



깃 특강 공유용

Git 소개

[https://ko.wikipedia.org/wiki/깃_\(소프트웨어\)](https://ko.wikipedia.org/wiki/깃_(소프트웨어))

시작소 경상, 파일 수사, 전자 허기와 함께

원저자

리눅스 토르발스^[1]

“리눅스를 만든 그 사람, 그분”

설명

깃(Git /gɪt/[5])은 컴퓨터 파일의 변경사항을 추적하고, 여러 명의 사용자들 간에 해당 파일들의 작업을 조율하기 위한 스냅샷 스트림 기반의 분산 버전 관리 시스템이다. 소프트웨어 개발에서 소스 코드 관리에 주로 사용되지만[6] 어떠한 파일 집합의 변경사항을 지속적으로 추적하기 위해 사용될 수 있다.

“파일의 변경사항을 추적하고, 관리하기 위한 프로그램”

책추천

first edition

<https://github.com/progit/progit>

second edition

<https://github.com/progit/progit2>

<https://git-scm.com/book/ko/v2>

“깃에 대한 깊은 이해와, 개념정리가 필요하다면 이책만 보면 끝이다.”

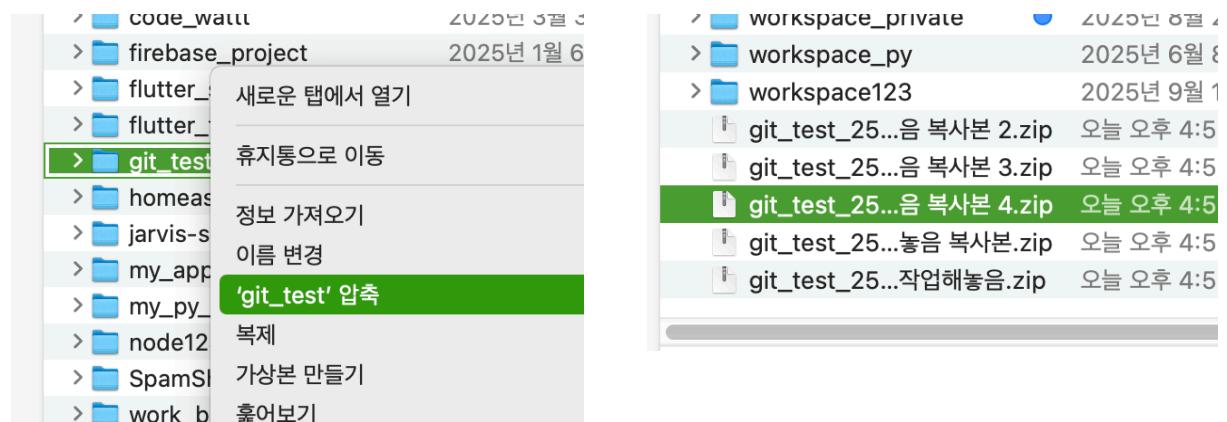
“심지어 무료로 볼수있다.”

왜 깃이 필요한가?

그냥 사용할수도 있지만, 이 복잡하고 귀찮은걸 왜 써야 될까?

1. 첫번째는 버전관리 측면입니다.
2. 두번째는 협업의 측면 입니다.

사례



"압축하고... 복사본을 만들고..."

→ 빠르고 단순하지만, 이런 방법으로는 한계가 있다.

깃 설치

맥에 기본 설정을 마쳤다면, git은 기본적으로 설치가 되어 있을 것이다.

직접 자신의 pc에서 확인해보자.

```
git --version
```

만약 없다면, brew 등을 통해서 설치하자.

```
brew install git
```

사용자 등록

"파일에 대한 변경사항을 기록하고, 추적, 관리하기 위해서는 누가 언제 어떻게 했는지에 대한 정보가 필요하다."

```
git config --global user.name "홍길동"  
git config --global user.email "hong@example.com"
```

설정을 완료하고 확인해보자.

```
git config --list
```

로컬에서 파일 관리하기

이제는 Git 을 통해서 관리해보자.

깃을 쓸때는 기본적으로 6가지를 기억하자.

1. git init
2. git status
3. git add
4. git commit
5. git push
6. git pull

git init

말그대로 우리가 작업할 폴더를 git 폴더로 초기화 하는 명령어 이다.

작업폴더를 만들고, 그 안에서 해당 명령어를 입력해보자.

```
mkdir [폴더명]
```

```
cd [해당폴더명]
```

```
git init
```

git status

해당 명령어는 현재 git 폴더의 상태를 체크해볼수 있다. 지금 내가 어떤 상태인지 확인하기 위해 자주 사용하면 좋다.

```
git status
```

해당 상태에 대한 힌트도 얻을수 있다.

```
Untracked files:  
(use "git add <file>..." to include in what will be committed)
```

git add

git add는 우리가 관리하고자 하는 파일을 추적 관리 하도록, 내부적으로 스테이징 이라는 공간에 올리는 작업이다.

```
git add [파일명]
```

전체파일을 add하고 싶다면 . 을 사용하자

```
git add .
```

git commit

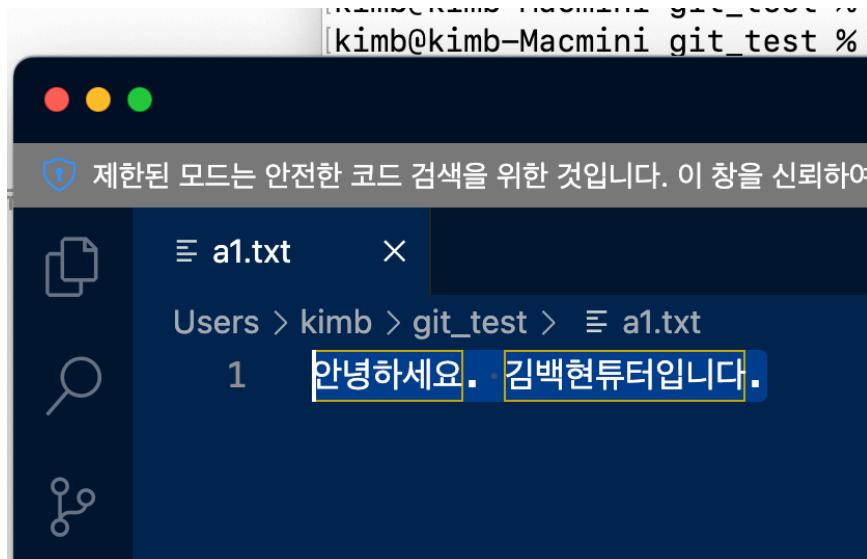
파일 수정과 add를 거쳐, 최종적으로 commit을 통해 변경사항을 최종 저장하는 단계이다.

```
git commit -m "커밋메세지"
```

변경사항(소스수정)

“결국 핵심은 변경사항에 대한 관리이다.”

“파일(소스)을 수정하고 어떤일이 생기는 확인해보자.”



```
kimb@kimb-Macmini git_test %
[+] 제한된 모드는 안전한 코드 검색을 위한 것입니다. 이 창을 신뢰하여

a1.txt
Users > kimb > git_test > a1.txt
1 안녕하세요. 김백현튜터입니다.
```

직접 파일의 내용을 수정하고, 상태를 확인해보자.

```
git status
```

```
kimb@kimb-Macmini git_test % git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      modified:   a1.txt
```

수정된 파일은 다시 add → commit 의 흐름으로 진행된다.

