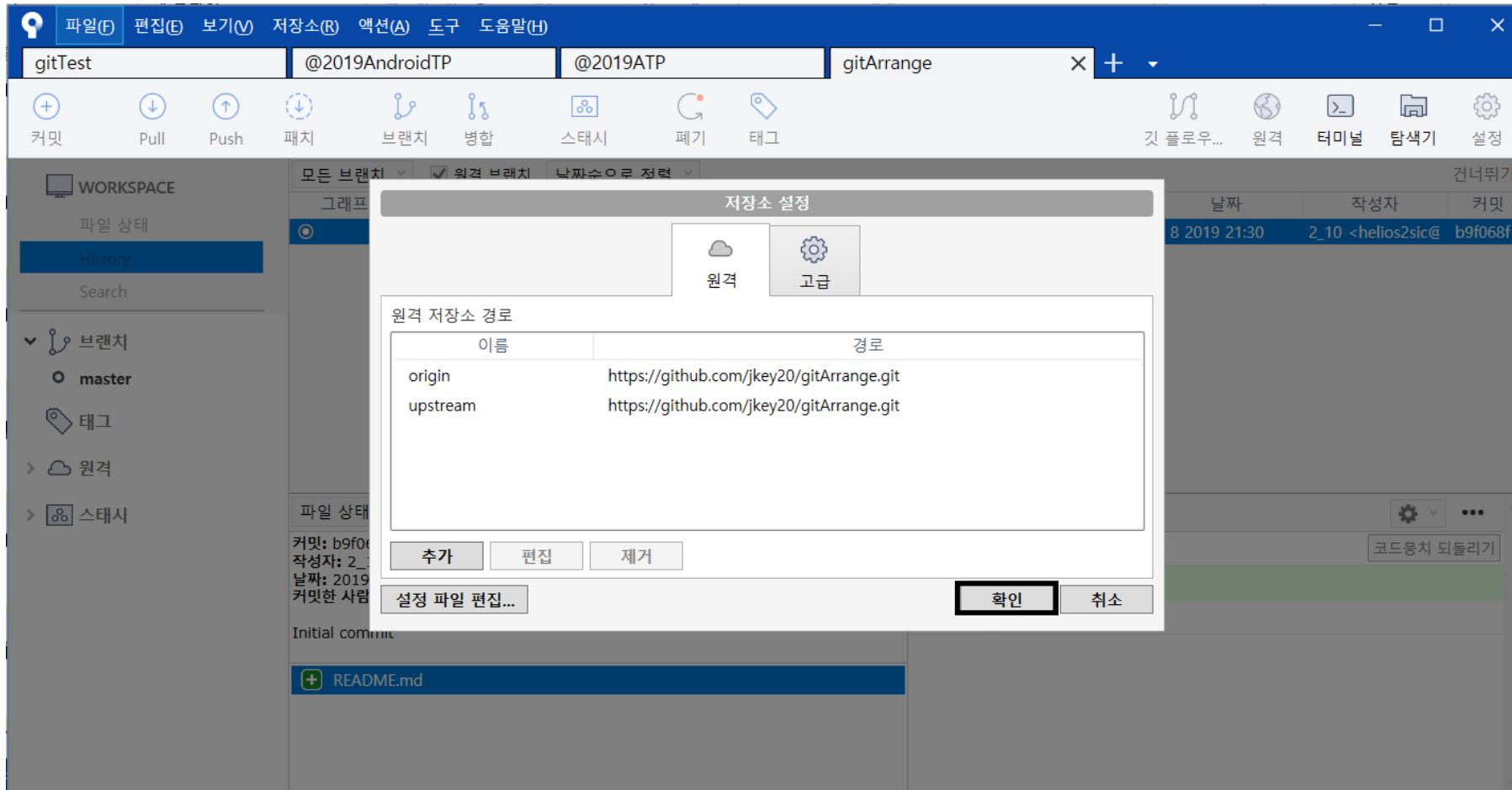
다시 포크를 뜯 페이지로 돌아가서 1번 네모 칸을 누른 뒤 그 다음 2번 네모 칸을 누릅니다.

# 3.Upstream 설정



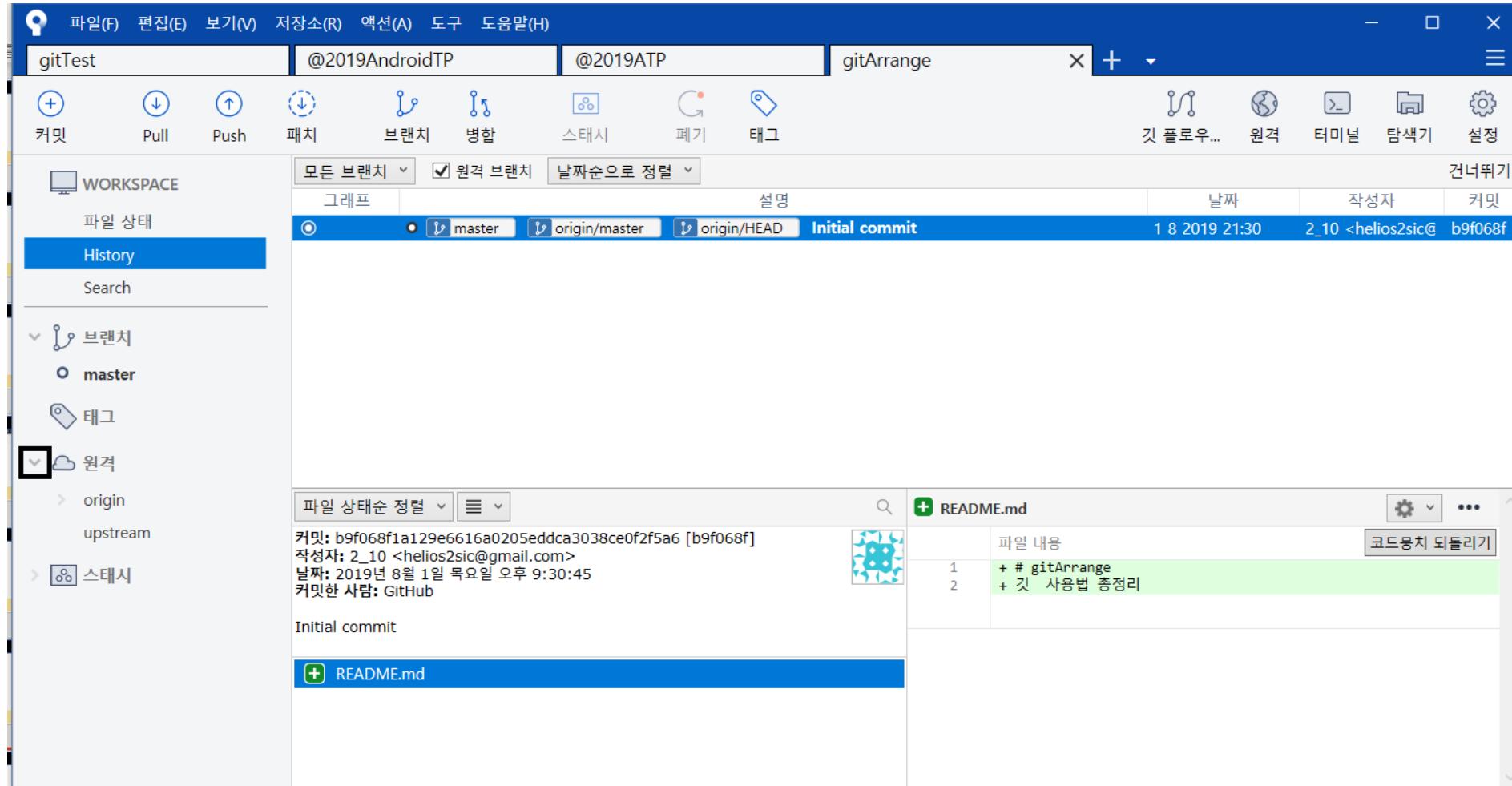
원격 이름에 upstream을 입력하고 URL/경로에 ctrl+v를 해줍니다.  
다 기입했으면 확인버튼을 눌러줍니다.

# 3.Upstream 설정



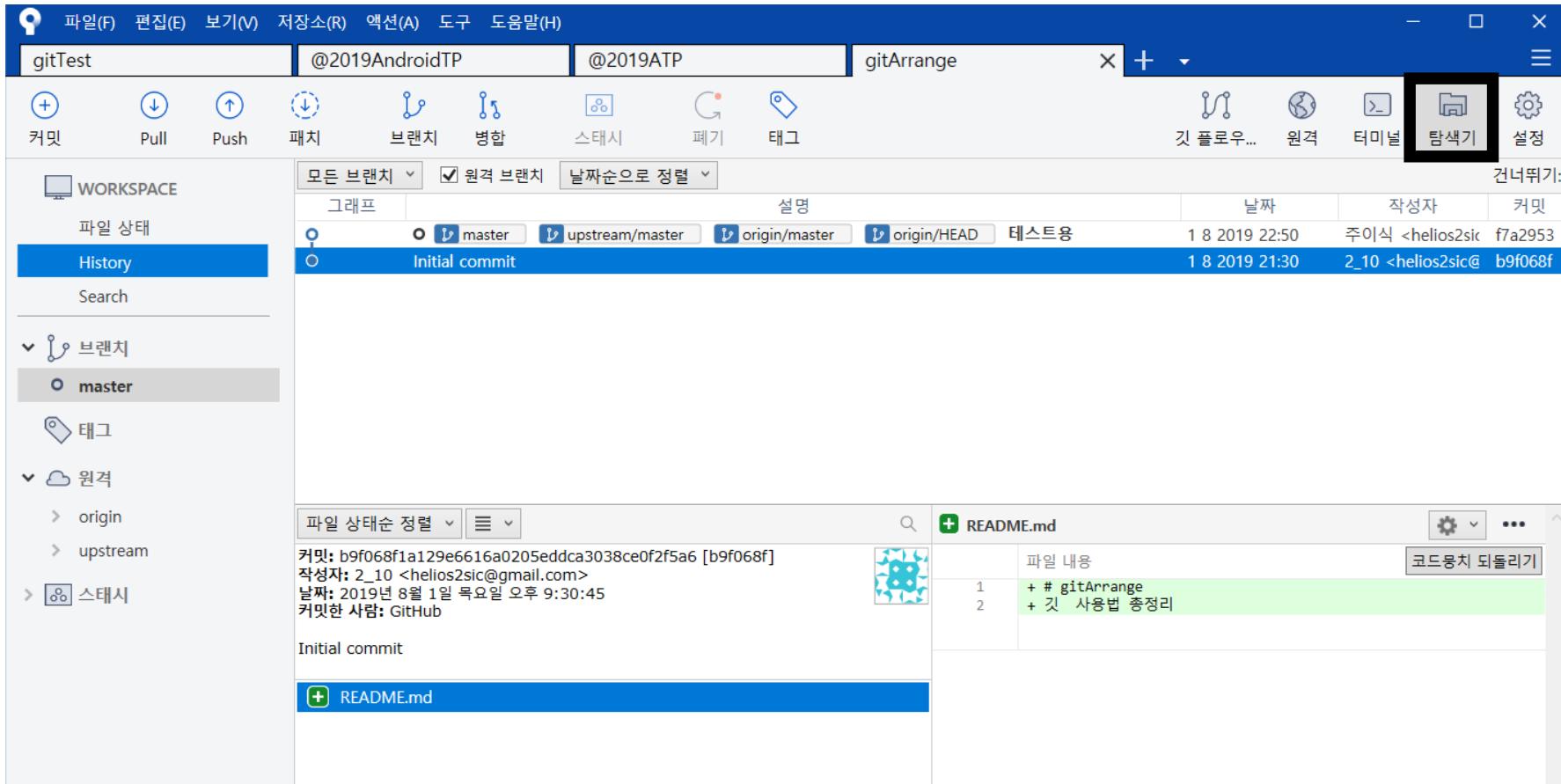
다음과 같이 upstream이 추가되면 확인버튼을 눌러줍니다.

# 3.Upstream 설정



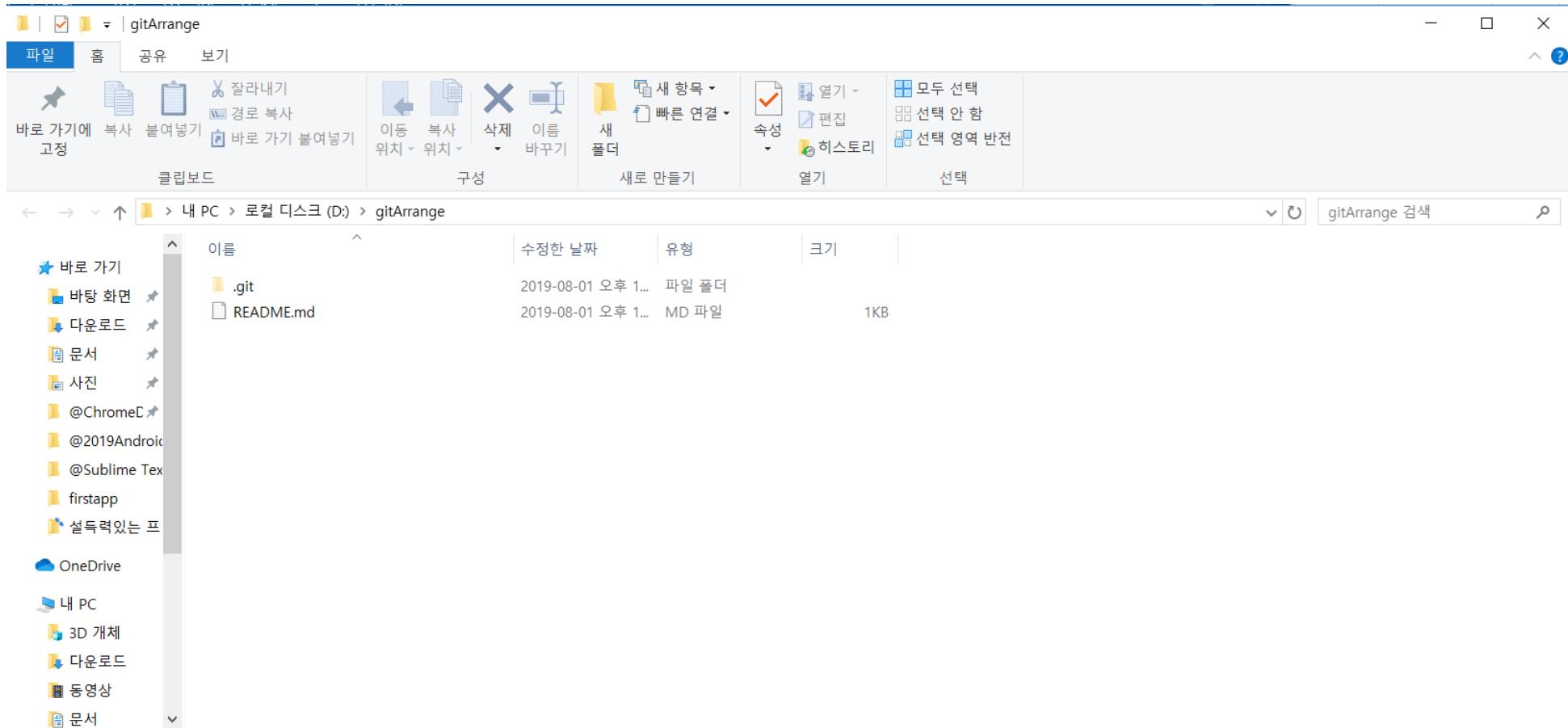
검정색 네모칸에 화살표를 눌러서 upstream이 나오면 정상적으로 추가된 겁니다.

# 4. Pull request 보내기



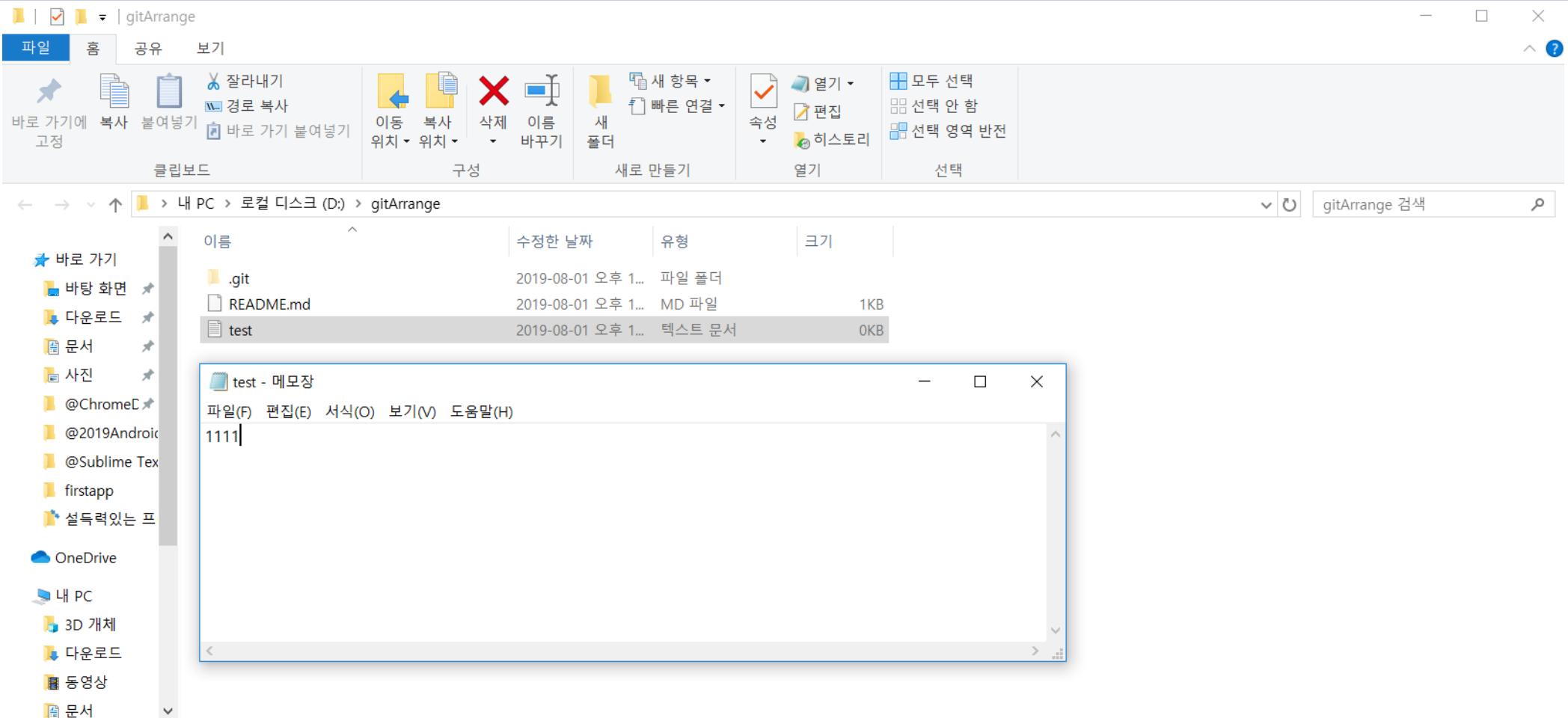
화면 오른쪽 상단에 탐색기를 눌러줍니다.

# 4. Pull request 보내기



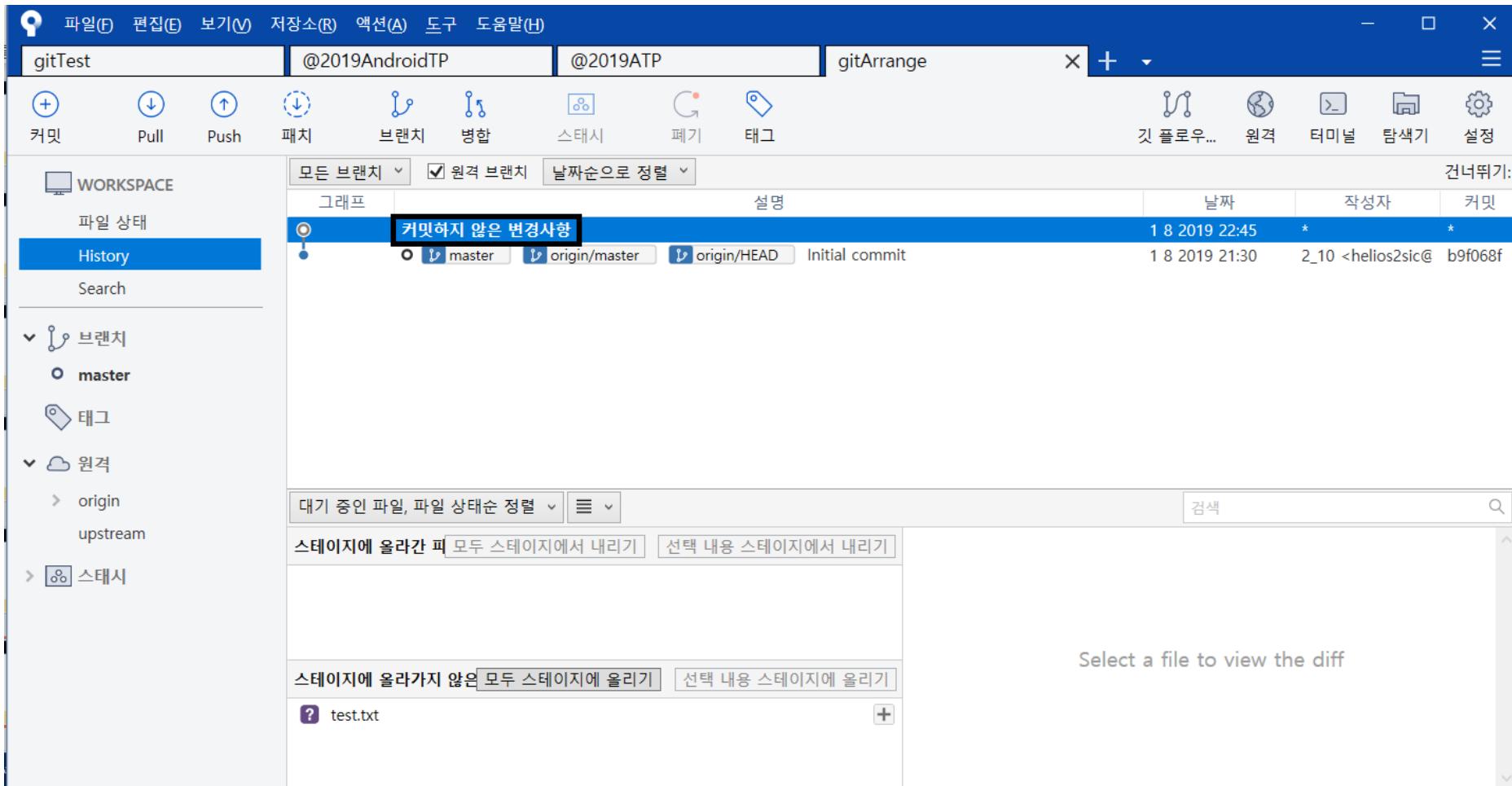
그러면 Clone을 생성할 때 지정해준 위치가 나옵니다. 여기서 테스트용으로 txt파일을 하나 만듭니다.

