

코드리뷰어를 위한 Git

CONTENTS

목차

Chapter1

git / github 소개

Chapter2

git 1단계 : Setting 부터 git commit 까지

Chapter3

git 2단계 : checkout 과 log view

Chapter4

git 3단계 : branch & merge

Chapter5

git 4단계 : github과 push, pull, clone

Chapter6

Markdown

Chapter7

git 5단계 : pull request

Chapter8

PR Template

Chapter1

git / github 소개

git (프로그램) + hub (모이는 중심지)

수 많은 오픈 소스코드 및
비공개 소스코드들이 저장 되어있는
Remote 소스코드 저장소

github을 사용하기 위해서는

git을 이용하여 소스코드를 Local에서 관리

git과 github을 연동해서,
git에 저장된 소스코드를 전부를
github에 업로드(push)를 하면 된다.

git 목적 - 버전관리

1. 버전관리

공유폴더 사용 대신, 소스코드들을 **버전별로 관리**

2. 동료와 함께 **하나의 프로젝트를 개발**

git의 Branch ~ Merge 기능 사용

3. 자동 백업된 **코드 복원**

과거 소스코드로 되돌아가기

1. git처럼 버전관리 가능

2. 서버에 소스코드 저장

로컬 컴퓨터가 아닌, 원격 서버에 소스코드를 안전하게 저장

3. 원격 협업

Issue(ticket) Open : 여러 사람에게 작업요청 / 질문 / 버그리포트 / 기능 추가 가능

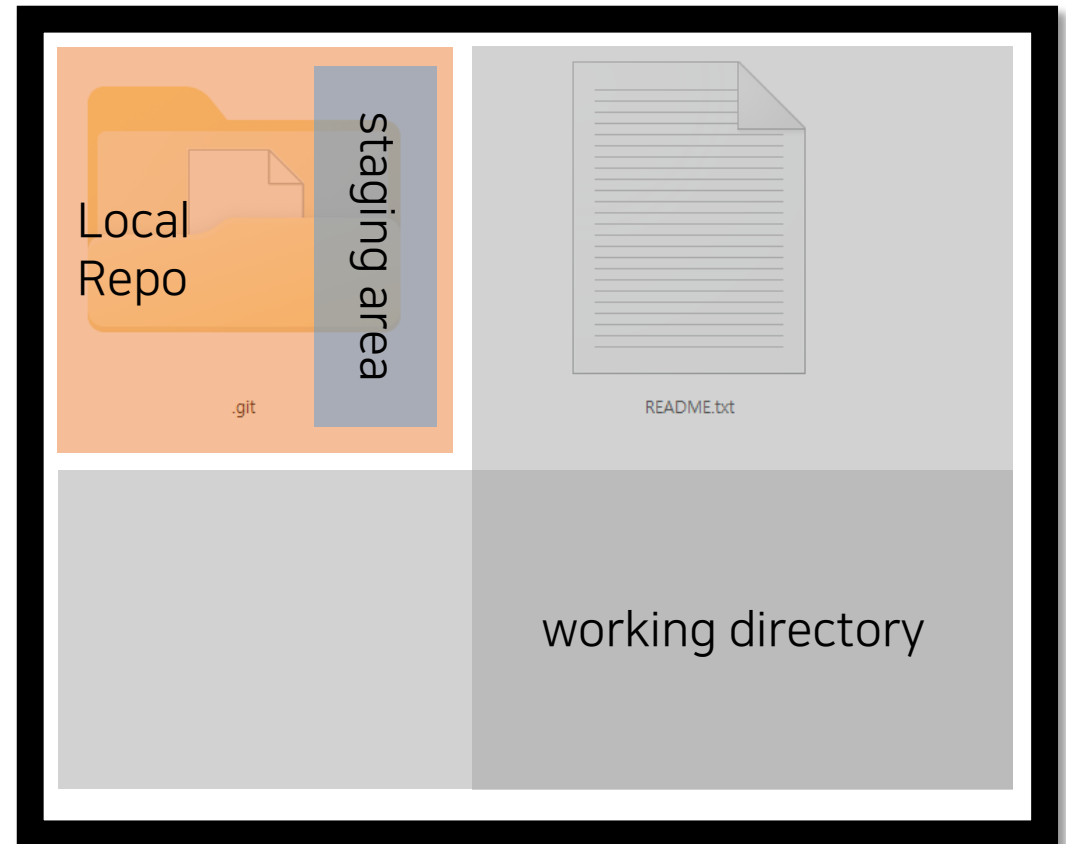
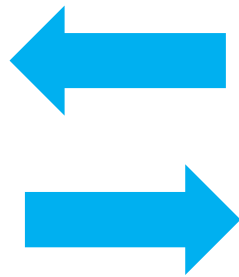
코드리뷰 : 업로드한 소스코드에 대해 토론

주요 내용 overview

1. Local Repo 에서 작업
2. Remote Repo 와의 연동



Remote Repo



소스코드 작성

txt문서도 상관없음

git에 버전관리

파일 복사 붙여넣기가 아닌,
파일들을 버전으로 관리해주는 git 사용

CLI 사용

github

git에 저장된 파일들을 github에 업로드(push)

git은 **Local** 저장소에 버전별 문서를 관리하는 프로그램이다.

문서 버전관리

컴퓨터 포맷되면 소스코드 사라짐

github은 **Remote** 저장소를 제공하는 Web Application이며,

협업하는 용도로 사용됨

git에서 저장된 내용 모두 github에 업로드(**push**)하여 안전하게 저장

remote 저장소 제공

Issue 관리 / **코드리뷰**

용어 암기

Local Repository : 내 작업 폴더 내부의 .git 폴더

Remote Repository

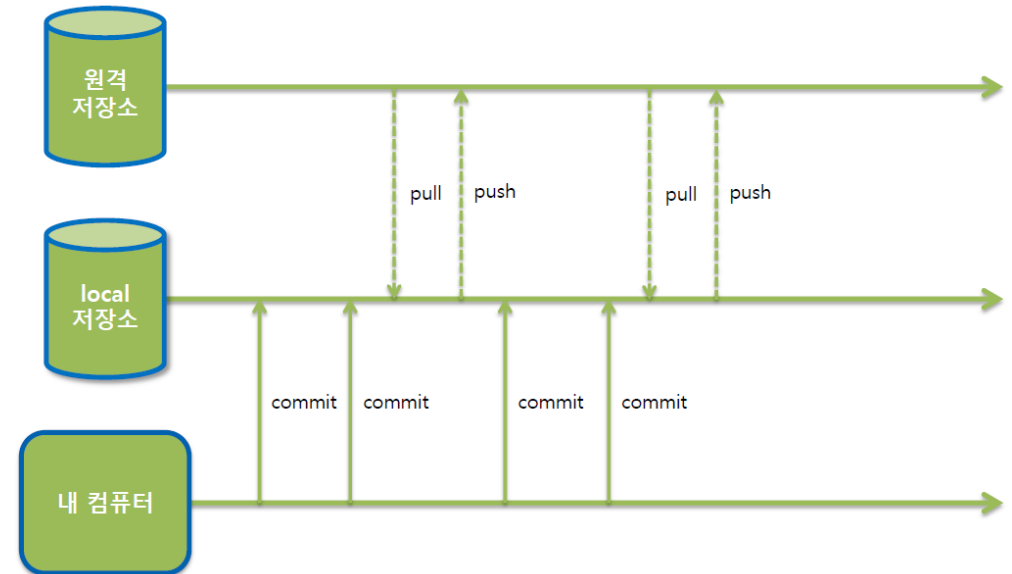
Commit : Local 저장

Push : 안전한 저장

Pull : 최신 결과 가져오기

분산환경

원격 저장소와 연결이 끊어져도 계속 버전관리가 가능합니다.



Git 1 ~ 4단계까지 진행합니다.

- 1단계 : Git Init부터 Commit까지
- 2단계 : checkout과 git log view
- 3단계 : git branch와 merge
- 4단계 : github과 push, pull, clone

매 단계별

단계별 Master를 위한 도전적인 미션이 존재합니다.

Chapter2

git 1 단계

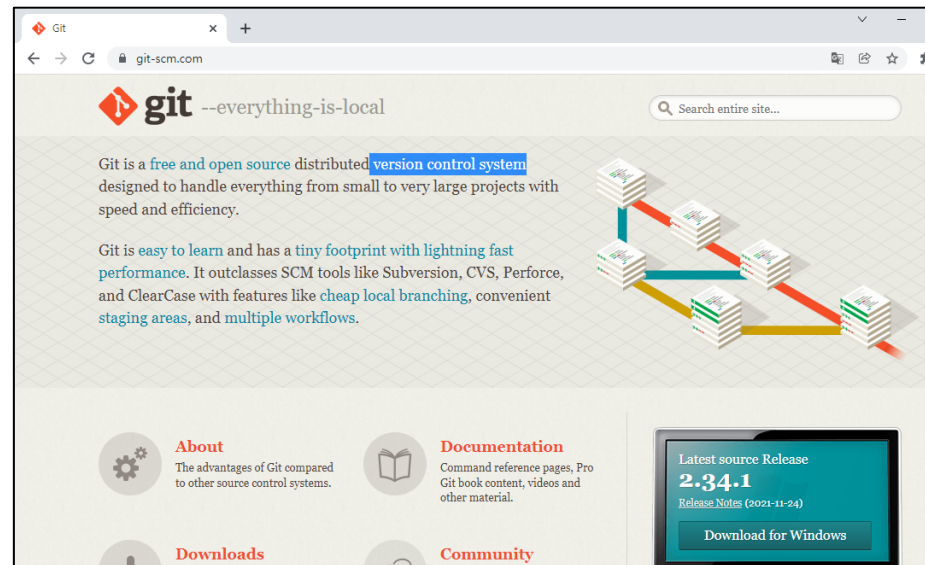
Setting 부터 git commit 까지

Git 설치하기

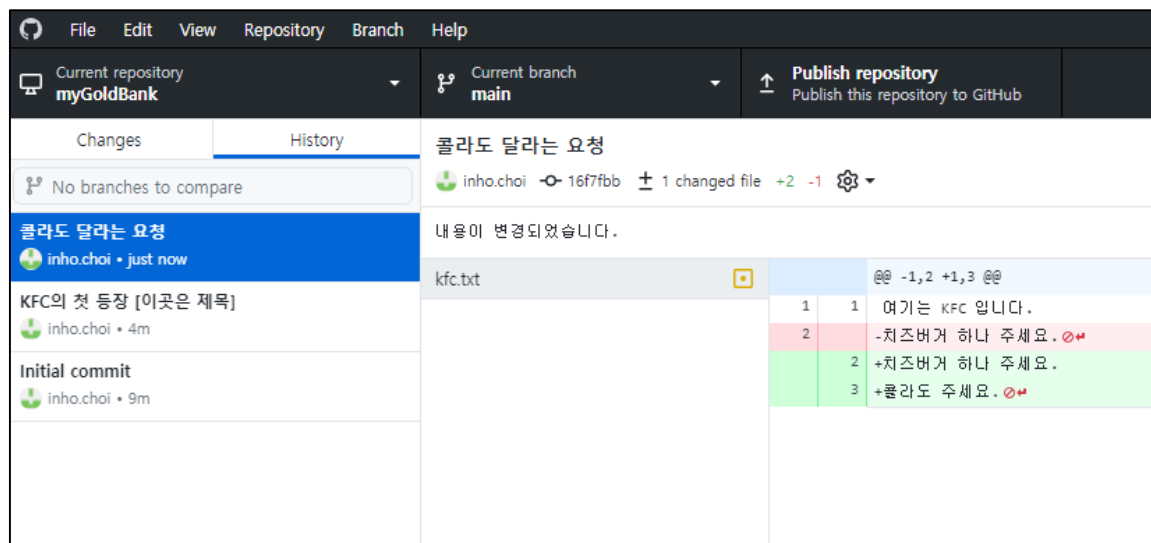
Version Control System

문서들을 게시판처럼 저장하는 곳

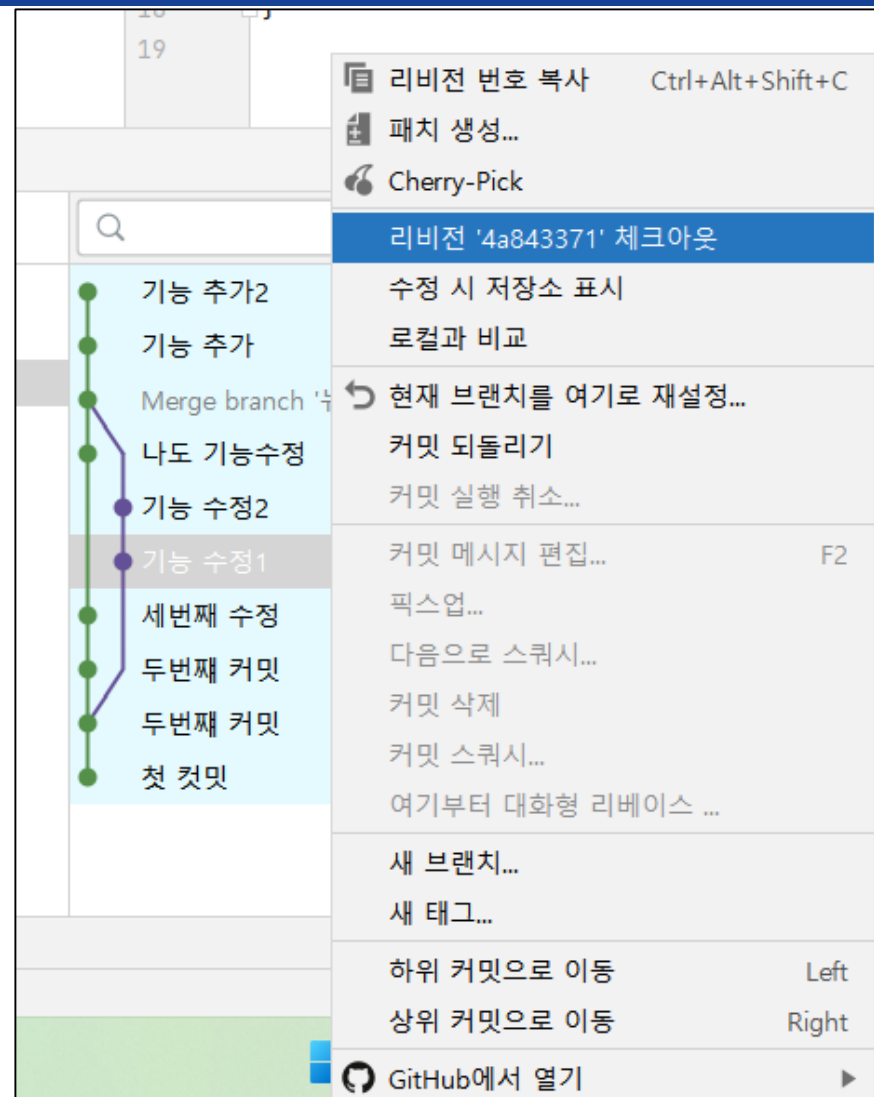
여기에 저장(업로드)하면 덮어쓰기로 파일을 날리는 경우 **없음!!!**



그래픽 기반으로 Git 사용



Github Desktop

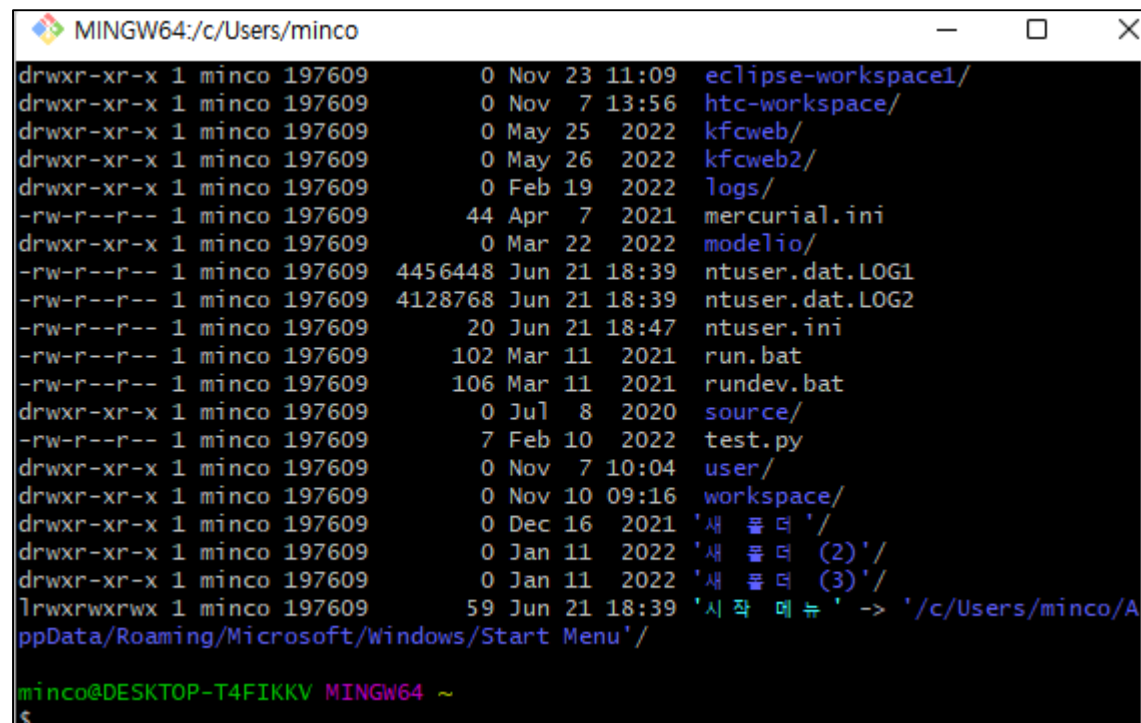


IntelliJ 에 포함된 Git GUI

Command Line Interface

글자기반으로

Git 명령어를 직접 입력



```
MINGW64:/c/Users/minco
drwxr-xr-x 1 minco 197609      0 Nov 23 11:09 eclipse-workspace1/
drwxr-xr-x 1 minco 197609      0 Nov  7 13:56 htc-workspace/
drwxr-xr-x 1 minco 197609      0 May 25 2022 kfcweb/
drwxr-xr-x 1 minco 197609      0 May 26 2022 kfcweb2/
drwxr-xr-x 1 minco 197609      0 Feb 19 2022 logs/
-rw-r--r-- 1 minco 197609    44 Apr  7 2021 mercurial.ini
drwxr-xr-x 1 minco 197609      0 Mar 22 2022 modelio/
-rw-r--r-- 1 minco 197609 4456448 Jun 21 18:39 ntuser.dat.LOG1
-rw-r--r-- 1 minco 197609 4128768 Jun 21 18:39 ntuser.dat.LOG2
-rw-r--r-- 1 minco 197609    20 Jun 21 18:47 ntuser.ini
-rw-r--r-- 1 minco 197609   102 Mar 11 2021 run.bat
-rw-r--r-- 1 minco 197609   106 Mar 11 2021 rundev.bat
drwxr-xr-x 1 minco 197609      0 Jul  8 2020 source/
-rw-r--r-- 1 minco 197609      7 Feb 10 2022 test.py
drwxr-xr-x 1 minco 197609      0 Nov  7 10:04 user/
drwxr-xr-x 1 minco 197609      0 Nov 10 09:16 workspace/
drwxr-xr-x 1 minco 197609      0 Dec 16 2021 '새 폴더' /
drwxr-xr-x 1 minco 197609      0 Jan 11 2022 '새 폴더 (2)' /
drwxr-xr-x 1 minco 197609      0 Jan 11 2022 '새 폴더 (3)' /
lrwxrwxrwx 1 minco 197609      59 Jun 21 18:39 '시작 메뉴' -> '/c/Users/minco/AppData/Roaming/Microsoft/Windows/Start Menu/'

minco@DESKTOP-T4FIKKV MINGW64 ~
$
```