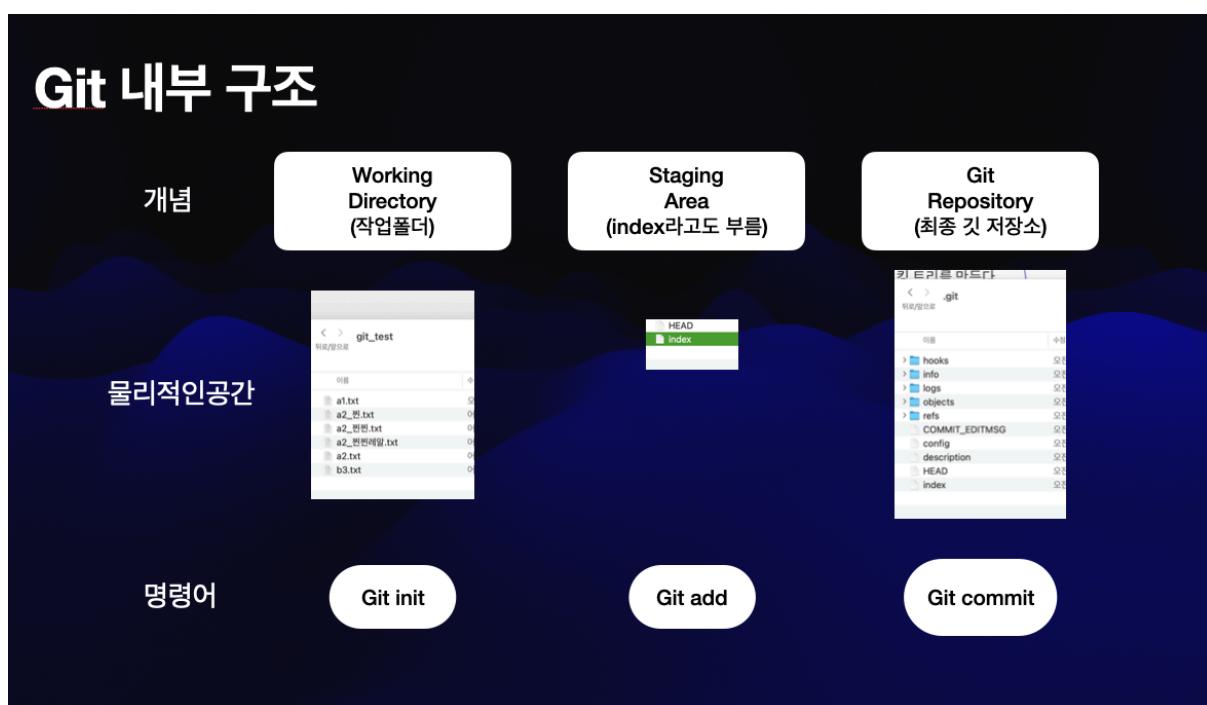
commit완료후 log를 확인해보자.

git log

구조

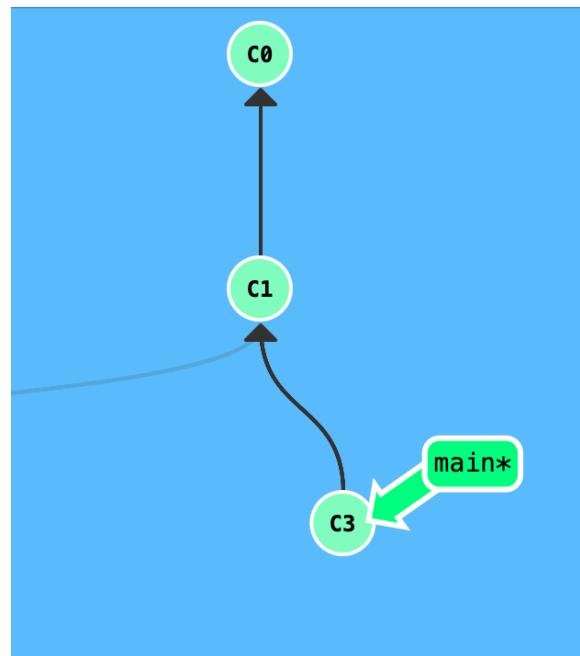
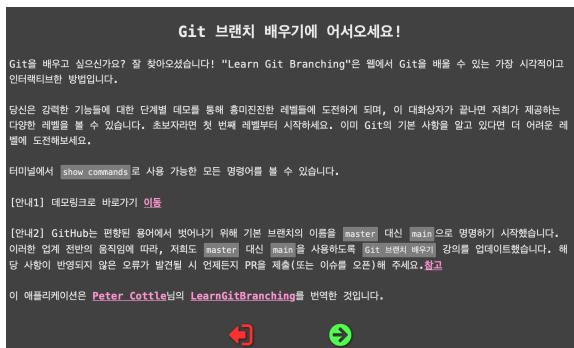
도저히 이해가 잘 안된다. 그림으로 한번 살펴보자.

실제 물리적인 폴더와 파일을 눈으로 보면서 이해하면 더 좋다.



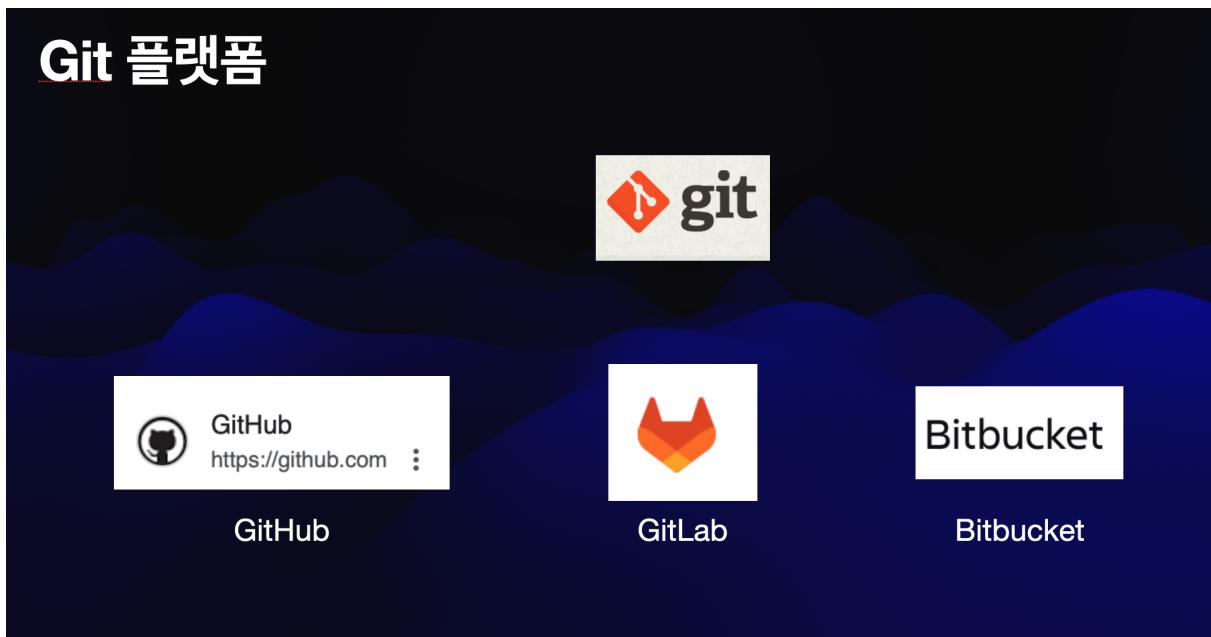
깃 연습하기 좋은 사이트

<https://learngitbranching.js.org/?locale=ko>



깃 파생 플랫폼들

오픈 소스인 Git을 기반으로, 여러가지 파생된 플랫폼들이 생겨났다.



Github

Github 는 전세계 최대의 오픈소스 공유 플랫폼이다.

<https://ko.wikipedia.org/wiki/깃허브>

<https://github.com/>

Gitlab, Bitbucket 등

주로 사내, 기업의 프로젝트에서 쓰인다.

<https://about.gitlab.com/>

<https://bitbucket.org/product/>

Github를 사용해보자

<https://github.com/>

1. 가입한다
2. 자신의 repository(저장소) 생성
3. git clone (저장소 복제, 동기화)
4. Collaborator(협업자) 등록
5. 소스 코드 작업
6. commit and push

repository 생성

순서대로 진행하고, repository를 만든다.

The screenshot shows the GitHub repository creation interface. On the left, there's a sidebar with 'Top repositories' and a search bar. In the center, the 'General' settings section is displayed. It includes fields for 'Owner' (set to 'johnkim-codewatt'), 'Repository name' (set to 'abcd_test'), and a note that it is available. A green checkmark indicates availability.

소스를 공개할지, 비공개로 할지 결정한다.

The screenshot shows the 'Choose visibility' section. It asks 'Choose who can see and commit to this repository' and provides a dropdown menu set to 'Public'.

맨처음 진입해서 읽게 되는 문서이다.

The screenshot shows the 'Add README' section. It says 'READMEs can be used as longer descriptions.' and has a toggle switch labeled 'On' which is turned on.

깃으로 관리하기 부담스러운 자잘한 파일이나, 복잡한 설정, 보안관련등은 .gitignore파일에 작성하면, 추적관리 되지 않게 할수 있다.