# 4. Pull request 보내기



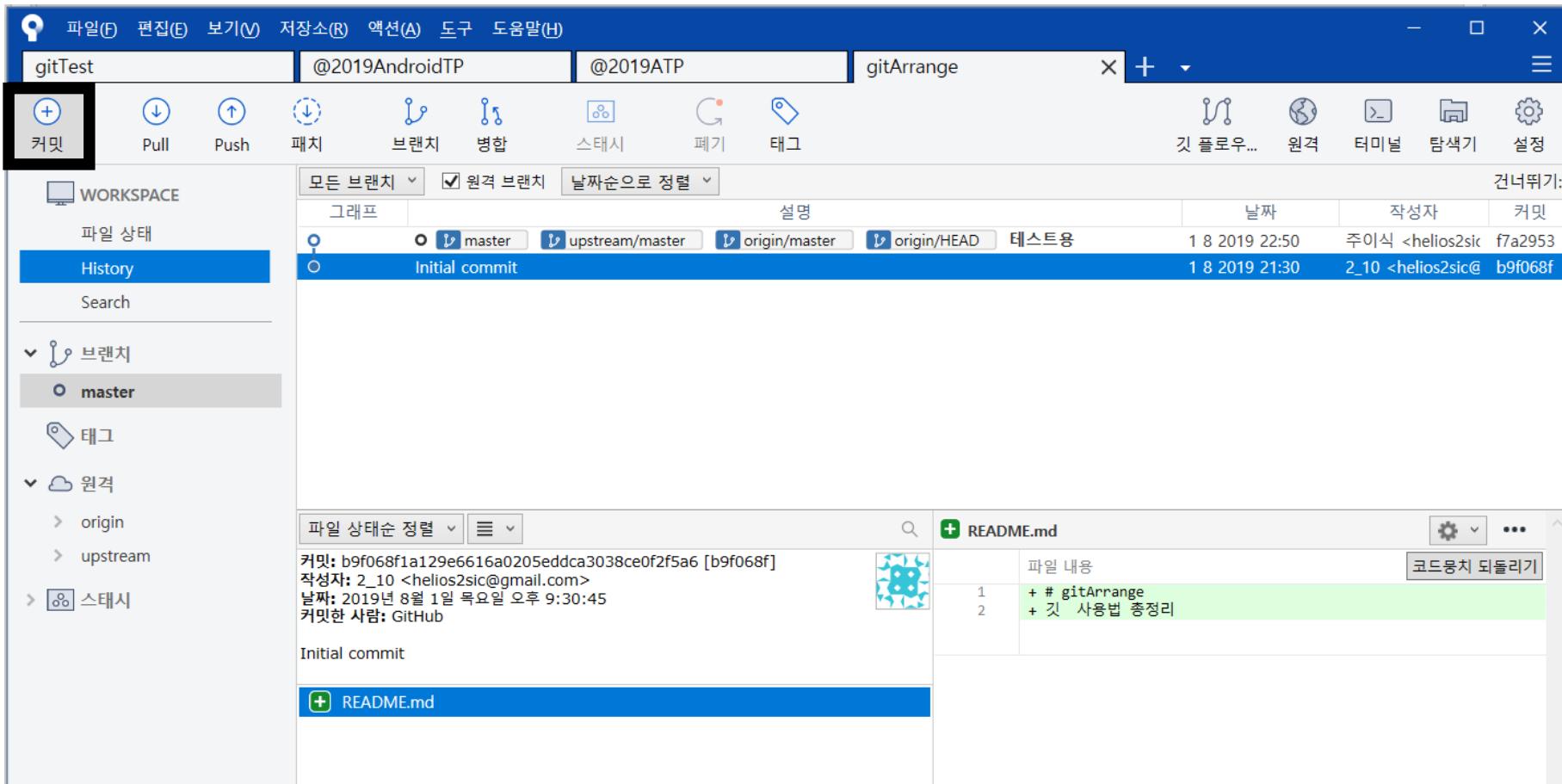
테스트용으로 텍스트 파일 하나를 만들고 저장합니다.

# 4. Pull request 보내기



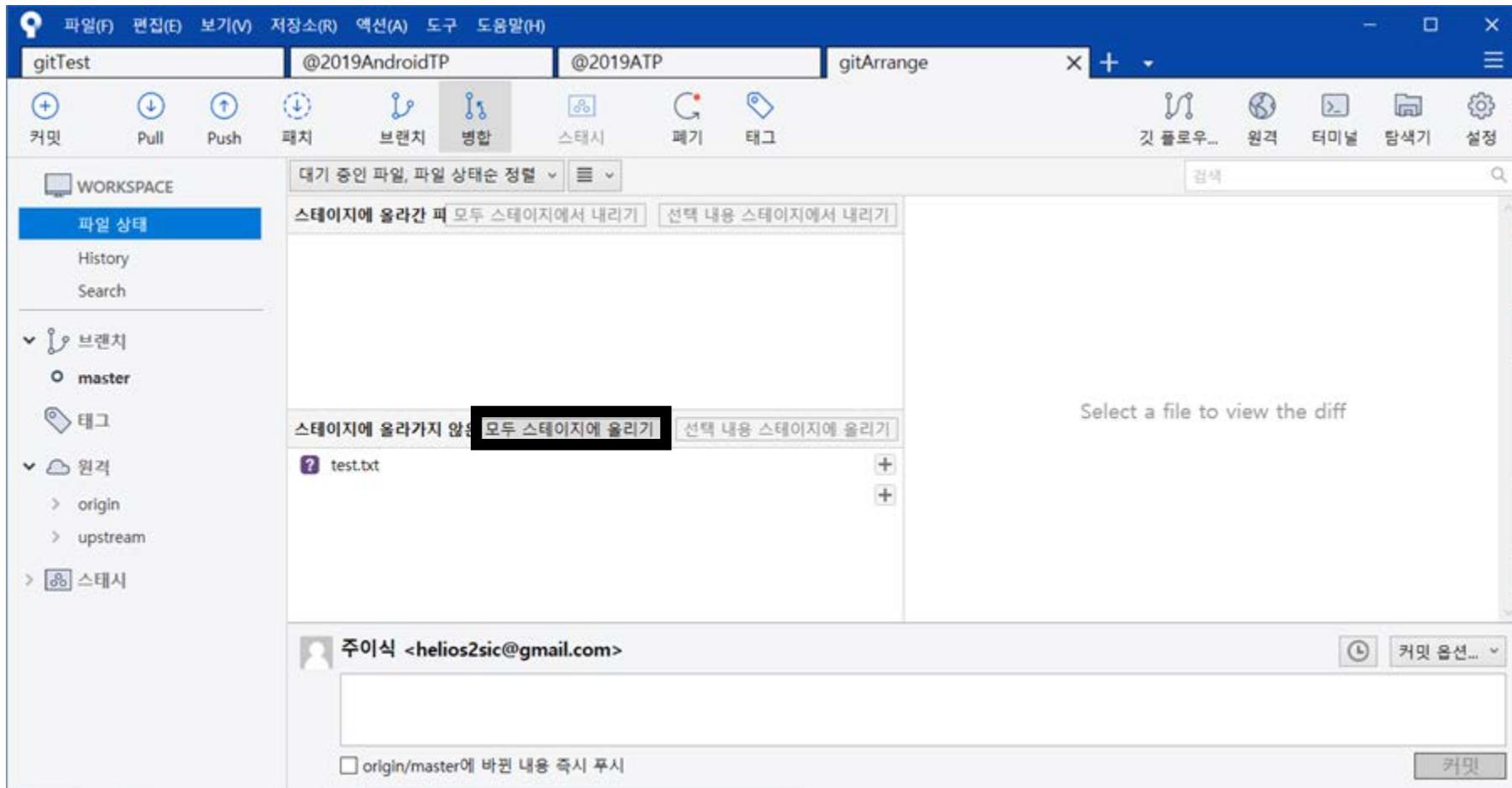
다시 소스트리로 돌아오면 다음과 같이 커밋하지 않은 변경사항 이라고 나옵니다.  
커밋하지 않은 변경사항이 나오지 않을 시 패치를 눌러줍니다.

# 4. Pull request 보내기



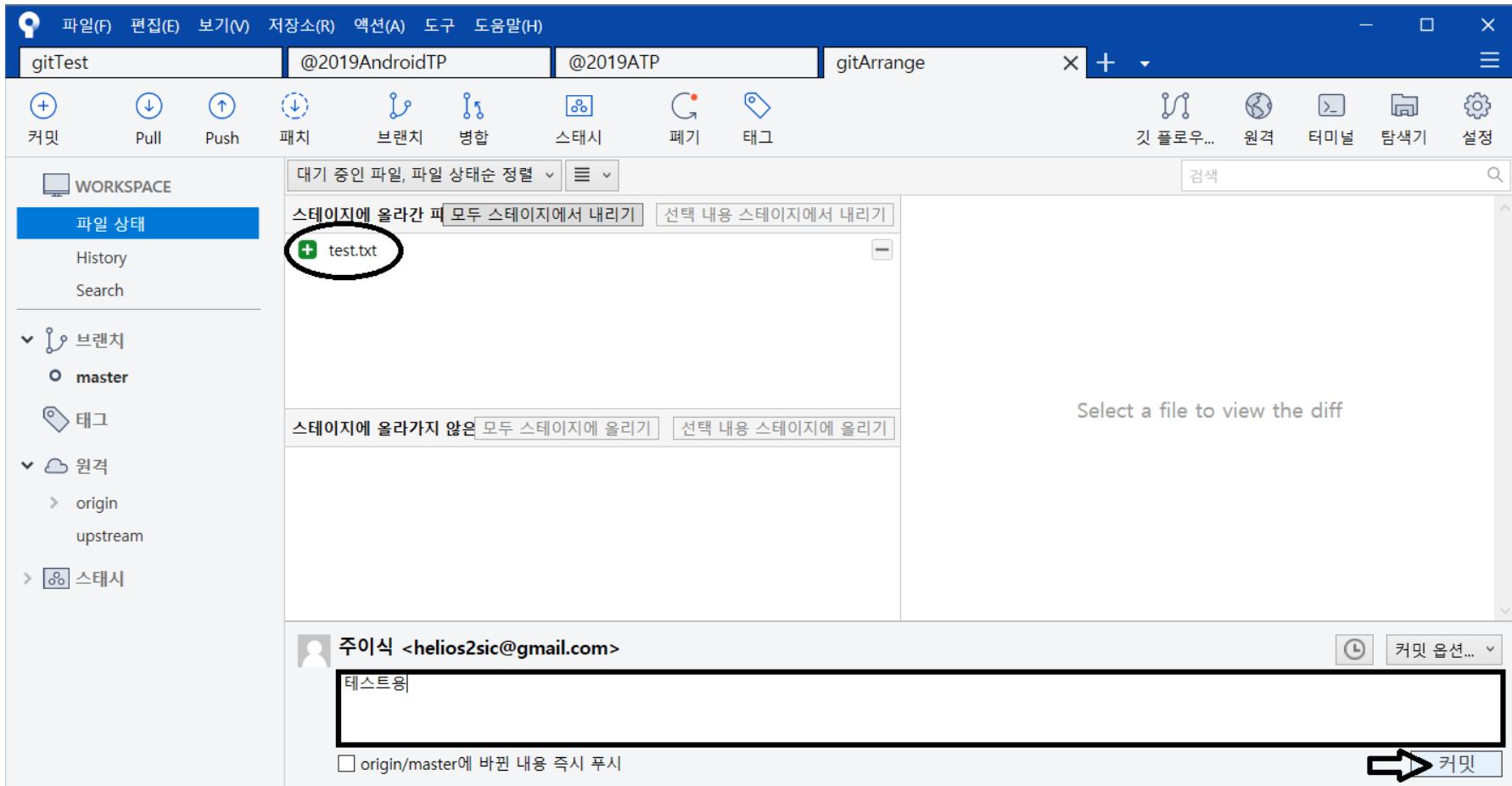
그 후 왼쪽 상단에 커밋을 누릅니다.

# 4. Pull request 보내기



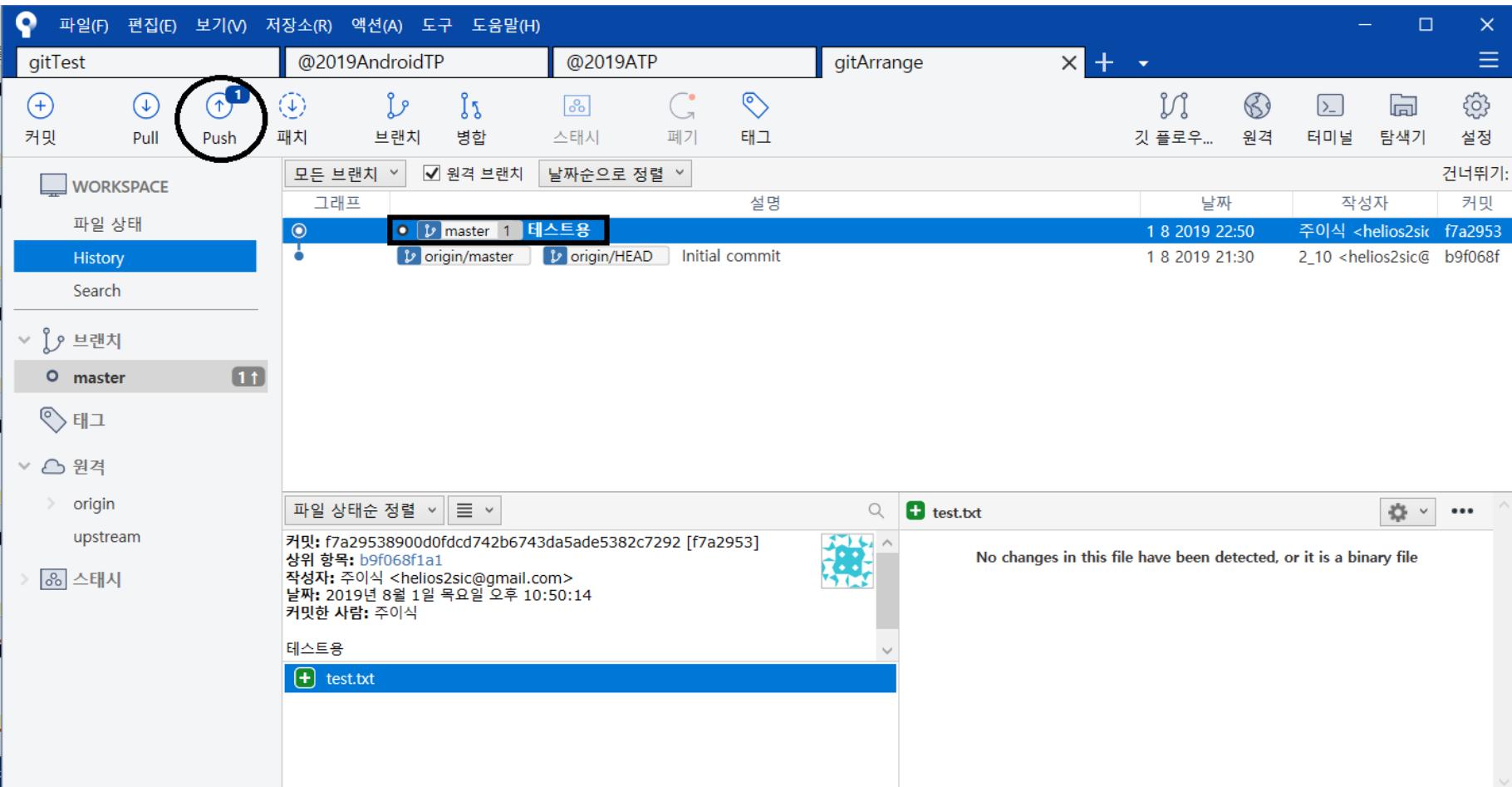
그 후 네모 친 모두 스테이지에 올리기를 누릅니다.

# 4. Pull request 보내기



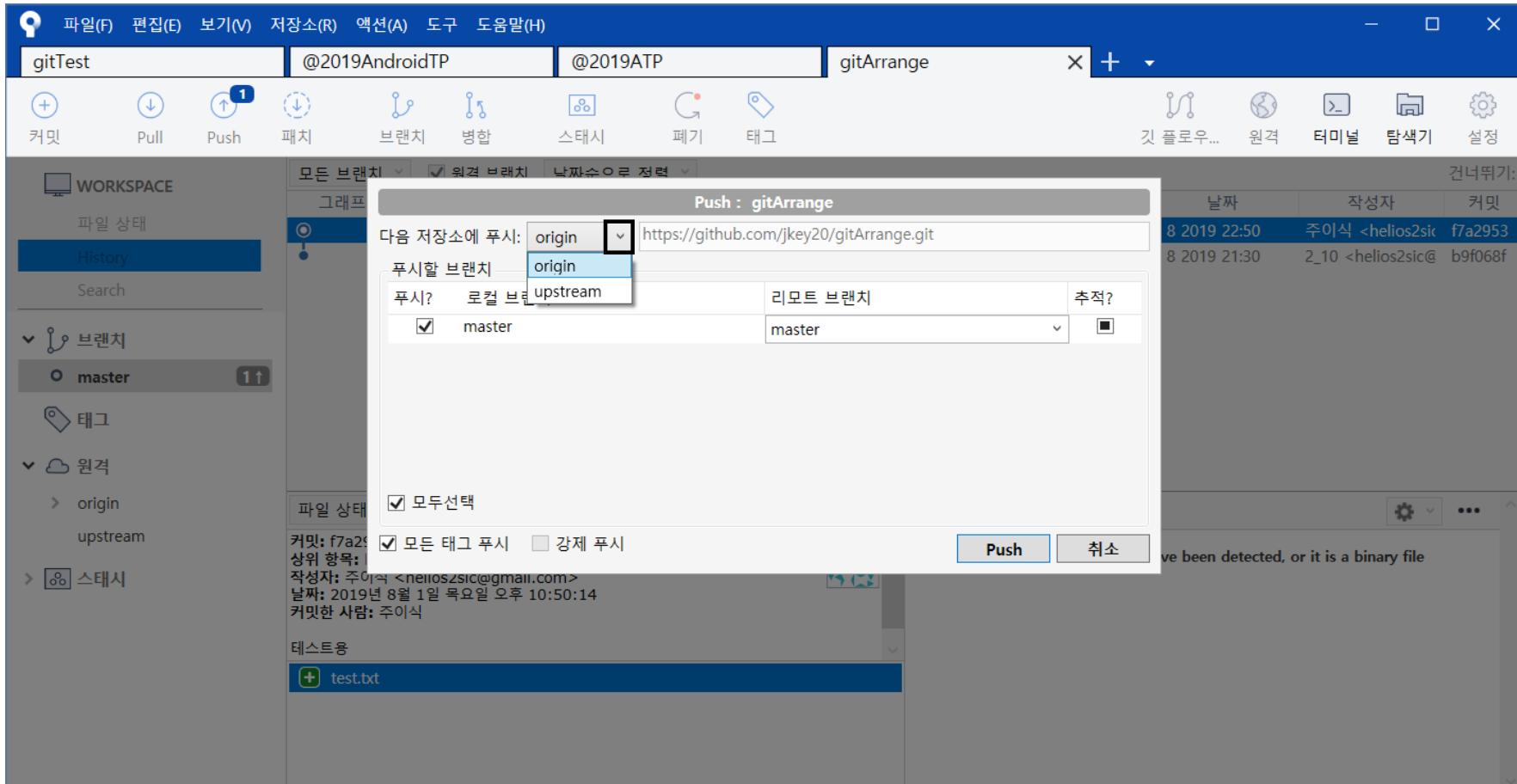
다음과 같이 동그라미 친 부분에 파일이 올라가면 됩니다.  
그 후 아래 네모 칸에 자신이 올린 파일이 무엇을 의미하는지 작성하고 커밋을 누릅니다.

# 4. Pull request 보내기



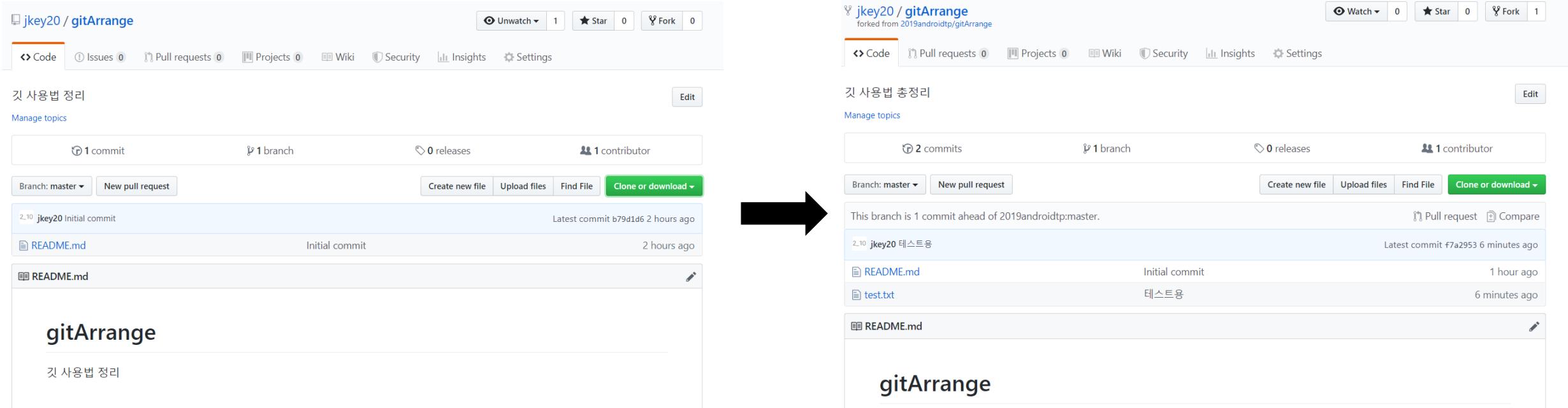
다음과 같이 네모 친 부분에 정상적으로 올라온걸 확인 할 수 있습니다.  
그 후 동그라미 친 칸에 숫자가 생기면 push버튼을 누릅니다.

# 4. Pull request 보내기



Push버튼을 누르면 다음과 같이 나오고 네모 친 부분을 누르면 origin과 upstream이 나옵니다.  
우리는 push할 때 origin만을 쓰므로 origin을 누르고 푸쉬를 해줍니다.

# 4. Pull request 보내기



Push가 끝나면 포크를 뜯 페이지로 다시 돌아갑니다.  
Push하기 전 왼쪽 페이지에서 오른쪽 페이지처럼 commit 개수가 늘고 파일이 정상적으로 올라온 것을 확인 할 수 있습니다.

# 4. Pull request 보내기

The screenshot shows a GitHub repository page for 'jkey20 / gitArrange'. The 'Pull requests' tab is highlighted with a red box. The page displays basic repository statistics: 2 commits, 1 branch, 0 releases, and 1 contributor. A 'New pull request' button is visible. Below the stats, it says 'This branch is 2 commits behind 2019androidtp:master.' and lists three files: 'README.md', 'test.txt', and another 'README.md'. The 'test.txt' file has a commit message '테스트용'. At the bottom, there's a text area containing the word 'gitArrange'.

jkey20 / gitArrange  
forked from 2019androidtp/gitArrange

Watch 0 Star 0 Fork 1

Code Pull requests 0 Projects 0 Wiki Security Insights Settings

깃 사용법 총정리 Edit

Manage topics

2 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find File Clone or download

This branch is 2 commits behind 2019androidtp:master. Pull request Compare

2\_10 jkey20 테스트용 Latest commit f7a2953 1 hour ago

README.md Initial commit 2 hours ago

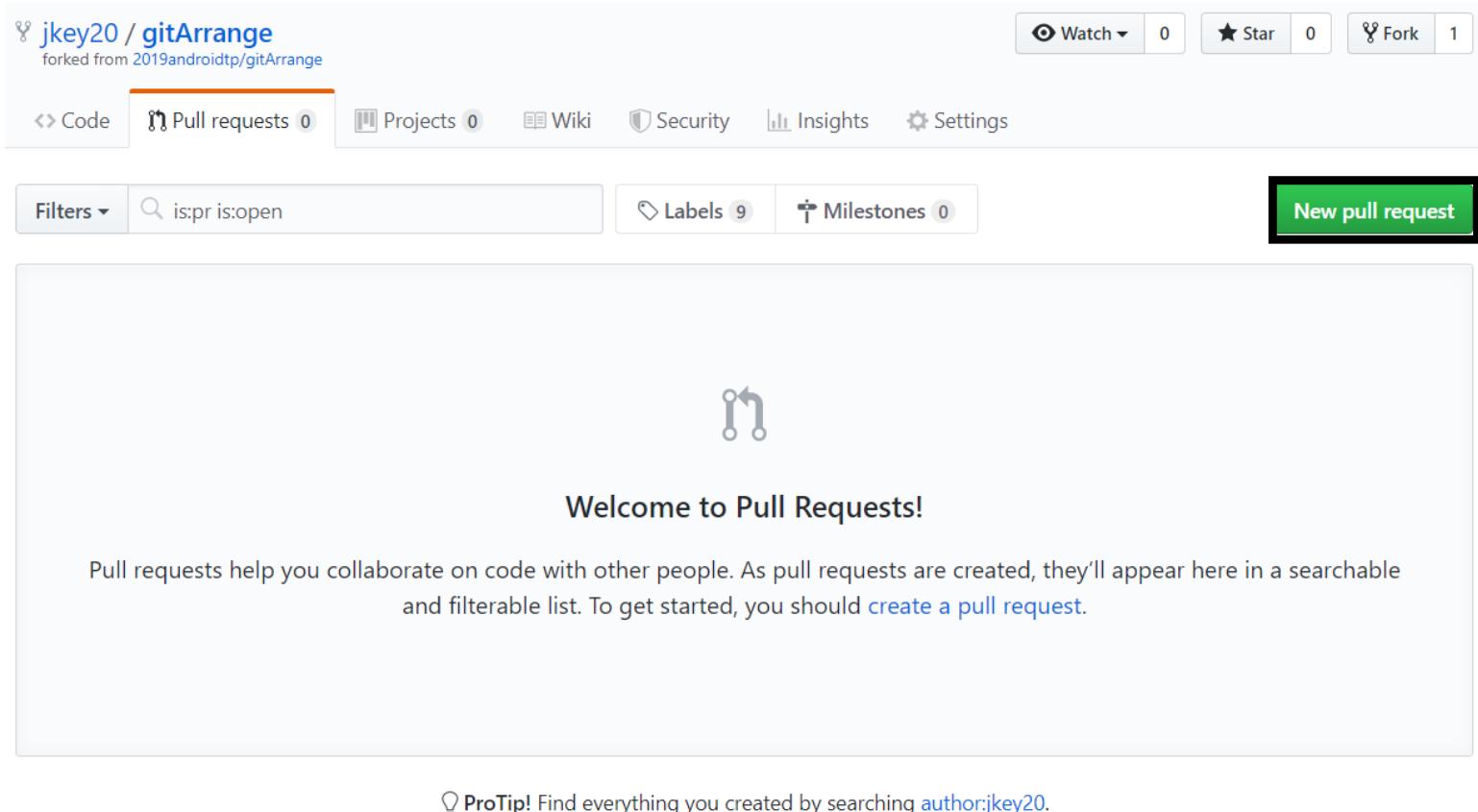
test.txt 테스트용 1 hour ago

README.md

gitArrange

그 다음 네모 친 Pull request 버튼을 누릅니다.