CLI로 수업하는 이유

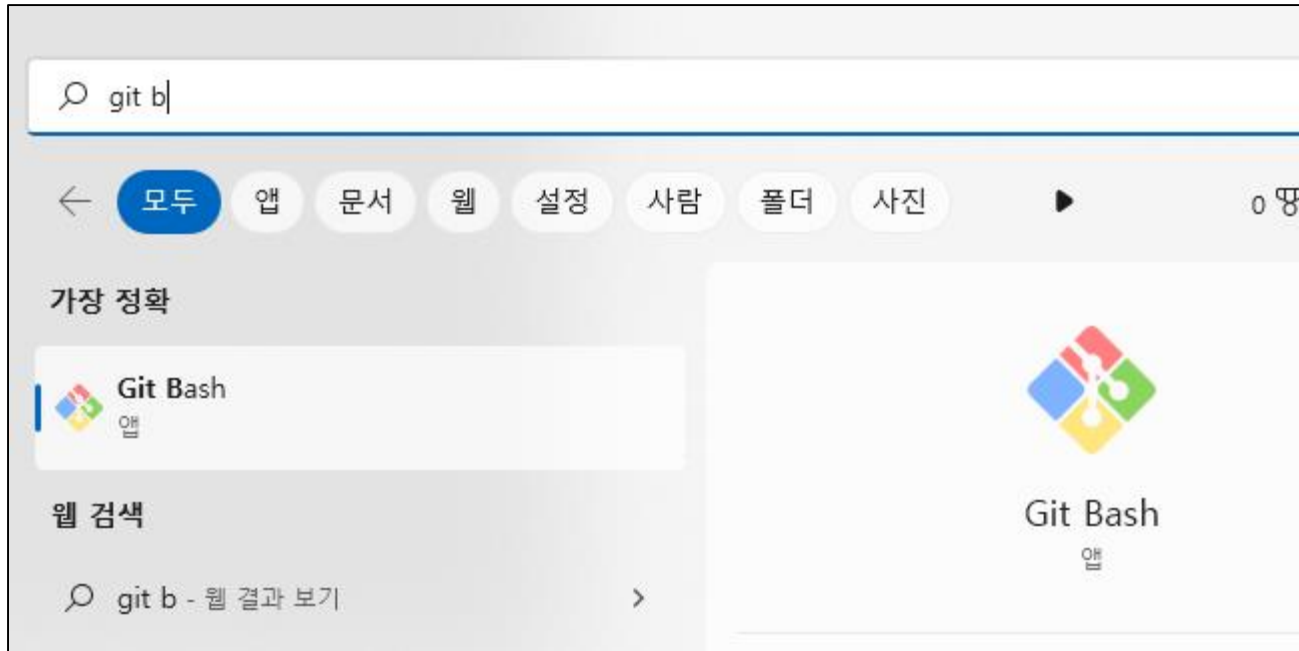
1. Git 원리를 알기 쉬움
2. GUI는 사람마다 / 팀마다 사용하는 Tool이 다름
CLI는 바뀌지 않음

수업은 처음에는 CLI로 하다가, TDD 이후에는 GUI로 수업예정

사용자 입장에서는 GUI가 편하면 GUI 쓰는 것이고,
CLI가 편하면 CLI를 쓰는 것이다.

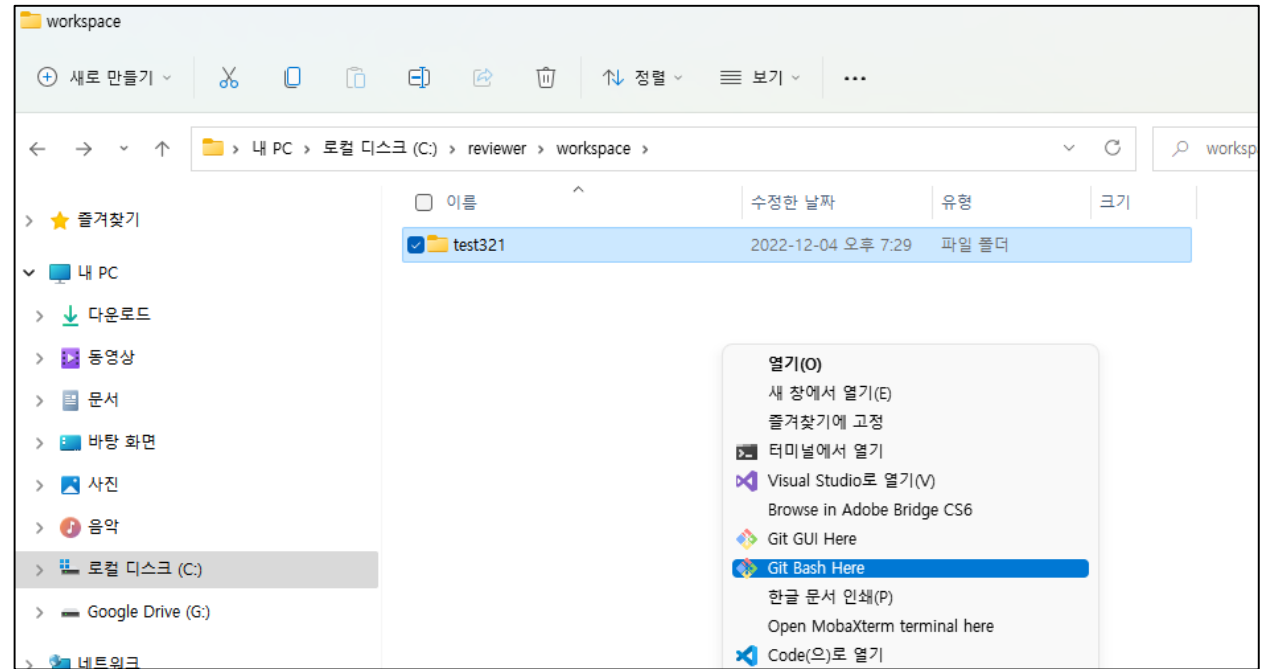
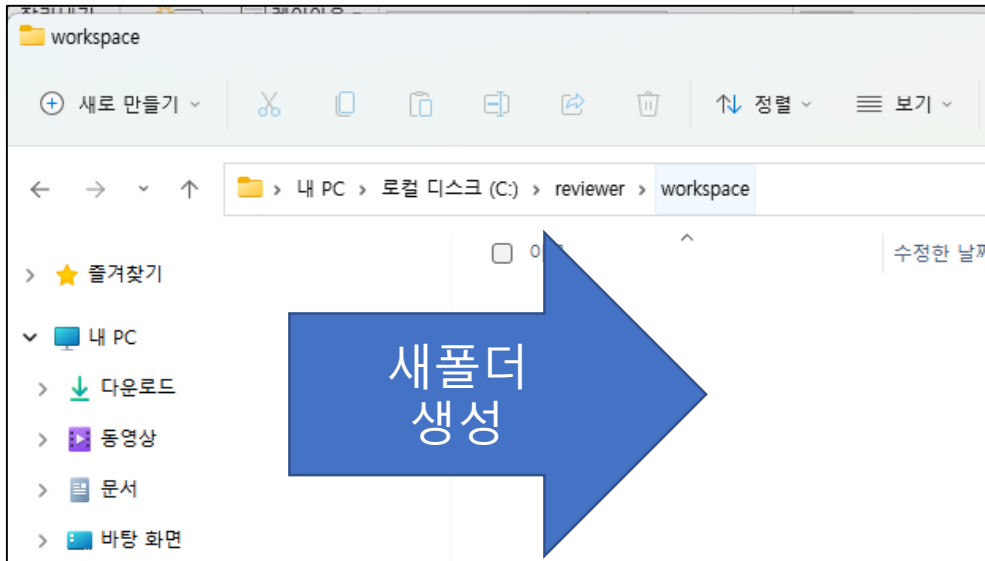
(뭐가 더 좋다 / 안 좋다는 개인차, 강요하지 말자!)

Git Bash



작업 폴더 생성 후 Git Bash 실행

c:\reviewer\workspace

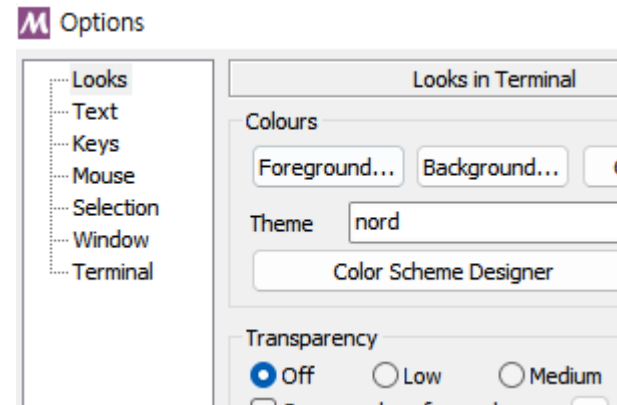
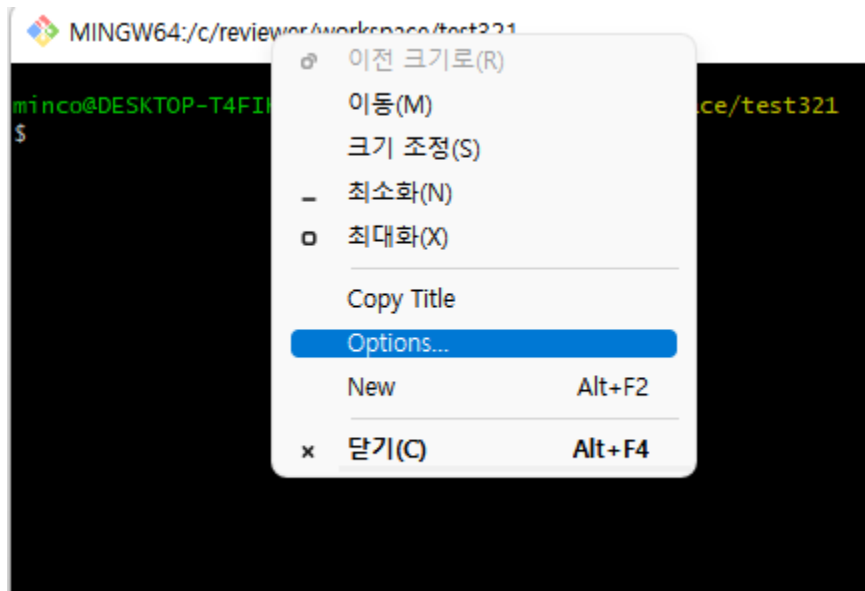


잘 보이게 폰트 및 테마 수정

학습자들도 실수를 줄이게끔 변경을 권장합니다.

Theme : nord

Font Size : 18



[복습] Git과 GitHub의 이해

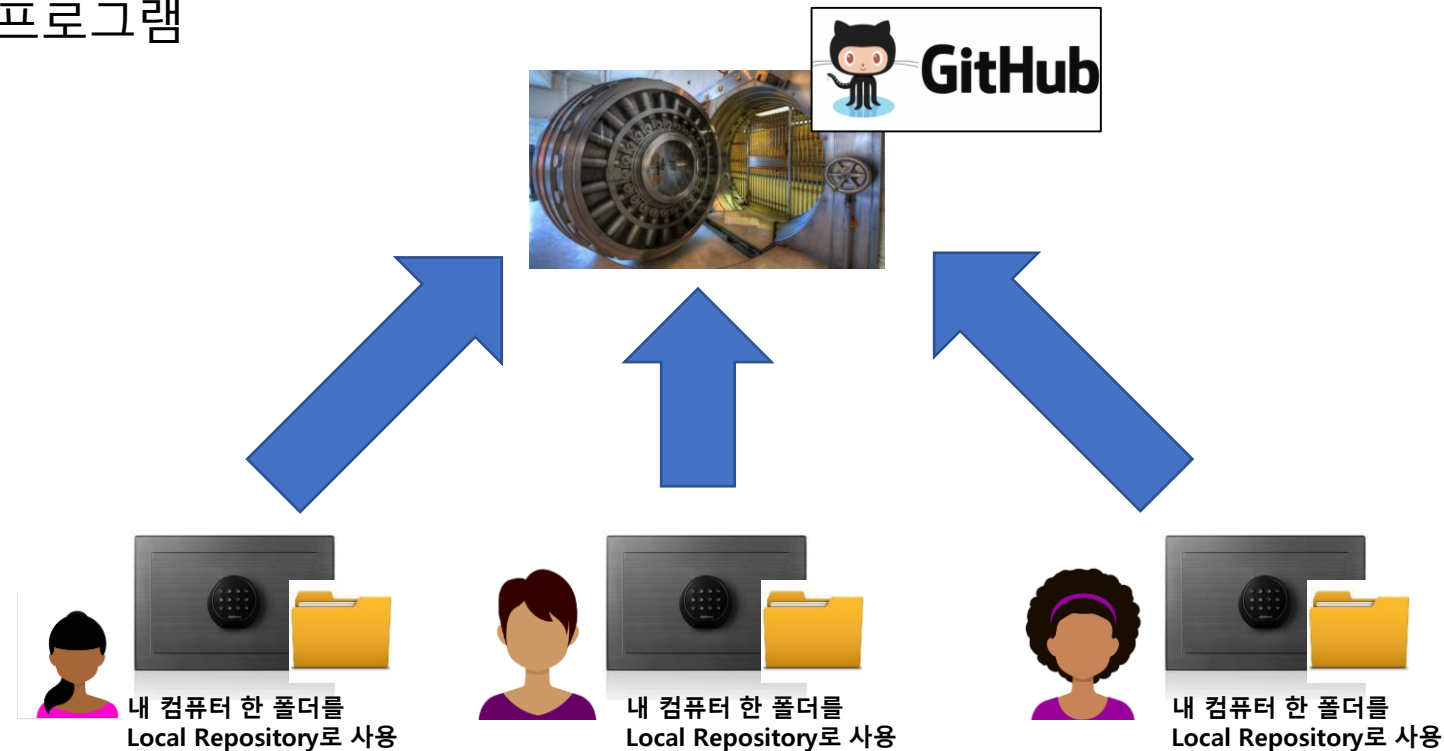
GitHub : 원격 저장소를 제공하는 Web Application

Git

내 컴퓨터 내부, 한 폴더를 **Local Repository**로 만들 수 있다.

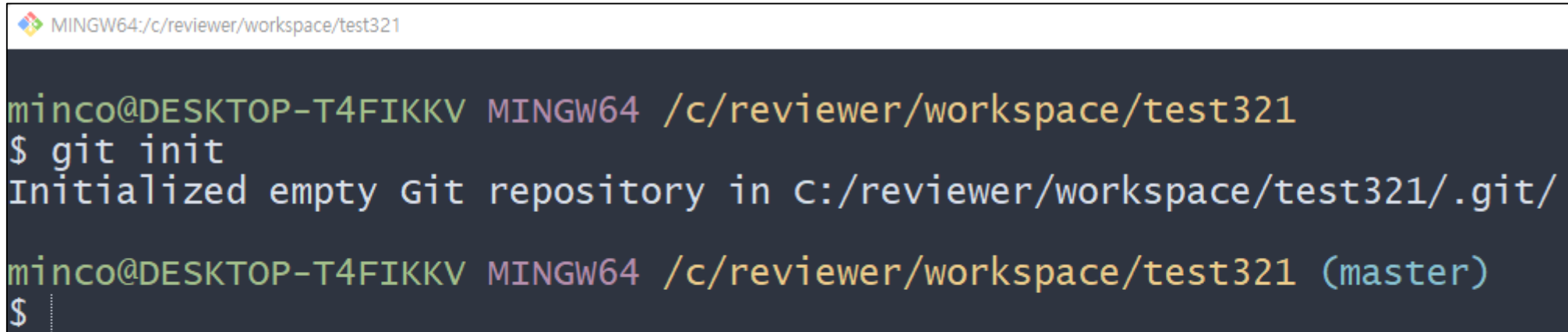
Local Repository 에 **Commit**도 할 수 있다.

GitHub에 소스코드도 **Push** 할 수 있는 프로그램



Local 저장소로 지정

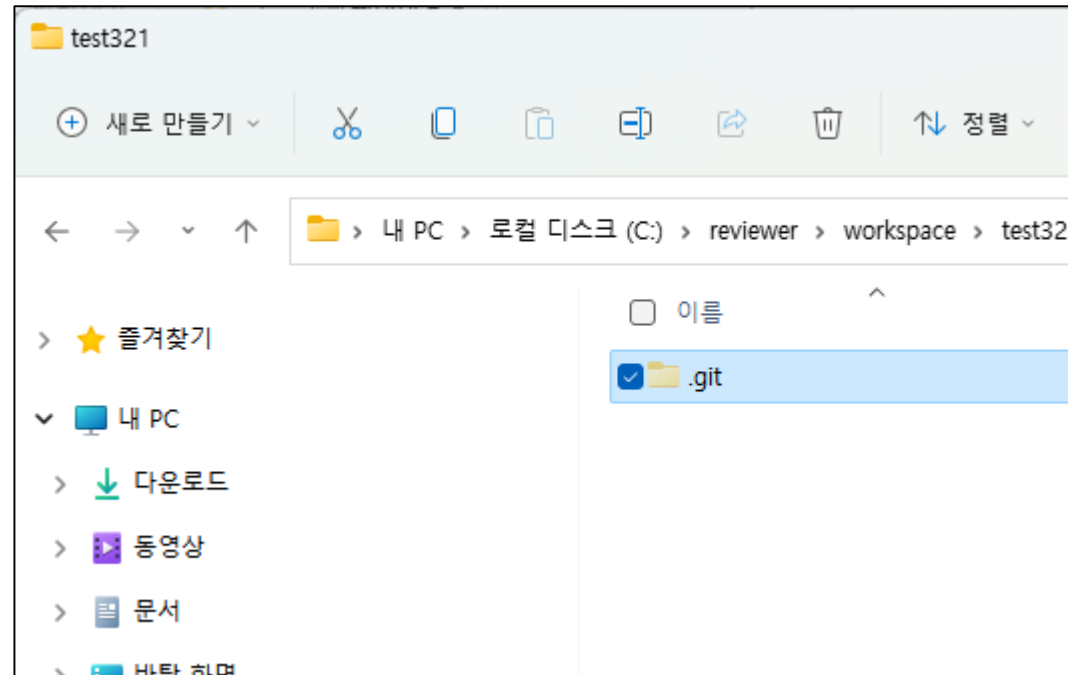
.git 폴더 생성이 됨
내 Local 저장소



```
MINGW64:/c/reviewer/workspace/test321  
  
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321  
$ git init  
Initialized empty Git repository in C:/reviewer/workspace/test321/.git/  
  
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)  
$ |
```

만약 Git Local 저장소 취소

그냥 git 폴더 삭제



Local Repo 내부 Local Repo 생성 금지

문제가 발생하는 구조의 예시)

- test321 폴더

- .git
- hello.txt 파일
- src 폴더
 - .git
 - abc.cpp



Local Repo 내부에
또 Local Repo 생성되는 문제 발생!

git 사용자 이름 / 정보 입력

git에 정보를 저장할 때 마다 기입 될,
사용자 기본 정보가 필요하다.

git config --global user.name [이름]

git config --global user.email [이메일]

git 초기 사용시
사용자 정보 입력을 필수로 기입하자.