The screenshot shows the 'Add .gitignore' section. It explains that '.gitignore tells git which files not to track.' and provides a dropdown menu currently set to 'No .gitignore'.

별로 어렵지 않게 나만의 원격저장소를 만들었다.

The screenshot shows a GitHub repository page for 'abcd_test'. At the top, there's a green circular icon with a white 'W', the repository name 'abcd_test', and a 'Public' button. To the right are 'Pin' and 'Watch' buttons. Below the header, there are navigation buttons for 'main' (with a dropdown arrow), a search bar ('Go to file'), and a 'Code' button. A sidebar on the right lists repository statistics: 'No branches', '1 Commit', 'No issues', 'No pull requests', and 'No discussions'. The main content area shows a single commit by 'johnkim-codewatt' with the message 'Initial commit' and timestamp '0b7661a · now'. Below the commit is a file named 'README.md' with the content 'Initial commit' and timestamp 'now'. The 'README' file itself is shown with the text 'abcd_test'.

git clone

우리 Pc에 새로운 폴더를 하나 만들자

```
mkdir git_test2
```

폴더 내부로 진입한다.

```
cd git_test2
```

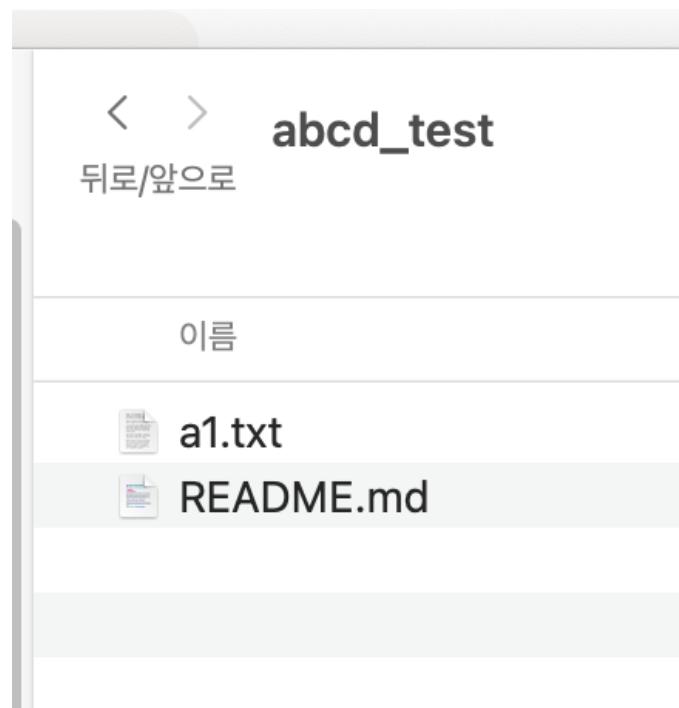
이제 여기가 우리의 새로운 작업폴더이다. 아래의 명령어를 입력한다.

```
git clone git_test2
```

```
[kimb@kimb-Macmini git_test2 % git clone https://github.com/johnkim-codewatt/abcd_test.git
Cloning into 'abcd_test'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
```

이렇게 하면 git init 과정없이도 .git 폴더 구조가 만들지고(확인해보자), 원격 저장소 복제가 완료 되었다.

테스트를 위해 기존에 있던 파일을 복사/붙여넣기했다.



새로운 파일을 추가하기 위해 순서대로 진행해보자.

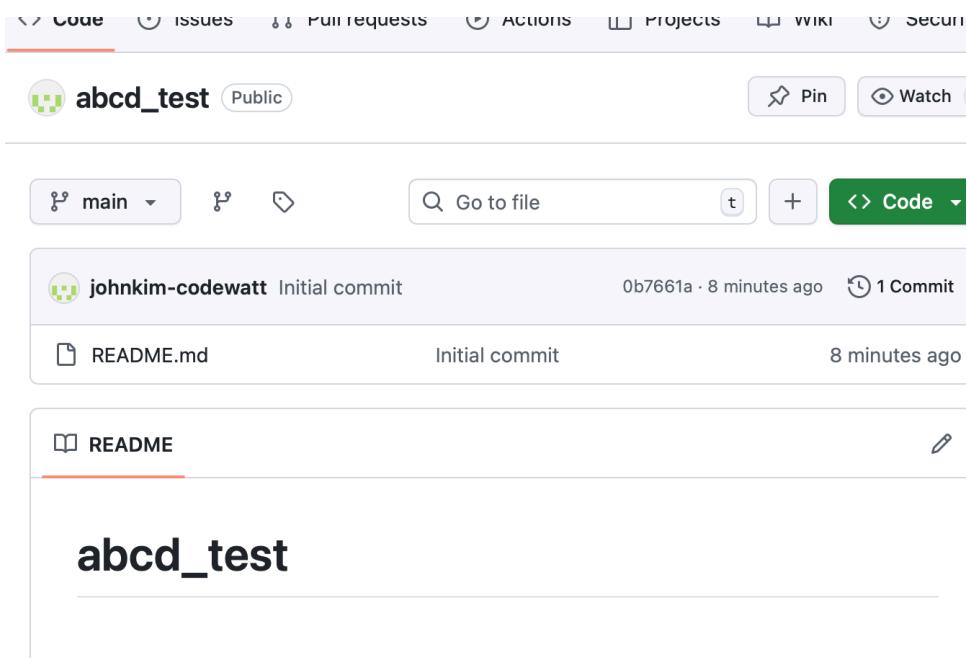
git status

git add

git commit

그러나, 원격 저장소에는 아무런 반응이 없다.

“서버가 느려서 그런걸까?”



git push

아니다. 원격저장소에 저장하기 위해서는 한가지 명령어를 추가해야한다.

```
git push
```

git push는 원격저장소로 해당 커밋을 밀어넣는다는 의미이다. 막상 해보면 문제가 발생한다.

```
k@kimkimb abcd_test % git push  
remote: Permission to johnkim-codewatt/abcd_test.git denied to kimback.  
fatal: unable to access 'https://github.com/johnkim-codewatt/abcd_test.git': The requested URL returned error: 403
```

엑세스 권한이 없다고 한다. 😅 (여러분들은 될수도 있다.)

만약 해당 메세지가 나온다면, Collaborator 등록을 참고 하자.

Collaborator 등록

이부분은 나중에 팀원들을 메인 저장소에 협력자로 등록할때 참고 하면된다.

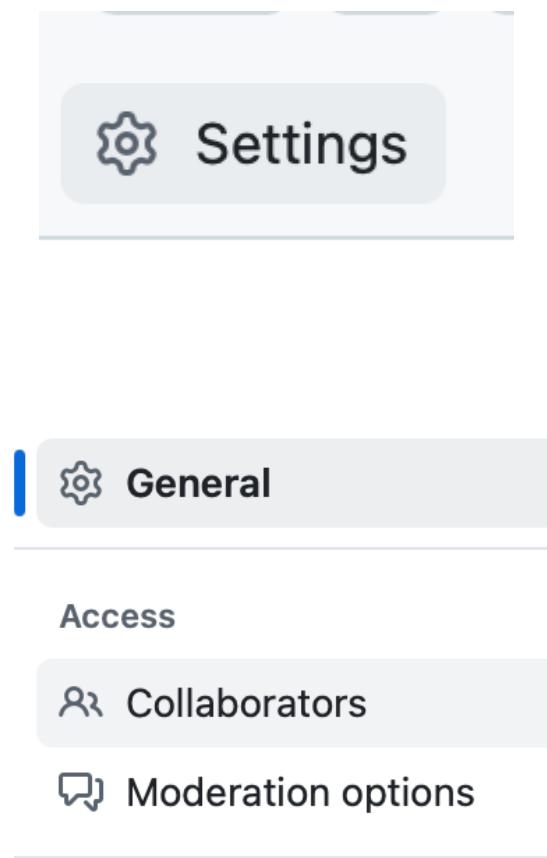
```
git config --list
```

우선 내가 누구인지 부터 확인한다.

```
user.name=kimback  
user.email=kimback03@naver.com
```

해당 이메일을 Collaborator로 등록하면 된다.

다시 웹으로 이동해서 원격저장소의 Settings로 진입한다.