# 4. Pull request 보내기



이렇게 창이 바뀌면 네모 칸 친 New pull request를 누릅니다.

# 4. Pull request 보내기

The screenshot shows a GitHub repository page for '2019androidtp/gitArrange'. The 'Code' tab is selected. At the top, there are buttons for 'Watch' (0), 'Star' (0), and 'Fork' (1). Below the navigation bar, there are links for 'Issues' (0), 'Pull requests' (0), 'Projects' (0), 'Wiki', 'Security', 'Insights', and 'Settings'. The main title is 'Comparing changes'. A sub-instruction says: 'Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.' Below this, there are dropdown menus for 'base repository: 2019androidtp/gitArrange', 'base: master', 'head repository: jkey20/gitArrange', and 'compare: master'. The 'compare: master' dropdown is circled in black. A green checkmark indicates that the branches are 'Able to merge'. Below this, a large green button says 'Create pull request'. To its right, a note says 'Discuss and review the changes in this comparison with others.' with a question mark icon. Below the button, there are stats: '1 commit', '1 file changed', '0 commit comments', and '1 contributor'. Under 'Commits on Aug 01, 2019', it shows a commit by 'jkey20' titled '테스트용' with hash 'f7a2953'. At the bottom, it says 'Showing 1 changed file with 0 additions and 0 deletions.' with 'Unified' and 'Split' buttons. The file 'test.txt' is listed with a note 'No changes.'

이렇게 창이 바뀌면 성공입니다.  
동그라미친 부분은 자신이 파일을 올린 브랜치를 선택하는 부분입니다.  
자신이 올리려는 파일의 위치가 master 브랜치 외에 다른 브랜치라면 바꿔줘야 합니다.  
그 후 네모 친 create pull request를 누릅니다.

# 4. Pull request 보내기

The screenshot shows the GitHub interface for creating a pull request. At the top, it displays the repository path `2019androidtp / gitArrange`. To the right are buttons for 'Watch' (0), 'Star' (0), and settings. Below the repository path is a navigation bar with links for 'Code', 'Issues 0', 'Pull requests 0', 'Projects 0', 'Wiki', 'Security', 'Insights', and 'Settings'. The main title 'Open a pull request' is centered above a sub-instruction: 'Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#)'. Below this, there are dropdown menus for 'base repository: 2019androidtp/gitArrange', 'base: master', 'head repository: jkey20/gitArrange', and 'compare: master'. A green checkmark indicates that the branches are 'Able to merge'. The main form area has a title '2\_10' and a text input field containing '테스트용'. It includes a rich text editor toolbar with buttons for bold, italic, and other styling options. Below the toolbar is a comment input field with the placeholder 'Leave a comment'. At the bottom of the form, there is a note about attaching files and a checkbox for 'Allow edits from maintainers' with a 'Learn more' link. A large green button labeled 'Create pull request' is positioned at the bottom right. To the right of the form, there are sections for 'Reviewers' (No reviews), 'Assignees' (No one—assign yourself), 'Labels' (None yet), 'Projects' (None yet), and 'Milestone' (No milestone).

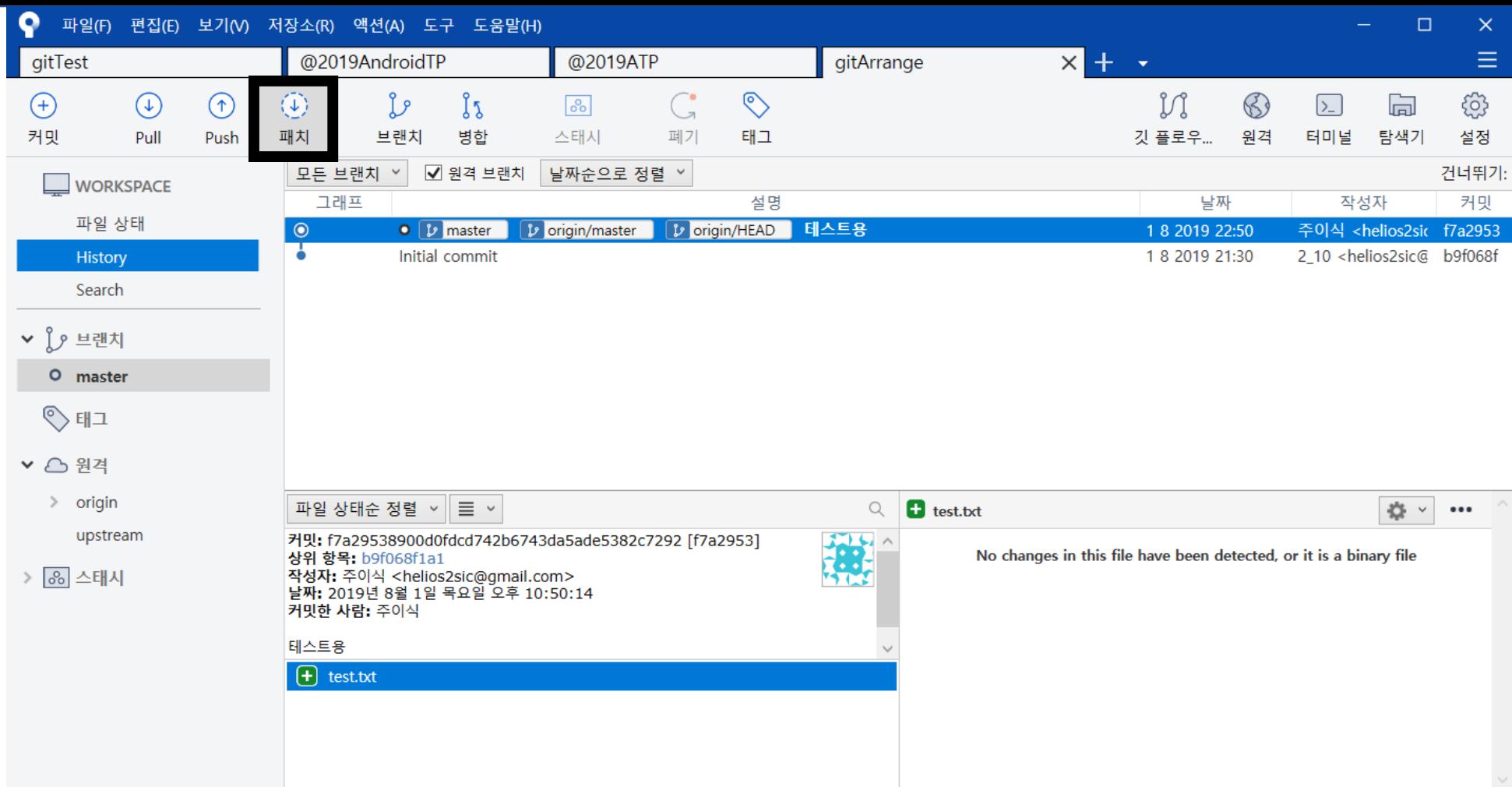
다음과 같이 창이 나오면 제목과 내용을 입력하고 네모 친 create pull request를 누릅니다.

# 4. Pull request 보내기

The screenshot shows a GitHub repository page for '2019androidtp / gitArrange'. The 'Pull requests' tab is selected, showing one open pull request titled '테스트용 #1' by user 'jkey20'. The pull request details indicate it merges 1 commit from 'jkey20:master' into '2019androidtp:master'. The commit message is 'No description provided.' and the commit hash is 'f7a2953'. A note at the bottom says 'Add more commits by pushing to the master branch on jkey20/gitArrange.' Below this, a green box highlights the message 'This branch has no conflicts with the base branch' and the button 'Merge pull request'. To the right, there are sections for Reviewers, Assignees, Labels, Projects, and Milestone, all currently set to 'None yet'. The top navigation bar includes 'Code', 'Issues 0', 'Pull requests 1', 'Projects 0', 'Wiki', 'Security', 'Insights', and 'Settings'.

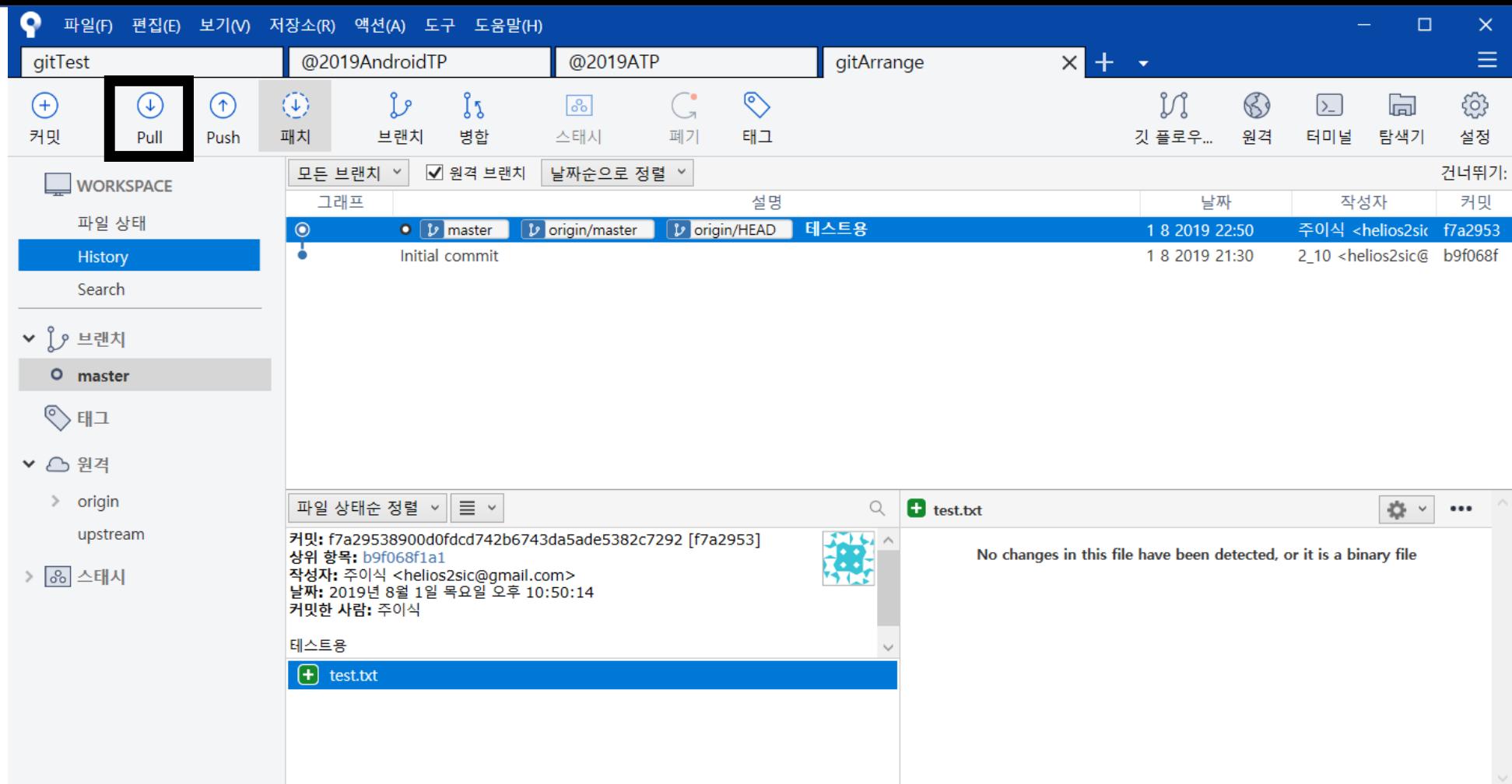
다음과 같이 pull request가 생깁니다.  
네모 친 부분은 Merge를 할 수 있는 권한이 있는 사용자만이 누를 수 있습니다.

# 5. 메인 페이지에서 받아오기



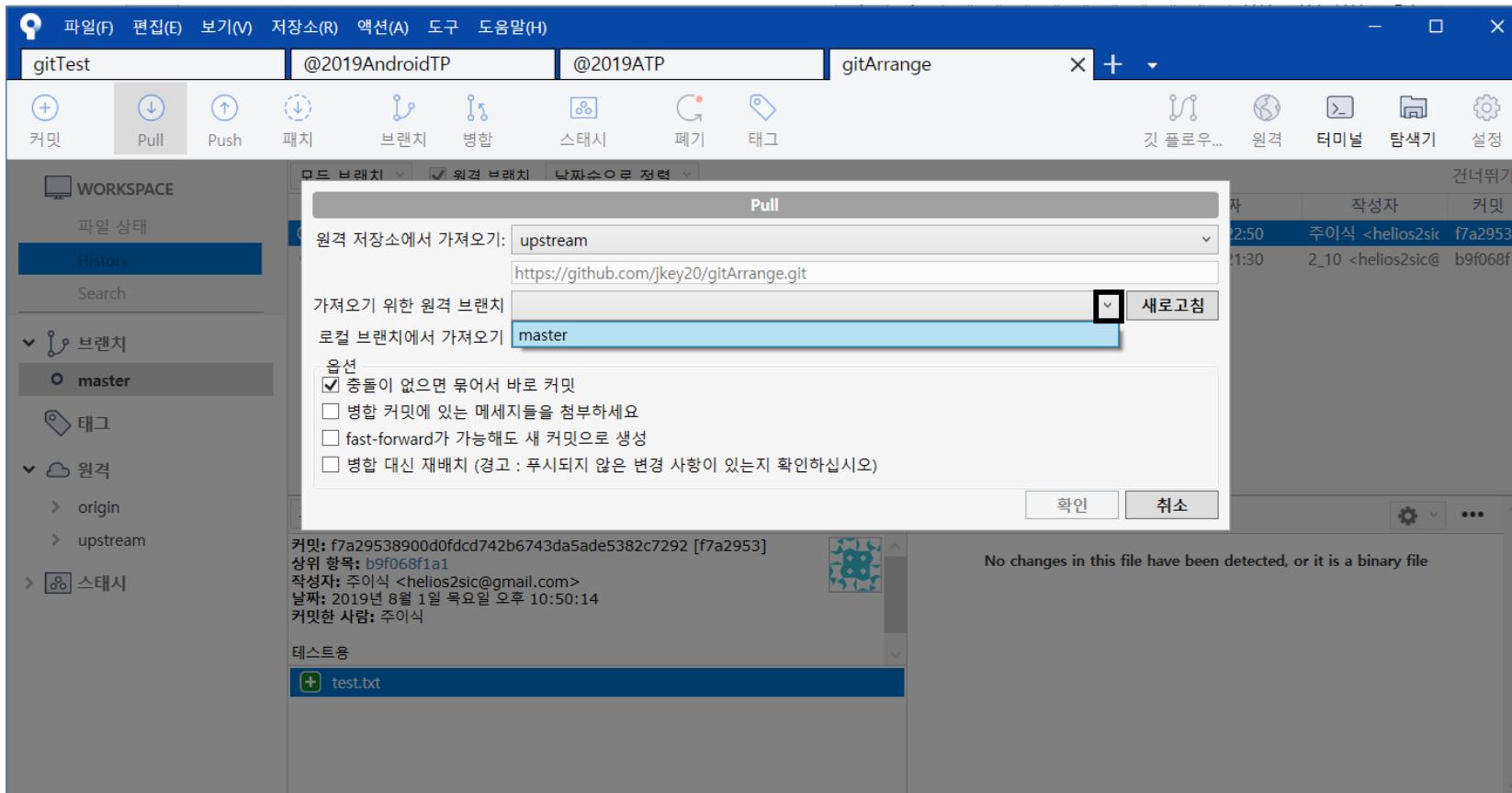
메인 페이지에서 파일들이 수정되면 로컬파일과 포크를 한 페이지에 수정사항을 업데이트 해줘야 합니다.  
소스트리에서 패치를 눌러 변경사항을 확인합니다.

# 5. 메인 페이지에서 받아오기



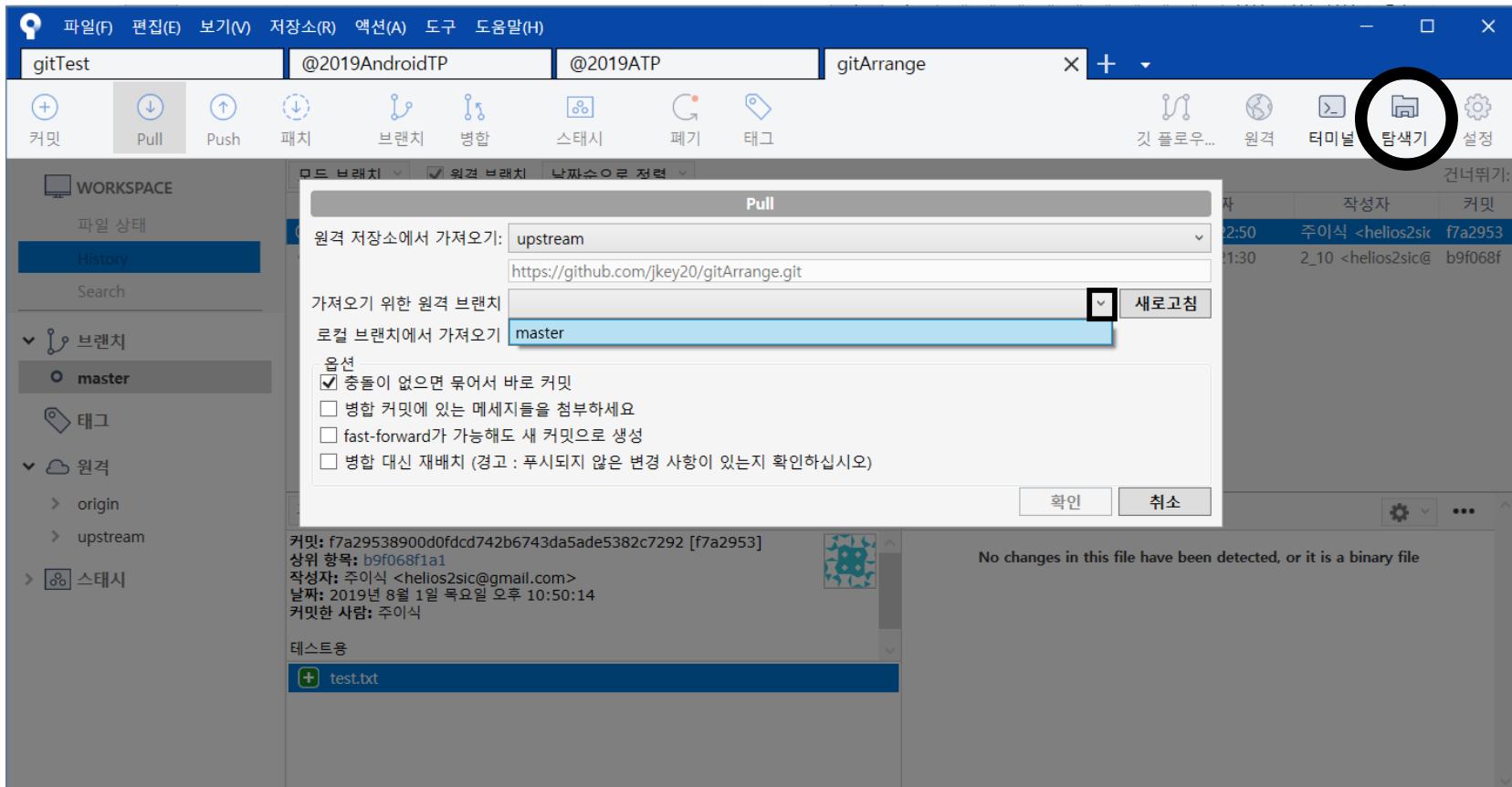
변경사항을 확인한 후 pull을 누릅니다.

# 5. 메인 페이지에서 받아오기



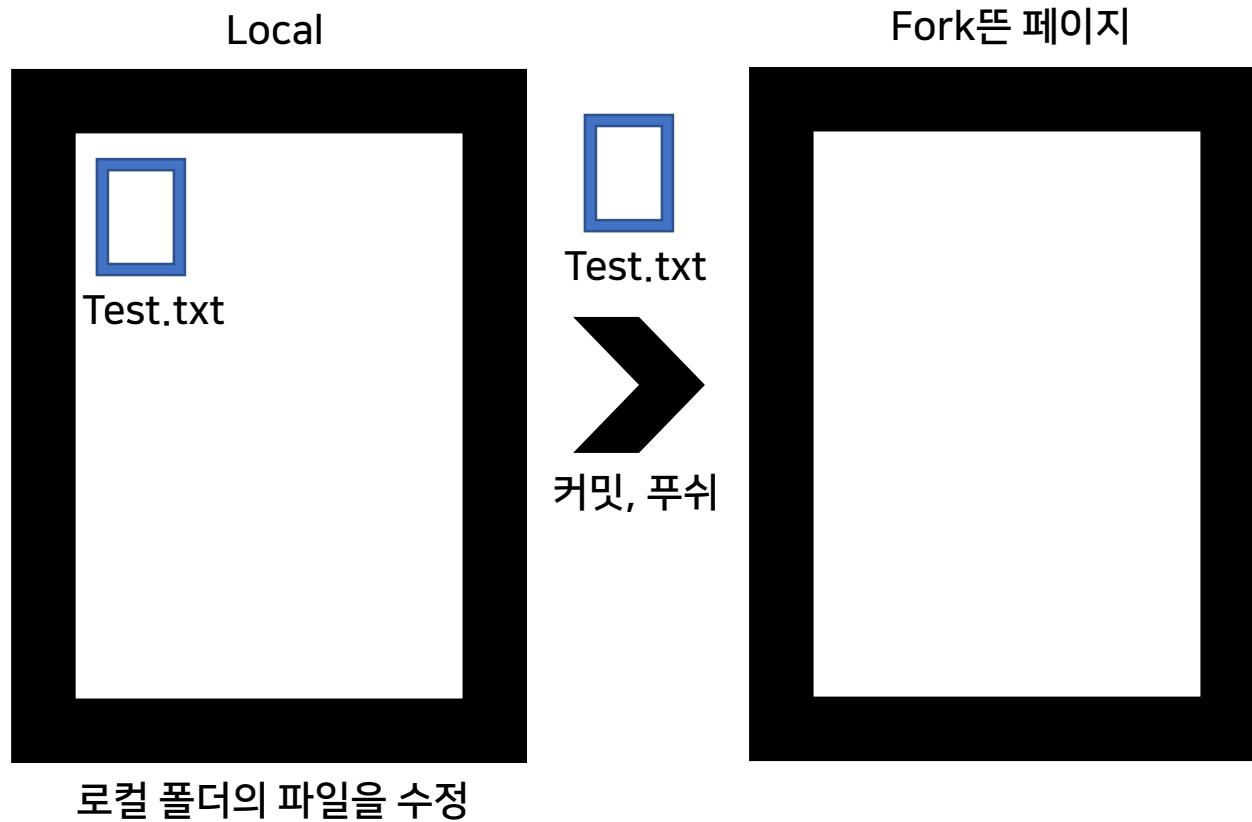
그 다음 네모 친 화살표를 누르고 master로 바꾸고 확인버튼을 누릅니다.