[trouble shooting] multiple values

git config 실수로, --global을 안쓰고 입력 후,
한번 더 --global로 입력하는 경우 "중복값 에러"가 발생한다.

이때는 --replace-all을 하나 넣어서 입력하면 된다.

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ git config --global user.name inho.choi
warning: user.name has multiple values
error: cannot overwrite multiple values with a single value
      Use a regexp, --add or --replace-all to change user.name.

minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ git config --global --replace-all user.name inho.choi

minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$
```

나오는 에러메세지 & 해결방법

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ git config --global user.name inho.choi

minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ git config --global user.email inho.choi@mincoding.co.kr
```

잘 입력되었는지 확인하는 방법

git status

branch에 대한 내용은 추후 진행할 예정

git 상태 출력

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ git status
on branch master

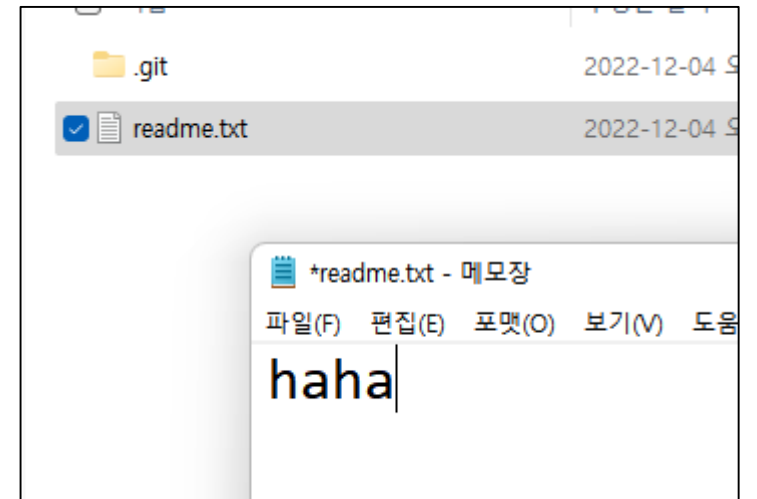
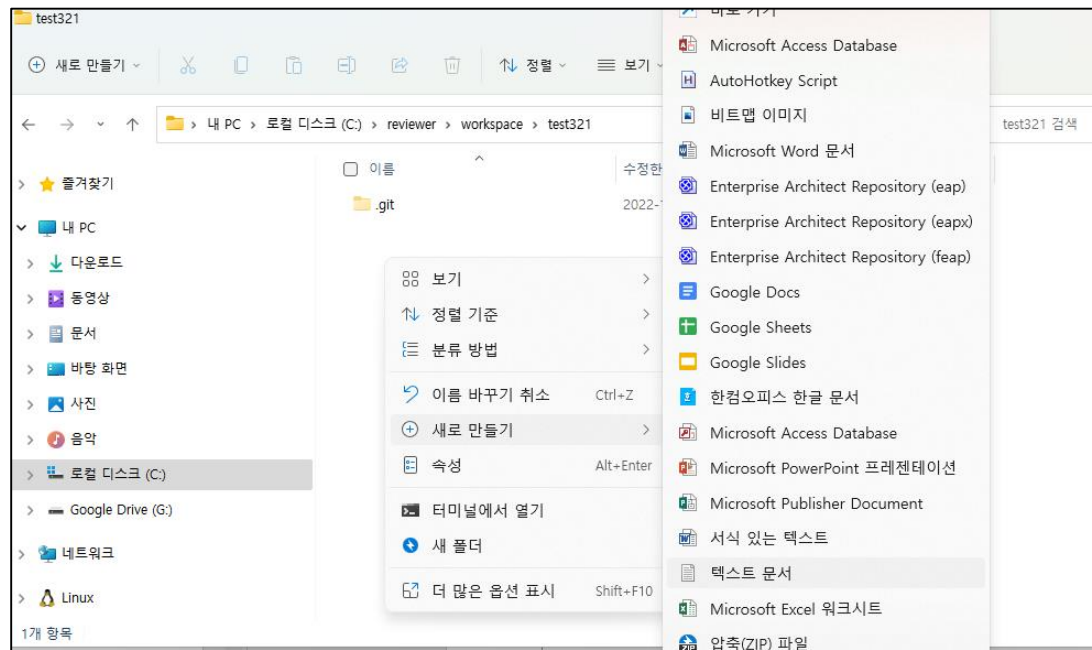
No commits yet

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

minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$
```

파일 생성

readme.txt 파일 생성



git status

readme 문서가 신규로 발견되었음

git으로 관리되고 있지 않은 신규파일을 **untracked file** 이라고 한다.

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)

minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$
```

아까 설명한 내용 중

파일을 Local Repo에 저장하는 것을 Commit 이라고 한다.

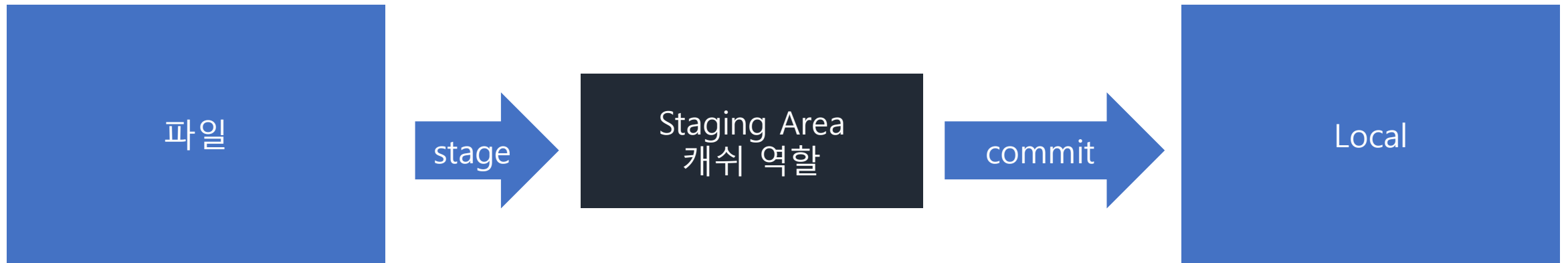
- (사실 한 단계 더 존재 : Staging Area)



Staging Area

신중하게 Local Repo (.git)에 저장하기 위한 단계
캐쉬 역할을 한다.

Staging Area에 올리는 것을 "Stage 하다" 라고 한다.



1. 신중한 Commit 관리 가능
2. 변경 내용을 선택하여 커밋 단위를 적절히 설정할 수 있다
 - 여러 개 파일 중에서 1번 파일과 2번 파일 작업했는데,
1번 파일만 작업 / 테스트 끝났고, 2번 파일은 조금 불안하면
1번 파일만 Stage하고, Commit한다.

git add

git status 입력해보자.

changes : 변경된 파일들을 뜻하는 용어

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ git add ./readme.txt

minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   readme.txt
```

[참고] stage 취소

Staging 되어있는 이력들을 unstage 하려면

git rm --cached readme.txt (최초 스테이징 했을 때)

or

git restore --staged readme.txt

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ git add ./readme.txt

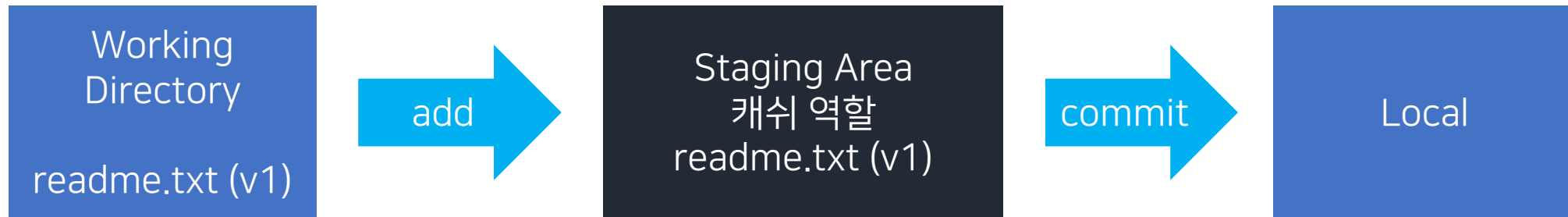
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   readme.txt
```

staging area 이해 -1

staging area 는 Local Repo에 저장하기 전 단계이다

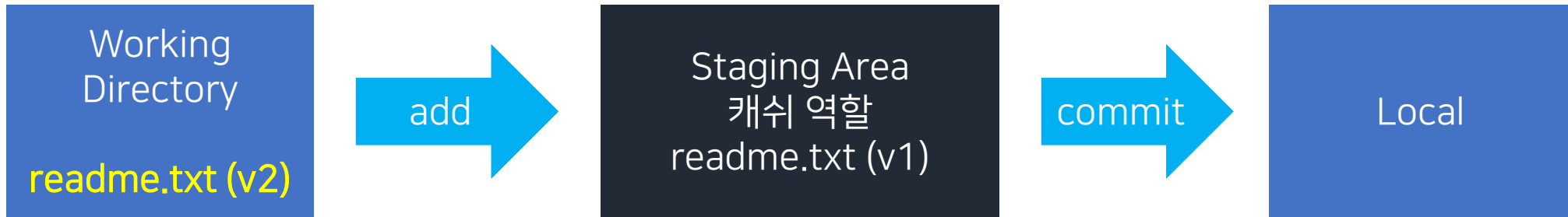


git add readme.txt 를 실행하면 staging area 에 readme.txt 가 올라간다

staging area 이해 -2

readme.txt 를 한번 더 수정한다. (version1 -> version2)

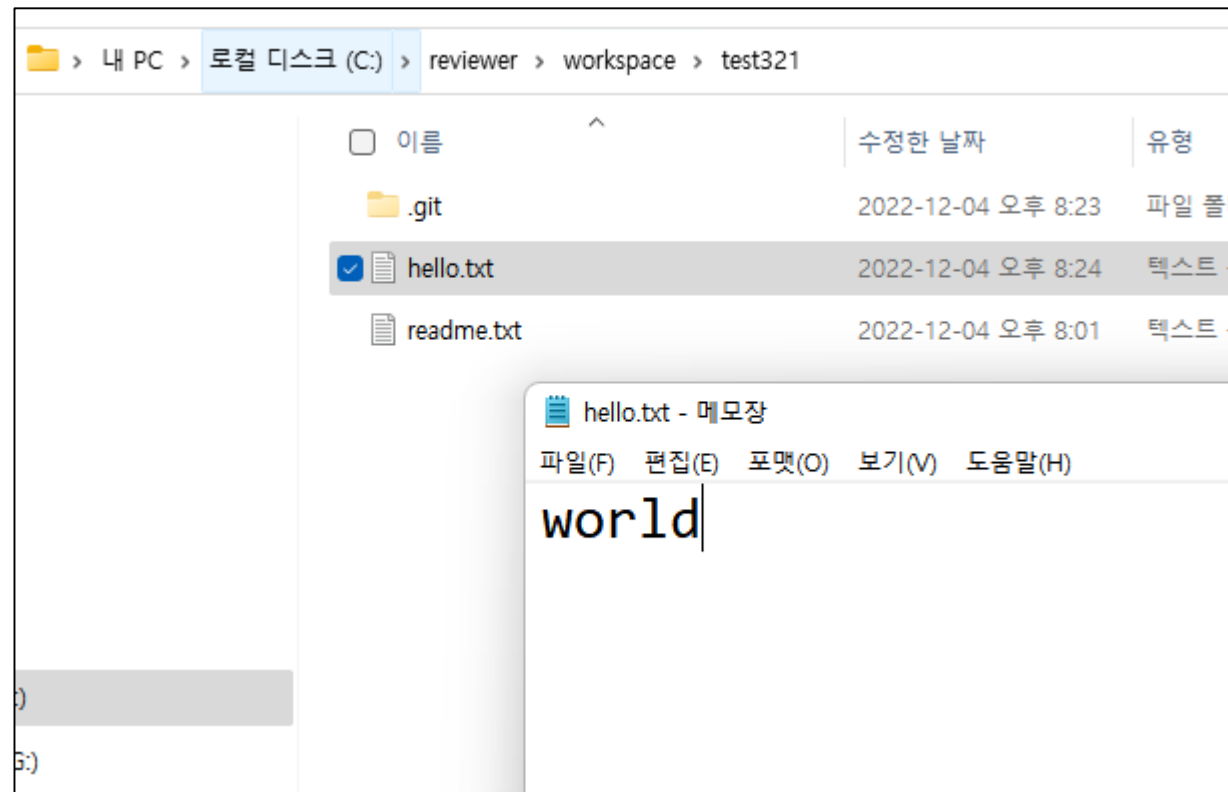
git status 입력 시 어떻게 출력 되는지 메시지를 해석해보자.



```
$ git status
on branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
       modified:   readme.txt

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:   readme.txt
```

파일 하나 더 추가



git add .

한꺼번에 stage 하기

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
gi
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ git add .

minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$
```

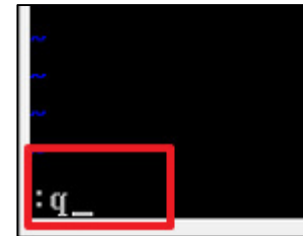
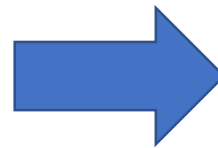
commit

vi 가 실행되기 때문에
이렇게 쓰지 말자.

vi 종료하자.

```
minco@DESKTOP-T4FIKKV M
$ git commit
Aborting commit due to
```

git commit 입력시
리눅스 editor인 "vi"가 실행된다.



vi 에디터 종료 방법

1. ESC 한번 누르고
2. :q 타이핑
3. Enter 입력

Commit -m “메세지”

`commit -m` 으로 명령어를 기억하자.

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ git commit -m 'kfc hello my first commit'
[master (root-commit) 04443ca] kfc hello my first commit
 2 files changed, 1 insertion(+)
 create mode 100644 hello.txt
 create mode 100644 readme.txt

minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$
```


Commit Hash

Commit의 고유 ID 값을 뜻한다.

아래 그림에서 04443ca

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ git commit -m 'kfc hello my first commit'
[master (root-commit) 04443ca] kfc hello my first commit
2 files changed, 1 insertion(+)
create mode 100644 hello.txt
create mode 100644 readme.txt

minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$
```

git의 log가 아닌, **git Commit한 History 보기**

commit hash 값 전체를 확인할 수 있음

앞자리 7자리만으로도 겹칠 일 없어, 앞자리로 관리한다.

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ git log
commit 04443caa18d0ef93fbef05789e2d34aff252b6cb (HEAD -> master)
Author: = <=>
Date:    Sun Dec 4 20:26:50 2022 +0900

    kfc hello my first commit

minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ |
```

[정리. 복습 키워드]

1. Working Directory
2. Staging Area
3. Local Repo.
4. Remote Repo.
5. Stage
6. `git add`
7. `git commit -m`
8. `git log`
9. `git status`
10. untracked files
11. changes
12. `git config`
13. `git init`
14. `push`
15. `pull`

[도전] 새롭게 Working Directory – 5분

1. Working Directory : test1541
2. git init
3. 신규파일 1개 추가
4. stage 후 commit
5. 생성한 파일 수정
6. 다시 stage 후 commit 하기

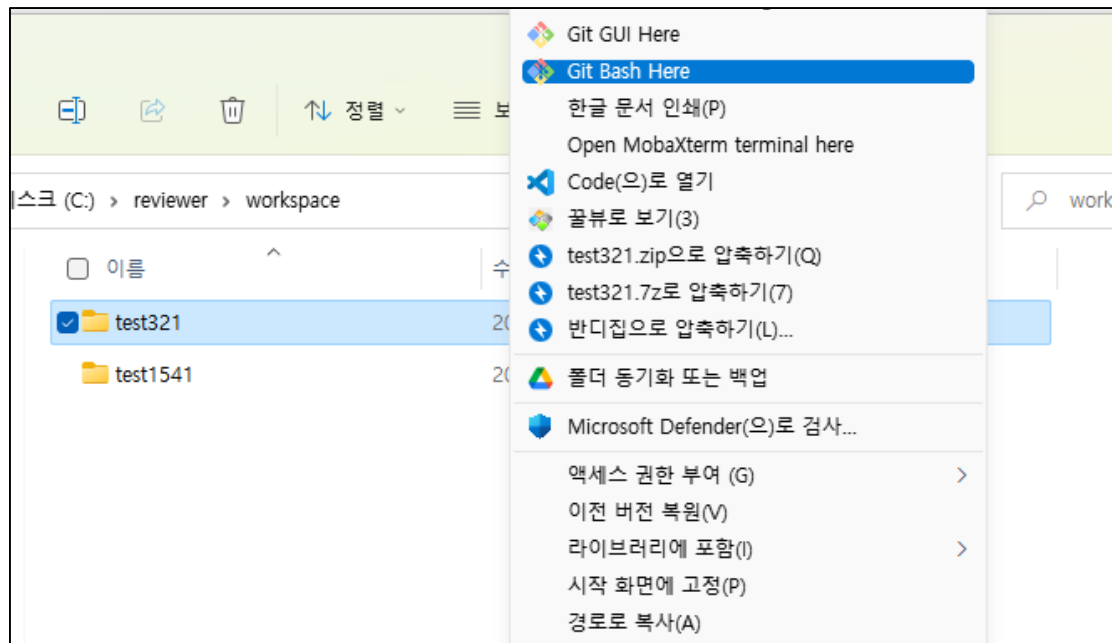
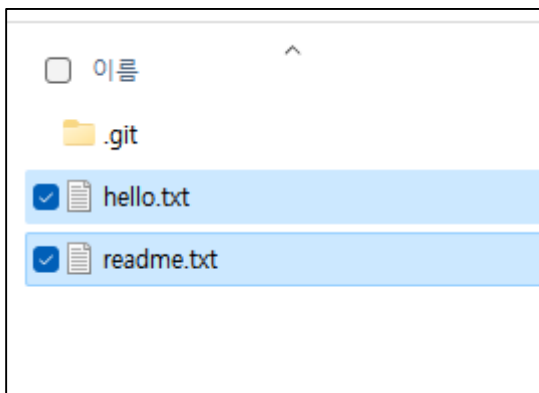
Chapter3

git 2단계

checkout 과 log view

파일 내용 수정하고 Commit 하기

다시 test321 폴더로 돌아오기
2개의 txt 파일 파일 내용 모두 수정
그리고 Git Bash 다시 켜기



Staging

1. git status
2. git add .
3. git status

```
minco@DESKTOP-T4FIKKV MINGW64 /c/review
$ git status
On branch master
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:   hello.txt
        modified:   readme.txt

no changes added to commit (use "git add" to update what will be committed)
```

```
minco@DESKTOP-T4FIKKV
$ git add .
```

```
minco@DESKTOP-T4FIKKV MINGW64 /c/review
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hello.txt
        modified:   readme.txt
```