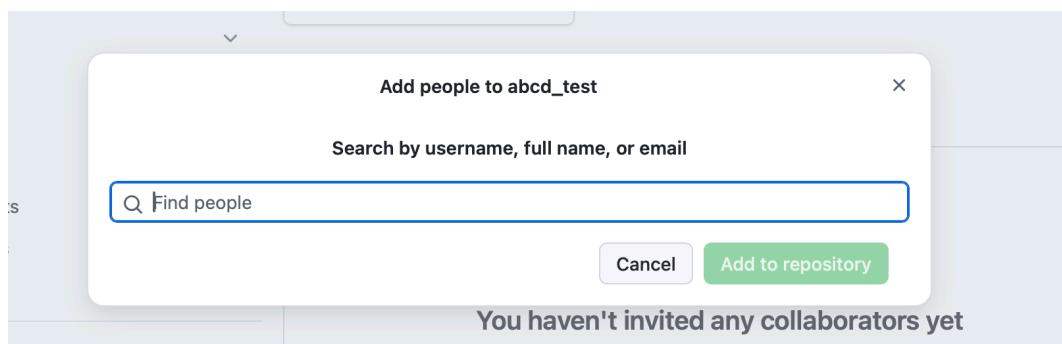


콜라보레이터를 등록하자.

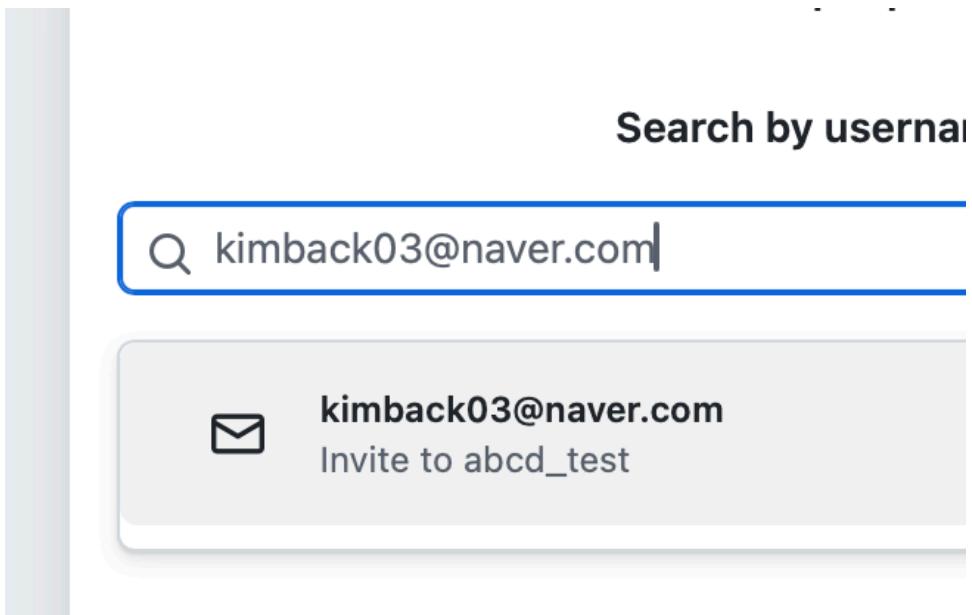


You haven't invited any collaborators yet

Add people



당연히 ...등록하려고 하는 이메일도 github에 가입되어있어야 한다.



The screenshot shows a GitHub invitation email. It starts with a GitHub logo and a plus sign, followed by two user icons. The text reads: '@johnkim-codewatt has invited you to collaborate on the [johnkim-codewatt/abcd_test](#) repository'. Below this, it says: 'You can [accept](#) or [decline](#) this invitation. You can also head over to https://github.com/johnkim-codewatt/abcd_test to check out the repository or visit [@johnkim-codewatt](#) to learn a bit more about them.' It also states: 'This invitation will expire in 7 days.' A blue 'View invitation' button is visible. At the bottom, there is a note: 'Note: This invitation was intended for [kimback03@naver.com](#). If you were not expecting this invitation, you can ignore this email. If @johnkim-codewatt is sending you too many emails, you can [block them](#) or [report abuse](#).' There is also a note about getting a 404 error and a link to manage GitHub email preferences. On the right side, there is a profile picture of 'johnkim-codewatt' and a '+' sign, with the text 'johnkim-codewatt invited you to collaborate on johnkim-codewatt/abcd_test'. Below this, there are 'Accept invitation' and 'Decline invitation' buttons. A note explains what owners of the repository can see: 'Owners of abcd_test will be able to see: Your public profile information, Certain activity within this repository, Country of request origin, Your access level for this repository, Your IP address.' Finally, there is a link to 'Block johnkim-codewatt' if the user is sending spam or malicious content.

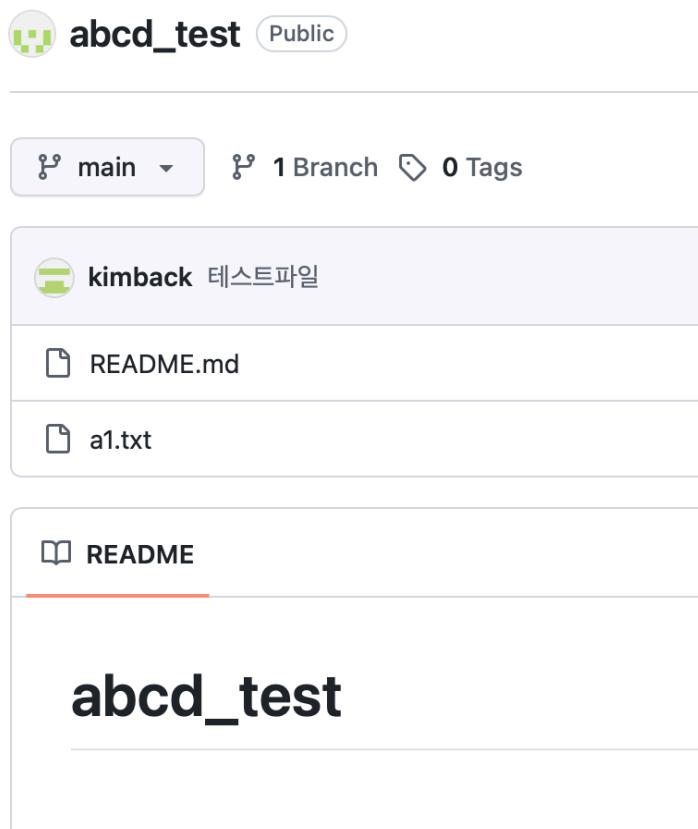
이런식으로 초대 메일이 온다.

초대메일을 통해 Accept 하면 Ok이다.

이제 다시 터미널로 돌아가서 push 해보자.

```
kimb@Kimb-Macmini abcd_test %
kimb@Kimb-Macmini abcd_test %
kimb@Kimb-Macmini abcd_test % git push
  Enumerating objects: 4, done.
  Counting objects: 100% (4/4), done.
  Delta compression using up to 10 threads
  Compressing objects: 100% (2/2), done.
  Writing objects: 100% (3/3), 331 bytes | 331.00 KiB/s, done.
    Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
  To https://github.com/johnkim-codewatt/abcd_test.git
    0b7661a..6e33f89  main -> main
kimb@Kimb-Macmini abcd_test %
```

성공! 완료 되었다! 원격저장소에도 잘 올라간걸 볼수있다.



협업실전

“정답은 없다, 다만 자주 사용하는 패턴은 있다”

첫번째

1. 작업자1(kimback03), 작업자2(codewatt)
2. github를 사용
3. branch를 dev, main, test 등 특정 목적, 기능 단위로 나누어서 작업
4. 최종적으로 소스를 main에 합친다.

두번째

1. 작업자1(kimback03), 작업자2(codewatt)
2. github를 사용
3. 서로의 각자의 branch를 나눠서 작업하고
4. 최종적으로 소스를 main에 합친다.