# 5. 메인 페이지에서 받아오기



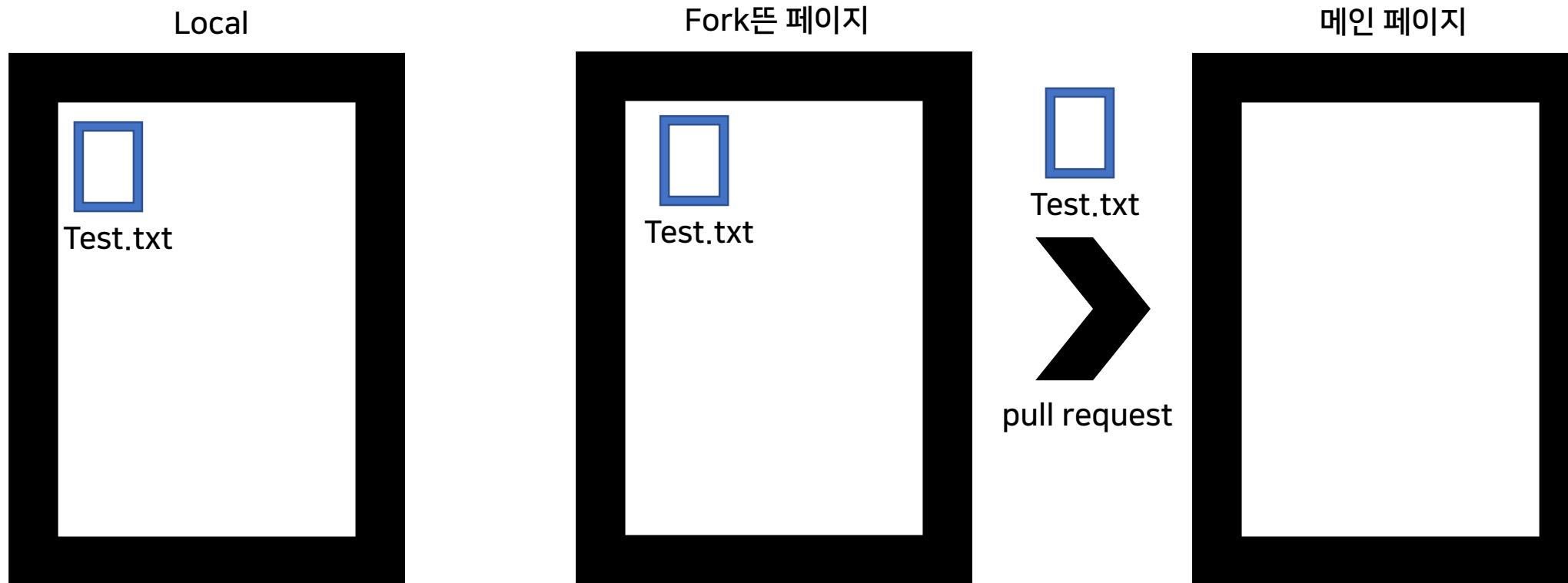
그 다음 네모 친 화살표를 누르고 master로 바꾸고 확인버튼을 누른 뒤  
그 다음 우측상단에 동그라미 친 탐색기를 눌러보면 파일들이 저장되어 있습니다.

# 파일 업로드 순서도



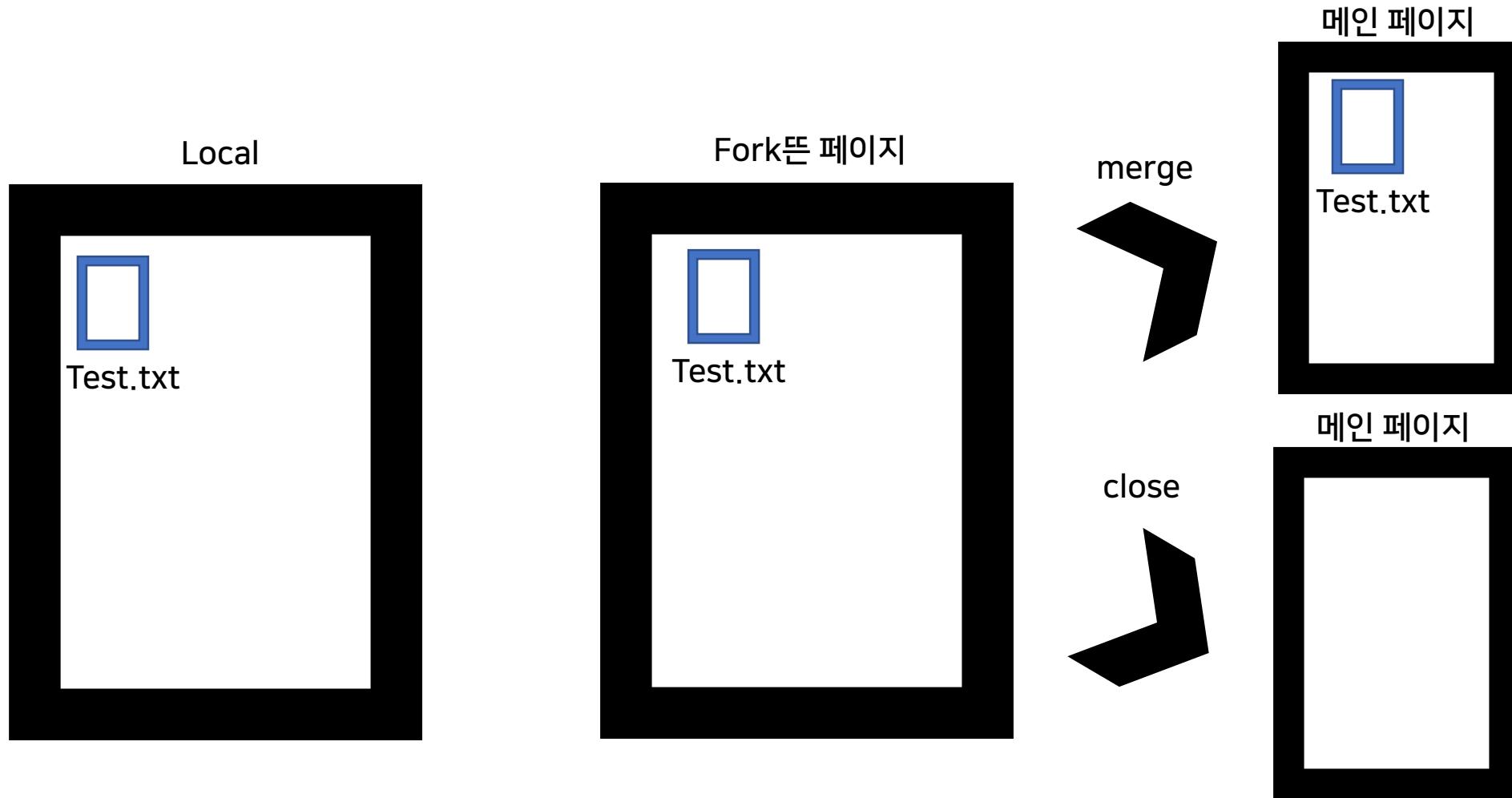
로컬 폴더의 파일을 수정한 후 해당 파일을 커밋하고 push합니다.

# 파일 업로드 순서도



푸쉬가 끝나면 fork뜬 페이지에 파일이 올라가게 됩니다.  
여기서 pull request를 보냅니다.

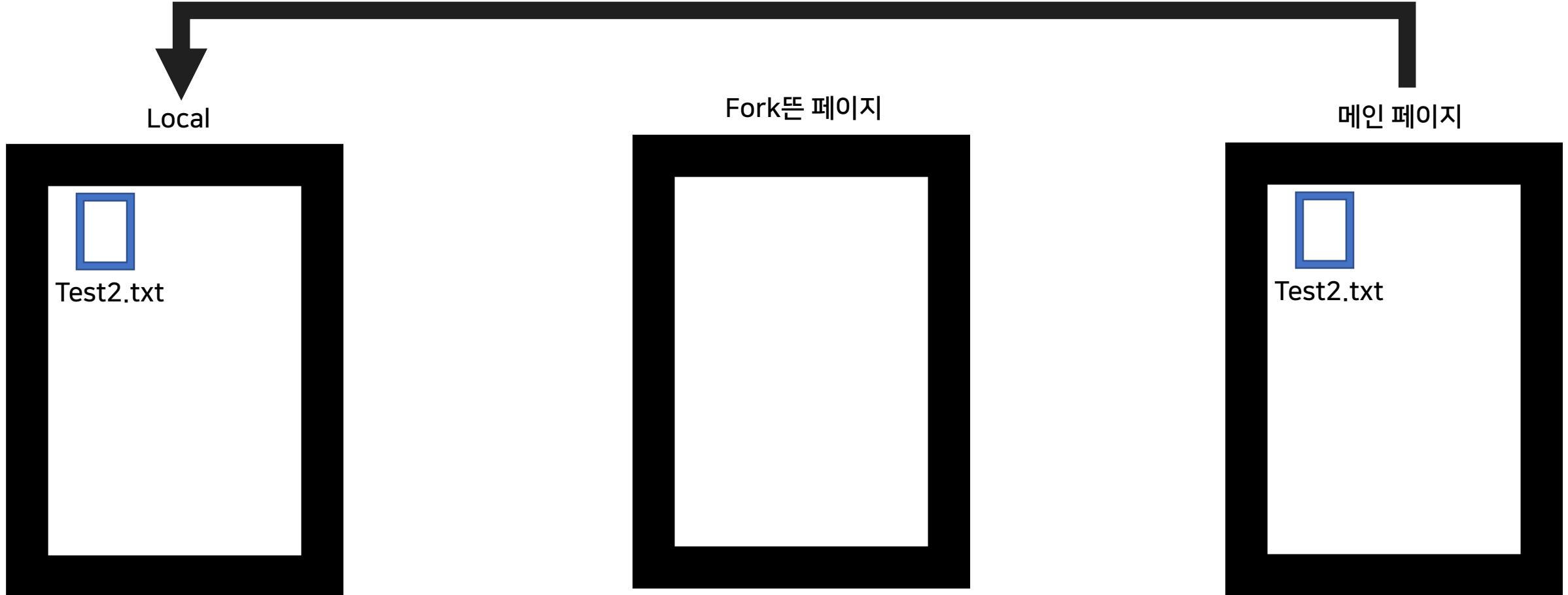
# 파일 업로드 순서도



Pull request를 소유자가 merge시키면 메인 페이지에 정상적으로 파일이 올라갑니다.  
만약 pull request를 close하면 파일이 올라가지 않습니다.

# 파일 다운로드 순서도

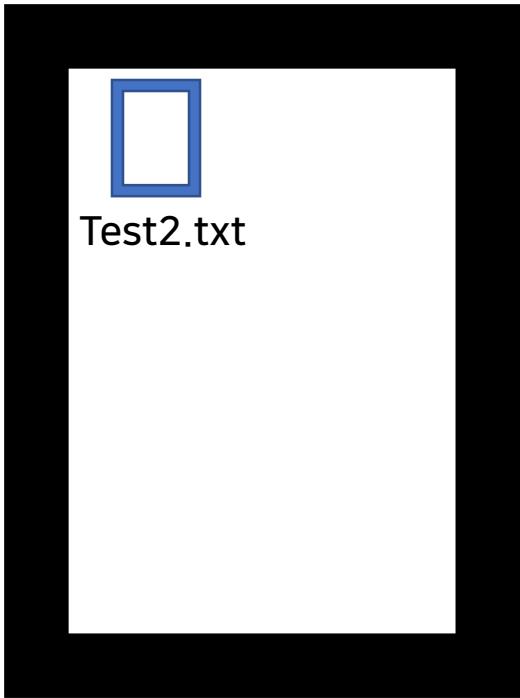
origin을 upstream으로 바꾸고 pull



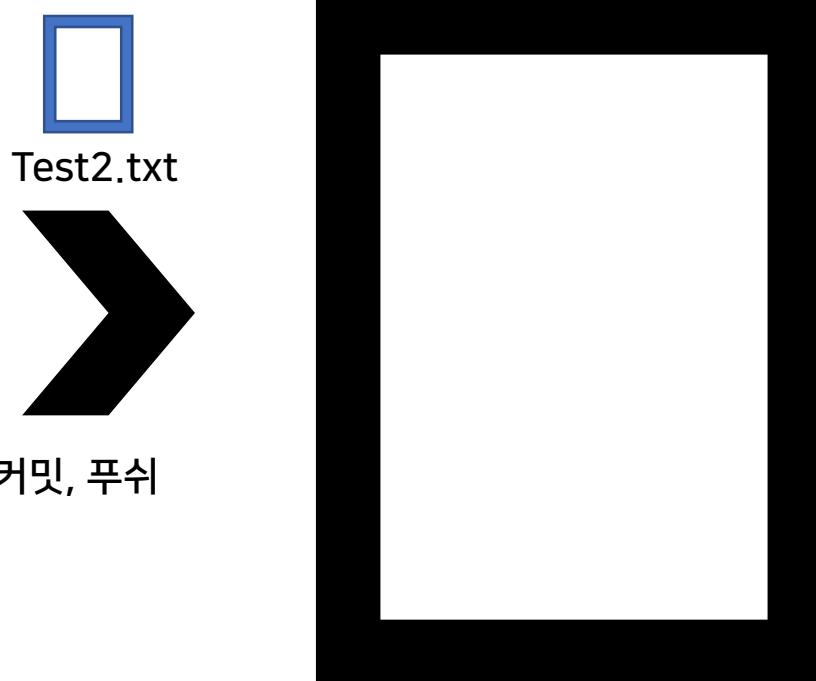
메인 페이지에 있는 파일을 다운받는 순서입니다.  
먼저 패치를 해주고 pull을 해서 로컬에 파일을 받습니다.

# 파일 다운로드 순서도

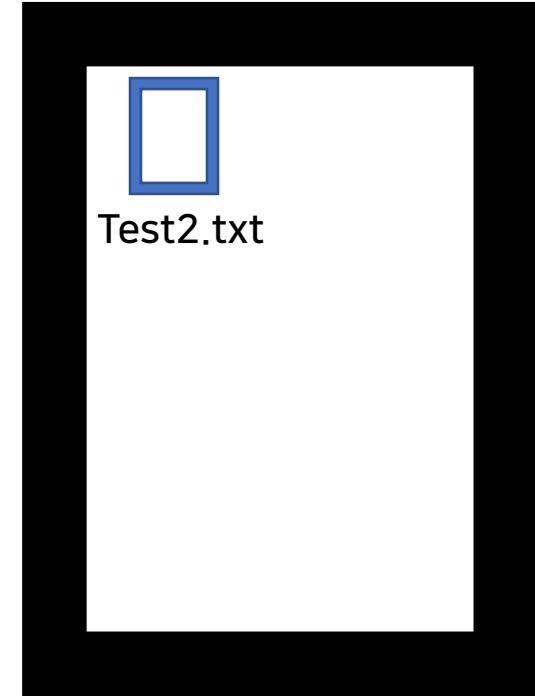
Local



Fork든 페이지



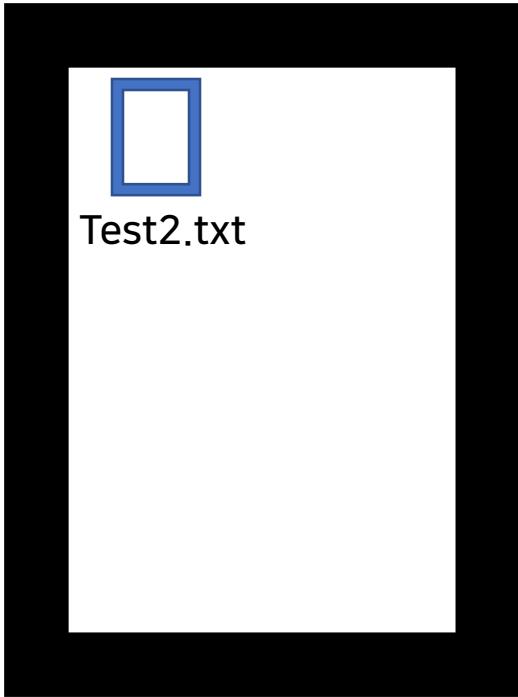
메인 페이지



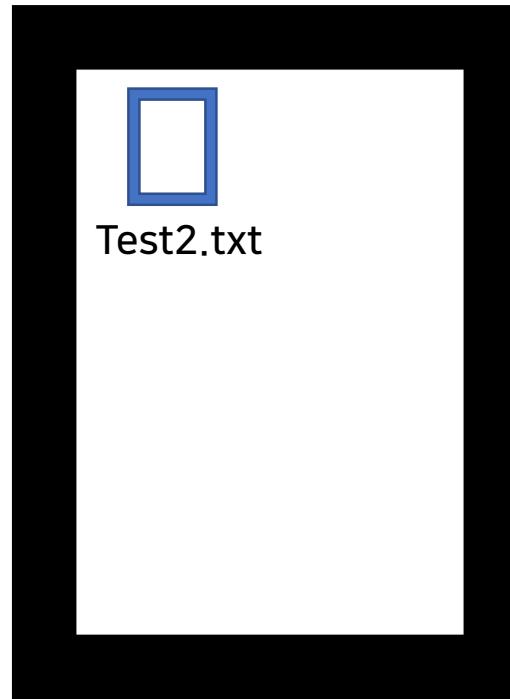
로컬에 새롭게 저장된 파일은 fork든 페이지에 올라가있지 않으므로 커밋되지 않은 변경사항으로 나옵니다.  
따라서 커밋 후 origin에 푸쉬를 해줍니다.

# 파일 다운로드 순서도

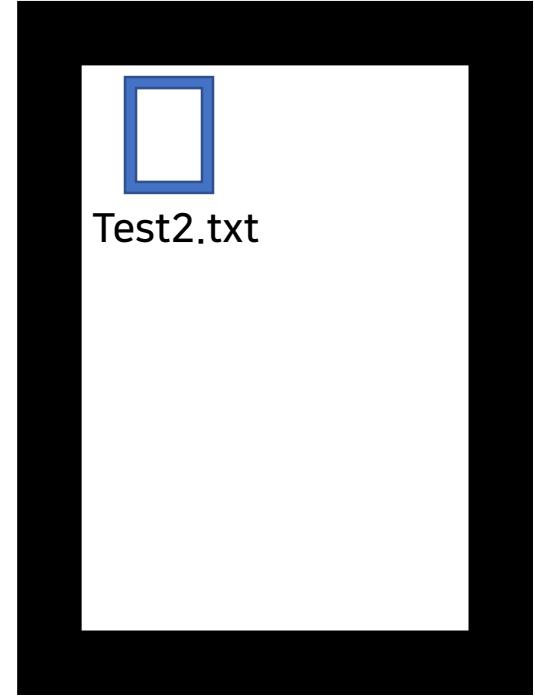
Local



Fork뜬 페이지

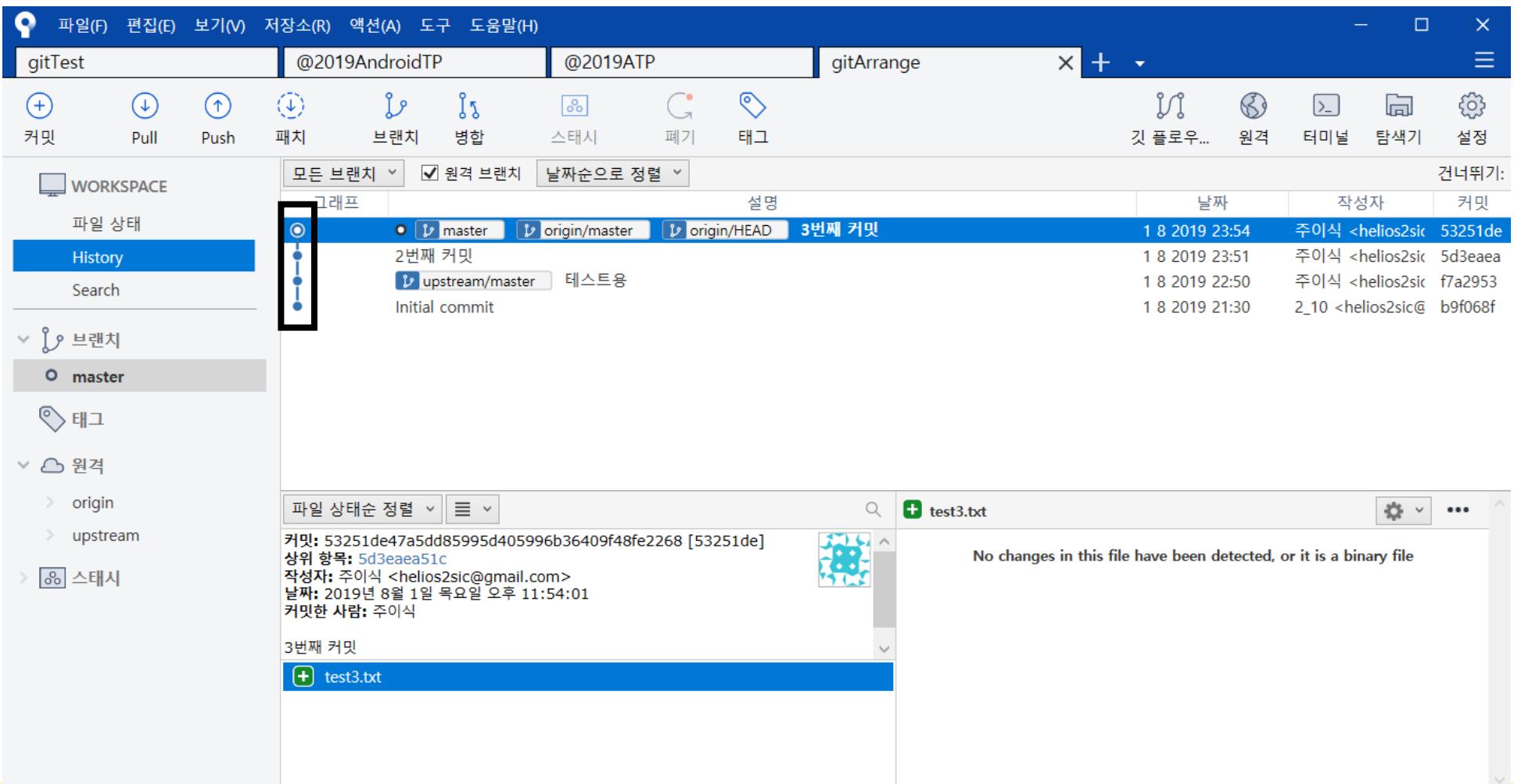


메인 페이지



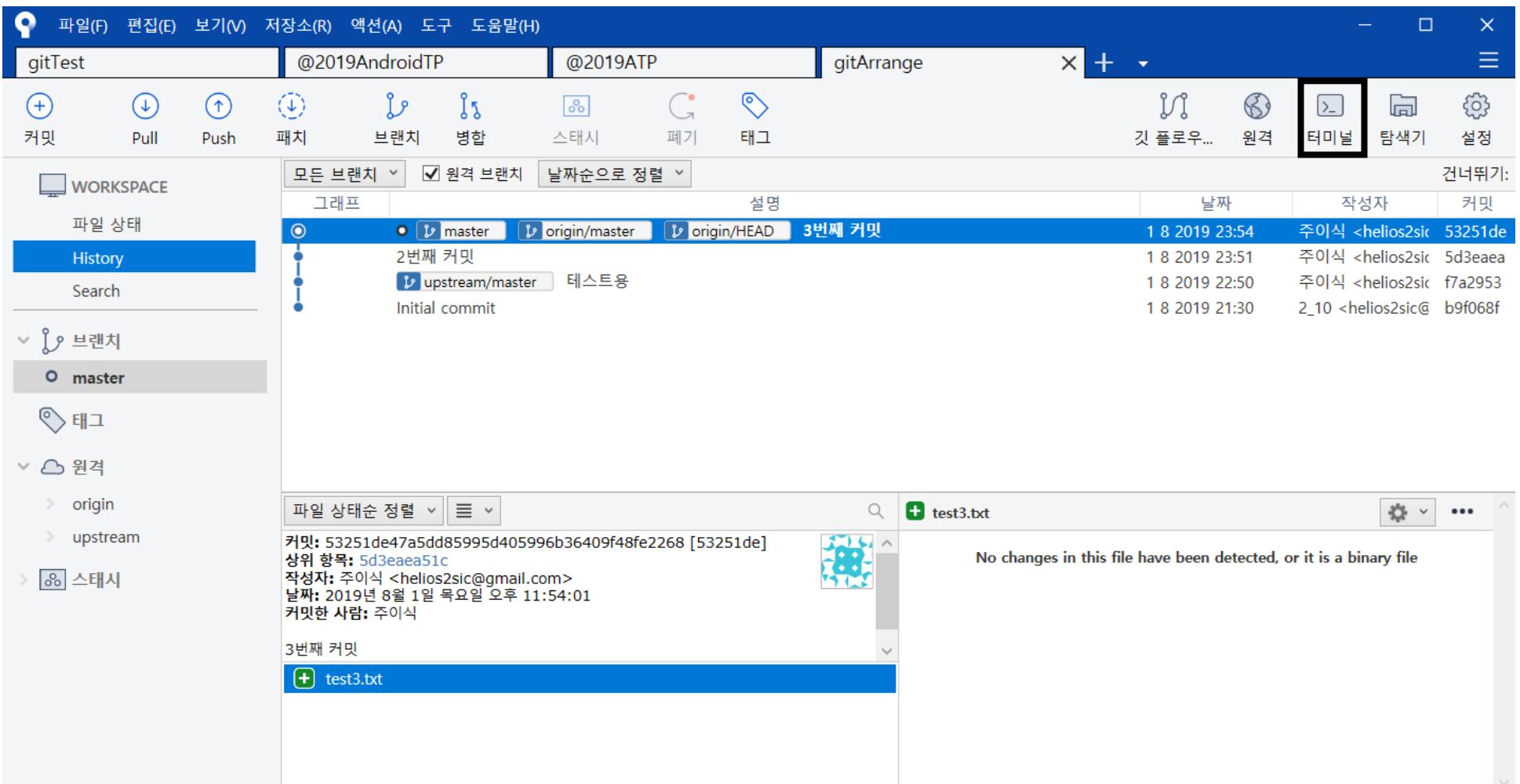
푸쉬가 끝나면 파일이 fork뜬 페이지에 들어가고 정상적으로 파일 다운로드가 완료됩니다.