Commit

git commit -m '메세지'

git log

```
minco@DESKTOP-T4FIKKV MINGW64 /c/rev
$ git commit -m 'fix content'
[master cb2f2f6] fix content
2 files changed, 2 insertions(+), 1
```

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/wor
$ git log
commit cb2f2f68b8e527cdbf7b72f8dcf3fccff78c49
Author: inho.choi <inho.choi@mincoding.co.kr>
Date: Sun Dec 4 21:07:52 2022 +0900
```

fix content

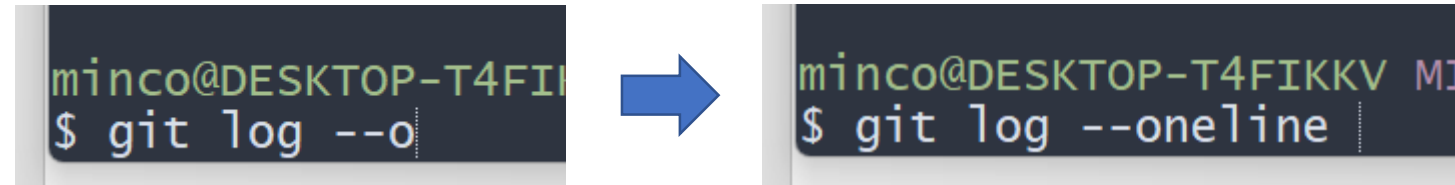
```
commit 04443caa18d0ef93fbeb05789e2d34aff252b6
Author: = <=>
Date: Sun Dec 4 20:26:50 2022 +0900
```

kfc hello my first commit

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/wor
$
```


git log --oneline

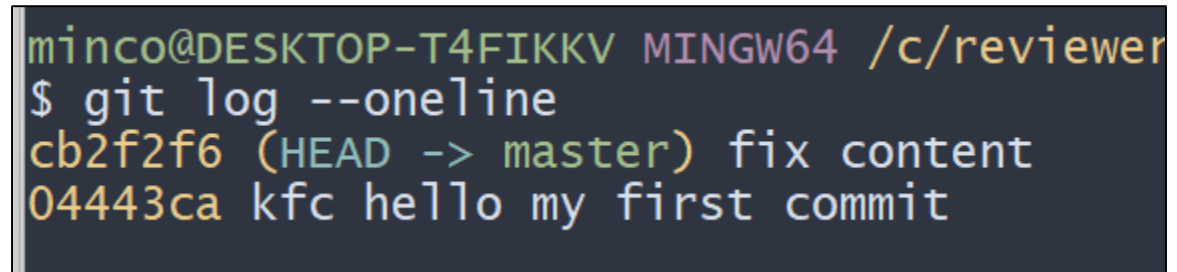
요약본



The diagram illustrates the transformation of the `git log --o` command into `git log --oneline`. It consists of two terminal window screenshots connected by a blue arrow pointing from left to right. The left window shows the command `$ git log --o` with a cursor at the end. The right window shows the command `$ git log --oneline` with a cursor at the end.

--o 입력 후 [Tab]키 입력하기

최신
↑
과거



The terminal output shows the result of running `git log --oneline`. It displays two commit entries in a list. The top entry is `cb2f2f6 (HEAD -> master) fix content` and the bottom entry is `04443ca kfc hello my first commit`. The output is shown in a terminal window with a dark background and light-colored text.

Commit 목록, 위가 가장 최신

git log -1

가장 최근 것 하나만 보기

하이픈(-) 하나만 입력

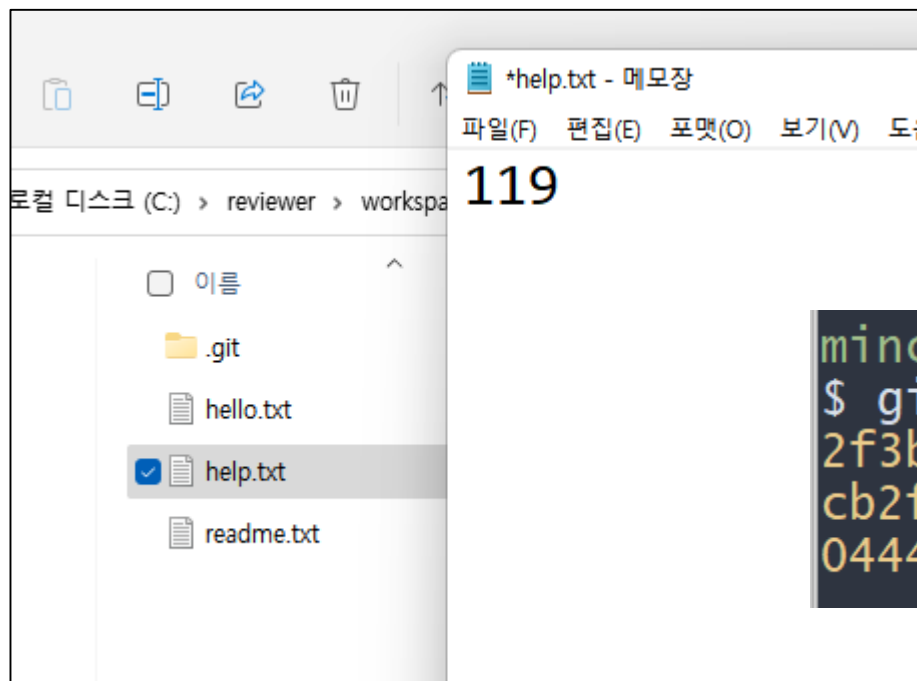
```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$ git log -1
commit cb2f2f68b8e527cdbf7b72f8dcf3fccff78c4964 (HEAD -> master)
Author: inho.choi <inho.choi@mincoding.co.kr>
Date:   Sun Dec 4 21:07:52 2022 +0900

    fix content

minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321 (master)
$
```

[도전] 새로운 파일 하나 더 만들어 Commit

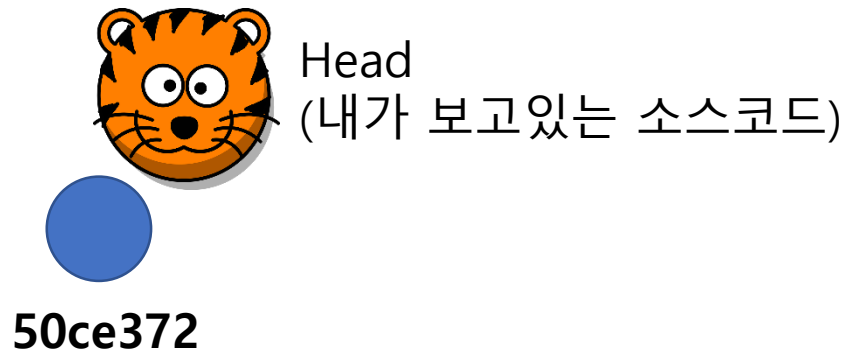
- help.txt 파일 하나 생성 후
- Commit 하기
- (1분)



```
minco@DESKTOP-T4FIKKV MINGW64 /c/rev
$ git log --oneline
2f3b90e (HEAD -> master) new file
cb2f2f6 fix content
04443ca kfc hello my first commit
```

Git log view 1

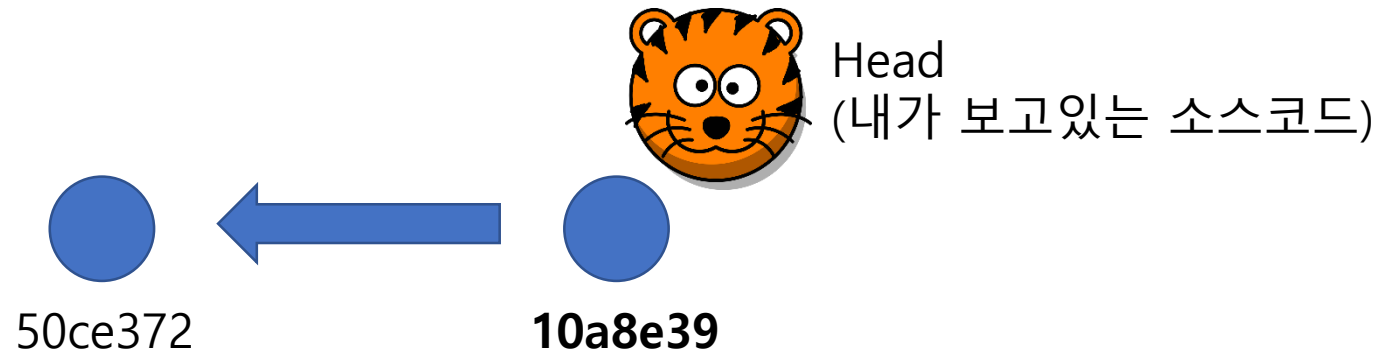
첫 Commit 상태



```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/bt (master)
$ git log --oneline
50ce372 (HEAD -> master) kfc hello my first commit
```

Git log view 2

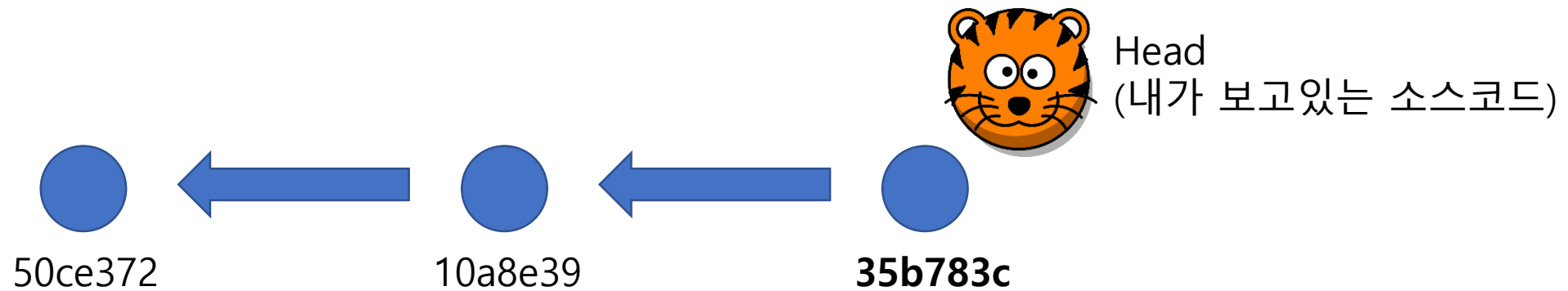
한번 더 Commit



```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/bt (master)
$ git log --oneline
10a8e39 (HEAD -> master) fix content
50ce372 kfc hello my first commit
```

Git log view 3

한번 더 Commit



```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/bt (master)
$ git log --oneline
35b783c (HEAD -> master) new file
10a8e39 fix content
50ce372 kfc hello my first commit
```

git visualizing

명령어 입력해보기

`git commit -m "next2"`

키보드 방향키 위로 한번 입력 후

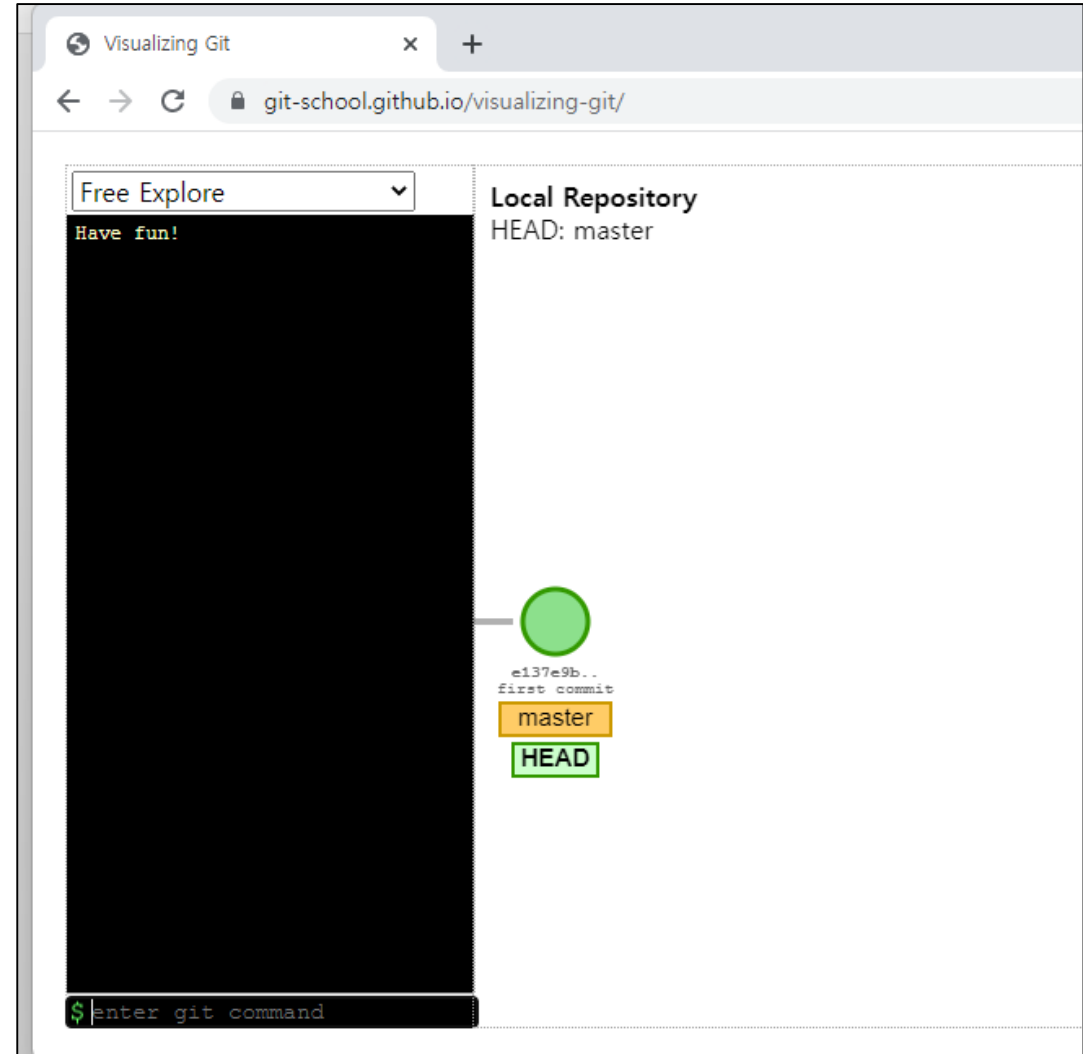
`git commit -m "next3"`

help 입력 시 web app 기능 나옴

undo

clear

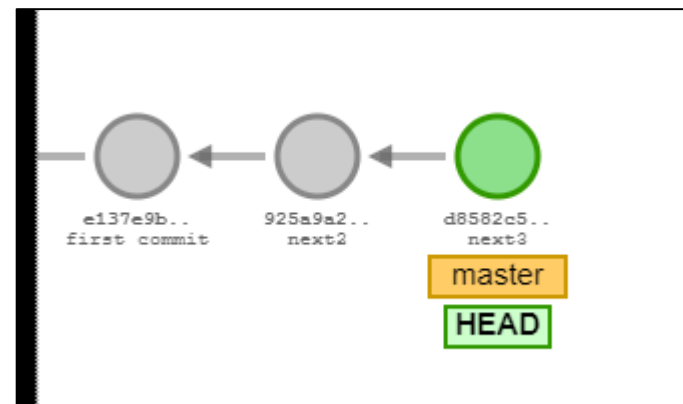
<https://git-school.github.io/visualizing-git/>



용어암기 1 : master branch

소스코드가 Commit되어 관리되는
Commit Line을 Branch 라고 한다.

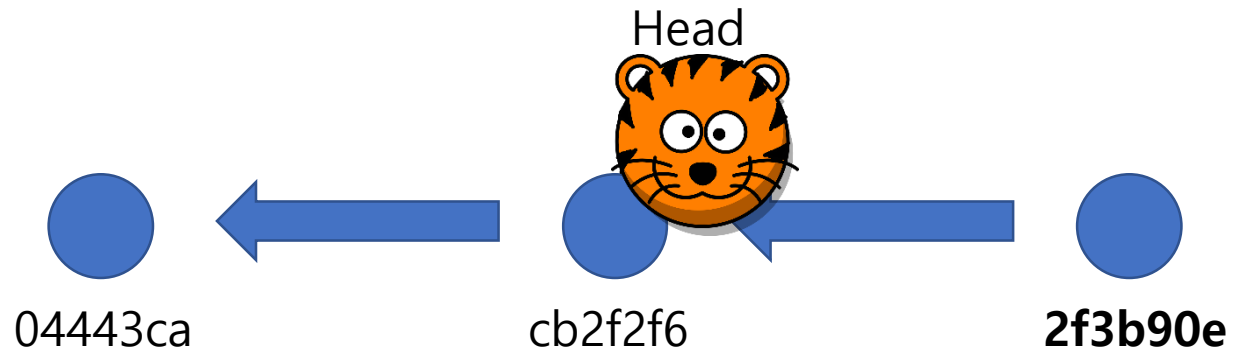
기본 Branch 이름이 Master이다.



용어암기 2 : Head

지금 바라보고있는 소스코드를 뜻한다.

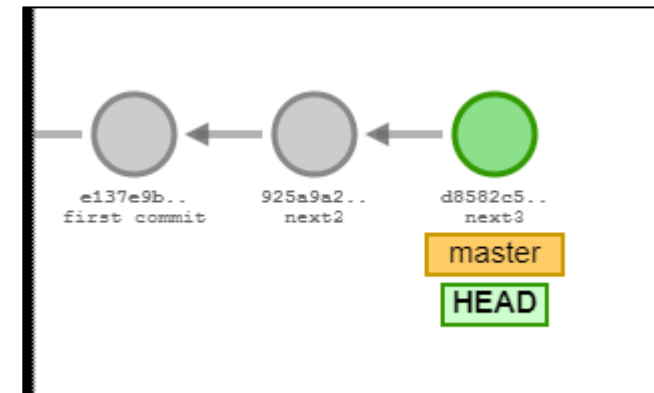
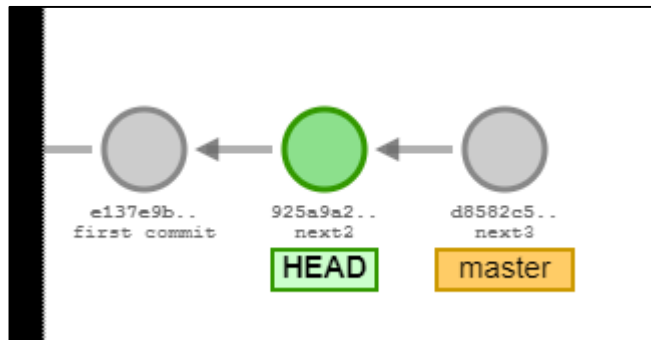
Head를 옮기면,
실제 소스코드가 변경된다.