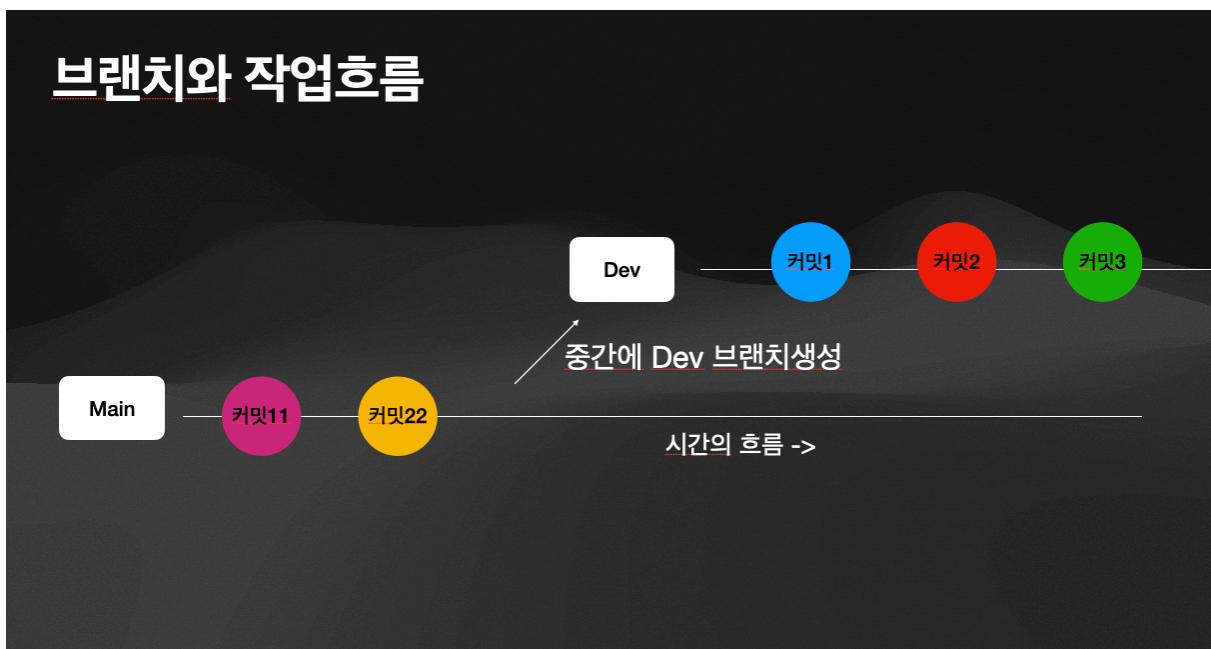
→ 결국 맥락은 동일하고, branch를 어떻게 나누고 관리할것인지는 핵심 포인트이다.

브랜치



“네. 그... 브런치 아닙니다...”

“브랜치는 소스관리를 위한 일종의 가상의 작업그룹을 의미한다.”



브랜치와 작업흐름



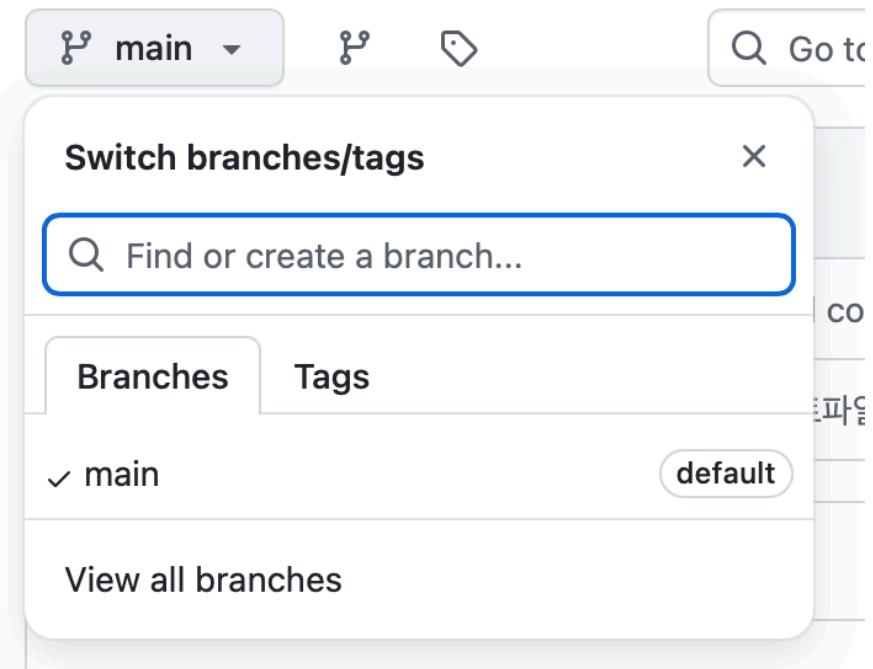
- Main 브랜치가 있고 일상적인 작업, 커밋을 하다가, 기능구현을 위해 Dev 브랜치를 만들었다.
- Main에서 어떤일이 벌어지든 상관없이, Dev 브랜치에서 자기 작업을 진행한다.
- 개발을 완료하고 최종적으로 합친다. (Merge)

실습

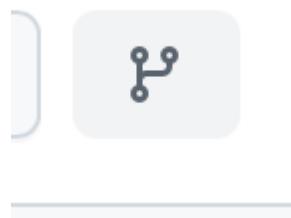
저장소의 master인 codewatt와 작업자인 kimback03은 각자의 브랜치에서 작업하기로 하고

최종적으로 소스를 main에서 머지해서 배포하기로 합니다.

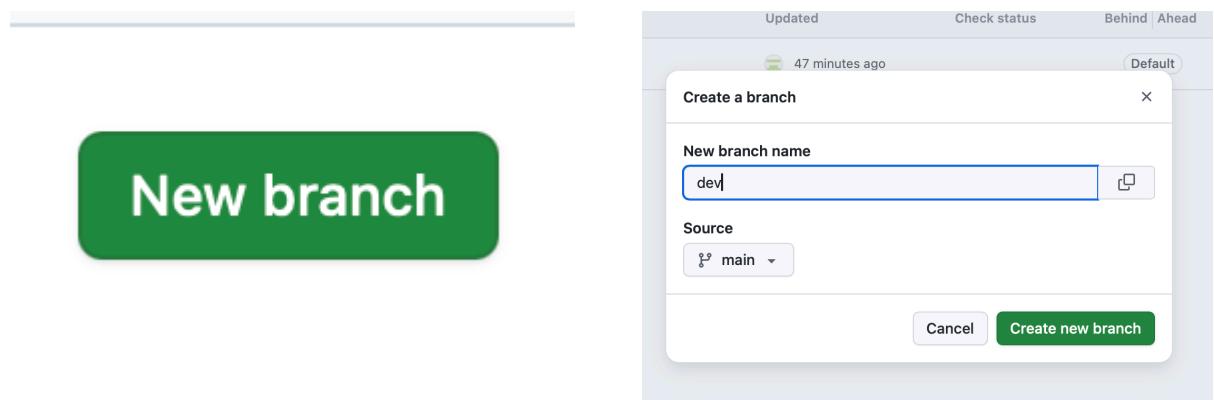
일단 브랜치를 생성하자. 웹 저장소로 이동해서 브랜치를 확인해보자.



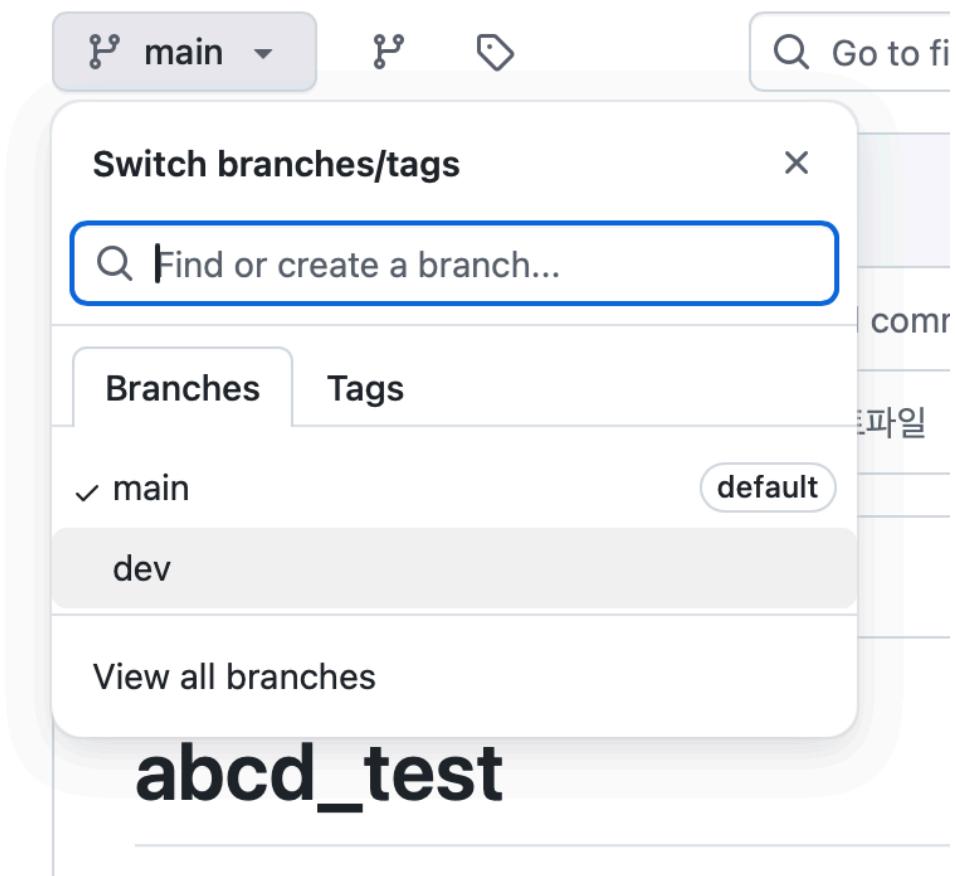
이게 브랜치 이다. 지금 브랜치가 main 하나밖에 없다.