# 6. 커밋 합치기



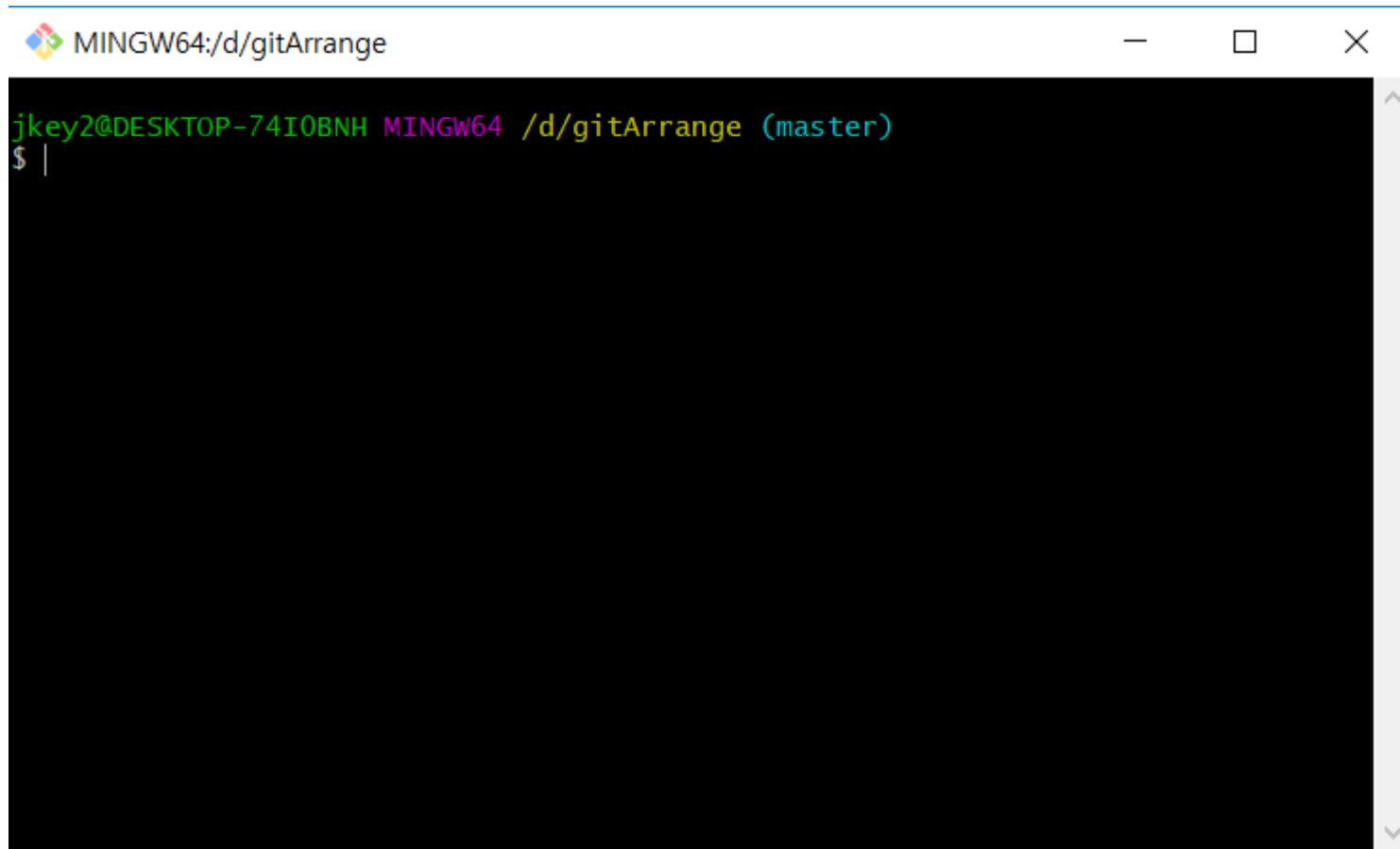
파일들을 많이 올리다보면 커밋들이 너무 많아지거나 쓸모없는 커밋들이 생깁니다.  
이때 커밋들을 합쳐서 깔끔하게 그래프를 정리할 수 있습니다.

# 6. 커밋 합치기



먼저 오른쪽 상단에 터미널을 눌러줍니다.  
꼭! Git bash가 깔려있어야 합니다!

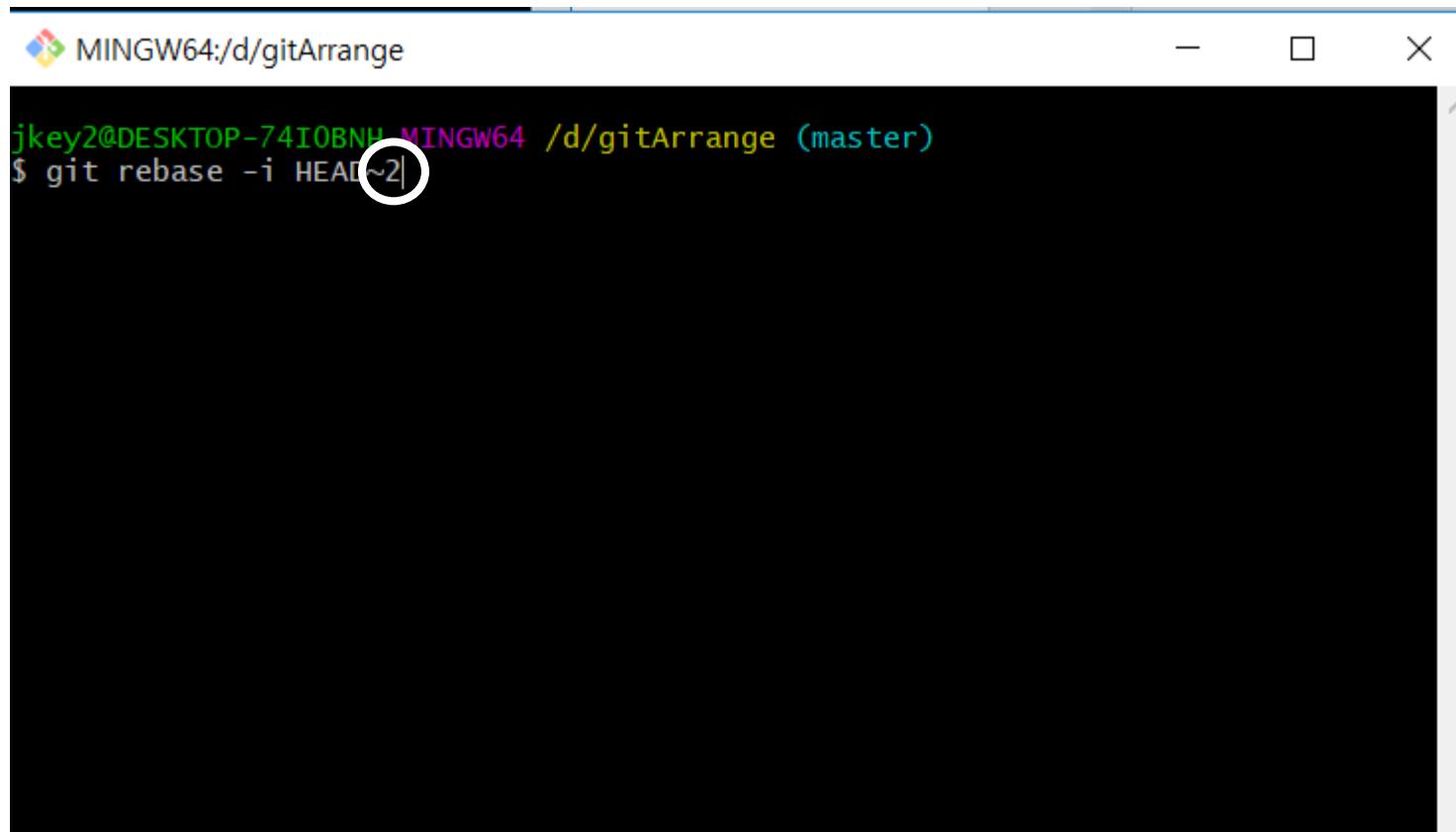
# 6. 커밋 합치기



The screenshot shows a terminal window titled "MINGW64:/d/gitArrange". The title bar also displays the user's name "jkey2@DESKTOP-74I0BNH" and the repository path "MINGW64 /d/gitArrange (master)". The main area of the terminal is black and contains a single character, a dollar sign (\$), indicating the command prompt. There are standard window controls (minimize, maximize, close) at the top right.

그럼 다음과 같이 cmd창이 나옵니다.

# 6. 커밋 합치기

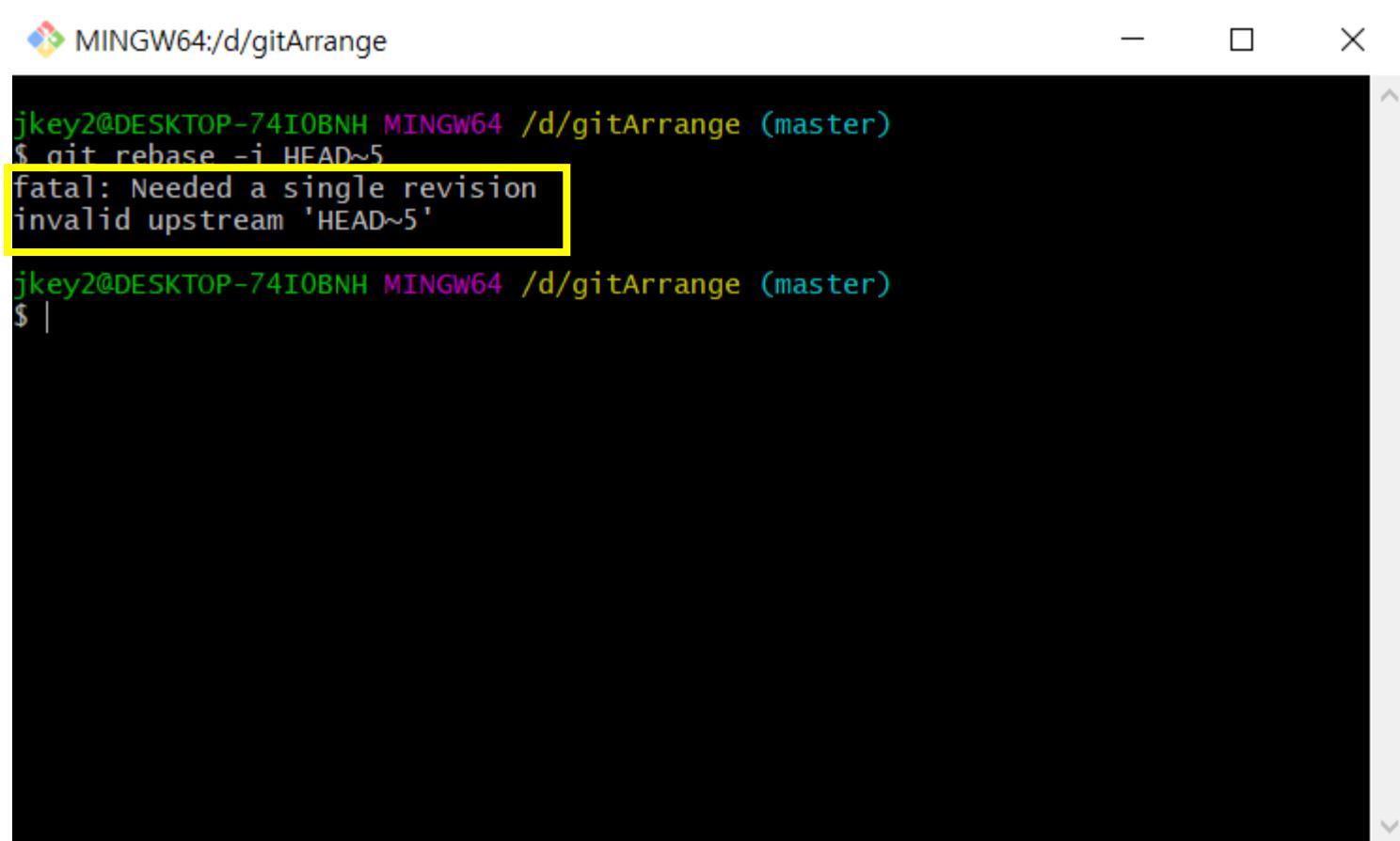


A screenshot of a terminal window titled "MINGW64:/d/gitArrange". The window shows a command line with the following text:  
jkey2@DESKTOP-74I0BNH MINGW64 /d/gitArrange (master)  
\$ git rebase -i HEAD~2|

The text "HEAD~2|" is highlighted with a white circle.

그다음 git rebase -i HEAD~ 를 입력합니다.  
흰색 동그라미 부분에 숫자는 합칠 커밋의 개수입니다.

# 6. 커밋 합치기



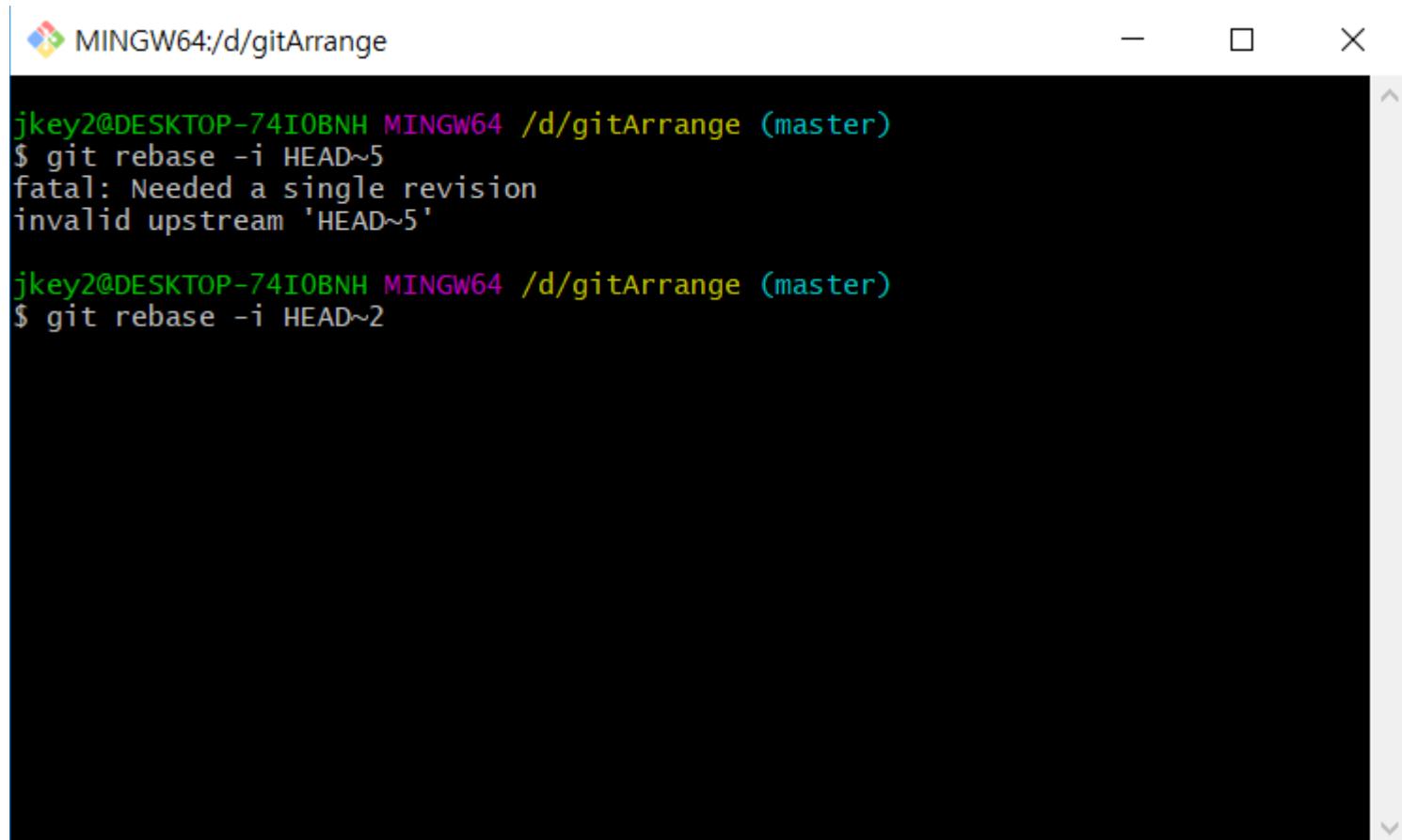
The screenshot shows a terminal window titled "MINGW64:/d/gitArrange". The user has run the command `git rebase -i HEAD~5`. The output shows an error message: "fatal: Needed a single revision" followed by "invalid upstream 'HEAD~5'". This error occurs because the user has specified a range of commits to rebase (HEAD~5), but has not provided a specific commit to merge into. A yellow box highlights the error message.

```
jkey2@DESKTOP-74I0BNH MINGW64 /d/gitArrange (master)
$ git rebase -i HEAD~5
fatal: Needed a single revision
invalid upstream 'HEAD~5'

jkey2@DESKTOP-74I0BNH MINGW64 /d/gitArrange (master)
$ |
```

만약 커밋의 개수보다 많은 숫자를 입력하면 노란색으로 오류가 납니다.  
이는 무시해도 되므로 자신의 커밋 개수를 모르겠으면 숫자를 하나씩 줄여가는 방법도 괜찮습니다.

# 6. 커밋 합치기

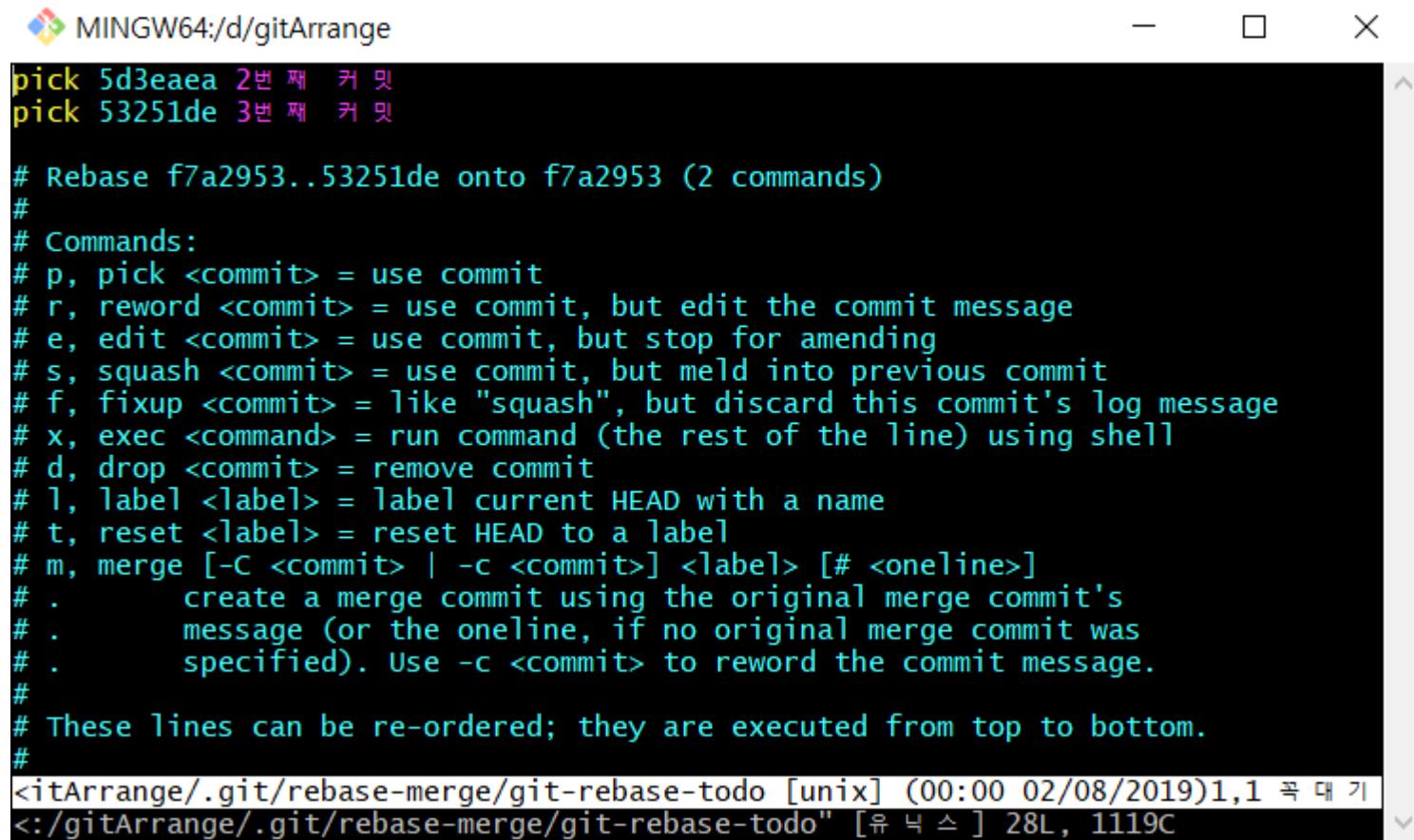


```
MINGW64:/d/gitArrange
jkey2@DESKTOP-74I0BNH MINGW64 /d/gitArrange (master)
$ git rebase -i HEAD~5
fatal: Needed a single revision
invalid upstream 'HEAD~5'

jkey2@DESKTOP-74I0BNH MINGW64 /d/gitArrange (master)
$ git rebase -i HEAD~2
```

2개의 커밋을 합쳐보겠습니다.  
커맨드로 git rebase -i HEAD~2를 치고 Enter를 누릅니다.

# 6. 커밋 합치기



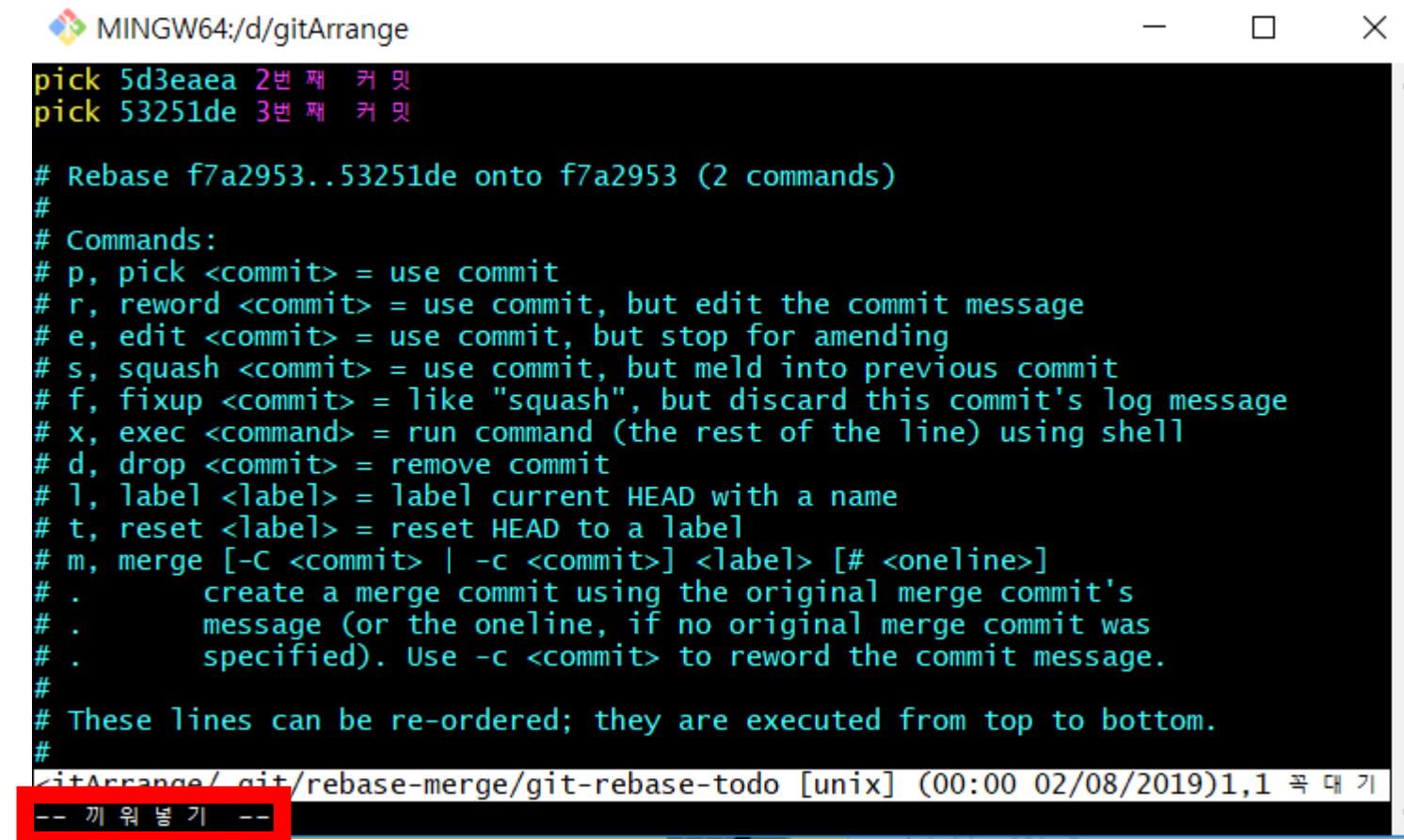
The screenshot shows a terminal window titled 'MINGW64:/d/gitArrange'. The command 'git rebase -i' has been run, resulting in the following output:

```
pick 5d3eaea 2번 째 커밋
pick 53251de 3번 째 커밋

# Rebase f7a2953..53251de onto f7a2953 (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [<oneline>]
# .
#       create a merge commit using the original merge commit's
#       message (or the oneline, if no original merge commit was
#       specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
<itArrange/.git/rebase-merge/git-rebase-todo [unix] (00:00 02/08/2019)1,1 꼭 대기
</itArrange/.git/rebase-merge/git-rebase-todo" [유닉스] 28L, 1119C
```

다음과 같이 창이 나오면 i를 눌러줍니다.