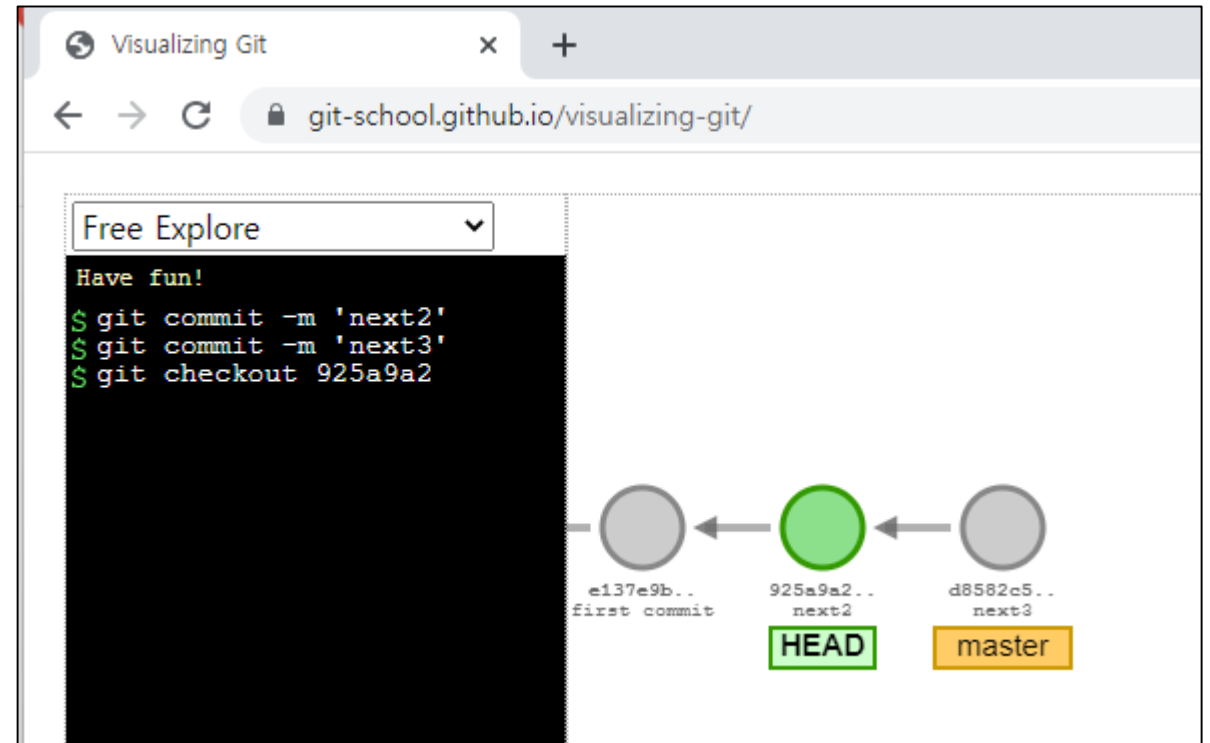
git checkout master

git checkout master

현재 branch (master라는 History Line) 의 최신본으로 HEAD를 옮기는 명령어

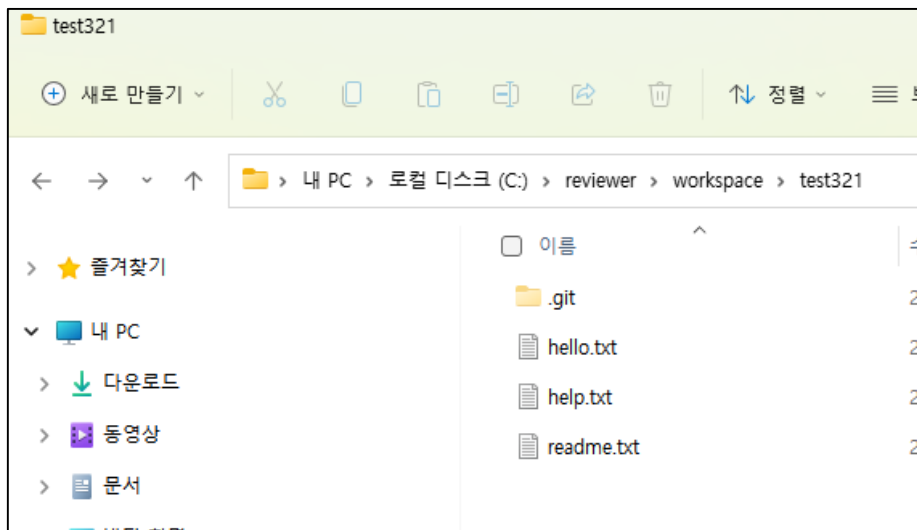


git checkout [commit hash번호]
Head가 이동한다.

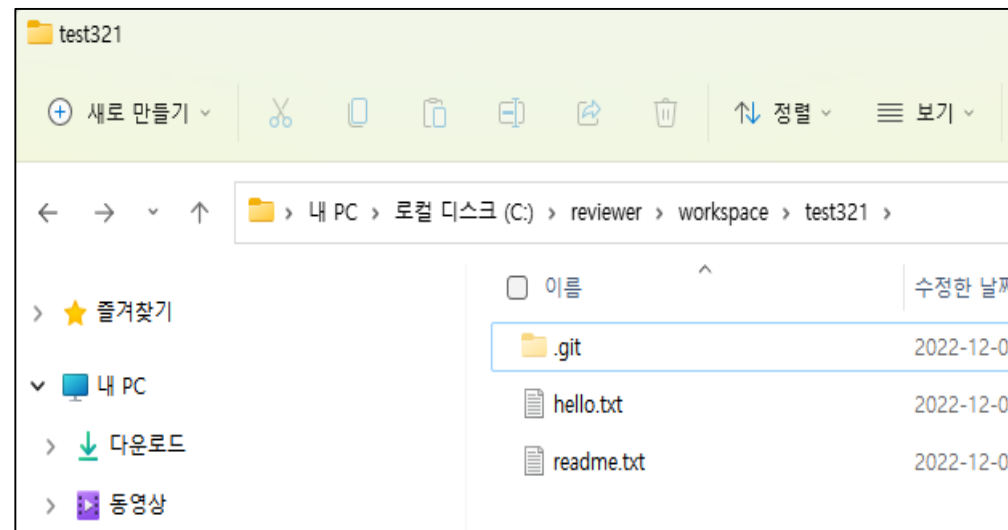


실제로 해보자.

Head를 두 번째 Commit 으로 옮기자.



세 번째 Commit 상태



두 번째 Commit 상태

git checkout [Commit Hash]

HEAD를 옮겨, 특정 Commit을 보러 가는 명령어

권장사항 : 여기서 한번 더 Commit 하지 말자. 복잡해진다.

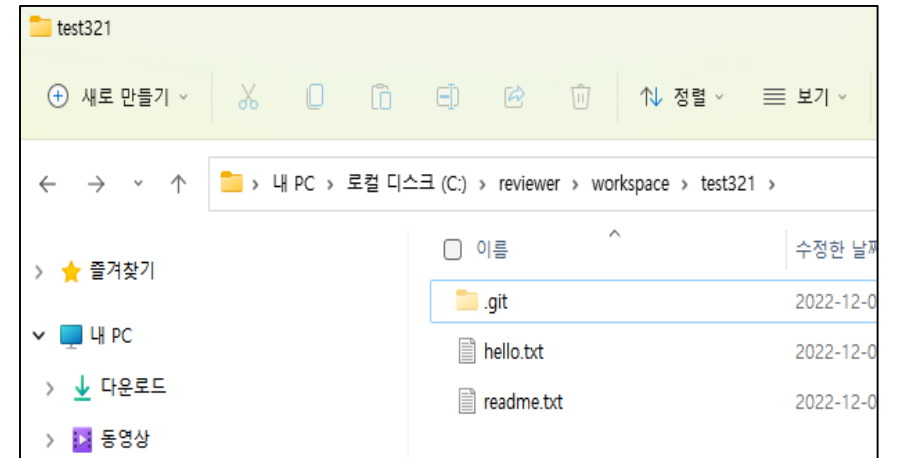
(detached HEAD 상태에 대한 처리 필요)

지금은 잠시 과거를 살펴보는 용도로 사용한다.

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321
$ git log --oneline
2f3b90e (HEAD -> master) new file
cb2f2f6 fix content
04443ca kfc hello my first commit

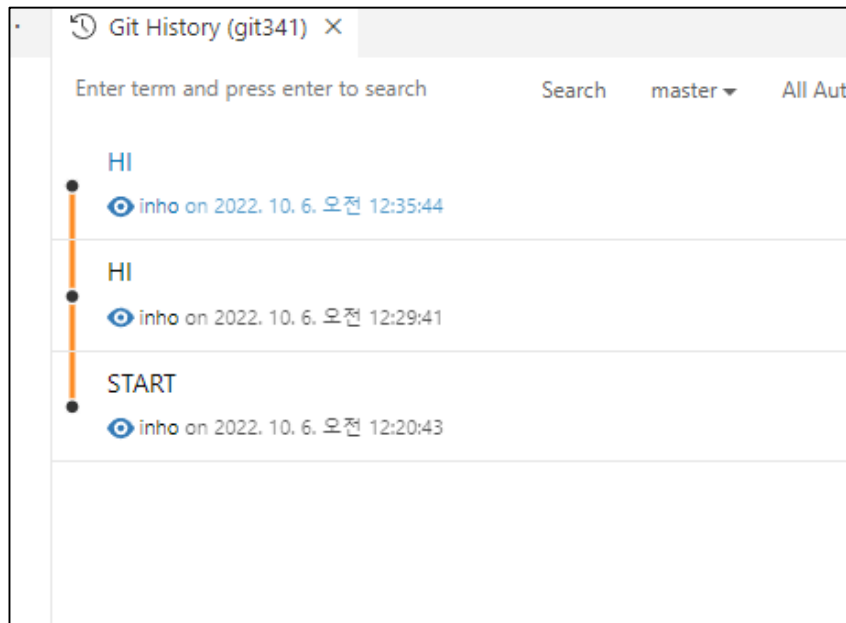
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test321
$ git checkout cb2f2f6
Note: switching to 'cb2f2f6'.