이 버튼을 누르고 들어간다.



dev 브랜치를 생성한다. 잘 만들어진것을 확인할수 있다.



이제 dev 브랜치에서 kimback03이 작업을 시작한다. 터미널로 가자.

git fetch/pull 로 브랜치 목록을 갱신한다.

```
git fetch
```

```
git branch -a
```

현재 브랜치 전체목록을 확인해보자.

```
* main
```

```
remotes/origin/HEAD -> origin/main  
remotes/origin/dev  
remotes/origin/main
```

```
git branch --track
```

*이 붙어있는 브랜치이름이 현재 내 브랜치라고 보면된다.

remotes ... 는 원격저장소의 브랜치목록이다.

dev브랜치를 만들고, *을 이동시켜보자.

checkout 키워드를 활용한다.

checkout은 해당시점 혹은 해당 브랜치의 작업내용을 내 작업공간에 맞추겠다는 뜻이다.

(실제로 체크아웃 하고 나서 탐색기를 통해 확인해보면된다.)

```
git checkout -b [로컬브랜치명] remotes/origin/[원격브랜치명]
```

이렇게 하면 3가지 동작을 한번에 처리하게 된다.

1. 해당브랜치를 기준으로 소스를 체크아웃하면서,
2. 로컬 브랜치를 생성하고
3. 만들어진 로컬 브랜치에 원격 브랜치를 맞추겠다는 뜻이된다.

확인해보자. 아래의 명령어를 통해 해당 브랜치가 원격지에 연결되어 있는지 확인 가능하다.

```
git branch -vv
```

```
[kimb@kimb-Macmini abcd_test % git branch -vv
* dev 6e33f89 [origin/dev] 테 스 트 파 일
  main 6e33f89 [origin/main] 테 스 트 파 일
```

→ 항상 내가 지금 어디 브랜치에서 작업하고 있는지 알아둬야 한다. 엉뚱한 다른 브랜치에서 작업하면...

이제 여기서 작업을 진행하고, 위에서 배운대로 status, add, commit, push 해보자.
제대로 완료되었다면, 웹 저장소에 새로운 브랜치에 반영된 파일을 볼수 있다.

abcd_test Public

Pin

dev ▾

Go to file t +

This branch is 1 commit ahead of main.

kimback 개발브랜치 9748c63 · now

README.md Initial commit

a1.txt 테스트파일

a2.txt 개발브랜치

README

abcd_test

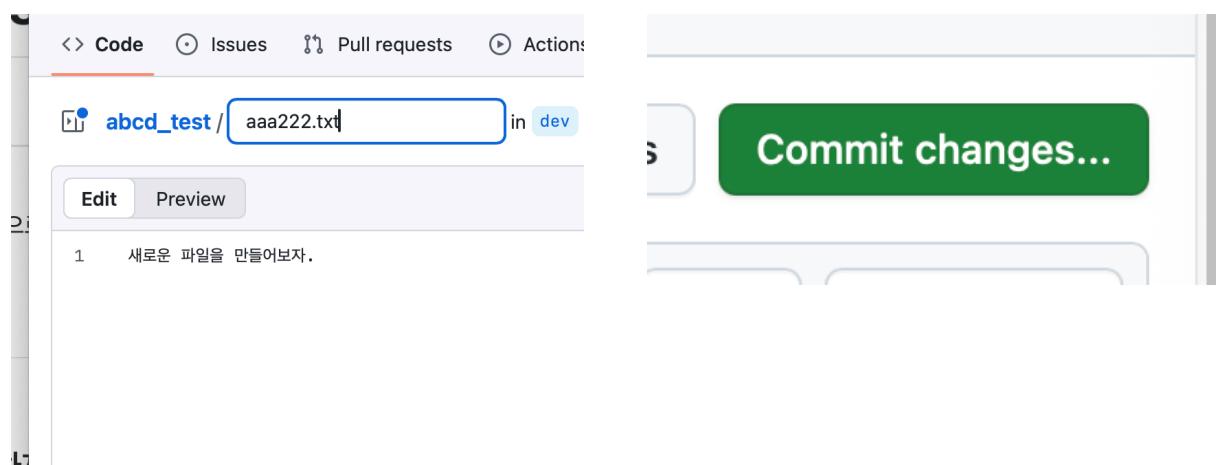
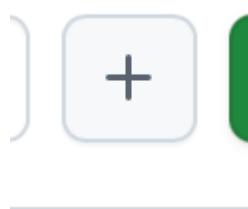
변경된 소스를 가져와서 작업하자 (Pull)

- Push : remote repository(원격 저장소)에 내 소스를 밀어넣는 명령어
- Pull : remote repository에 있는 최신의 소스를 내 PC의 작업 공간으로 가져오는 작업.

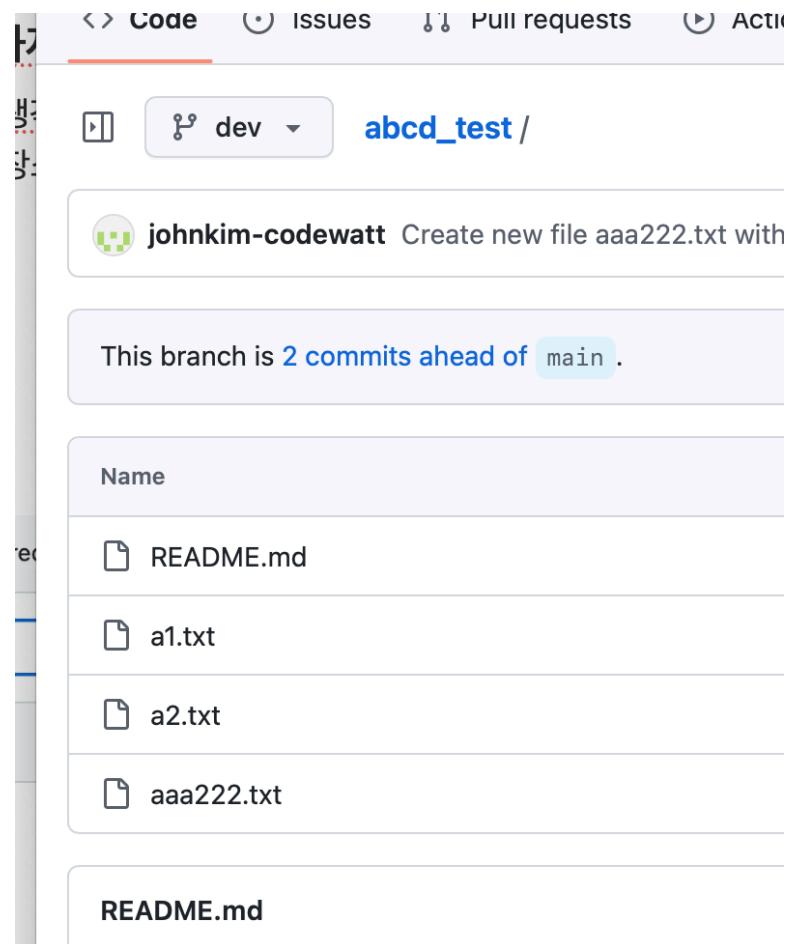
위에서 배운 push는 직관적으로 생각하면, 내 작업물을 저장소로 밀어넣는다는 의미이다.
“그렇다면 반대로 저장소에 있는 변경된 사항을 내 로컬 저장소로 가져오려면 어떻게 할까?”