# 6. 커밋 합치기



The screenshot shows a terminal window titled 'MINGW64:/d/gitArrange'. The command 'git rebase -i' has been run, displaying a list of commits to be rebased. The first two commits are explicitly selected with 'pick' (the others are shown with '#'). Below the commits is a detailed explanation of the rebase command's options, including pick, reword, edit, squash, fixup, exec, drop, label, reset, merge, and create. A note at the bottom states that the lines can be reordered. The status bar at the bottom right indicates the session is on 'gitArrange / git/rebase-merge/git-rebase-todo [unix]' and it started at '00:00 02/08/2019'. A red box highlights the text '-- 끼워 넣기 --' in the status bar.

```
pick 5d3eaea 2번 째 커밋
pick 53251de 3번 째 커밋

# Rebase f7a2953..53251de onto f7a2953 (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [<oneline>]
# .      create a merge commit using the original merge commit's
# .      message (or the oneline, if no original merge commit was
# .      specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
-- 끼워 넣기 --
```

다음과 같이 끼워넣기가 나오면 잘 하고 있는 겁니다.

# 6. 커밋 합치기

```
pick 5d3eaee 2번 째 커밋
pick 53251de 3번 째 커밋
#
# rebase T/aZ953..53251de onto T/aZ953 (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [<oneline>]
# .      create a merge commit using the original merge commit's
# .      message (or the oneline, if no original merge commit was
# .      specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
<rrange/.git/rebase-merge/git-rebase-todo[+] [unix] (00:00 02/08/2019)3,2 꼭 대기
-- 끄 워 넣기 --
```

빨간색 줄 위부분이 우리가 수정할 부분입니다.  
아래부분은 커맨드 설명입니다.

# 6. 커밋 합치기

```
MINGW64:/d/gitArrange
pick 5d3eaea 2번 째 커밋
pick 53251de 3번 째 커밋

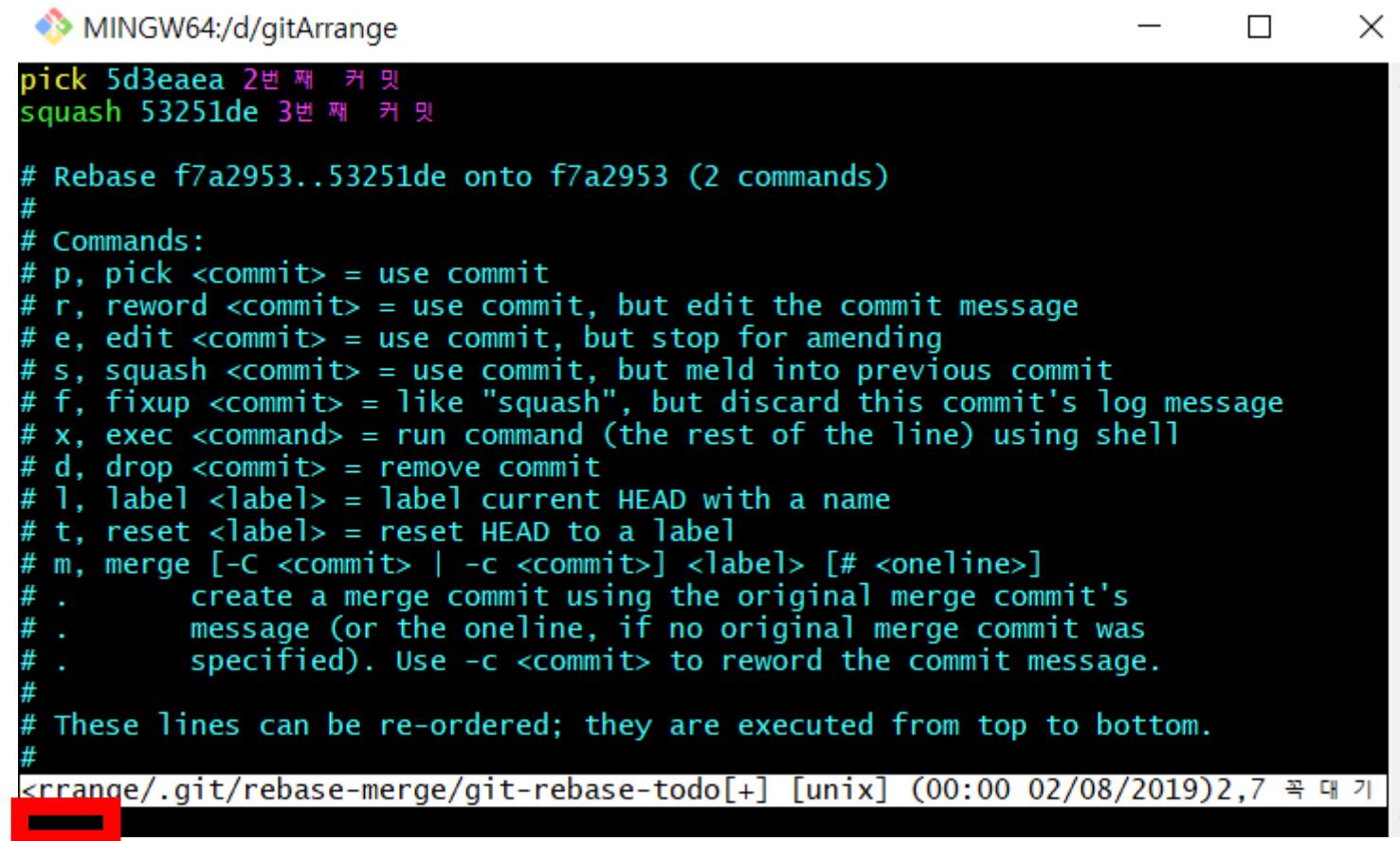
# Rebase f7a2953..53251de onto f7a2953 (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [<oneline>]
# .           create a merge commit using the original merge commit's
# .           message (or the oneline, if no original merge commit was
# .           specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
<range/.git/rebase-merge/git-rebase-todo[+] [unix] (00:00 02/08/2019)3,2 폭 대기
-- 끄워 넣기 --
```

```
MINGW64:/d/gitArrange
pick 5d3eaea 2번 째 커밋
squash 53251de 3번 째 커밋

# Rebase f7a2953..53251de onto f7a2953 (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [<oneline>]
# .           create a merge commit using the original merge commit's
# .           message (or the oneline, if no original merge commit was
# .           specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
<range/.git/rebase-merge/git-rebase-todo[+] [unix] (00:00 02/08/2019)2,7 폭 대기
-- 끄워 넣기 --
```

빨간색 네모부분을 지우고 squash로 바꿉니다.  
다수의 커밋을 합치려면 가장 위의 pick만 남기고 모든 pick부분을 squash로 바꾸면 됩니다.

# 6. 커밋 합치기



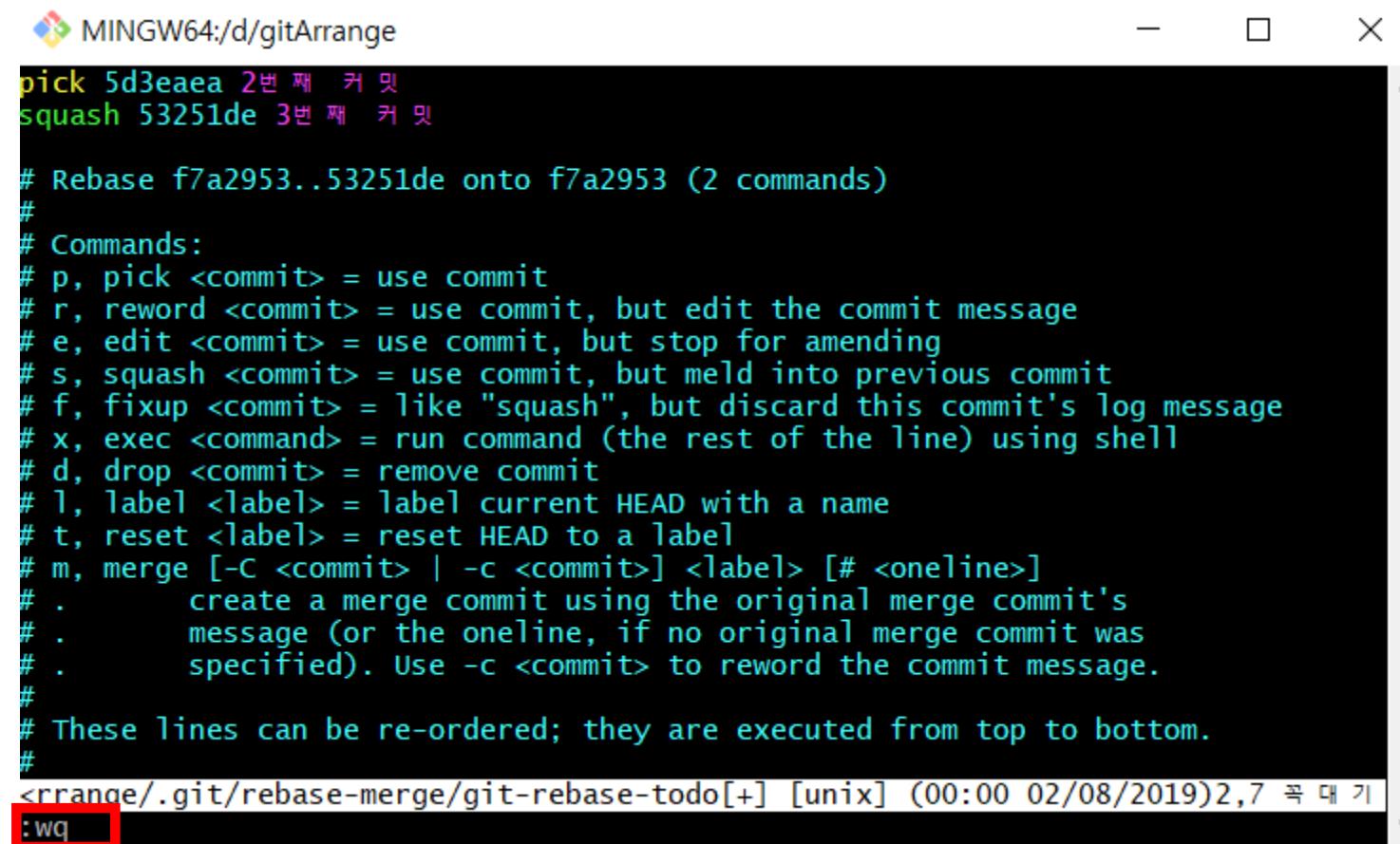
The screenshot shows a terminal window titled 'MINGW64:/d/gitArrange'. The window contains a list of Git rebase commands:

```
pick 5d3eaea 2번째 커밋
squash 53251de 3번째 커밋

# Rebase f7a2953..53251de onto f7a2953 (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [<oneline>]
# .
#   create a merge commit using the original merge commit's
#   message (or the oneline, if no original merge commit was
#   specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
<rrange/.git/rebase-merge/git-rebase-todo[+] [unix] (00:00 02/08/2019)2,7 폭 대 기
```

그후 ESC를 눌러 빨간색 부분에 있던 끼워넣기가 사라지게 만듭니다.

# 6. 커밋 합치기



The screenshot shows a terminal window titled "MINGW64:/d/gitArrange". The window displays a list of git commands for rebasing, starting with "pick 5d3eaea 2번째 커밋" and "squash 53251de 3번째 커밋". Below these, a detailed explanation of the "rebase" command is provided, listing various options like p, r, e, s, f, x, d, l, t, m, ., and their descriptions. At the bottom of the list, it says "# These lines can be re-ordered; they are executed from top to bottom." A red box highlights the command ":wq" at the bottom of the terminal window.

```
pick 5d3eaea 2번째 커밋
squash 53251de 3번째 커밋

# Rebase f7a2953..53251de onto f7a2953 (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [<oneline>]
# .
#     create a merge commit using the original merge commit's
#     message (or the oneline, if no original merge commit was
#     specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
<rrange/.git/rebase-merge/git-rebase-todo[+] [unix] (00:00 02/08/2019)2,7 품 대 기
:wq
```

그 다음 :wq를 입력하고 Enter를 누릅니다.

# 6. 커밋 합치기

리베이스 도중에 git bash를 꺼버렸어요!

- git rebase --continue를 치시고 리베이스를 계속하세요

리베이스를 하는데 visual or 다른 프로그램으로 연결이 됩니다!

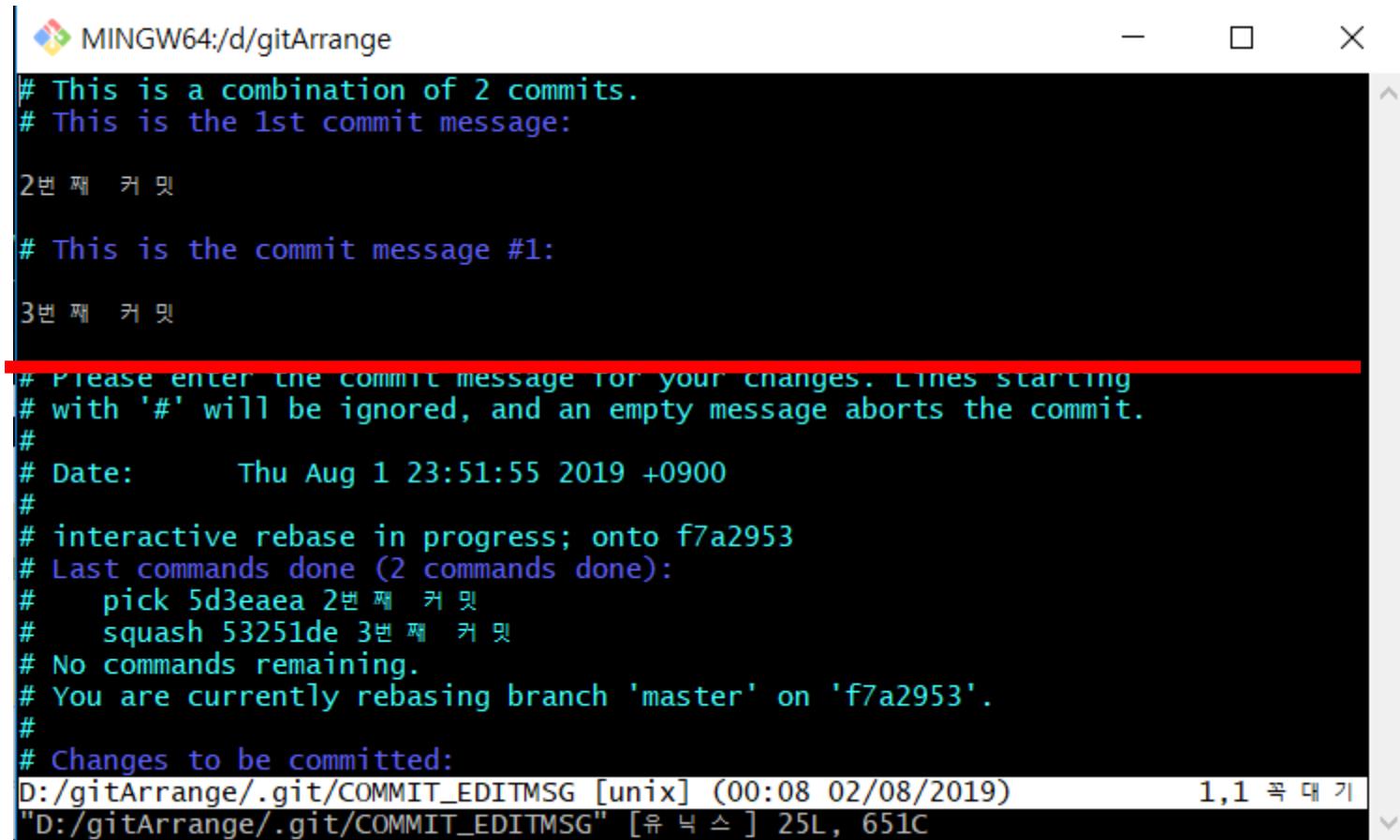
- git config --list를 입력하고 core.editor라고 적혀져 있는 부분이 있다면  
git config --global core.editor "vim"을 쳐주세요

리베이스 도중에 could not apply ~ 라고 뜨면서 리베이스가 멈춥니다!

- 자신이 합치려는 커밋들 중 이름이 같은 파일들이 있는지 확인하고 파일을 삭제해주세요.  
만약 삭제를 해도 안된다면 pick을 squash로 바꾸는 부분에서 리베이스가 멈추는 부분의 커밋을 squash가  
아닌 drop으로 바꿔서 진행해보세요.

만약 리베이스 도중 오류가 난다면 다음과 같이 해결해 보세요.

# 6. 커밋 합치기



The screenshot shows a terminal window titled 'MINGW64:/d/gitArrange'. The terminal displays a series of commit messages and a rebase status:

```
# This is a combination of 2 commits.
# This is the 1st commit message:

2번째 커밋

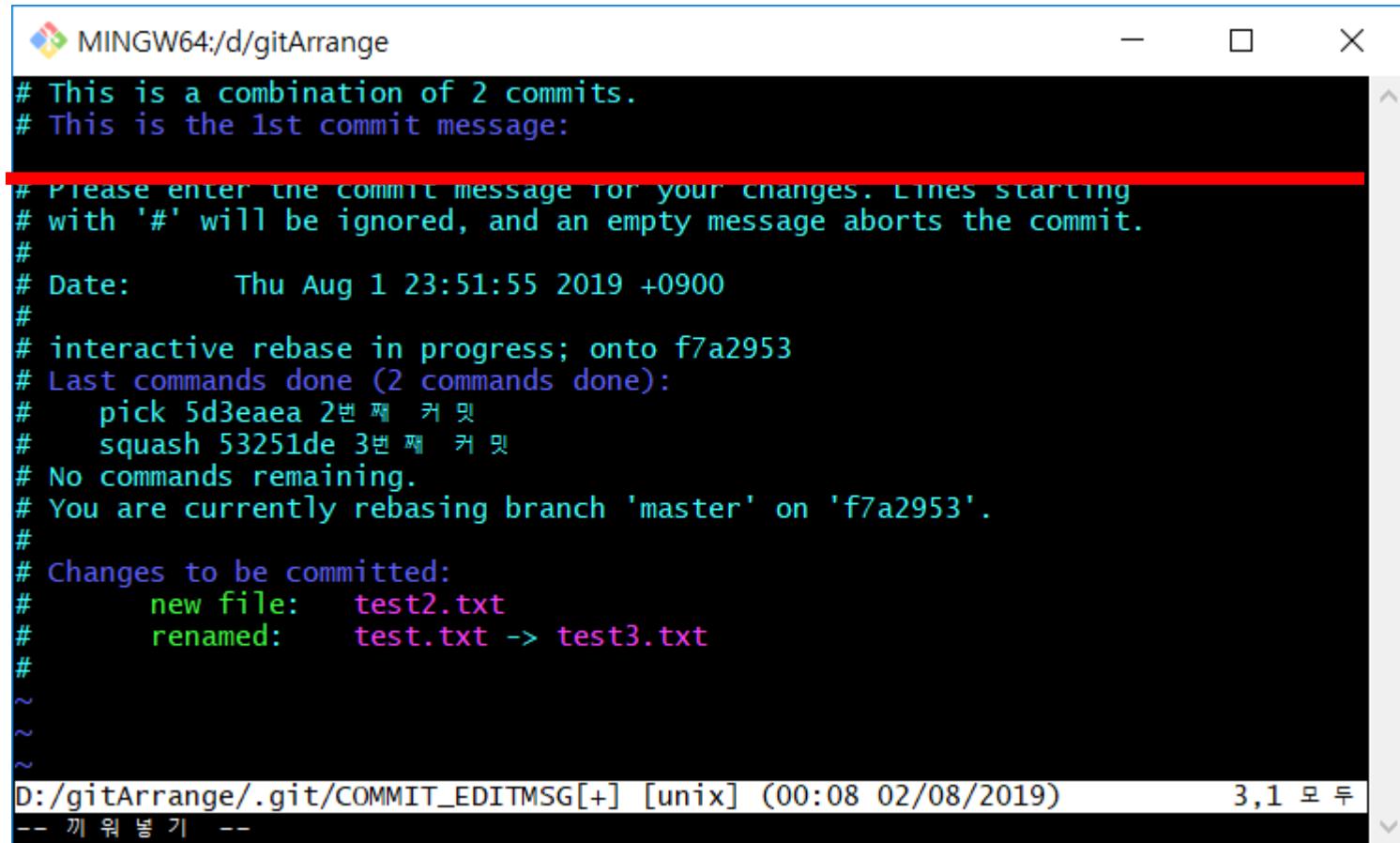
# This is the commit message #1:

3번째 커밋

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Thu Aug 1 23:51:55 2019 +0900
#
# interactive rebase in progress; onto f7a2953
# Last commands done (2 commands done):
#   pick 5d3eaea 2번째 커밋
#   squash 53251de 3번째 커밋
# No commands remaining.
# You are currently rebasing branch 'master' on 'f7a2953'.
#
# Changes to be committed:
D:/gitArrange/.git/COMMIT_EDITMSG [unix] (00:08 02/08/2019)           1,1 꼭 대기
"D:/gitArrange/.git/COMMIT_EDITMSG" [유닉스] 25L, 651C
```

다음과 같이 화면이 나오면 정상적으로 리베이스가 진행된 겁니다.  
아까와 같이 i를 누르고 빨간색 줄 위부분에 커서를 맞춥니다.

# 6. 커밋 합치기



The screenshot shows a terminal window titled 'MINGW64:/d/gitArrange'. The window contains a git commit message editor. The message starts with '# This is a combination of 2 commits.' and '# This is the 1st commit message:'. It then provides instructions for entering the commit message, mentioning that lines starting with '#' will be ignored. Below these instructions, the commit message continues with '# Date: Thu Aug 1 23:51:55 2019 +0900', '# interactive rebase in progress; onto f7a2953', '# Last commands done (2 commands done):', '# pick 5d3eaea 2번째 커밋', '# squash 53251de 3번째 커밋', '# No commands remaining.', '# You are currently rebasing branch 'master' on 'f7a2953'.', '#', '# Changes to be committed:', '# new file: test2.txt', '# renamed: test.txt -> test3.txt', '#', '~', '~', '~'. At the bottom of the editor, the path 'D:/gitArrange/.git/COMMIT\_EDITMSG[+]' is shown along with the date '(00:08 02/08/2019)' and the status '3,1 모두'. A status bar at the bottom of the terminal window displays the Korean text '-- 끼워 넣기 --'.

This is the 1st commit message: 아랫부분까지 지웁니다.