You are in 'detached HEAD' state. You can look around, make
changes and commit them, and you can discard any commits you
state without impacting any branches by switching back to a
```

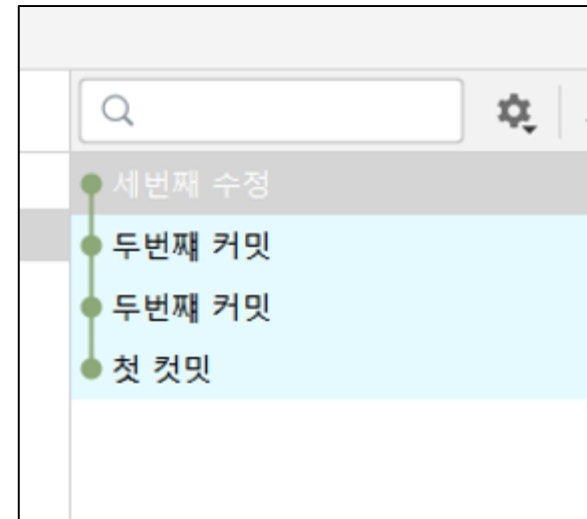


[참고] Git Log view

(GUI 장점) Git Log View가 편리하다.



Visual Studio Code (with Plugin)



IntelliJ Git Log view

Git 활용 : Git으로 디버깅이 가능하다.

De : 죽이다.

Bug : 벌레

ing : ~ 하다.

Debugging : 버그 죽이는 활동

Git을 사용해서 디버깅을 할 수 있다.

예시를 들어보자.

Git History에 다음과 같다.

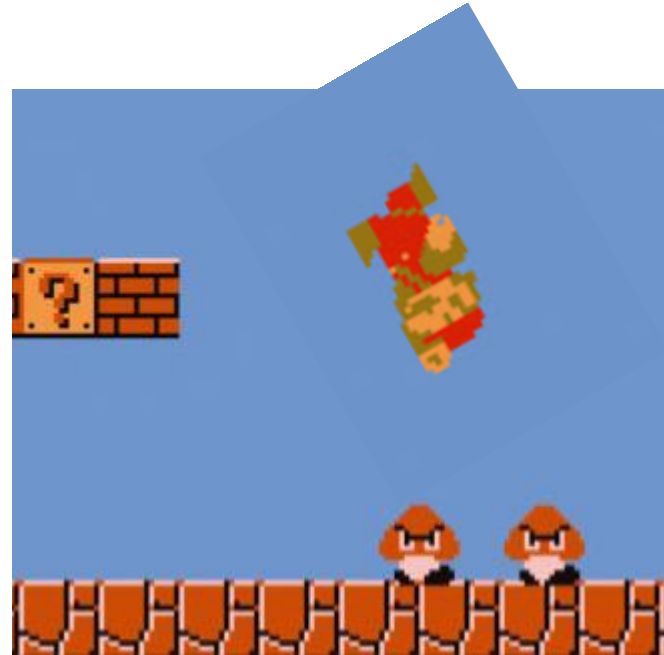
12/21일, W와리오 점프 기능 추가
12/20일, W와리오 캐릭터 추가
12/19일, 슈퍼 점프 기능 추가.
12/16일, 마리오 점프, 버그 수정
12/15일, 점프 기능 업그레이드



갑자기 마리오 점프가 이상하다!

버그가 갑자기 발견되었다!
오늘 날짜는 12/22일이다.

12/21일, W와리오 점프 기능 추가
12/20일, W와리오 캐릭터 추가
12/19일, 슈퍼 점프 기능 추가.
12/16일, 마리오 점프, 버그 수정
12/15일, 점프 기능 업그레이드



언제부터 이 버그가 발견되었을까?

소스코드를 과거로 돌려보자.

12/21일, W와리오 점프 기능 추가
12/20일, W와리오 캐릭터 추가
12/19일, 슈퍼 점프 기능 추가.
12/16일, 마리오 점프, 버그 수정
12/15일, 점프 기능 업그레이드

잘 됨



언제부터 이 버그가 발견되었을까?

소스코드를 과거로 돌려보자.

12/21일, W와리오 점프 기능 추가
12/20일, W와리오 캐릭터 추가
12/19일, 슈퍼 점프 기능 추가.
12/16일, 마리오 점프, 버그 수정
12/15일, 점프 기능 업그레이드

잘 됨

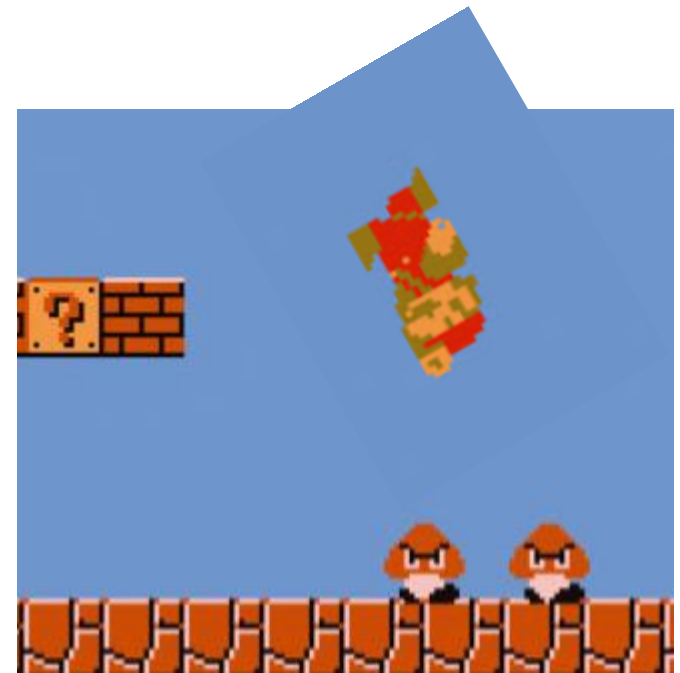


언제부터 이 버그가 발견되었을까?

소스코드를 과거로 돌려보자.

12/21일, W와리오 점프 기능 추가
12/20일, W와리오 캐릭터 추가
12/19일, 슈퍼 점프 기능 추가.
12/16일, 마리오 점프, 버그 수정
12/15일, 점프 기능 업그레이드

버그 재현
성공



그렇다면 버그가 발생한 Commit은?

며칠에 진행한 Commit에서 버그를 만들어 낸 것일까?

12/21일, W와리오 점프 기능 추가
12/20일, W와리오 캐릭터 추가
12/19일, 슈퍼 점프 기능 추가.
12/16일, 마리오 점프, 버그 수정
12/15일, 점프 기능 업그레이드

VS

12/21일, W와리오 점프 기능 추가
12/20일, W와리오 캐릭터 추가
12/19일, 슈퍼 점프 기능 추가.
12/16일, 마리오 점프, 버그 수정
12/15일, 점프 기능 업그레이드

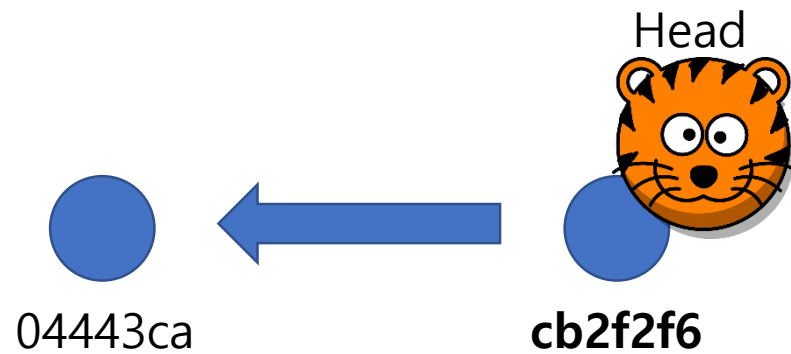
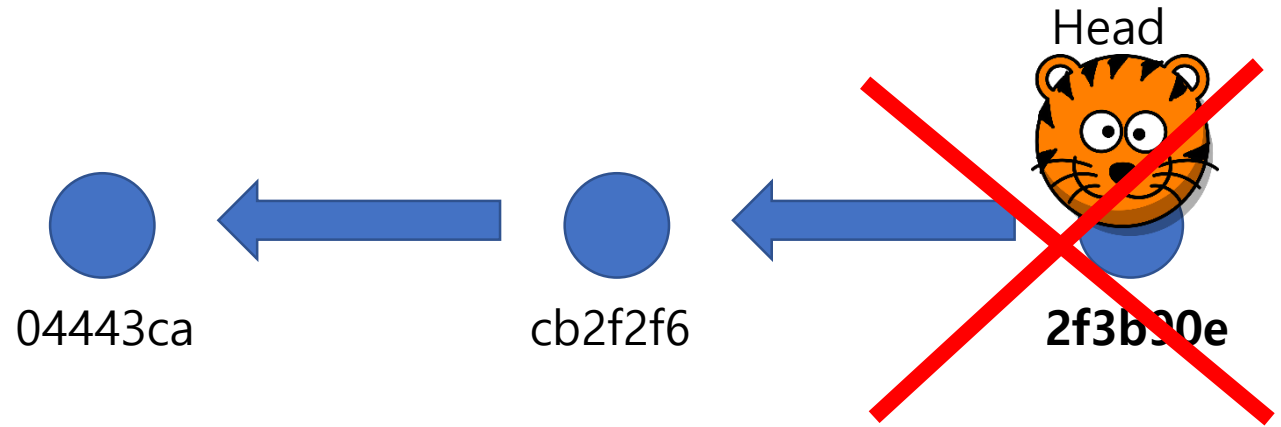
Git의 History 기능

- 타임머신타고, 그 당시 소스코드로 회기 할 수 있음

Git Hard Reset

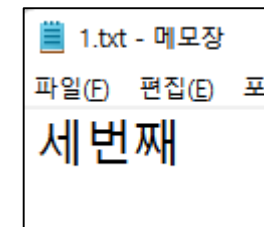
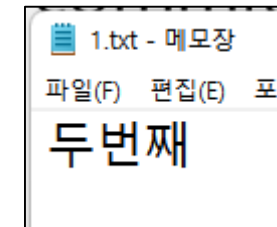
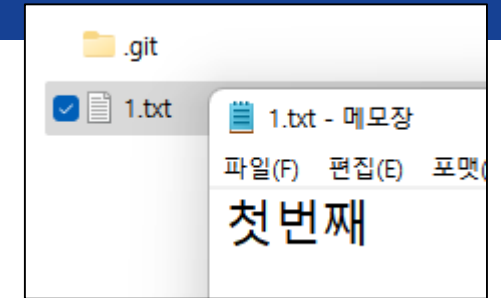
checkout이 아닌,
정말로 되돌리는 방법

git reset --hard [Commit Hash 값]



[도전] git commit – 5분

1. 새로운 Work dir 생성
2. 1.txt 파일 생성 후 “첫번째” 파일 내용 삽입
3. 최초의 Commit 수행
4. 파일 내용 수정 후 두 번째 Commit하기
5. 파일 내용 수정 후 세 번째 Commit하기
6. 최초의 Commit으로 checkout하기
7. 파일 내용 확인하기
8. 최근 Commit으로 checkout하기
9. 파일 내용 확인하기



Chapter4

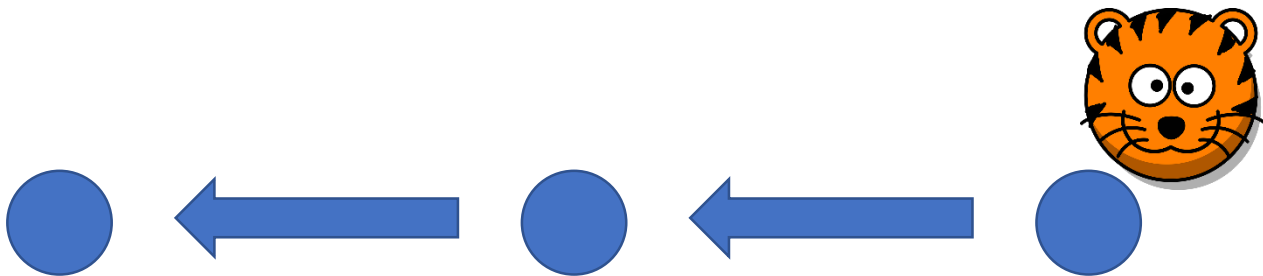
git 3단계

branch & merge

Branch란?

독립적으로 작업을 수행하기 위해 사용된다.
여러 사람이 한 프로젝트를 함께 개발할 때 유용

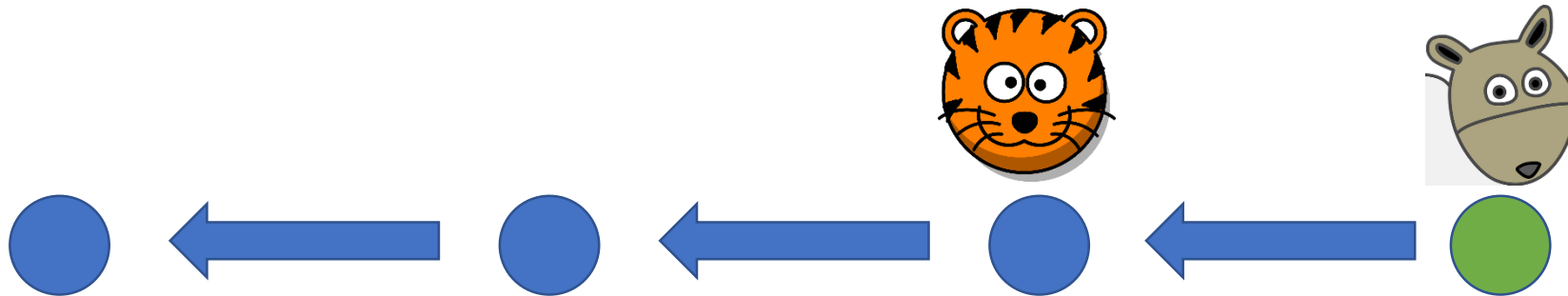
양은 호랑이를 위해,
기능 추가를 도와주러 왔다고 가정하며 예시를 살펴보자.



하나의 소스코드를 공동 작업시.. 1

호랑이가 작성한 코드를 수정하여
양이 Commit을 했다.

그런데, 호랑이는 Commit했던 사실을 모르고
본인의 수정 작업을 계속한다.



하나의 소스코드를 공동 작업시.. 2

호랑이가 열심히 수정한 소스코드를 Commit 하려고 하는데,
Git이 양이 Commit한 내용과
소스코드가 충돌 난다고 에러가 뜨면서 Commit 이 안된다.

→ 소스코드를 하나씩 비교해보며, 충돌 해결작업이 필요



하나의 소스코드를 공동 작업시.. 3

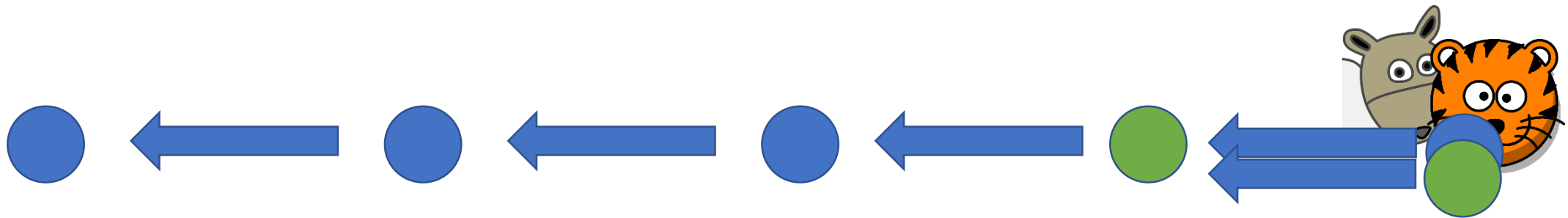
한번 두번은..

충돌날때마다 충돌을 해결하는 소스코드

Merge 작업을 하면 되지만,

이게 서로 조금 수정하고 Commit 자주 하고

매번 충돌이 발생해서 Merge를 해야한다면, 필요없는 업무량 증가한다.

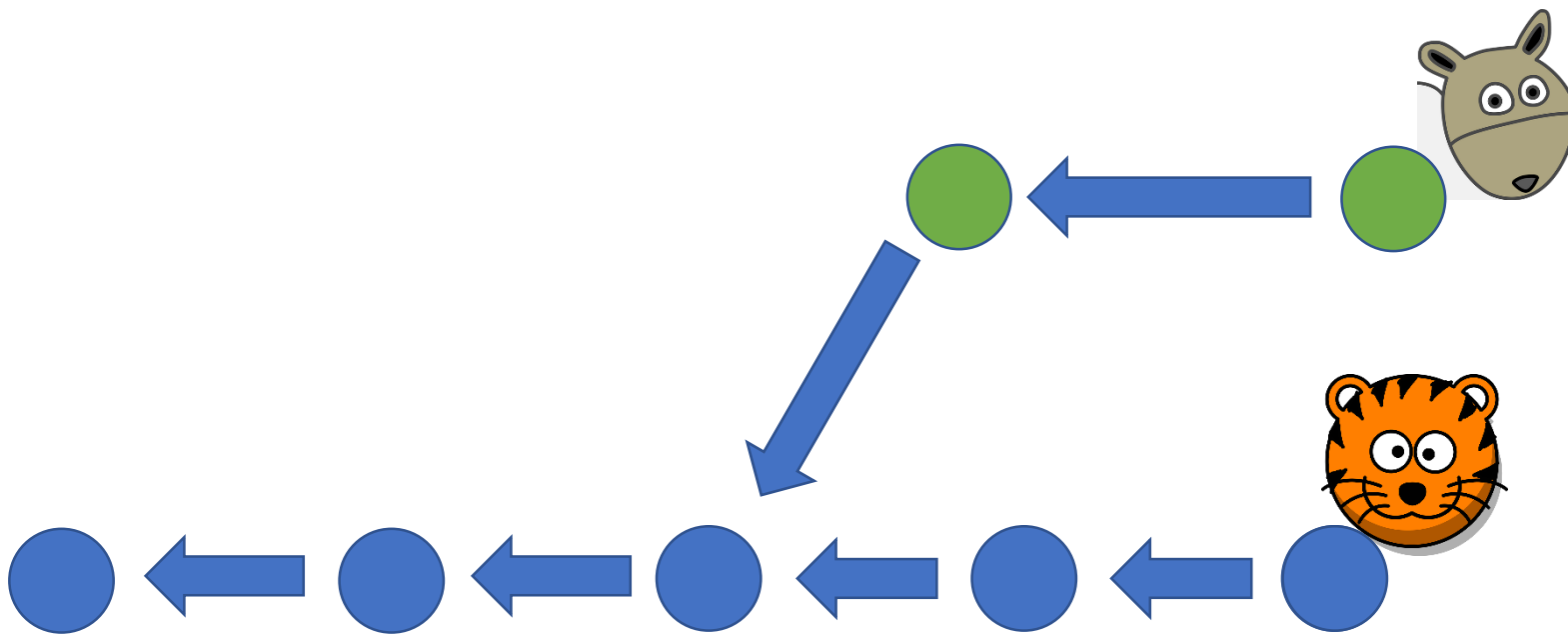


해결방안은 Branch

독립적인 소스코드를 갖고,

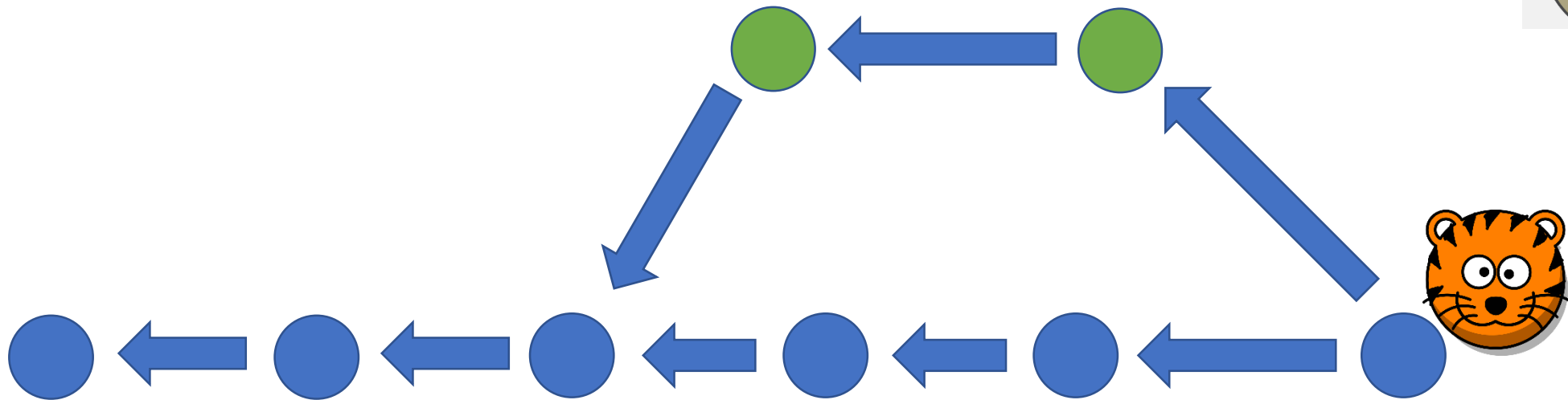
나만의 Time Line을 만든다. = Branch를 하나 만든다.

= 나만의 Branch에서 충돌걱정 없이 편안하게 Commit 한다.



소스코드를 하나로 합쳐야 할 때는 Merge

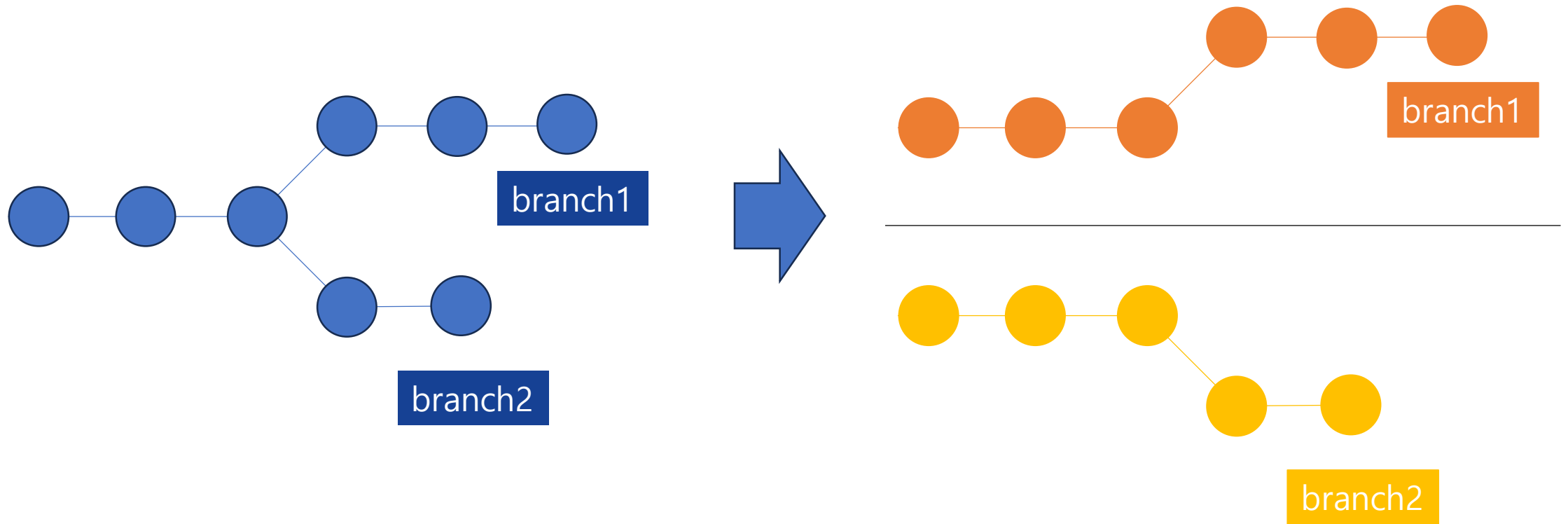
어느정도 완성도를 보인 후에,
두 사람의 소스코드를 하나로 합쳐 적용시키면 된다.



난 개발 끝~
안녕

독립적인 타임라인

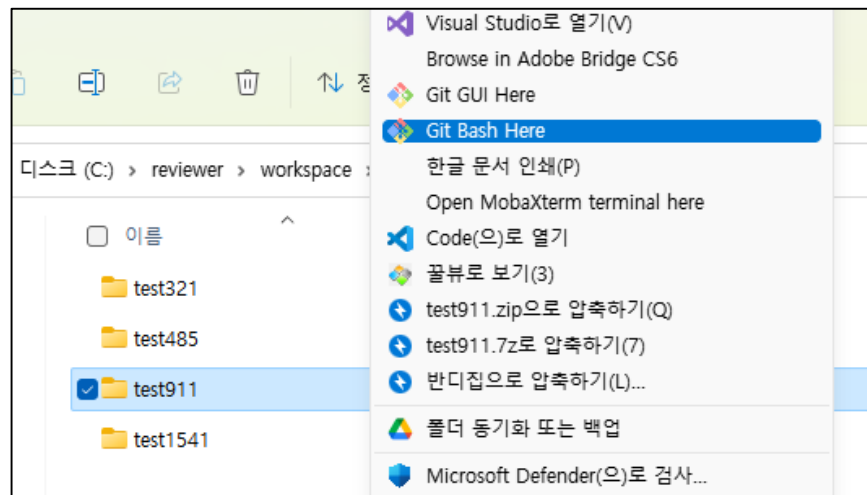
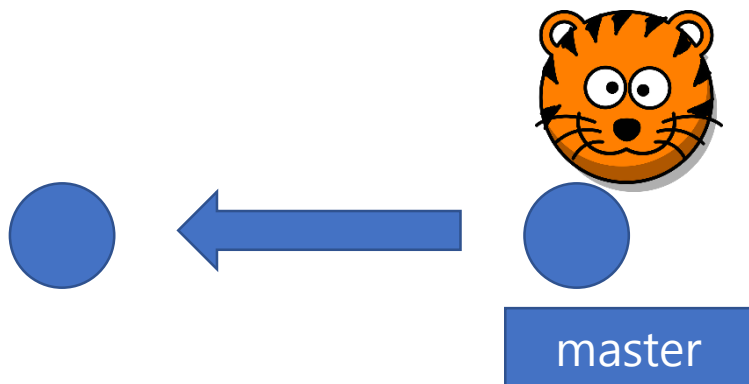
- branch 를 이용하면 독립적인 타임라인에서 작업을 할 수 있다.



새로운 Working Dir. 준비

test911 폴더 생성

1. git init
2. sample.txt 파일 생성 후 Commit
3. 내용 수정 후 Commit



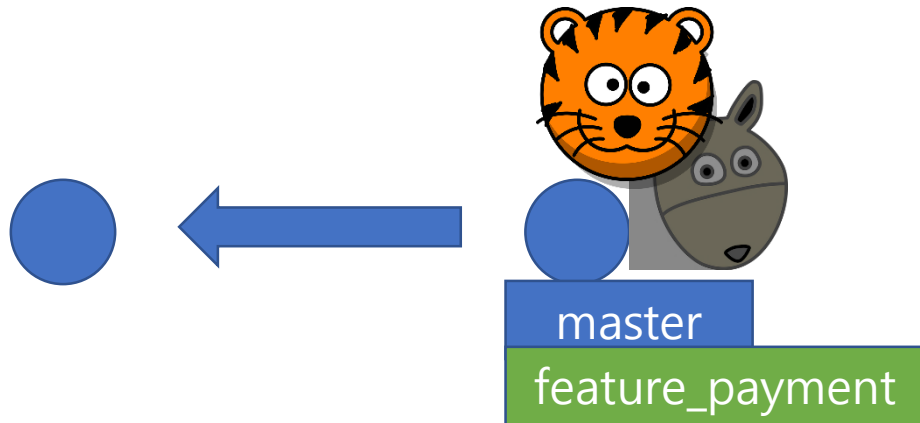
```
minco@DESKTOP-T4FIKKV MINGW64 /c/
$ git log --oneline
9fa6fc0 (HEAD -> master) second
ed5ee99 first file
```

branch 생성하기

git branch [branch 이름]

새로운 branch를 만들면,
양 입장에서는 최초의 branch가 생성되었다.