웹상으로 이동해서 + 버튼을 누른다.

누르고 신규 파일을 생성해보자.



파일을 작성하고 커밋하자.



dev 브랜치를 기준으로 새로운 파일이 생성되었다. (웹에서 생성)

다시 터미널로 가보자.

원격 저장소에 새롭게 업데이트된 aaa222.txt 파일을 로컬저장소와 동기화 시키기 위해 아래의 명령어로...

“당겨서 가져온다”

```
git pull
```

```
[kimb@kimb-Macmini abcd_test %  
[kimb@kimb-Macmini abcd_test % git pull  
remote: Enumerating objects: 4, done.  
remote: Counting objects: 100% (4/4), done.  
remote: Compressing objects: 100% (2/2), done.  
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0  
Unpacking objects: 100% (3/3), 980 bytes | 490.00 KiB/s,  
From https://github.com/johnkim-codewatt/abcd_test  
  9748c63..8ccd92a dev      -> origin/dev  
Updating 9748c63..8ccd92a  
Fast-forward  
  aaa222.txt | 1 +  
  1 file changed, 1 insertion(+)  
  create mode 100644 aaa222.txt
```

해당 명령어를 통해 해당 원격 저장소와 내 로컬 저장소를 동기화 시켰다.

즉 pull의 의미를 생각해보면 된다.

이렇게 push&pull을 통해서 원격저장소와 싱크를 맞출수 있다.

```
[kimb@kimb-Macmini abcd_test % ls  
a1.txt          a2.txt          aaa222.txt          README.md  
[kimb@kimb-Macmini abcd test %
```

생성된 파일이 내 로컬에도 생겼다.

협업간의 꿀팁 (충돌예방)

1. 내 작업을 시작하기 전에 원격저장소의 소스를 먼저 pull 해서 최신화 하고 시작한다.
2. 내 로컬 저장소가 add commit 등이 완료된 clean 상태에서 시작하는것이 좋다.
3. 항상 내가 누구이고, 내가 어디브랜치에 있는지 확인하고 작업을 시작하자.

Pull Request (DEV → MAIN)

각자의 브랜치에서 작업을 마치고, 마지막 main 브랜치에 합쳐보자.

어떻게?

“github의 pull request 기능을 통해서”

풀리퀘스트는 쉽게 말해서

“내가 이렇게 작업을 했으니, 마스터님 검토하고 main 브랜치에 합쳐주세요~”

이렇게 요청하는 것과 같다.

웹 저장소로 이동해보자. 아래의 버튼을 누르면 풀리퀘스트를 진행 할수있다.

Compare & pull request

git dev2 had recent pushes 2 minutes ago

Compare & pull request

상단에 중요한 정보가 나온다.

Dev #1

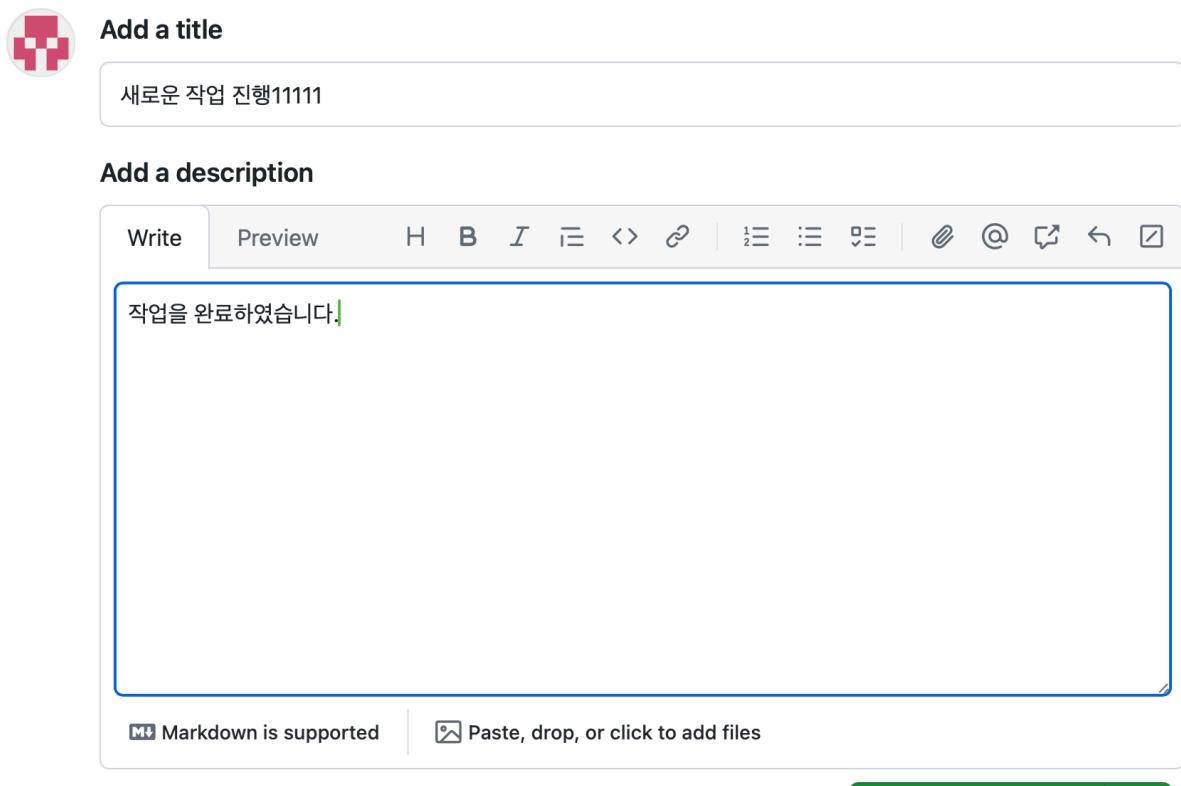
 Open

johnkim-codewatt wants to merge 2 commits into `main` from `dev` 

→ 작업자가 dev에서 main으로 합치길 원한다!

요청을 작성해보자.

“마스터님, 제가 이렇게 저렇게 작업했으니 검토해보시고 main에 합쳐주시면 감사하겠습니다.” 😊



The screenshot shows the GitHub interface for creating a pull request. At the top, there's a green button labeled "Open". Below it, a message from "johnkim-codewatt" says they want to merge two commits from the "dev" branch into the "main" branch. The main area is a form for creating a new issue or pull request. It has fields for "Add a title" (containing "새로운 작업 진행11111") and "Add a description" (containing "작업을 완료하였습니다."). There are buttons for "Write" and "Preview", and a rich text editor toolbar with various icons. At the bottom, there are buttons for "Create pull request" and "Markdown is supported". A note at the bottom says "Remember contributions to this repository should follow our [Code of Conduct](#)".

하단에 댓글을 달수 있다.



Add a comment

Write Preview **H** **B** *I* U <>  |    |    

> 작업을 완료하였습니다.

-> 확인하였습니다.

 Markdown is supported  Paste, drop, or click to add files

  Close with comment  Comment

소스를 합칠 타당성이 없으면 거절하기도 한다...



Add a comment

Write Preview 

다시수정하세요.