# 6. 커밋 합치기

The terminal window shows the following output:

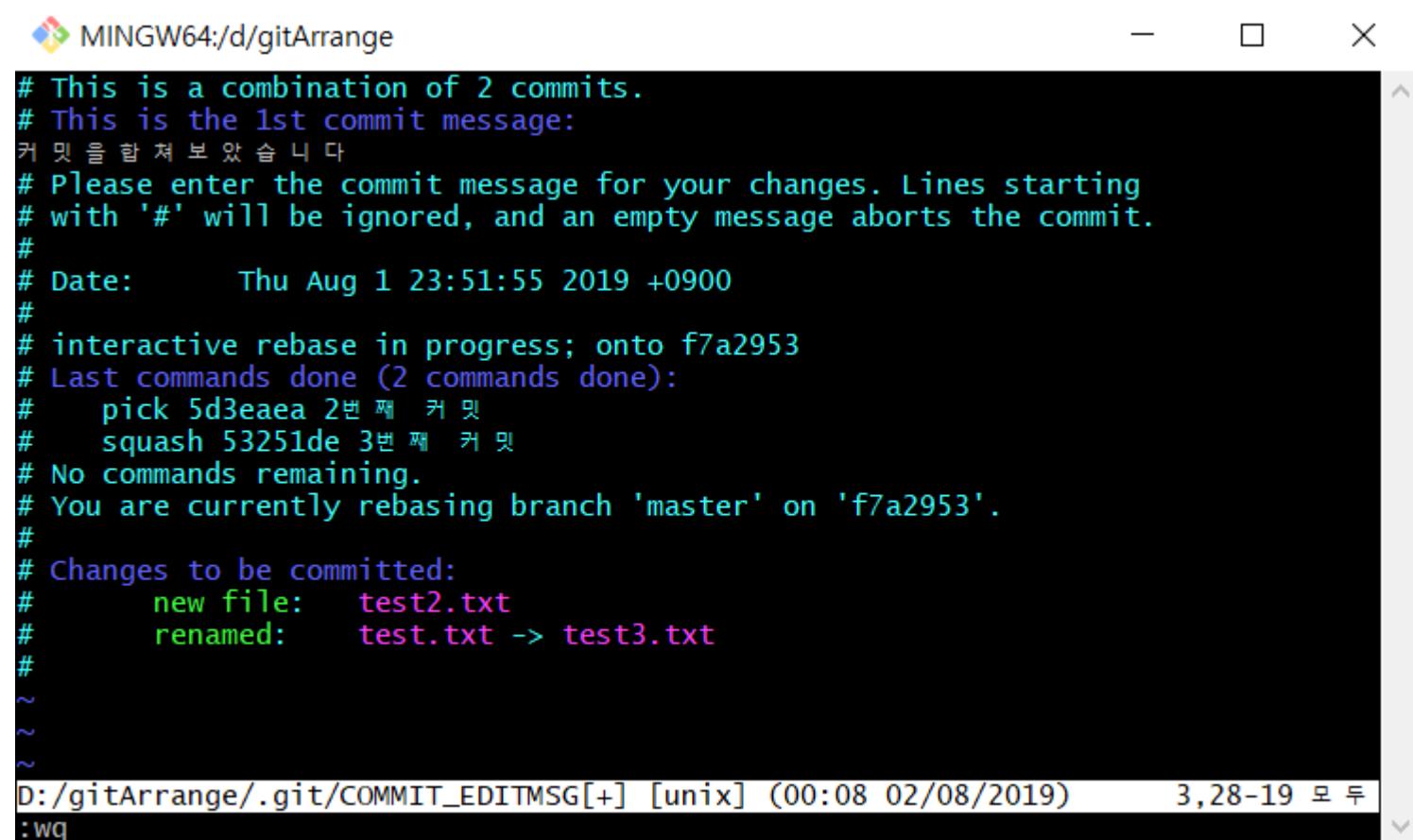
```
MINGW64:/d/gitArrange
# This is a combination of 2 commits.
# This is the 1st commit message:

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Thu Aug 1 23:51:55 2019 +0900
#
# interactive rebase in progress; onto f7a2953
# Last commands done (2 commands done):
#   pick 5d3eaea 2번째 커밋
#   squash 53251de 3번째 커밋
# No commands remaining.
# You are currently rebasing branch 'master' on 'f7a2953'.
#
# Changes to be committed:
#       new file:    test2.txt
#       renamed:    test.txt -> test3.txt
#
~ ~ ~
D:/gitArrange/.git/COMMIT_EDITMSG[+] [unix] (00:08 02/08/2019) 3,1 모두
-- 끼워 넣기 --
```

To the right of the terminal is a diagram of a Git commit history. It shows a vertical line of dots representing commits. Above the line, there are several branches listed: master (selected), origin/master, origin/HEAD, upstream/master, and 테스트용 (Test). The commit '2번째 커밋' (2nd Commit) is highlighted with a red box. Below the diagram, the text 'Initial commit' is visible.

빨간 줄 위부분에 변경할 커밋 메시지를 적습니다.  
커밋 메시지란 오른쪽 그림에 보여지는 네모 친 부분과 같습니다.

# 6. 커밋 합치기

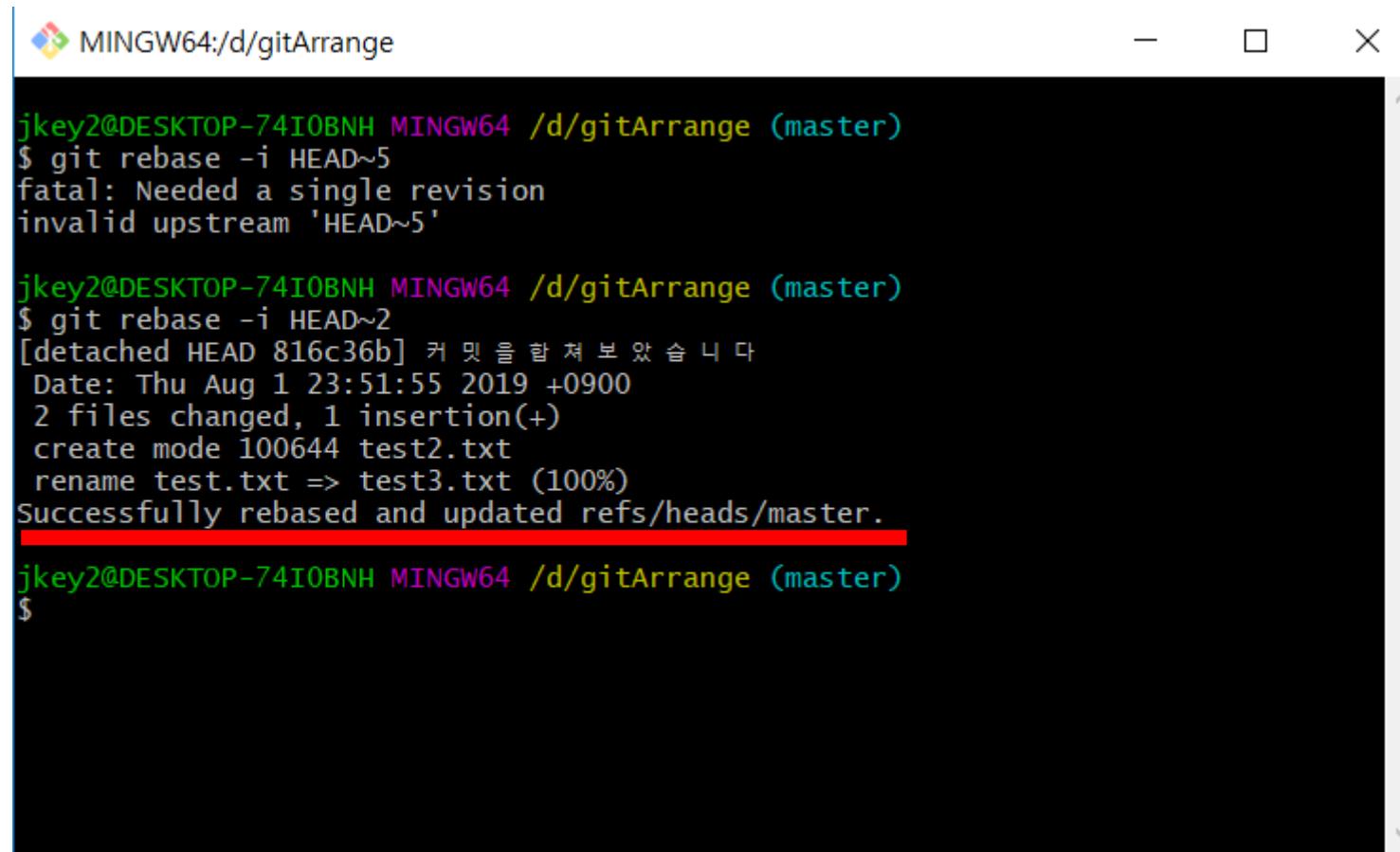


The screenshot shows a terminal window titled "MINGW64:/d/gitArrange". The window contains the following text:

```
# This is a combination of 2 commits.
# This is the 1st commit message:
커밋을 합쳐 보았습니다
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Thu Aug 1 23:51:55 2019 +0900
#
# interactive rebase in progress; onto f7a2953
# Last commands done (2 commands done):
#   pick 5d3eaea 2번째 커밋
#   squash 53251de 3번째 커밋
# No commands remaining.
# You are currently rebasing branch 'master' on 'f7a2953'.
#
# Changes to be committed:
#       new file:    test2.txt
#       renamed:    test.txt -> test3.txt
#
~ ~ ~
D:/gitArrange/.git/COMMIT_EDITMSG[+] [unix] (00:08 02/08/2019) 3,28-19 모두
:wq
```

수정을 다 마친 후 ESC를 누르고 :wq를 입력하고 Enter를 누릅니다.

# 6. 커밋 합치기



The screenshot shows a terminal window titled 'MINGW64:/d/gitArrange'. It displays the following command and its output:

```
jkey2@DESKTOP-74I0BNH MINGW64 /d/gitArrange (master)
$ git rebase -i HEAD~5
fatal: Needed a single revision
invalid upstream 'HEAD~5'

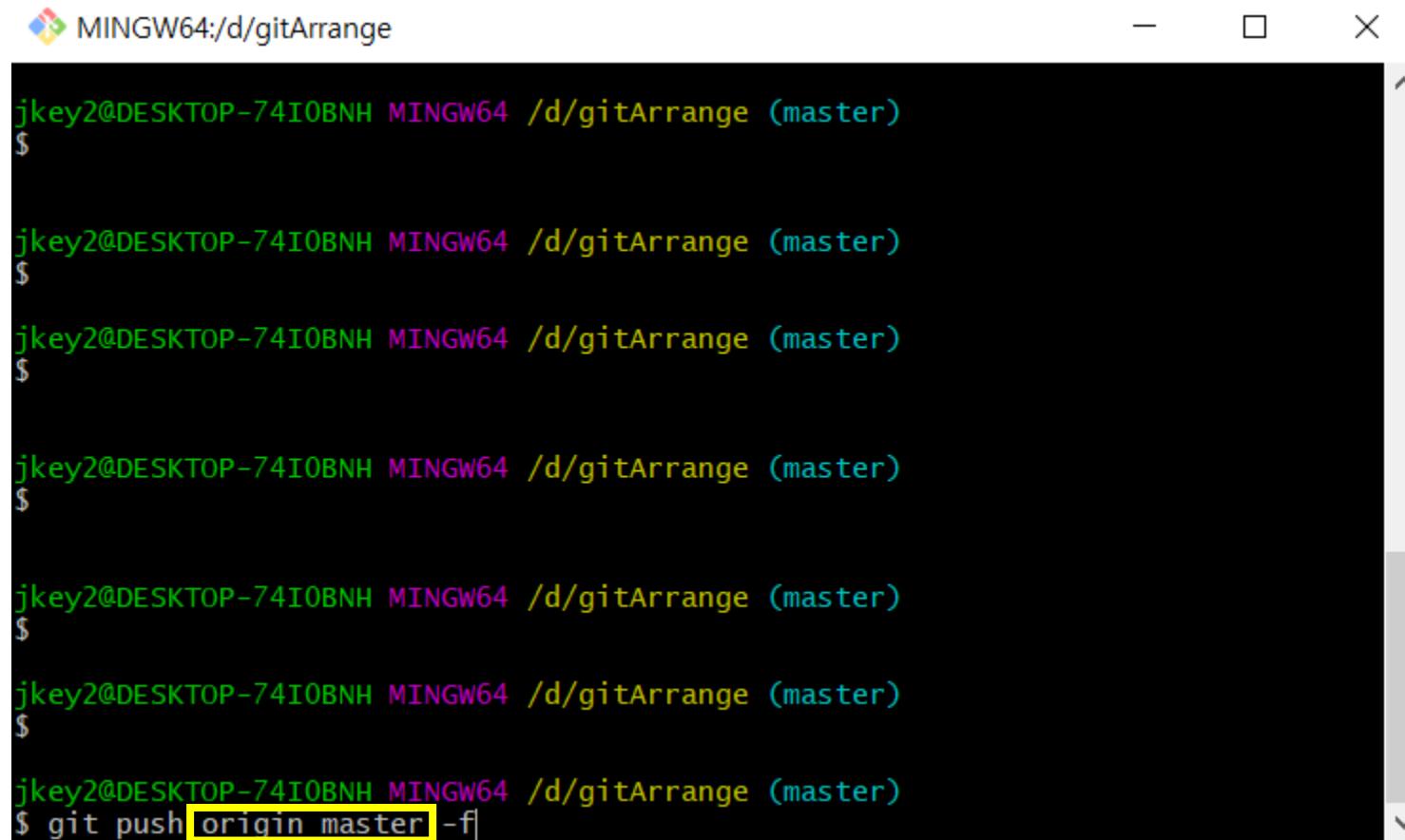
jkey2@DESKTOP-74I0BNH MINGW64 /d/gitArrange (master)
$ git rebase -i HEAD~2
[detached HEAD 816c36b] 커밋을 합쳐보았습니다
Date: Thu Aug 1 23:51:55 2019 +0900
2 files changed, 1 insertion(+)
create mode 100644 test2.txt
rename test.txt => test3.txt (100%)
Successfully rebased and updated refs/heads/master.

jkey2@DESKTOP-74I0BNH MINGW64 /d/gitArrange (master)
$
```

The last line of the output, 'Successfully rebased and updated refs/heads/master.', is highlighted with a red rectangle.

빨간 줄 윗부분처럼 출력되면 정상적으로 리베이스가 된 것입니다.

# 6. 커밋 합치기



The screenshot shows a terminal window titled "MINGW64:/d/gitArrange". It displays a series of identical commit messages from a user named "jkey2" on a machine named "DESKTOP-74I0BNH". The commits are in English and mention "gitArrange" and "master". The terminal then shows the command "\$ git push origin master -f", where "origin master" is highlighted in yellow.

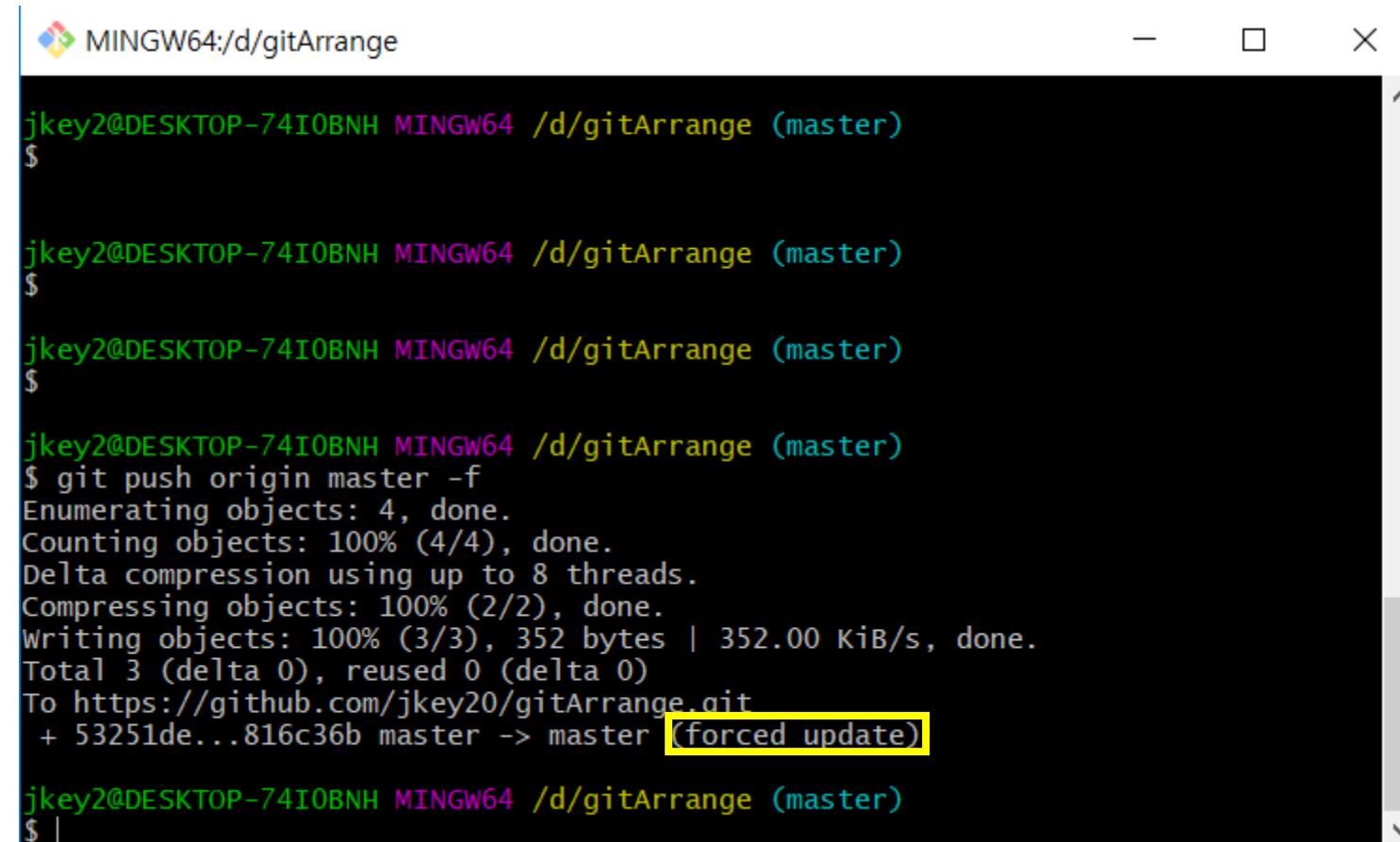
```
jkey2@DESKTOP-74I0BNH MINGW64 /d/gitArrange (master)
$  
  
jkey2@DESKTOP-74I0BNH MINGW64 /d/gitArrange (master)
$ git push origin master -f
```

그 후 git push origin master -f로 강제푸쉬를 해줍니다.

노란색 네모 부분은 푸쉬를 넣을 곳을 지정해주는 부분입니다.

만약 origin에있는 feature 브랜치에 있는 커밋들을 합치고 있다면 master를 feature로 바꿔주어야 합니다.

# 6. 커밋 합치기

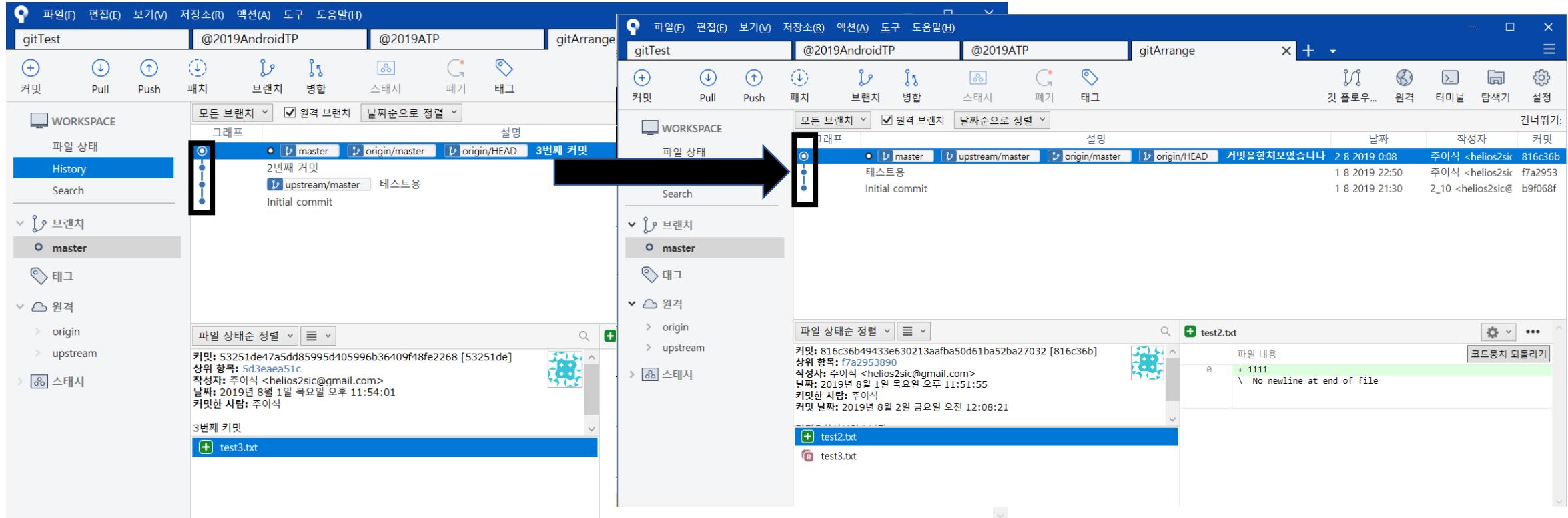


The screenshot shows a terminal window titled "MINGW64:/d/gitArrange". The user is at the prompt "jkey2@DESKTOP-74I0BNH MINGW64 /d/gitArrange (master)". They run the command "\$ git push origin master -f". The output shows the process of pushing three objects to a GitHub repository, with the final line indicating a forced update: "To https://github.com/jkey20/gitArrange.git + 53251de...816c36b master -> master (forced update)". The phrase "(forced update)" is highlighted with a yellow box.

```
jkey2@DESKTOP-74I0BNH MINGW64 /d/gitArrange (master)
$ git push origin master -f
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 352 bytes | 352.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/jkey20/gitArrange.git
 + 53251de...816c36b master -> master (forced update)
```

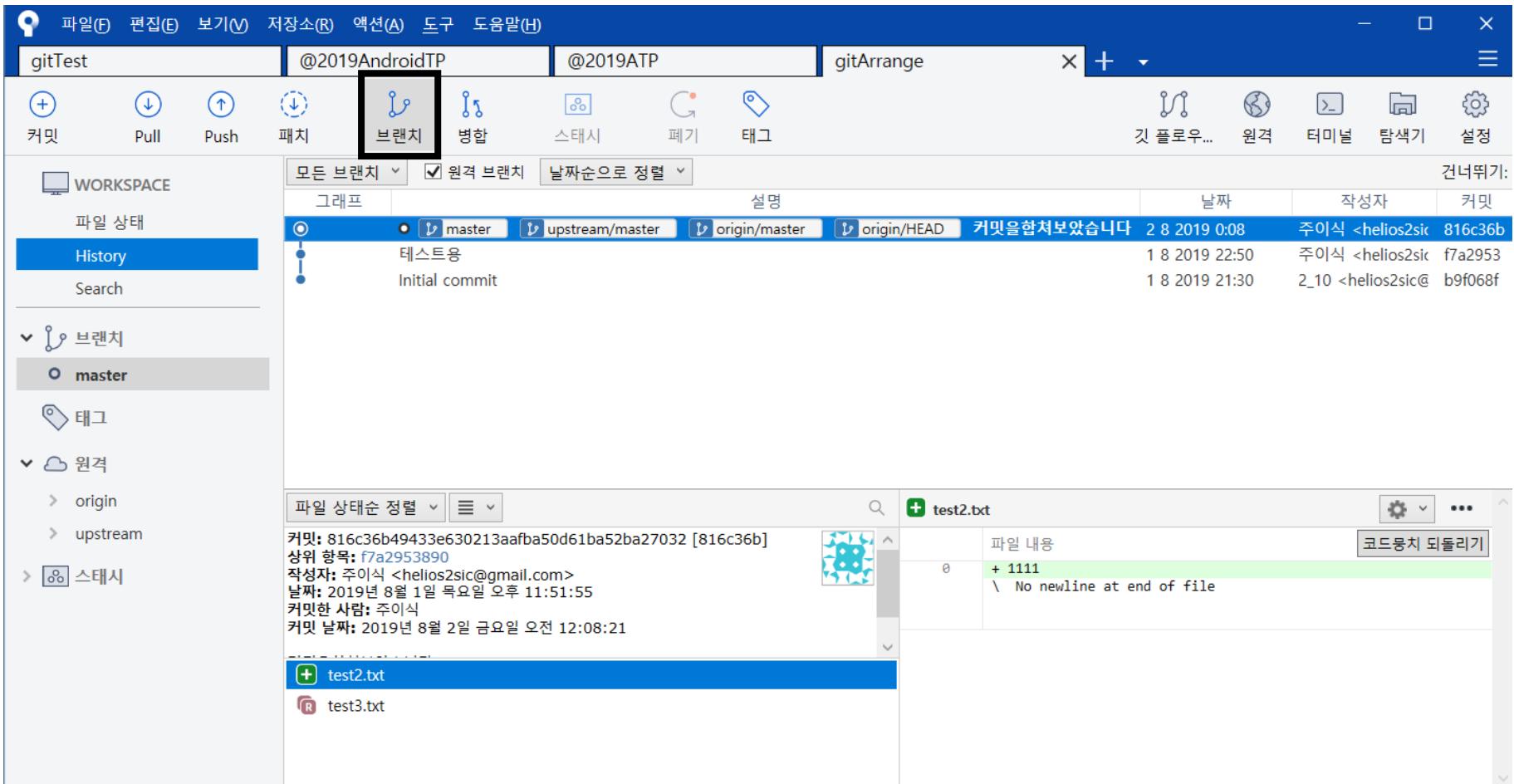
다음과 같이 forced update가 뜨면 정상적으로 푸쉬가 들어간 겁니다.