```
minco@DESKTOP-T4FIKKV MINGW64 /  
$ git branch feature_payment
```



git branch 보기

생성 후 git branch라고 입력하면, 목록을 확인할 수 있음
잘못 만든 경우는 git branch -d [브랜치명]

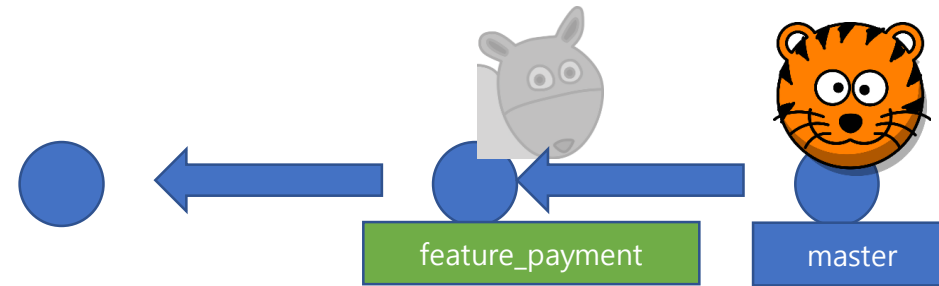
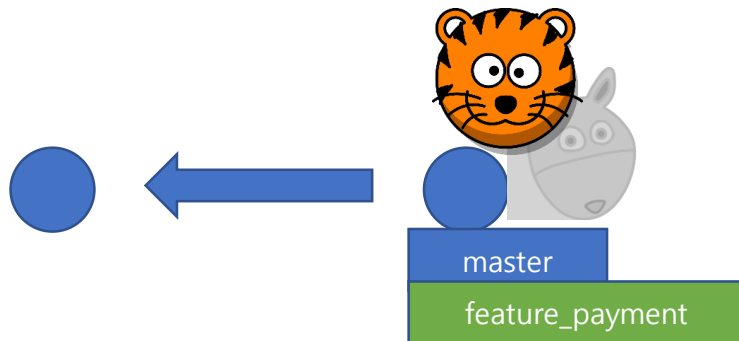
branch를 지우고 다시 만들어보자.

1. git branch
2. git branch -d feature_payment
3. git branch
4. git branch feature_payment
5. git branch

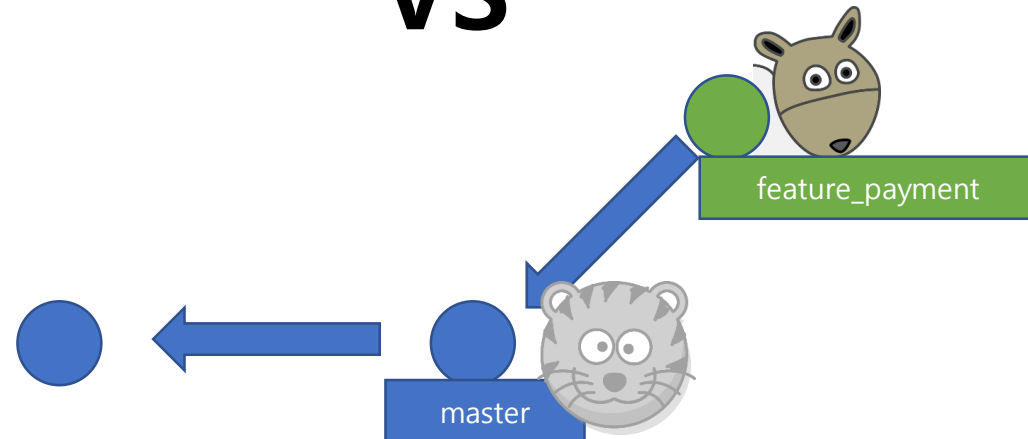
지금 상태에서 Commit하면?

master branch에서 commit하면, 어떤 그림의 형태로 commit이 되는 것일까?

```
minco@DESKTOP-T4FIK  
$ git branch  
feature_payment  
* master
```

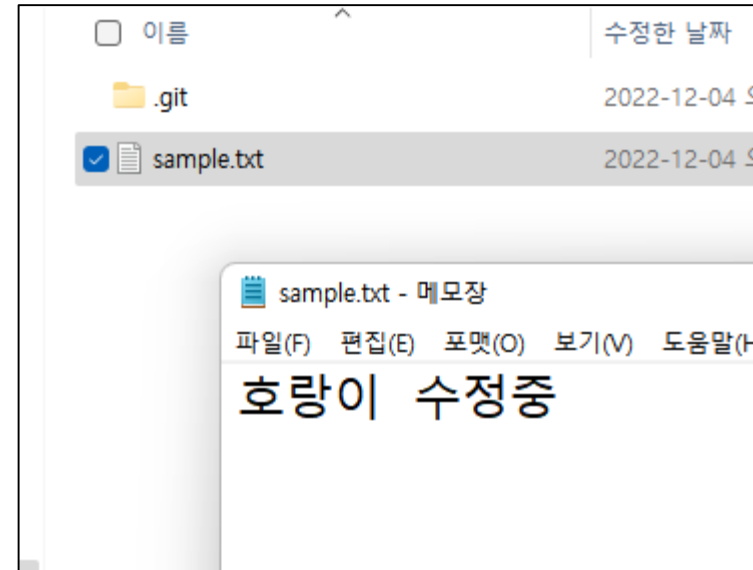
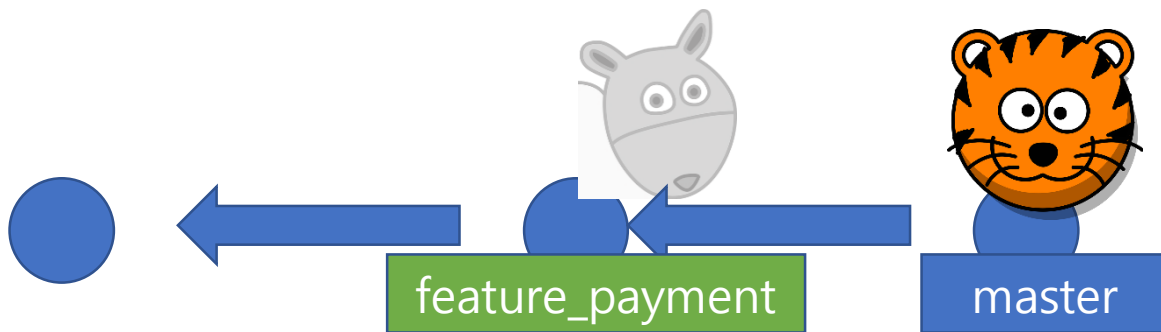


VS



Commit 한다.

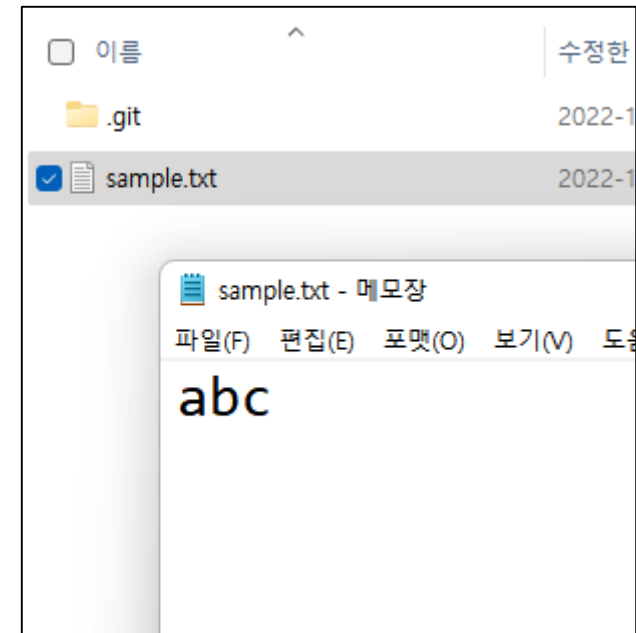
파일 내용을 수정 후
Commit 하자.



양이 작업을 시작한다.

새로운 기능 개발을 위해
소스코드 작업을 시작한다.

git checkout **feature_payment**

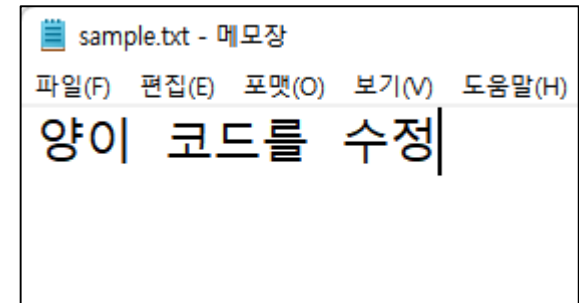
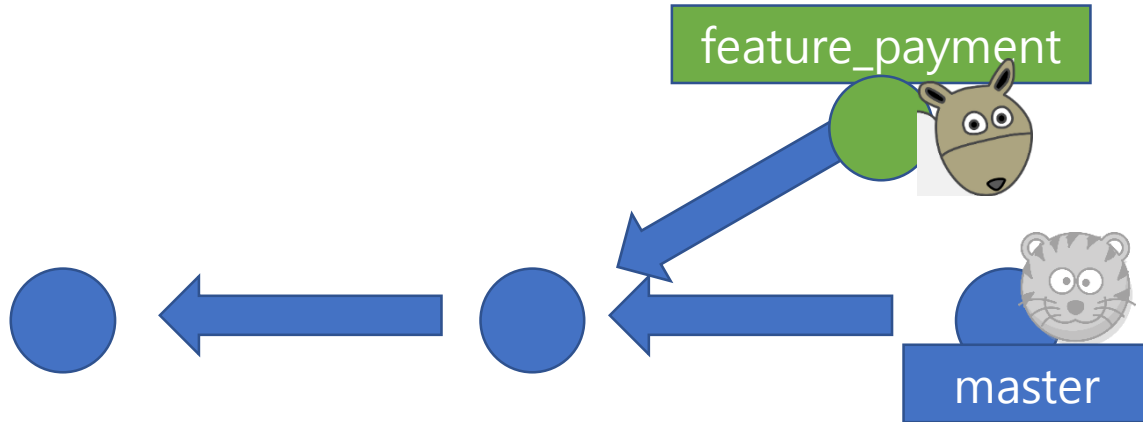


파일이 다시 수정되어있다.

양의 새로운 Commit

소스코드 수정 후 Commit

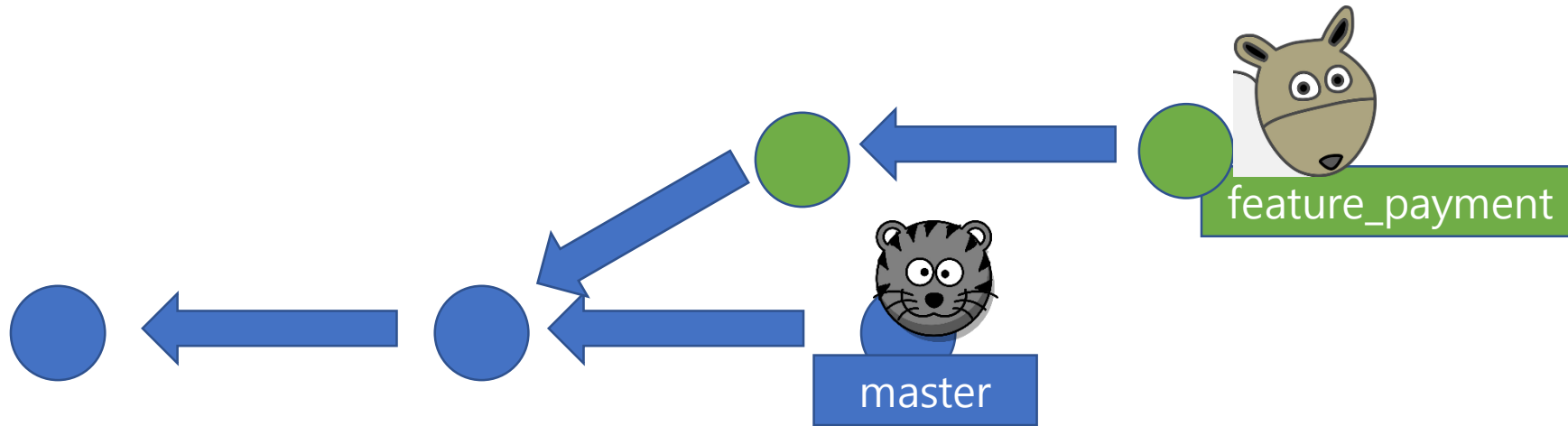
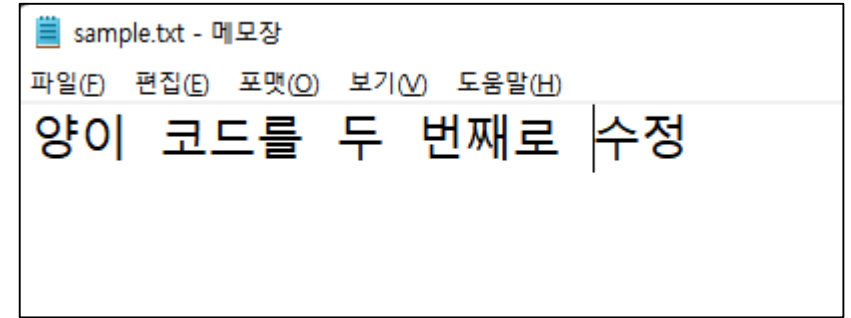
```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/worksp  
$ git add .  
  
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/worksp  
$ git commit -m 'payment1'  
[feature_payment 8ad3681] payment1  
1 file changed, 1 insertion(+), 1 deletion(-)
```



한번 더 수정 후 Commit

소스코드 수정 후 Commit

이제 양은 개발이 완료되었다.



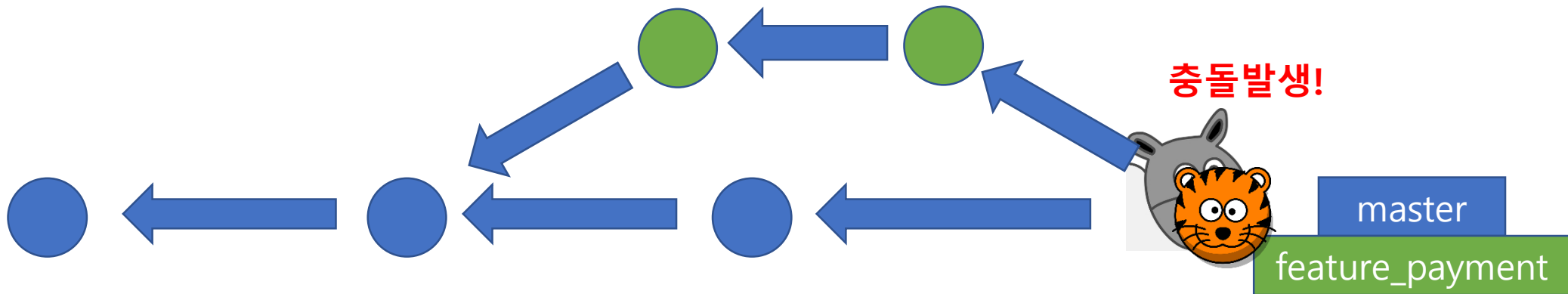
Merge하기

master로 체크아웃 후,
git merge [합칠 브랜치명] 입력

자동 Merge에 실패했다.
호랑이와 양의 소스코드를 수동으로
Merge 해야 한다.

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test911 (feature_payment)
$ git checkout master
Switched to branch 'master'

minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test911 (master)
$ git merge feature_payment
Auto-merging sample.txt
CONFLICT (content): Merge conflict in sample.txt
Automatic merge failed; fix conflicts and then commit the result.
```



Merge 작업을 위한 충돌 해결 1

git status를 입력하면
sample.txt 파일에서 충돌났음을 알 수 있다.



```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test911 (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   sample.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

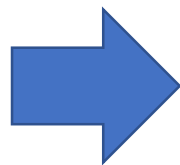
Merge 작업을 위한 충돌 해결 2

1. 현재 HEAD 소스코드 (master)
2. 합쳐질 feature_payment 소스코드

둘 다 표시되어 있다.

수작업으로 어떻게 합칠지 소스코드를 수정해준다.

```
sample.txt - 메모장
파일(F) 편집(E) 포맷(O) 보기(V) 도움말(H)
<<<<<<< HEAD
호랑이 수정중
=====
양이 코드를 두 번째로 수정
>>>>>>> feature_payment
```



```
sample.txt - 메모장
파일(F) 편집(E) 포맷(O) 보기(V) 도움말(H)
호랑이 수정중
양이 코드를 두 번째로 수정|
```

Merge 작업을 위한 충돌 해결 3

이제 add 해주고, status를 확인한다.

```
minco@DESKTOP-T4FIKKV MINGW64 /c/review
$ git add .

minco@DESKTOP-T4FIKKV MINGW64 /c/review
$ git status
On branch master
All conflicts fixed but you are still in merge mode
  (use "git commit" to conclude merge)

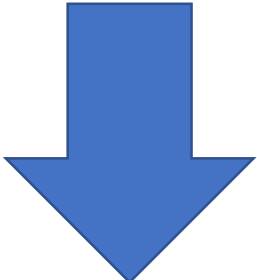
changes to be committed:
  modified:   sample.txt
```

Merge 작업을 위한 충돌 해결 4

Commit을 해주면


정상적으로 merge 된다.

branch가 제거된 것이 아니다.