 Markdown is supported  Paste, drop, or c

The screenshot shows a GitHub pull request interface. At the top, there are tabs for Conversation (0), Commits (2), Checks (0), and Files changed (2). The main area displays a conversation between two users:

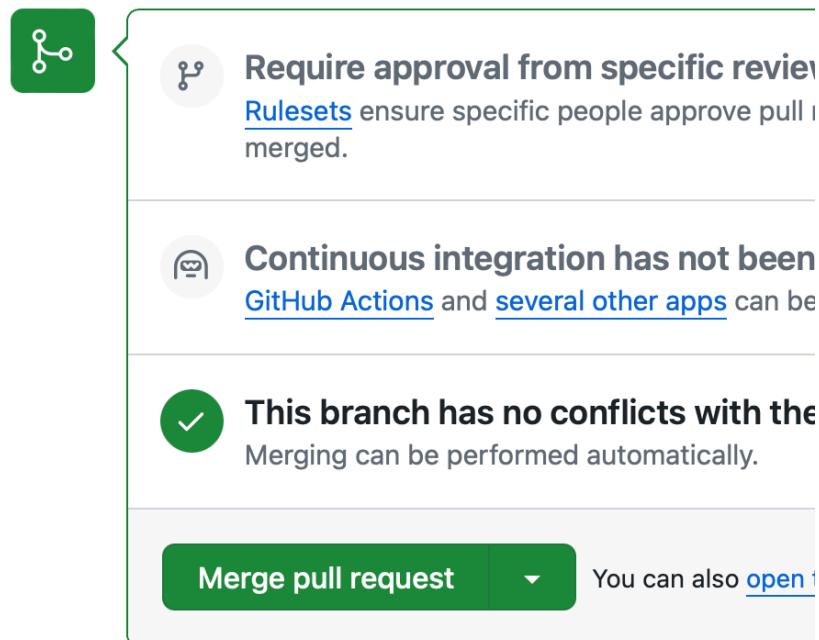
- johnkim-codewatt commented 3 minutes ago**: 마스터님 합쳐주세요.
- kimback and others added 2 commits 3 hours ago**:
 - o 개발브랜치 (Commit hash: 9748c63)
 - o Create new file aaa222.txt with initial content (Verified, Commit hash: 8ccd92a)
- johnkim-codewatt commented now**: 거절합니다.
- johnkim-codewatt commented now**: 다시수정하세요.

On the right side of the interface, there are several status indicators and buttons:

- Owner, Author, ...
- Review, Sugg, C, Still in
- Assign, No or
- Label, None
- Project, None
- Miles, No m
- Deve, Succ, these, None
- Notif

원활한 작업을 위해 팀원들간에 협의가 필요하다.

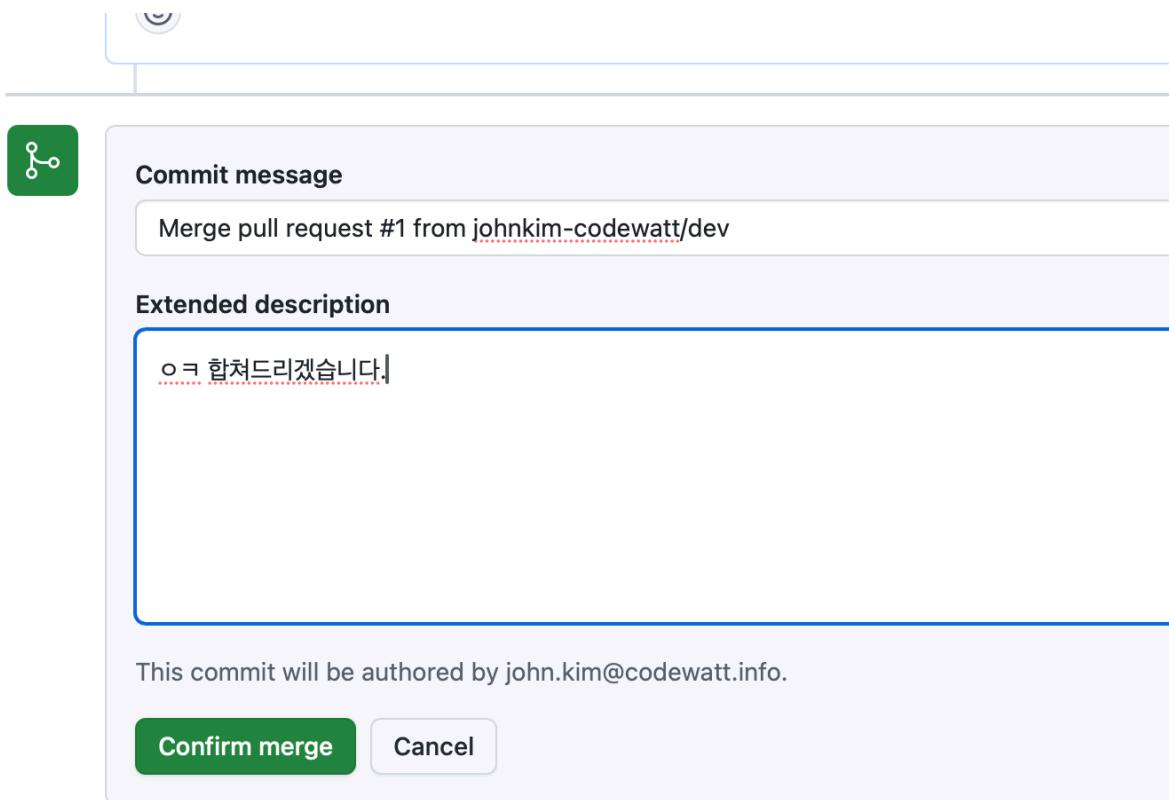
Add more commits by pushing to the [dev2](#) branch or



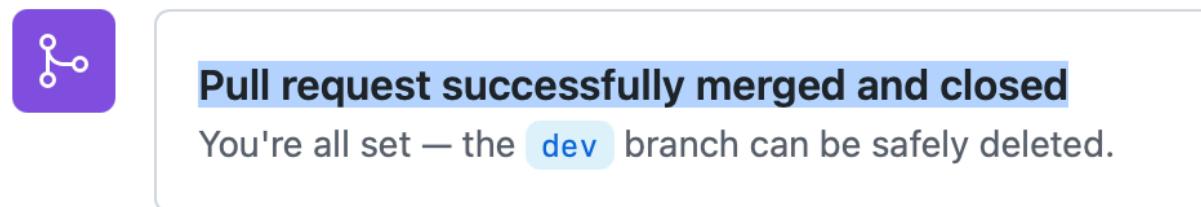
The image shows a screenshot of a GitHub pull request merge interface. At the top, there's a green button labeled "Merge pull request". To its right is a dropdown arrow. Below the button, the text "You can also [open](#) this pull request" is visible. The main area contains three items in a list:

- Require approval from specific reviewers**: This item includes a link to "Rulesets" and a note that specific people will approve the pull request before it's merged.
- Continuous integration has not been triggered**: This item links to "GitHub Actions" and mentions "several other apps" that can trigger CI.
- This branch has no conflicts with the base branch**: This item indicates that merging can be performed automatically.

별 문제가 없다면 머지 풀리퀘스트로 소스를 합칠수 있다.



문제가 없다면 최종적으로 머지가 완료된다.



A screenshot of a GitHub repository interface. At the top, there are buttons for 'main' (with a dropdown arrow), '3 Branches' (with a dropdown arrow), '0 Tags', a search bar with 'Go to file' and a 't' icon, a '+' icon, and a 'Code' button with a dropdown arrow. Below this, a card displays a merge commit from user 'kimback05'. The commit message is 'Merge pull request #3 from kimback05/dev2'. It shows three files: 'README.md' (Initial commit, 3 weeks ago), 'test_abc.txt' (23 minutes ago), and 'test_abcdefg.txt' (10 minutes ago). The commit hash is d82e782 and it was made 'now'.

저장소에 쓰기 권한이 있는 사람만 합칠수 있기 때문에,
팀원들이 작업하고, 마스터(리더)가 최종 합치는것이 좋다.

이외에 참고할것 (툴사용)

"특별한 이유가 없다면, 그냥 툴을 쓰도록 하자" 😊

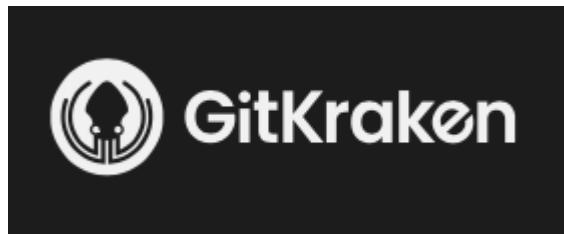
sourcetree

<https://www.sourcetreeapp.com>



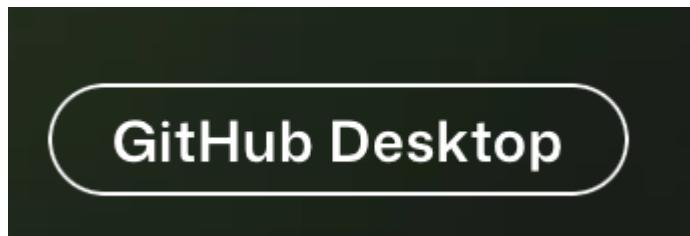
gitkraken

<https://www.gitkraken.com>



github desktop

<https://github.com/apps/desktop?locale=ko-KR>



끝.