# 6. 커밋 합치기



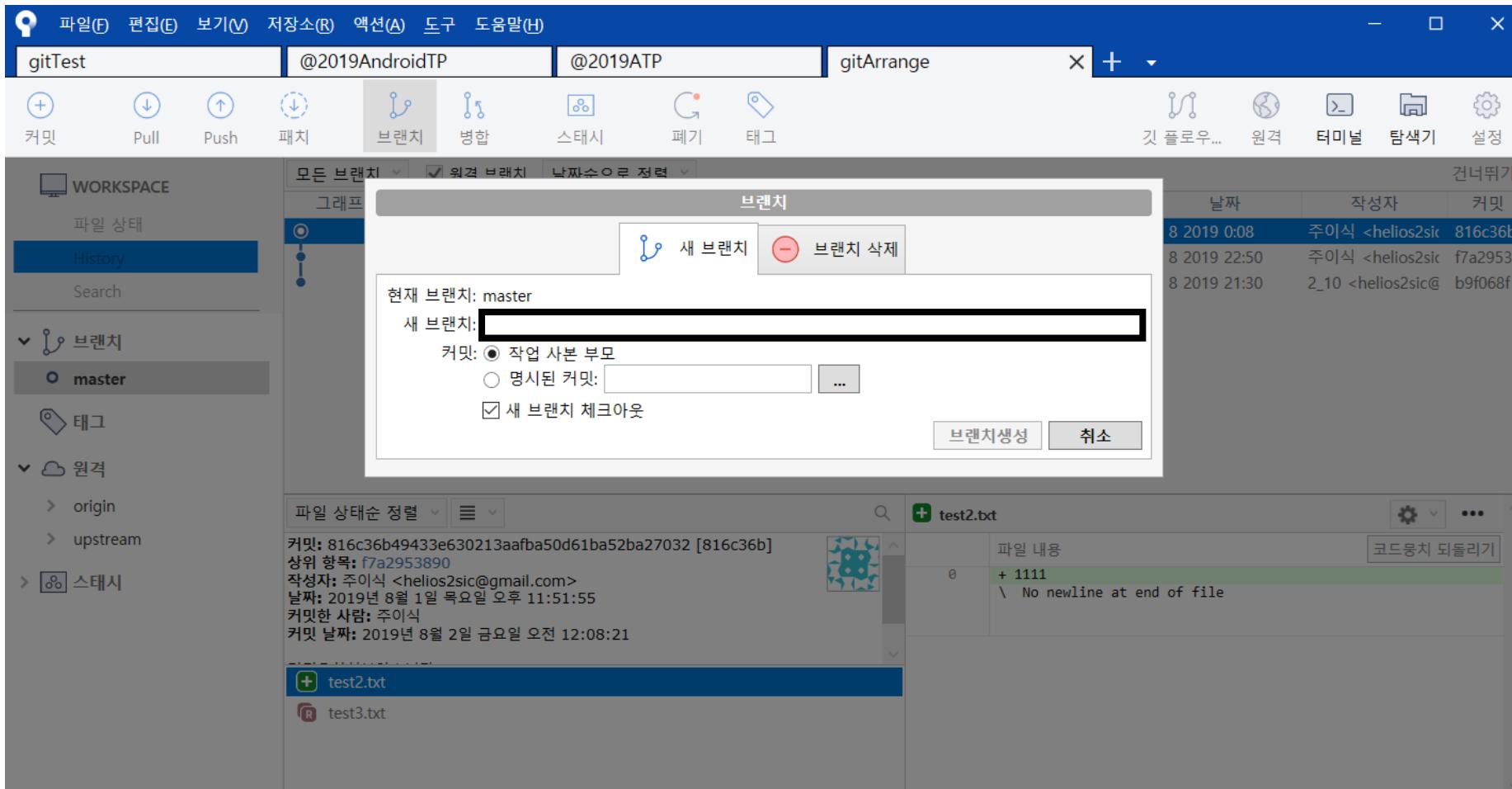
그 다음 소스트리로 돌아가서 패치를 누르면 커밋이 합쳐져 있습니다.

# 7. 브랜치 생성



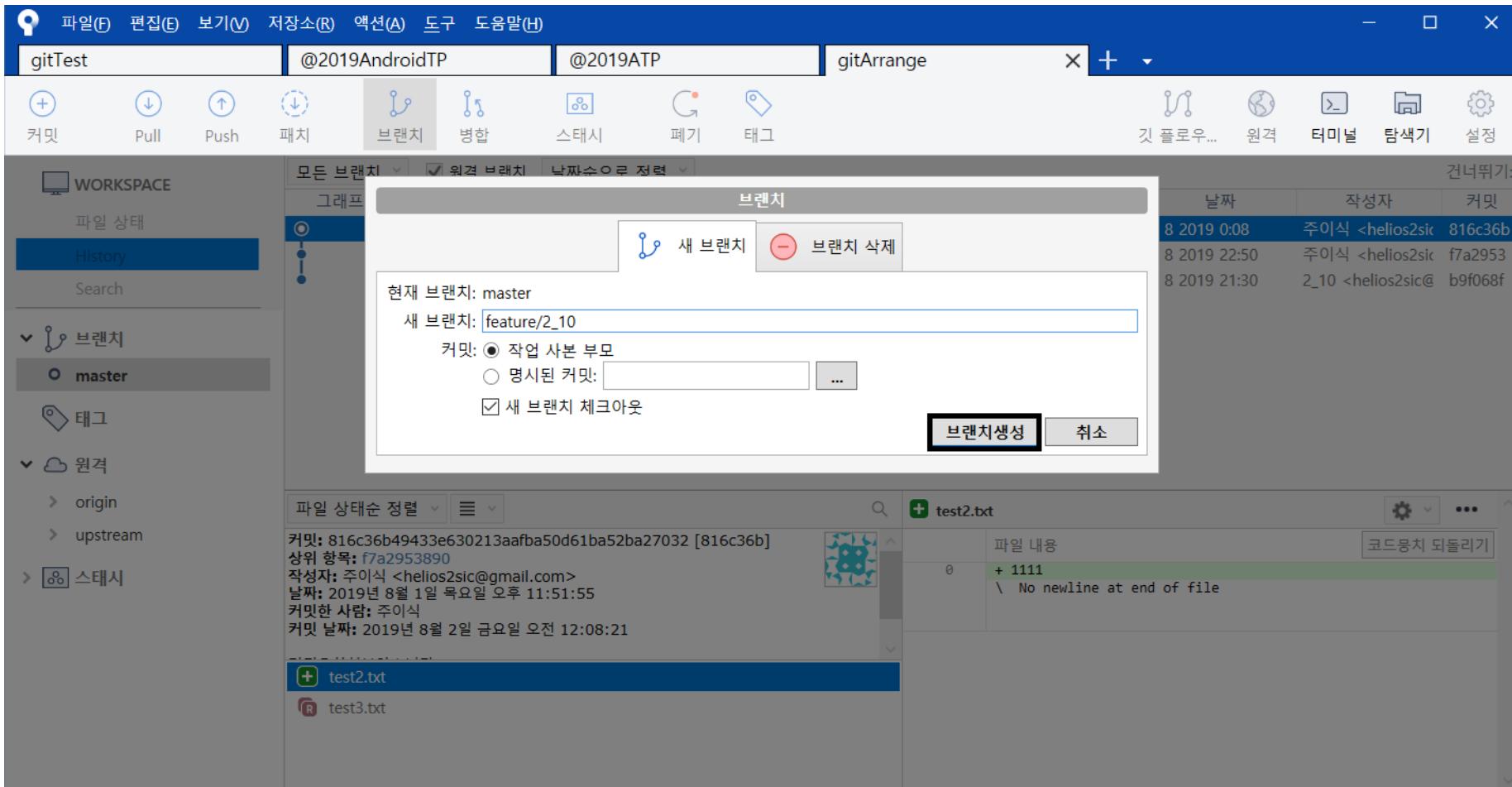
개발을 계속 하다보면 기능별로 파일들을 모아놓고 싶은 경우가 생깁니다.  
그럴 때 브랜치를 만들어서 분류를 할 수 있습니다.  
네모 칸을 눌러 브랜치를 생성합니다.

# 7. 브랜치 생성



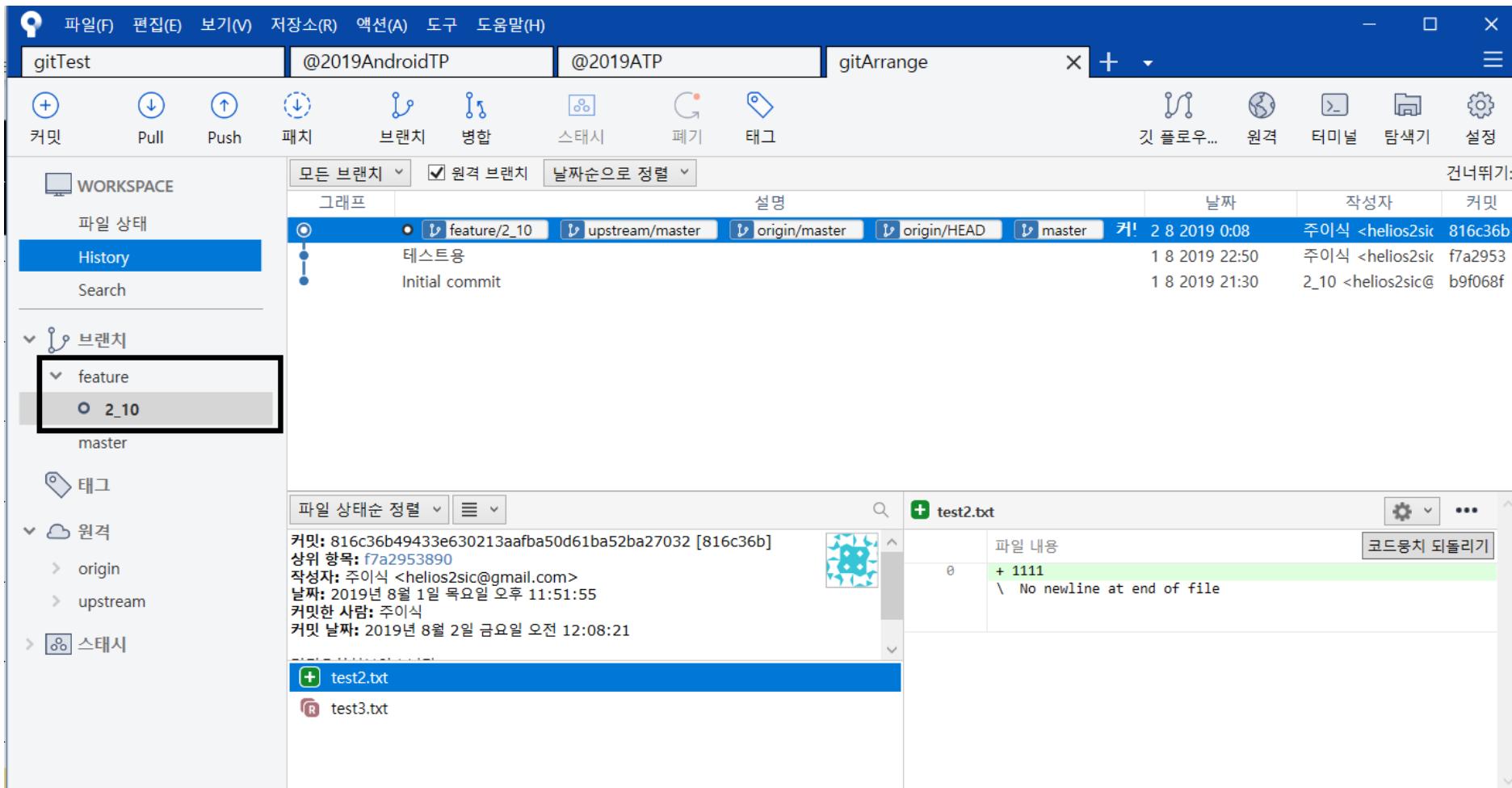
다음 화면이 나오면 새 브랜치 부분에 새로 만들어질 브랜치의 이름을 입력합니다.

# 7. 브랜치 생성



입력이 끝나면 브랜치를 생성합니다.

# 7. 브랜치 생성



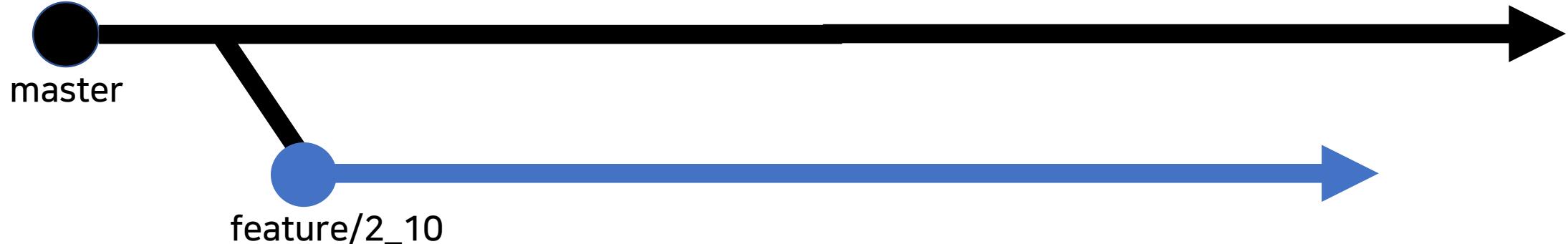
네모 친 부분처럼 새로운 브랜치가 생기면 성공입니다.

# 브랜치를 이해하자



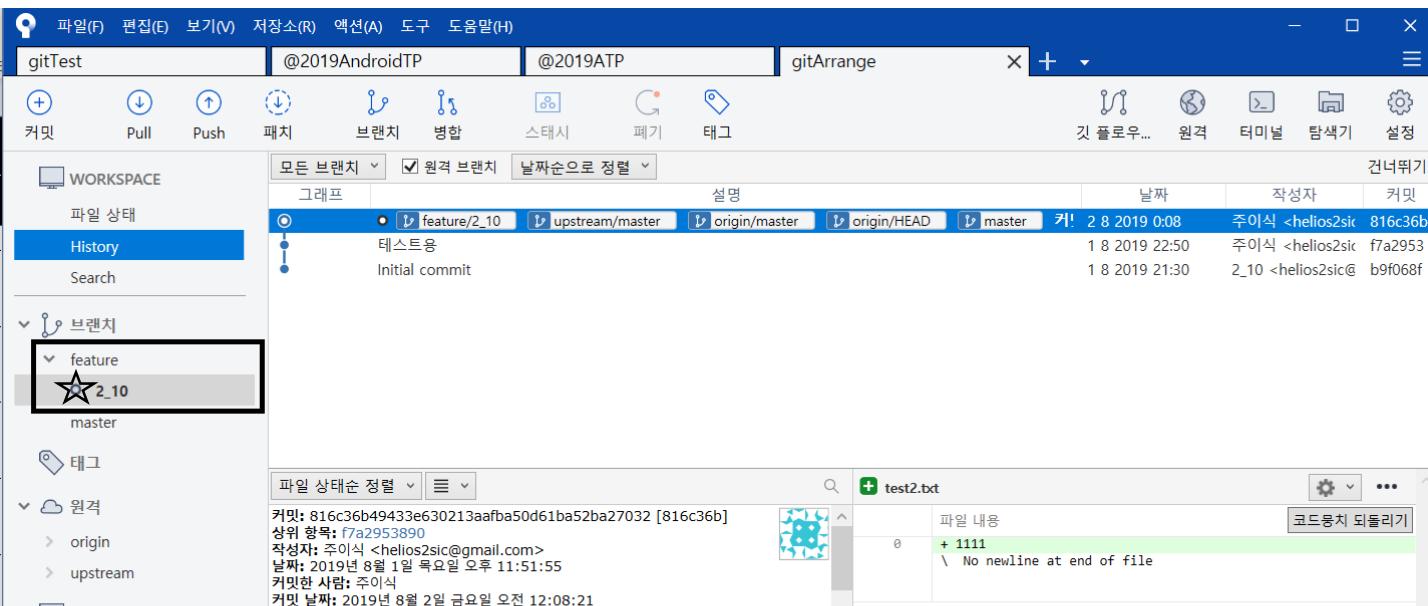
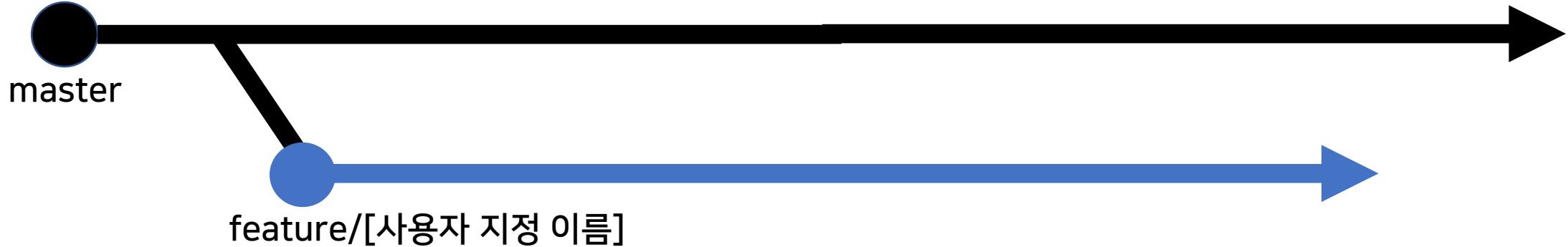
우리가 처음 clone을 만들면 master 브랜치 하나만 존재합니다.

# 브랜치를 이해하자



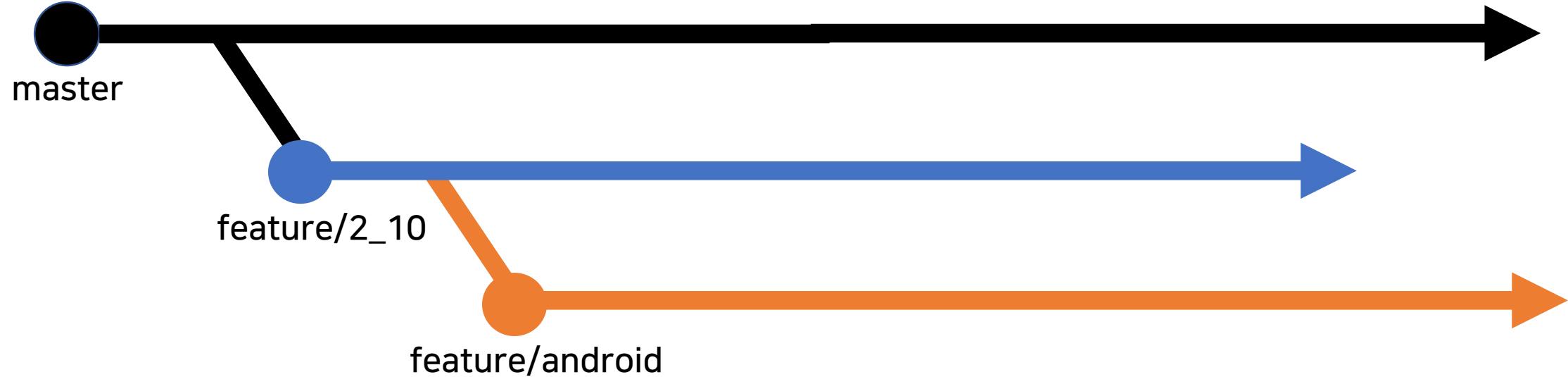
여기서 새로운 브랜치인 feature/[사용자 지정 이름]을 만들어주면  
다음과 같이 master에서 브랜치가 생성됩니다.  
여기서는 2\_10으로 생성했습니다.

# 브랜치를 이해하자



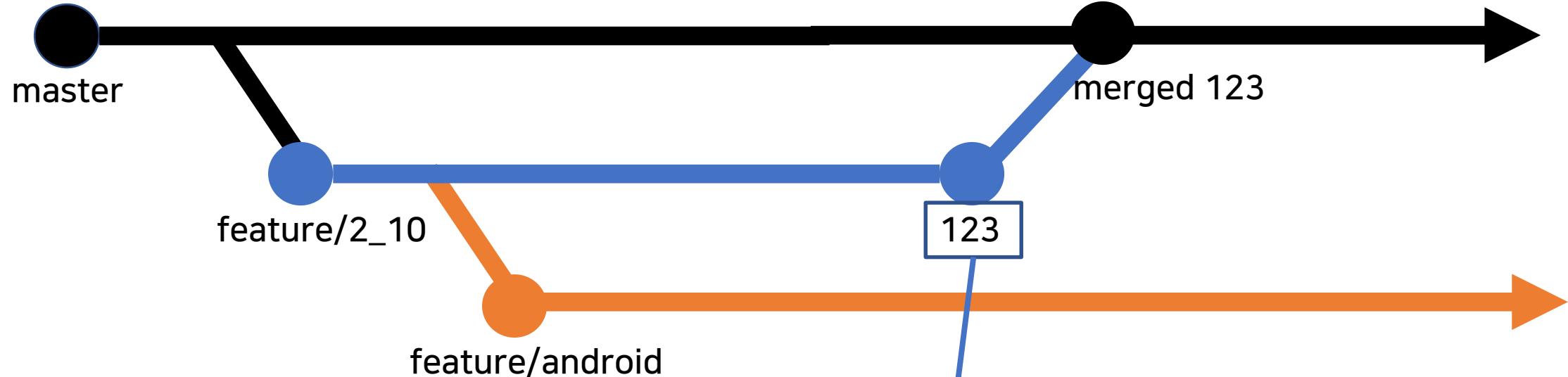
이 상태에서 feature/[사용자 지정 이름]에 체크아웃(별표 친 부분)이 되어있는 상태에서 새롭게 브랜치를 생성해 보겠습니다.

# 브랜치를 이해하자



새롭게 feature/android로 이름을 지정해주고 브랜치를 만들면 다음과 같이 브랜치가 생성됩니다.

# 브랜치를 이해하자



여기서 feature/2\_10에 커밋을 한 개 올리고 올라와 있는 커밋을 merge시키면 다음과 같이 진행됩니다.

# 주의사항

Readme.md 오류가 나면 Readme.md파일을 꼭 수정해서 conflict를 해결해주세요.

메인 페이지 소유자는 pull request를 꼼꼼히 읽어보고 협업자들이 수정한 사항들을 확인해주세요.

## 소스트리에서 왼쪽 그래프 부분은 자신이 체크아웃한 부분입니다!

팀원들과 협업시에 각자의 그래프들이 달라보이는 것은 지극히 정상이며 fork를 뜯 시점에 따라서도 그래프가 달라질 수 있습니다!

## 오류가 나면 꼭 구글링으로 원인을 찾은 후 커밋 푸쉬 pull request 등을 진행하세요!

급하게 오류를 해결하려고 하면 오히려 오류가 나서 파일들이 날아갈 수 있습니다!

## 기능개발용 브랜치를 꼭 만드세요!

기능개발용 브랜치(feature/~)를 만들지 않고 협업을 진행할 시 master에 오류가 생기면 모든 파일들이 날아갈 수 있습니다!

꼭 브랜치를 나눠서 오류가 생겨도 백업을 할 수 있게 협업을 해주세요!

## Pull request의 제목과 내용은 수정이 되지 않습니다!

Pull request를 보낼때는 자신이 수정한 부분과 상세한 설명을 적어주세요!

# 주의사항

Readme.md 오류가 나면 Readme.md파일을 꼭 수정해서 conflict를 해결해주세요.

메인 페이지 소유자는 pull request를 꼼꼼히 읽어보고 협업자들이 수정한 사항들을 확인해주세요.

## 소스트리에서 왼쪽 그래프 부분은 자신이 체크아웃한 부분입니다!

팀원들과 협업시에 각자의 그래프들이 달라보이는 것은 지극히 정상이며 fork를 뜯 시점에 따라서도 그래프가 달라질 수 있습니다!

## 오류가 나면 꼭 구글링으로 원인을 찾은 후 커밋 푸쉬 pull request 등을 진행하세요!

급하게 오류를 해결하려고 하면 오히려 오류가 나서 파일들이 날아갈 수 있습니다!

## 기능개발용 브랜치를 꼭 만드세요!

기능개발용 브랜치(feature/~)를 만들지 않고 협업을 진행할 시 master에 오류가 생기면 모든 파일들이 날아갈 수 있습니다!

꼭 브랜치를 나눠서 오류가 생겨도 백업을 할 수 있게 협업을 해주세요!

## Pull request의 제목과 내용은 수정이 되지 않습니다!

Pull request를 보낼때는 자신이 수정한 부분과 상세한 설명을 적어주세요!

“혼자서는 우리는 거의 아무 것도 못한다.  
함께 하면 우리는 그렇게 많은 것을 할 수 있다”  
-헬렌 켈러-

수고하셨습니다!