```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test911 (master|MERGING)
$ git commit -m 'merge_feature_payment'
[master c50b9f8] merge_feature_payment

minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test911 (master)
$
```



현재까지 상황

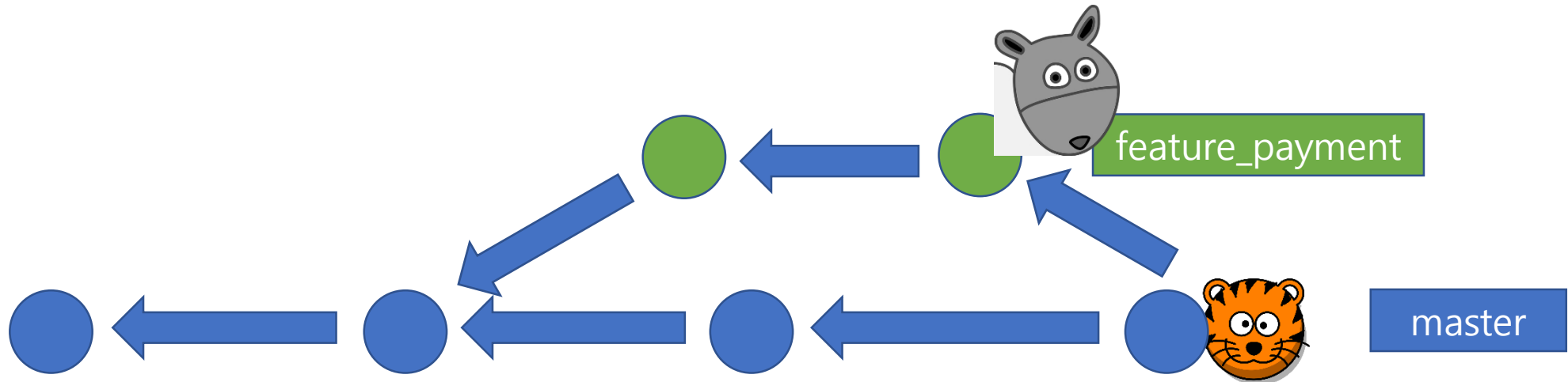
git branch를 하면

아직 feature_payment branch가 살아있음을 알 수 있다.

더 이상 필요하지 않기 때문에 제거하자.

`git branch -d feature_payment`
(삭제해도 복구할 수 있음)

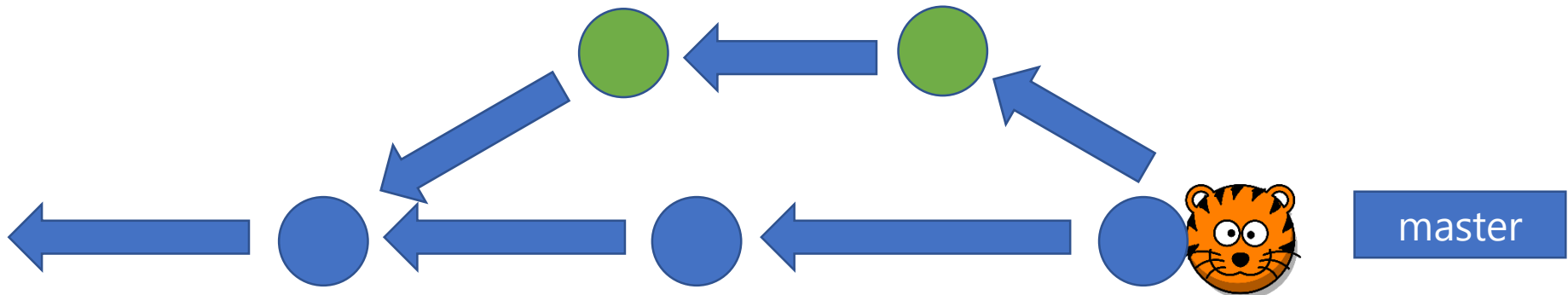
```
minco@DESKTOP-T4FIKKV MIN  
$ git branch  
feature_payment  
* master
```



Branch 종료

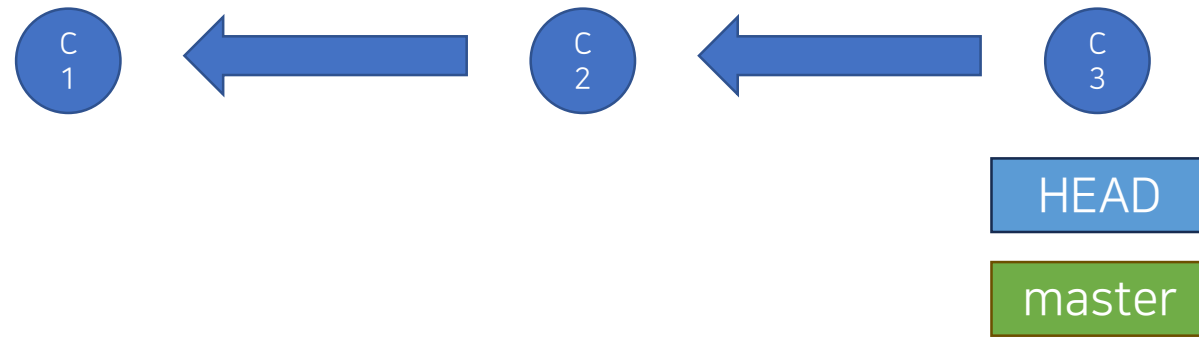
기능 추가 완료

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspa  
$ git branch -d feature_payment  
Deleted branch feature_payment (was 033655a).  
  
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspa  
$ git branch  
* master
```

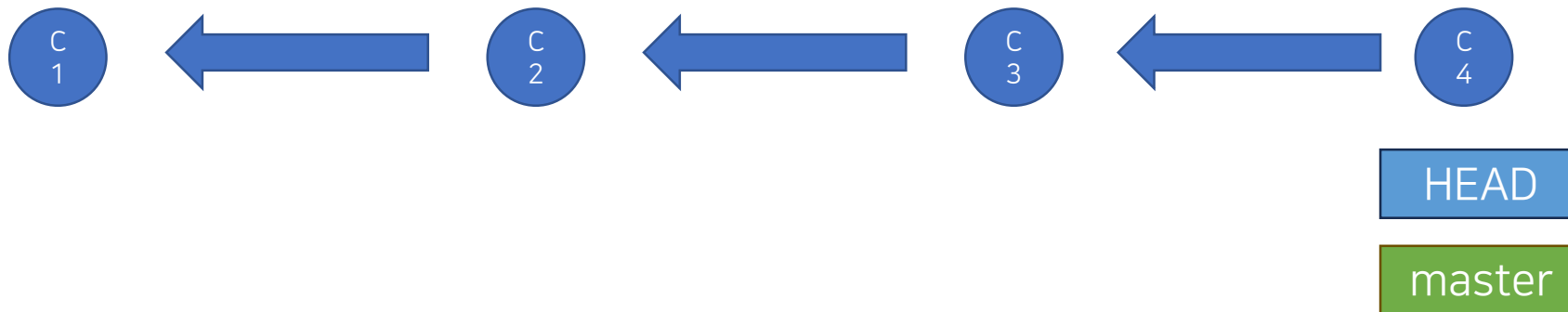


branch

branch 는 HEAD 와 마찬가지로 특정 커밋을 가리키는 포인터다



새로운 커밋을 생성하면 HEAD와 함께 이동한다(웃말 었어가듯)



git branch

현재 branch들 확인하기(*master 는 현재 checkout 되어있는 브랜치다)

```
MINGW64:/c/Users/jeong/Desktop/git_sample/branchSample
jeong@mincoding MINGW64 ~/Desktop/git_sample/branchSample (master)
$ git branch
  feature1
  feature2
* master
```

특정 커밋에 새로운 branch 포인터 두기

1. HEAD 위치에 branch 두기
2. 다른 커밋으로 checkout 하고 branch 두기

```
MINGW64:/c/Users/jeong/Desktop/git_sample/branchSample
jeong@mincoding MINGW64 ~/De
$ git branch feature
```

새로 생성할 브랜치 이름

branch 포인터 삭제하기

```
MINGW64:/c/Users/jeong/Desktop/git_sample/branchSample
jeong@mincoding MINGW64 ~/Desktop/git_
$ git branch -d feature
Deleted branch feature (was bfa15b1).
```

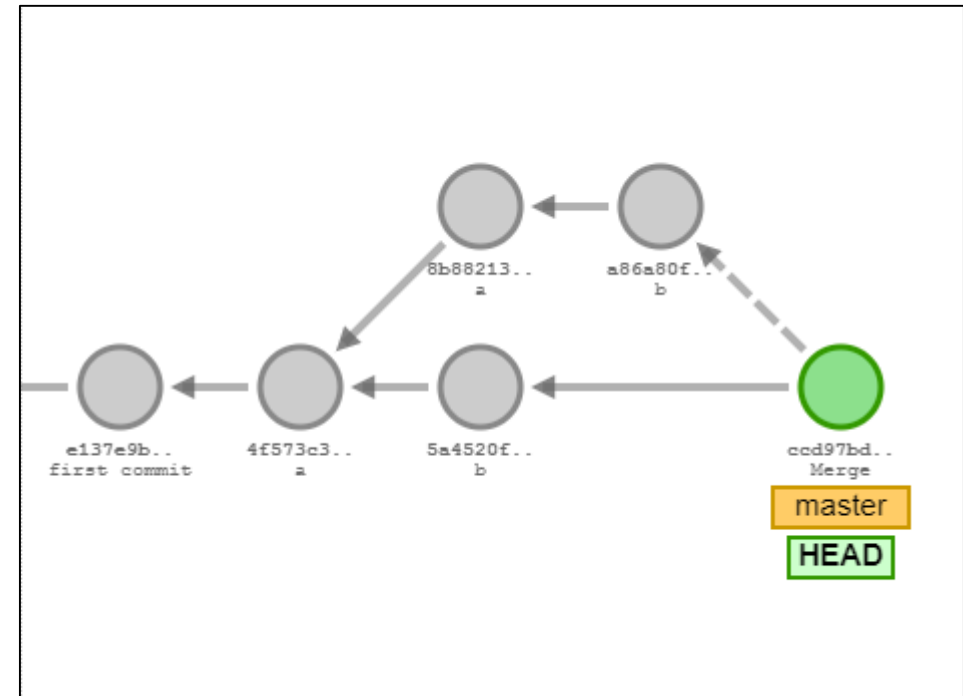
[도전] branch 후 merge하기 – 5분

- <https://git-school.github.io/visualizing-git/>

clear 명령어 = 전체 초기화

redo 명령어 = 실행취소

오른쪽 그림과 같이 만들어보기



[도전] 다음과 같이 만들어보자. – 10분

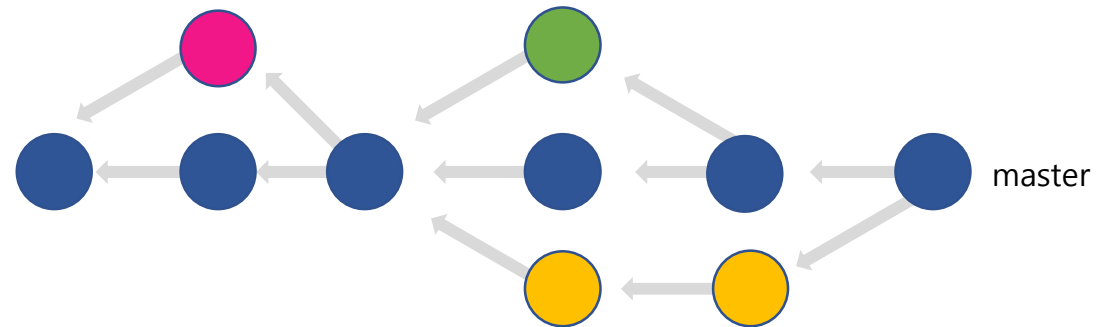
<https://git-school.github.io/visualizing-git/>

규칙

branch명

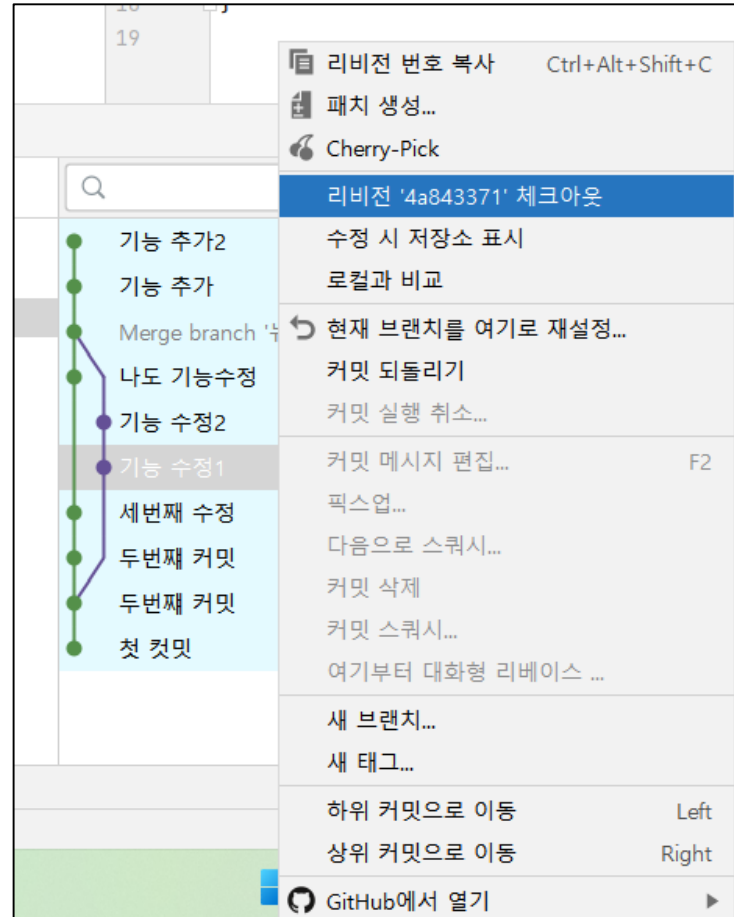
- 다홍 : `feature_payment`
- 연두 : `feature_printer`
- 노랑 : `feature_jsonParser`

merge 후 branch는 삭제한다.

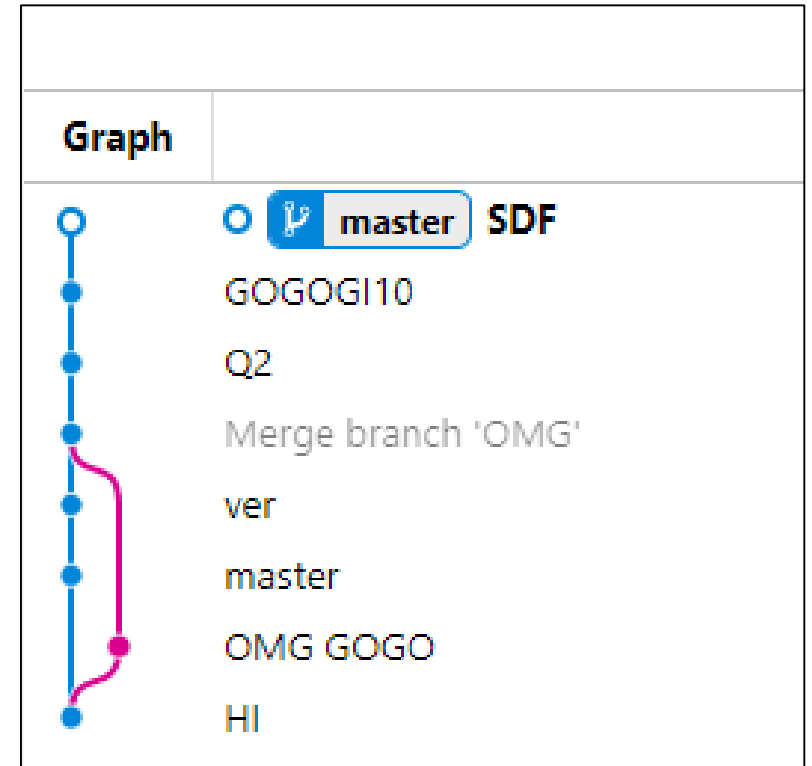


[참고] GUI 에서 Git log view

한줄에
하나의 Commit만
표시하는 형태



IntelliJ Git Log

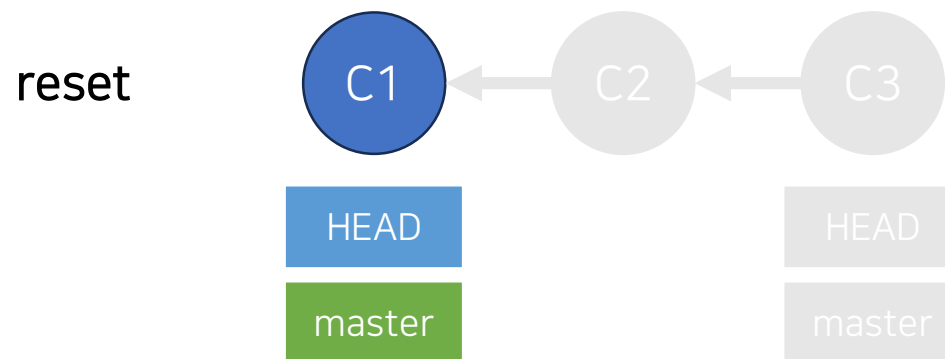
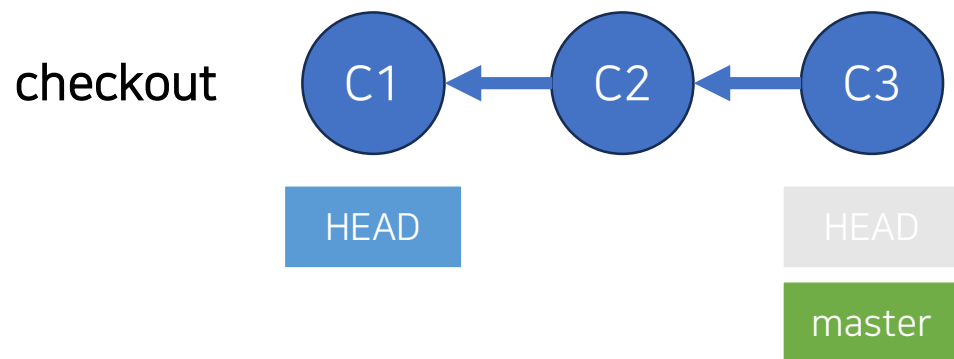


VSCode Plugin

[참고] Git Reset

checkout이 아닌, 정말로 되돌리는 방법 “git reset --hard [Commit Hash 값]”
“checkout” 과 “reset --hard” 은 HEAD 를 이동하고 Work Dir. 가 바뀐다 (staging area 또한 바뀐다)

차이점은 브랜치 포인터로 reset 은 브랜치 포인터가 옮겨지고 checkout 은 그대로 유지된다



Chapter5

git 4단계

github과 push, pull, clone

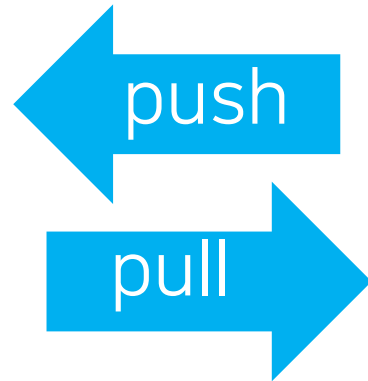
git 과 github

GitHub : 원격 저장소(Remote Repo) 를 제공하는 Web App

Git : Local Repo, GitHub와 연동해서 사용할 수 있다

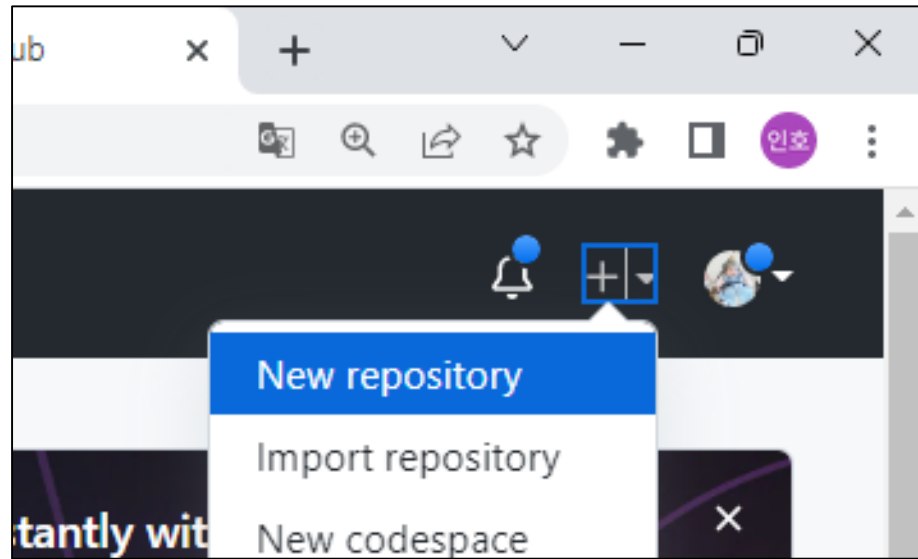


Remote Repo



github 접속 후 Remote Repo 만들기

remote repository 만들기



github 사용 규칙

1. Owner : 실습용 organization 선택

2. Repository 이름

프로젝트명-번호

Description : 이름

3. Internal로 선택

Create a new repository

A repository contains all project files, including the revision history.

Owner * Repository name *

CRA23C3 / PJT-123456

Great repository names are short and memorable. Need inspiration? How about ubiquitous-winner

Description (optional)

한글 성찰

☐ Public
Any logged in user can see this repository. You choose who can commit.

☒ Internal
Samsung Electronics DS enterprise members can see this repository. You choose who can commit.

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

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

README 파일을 생성하지 않는다!

☐ Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

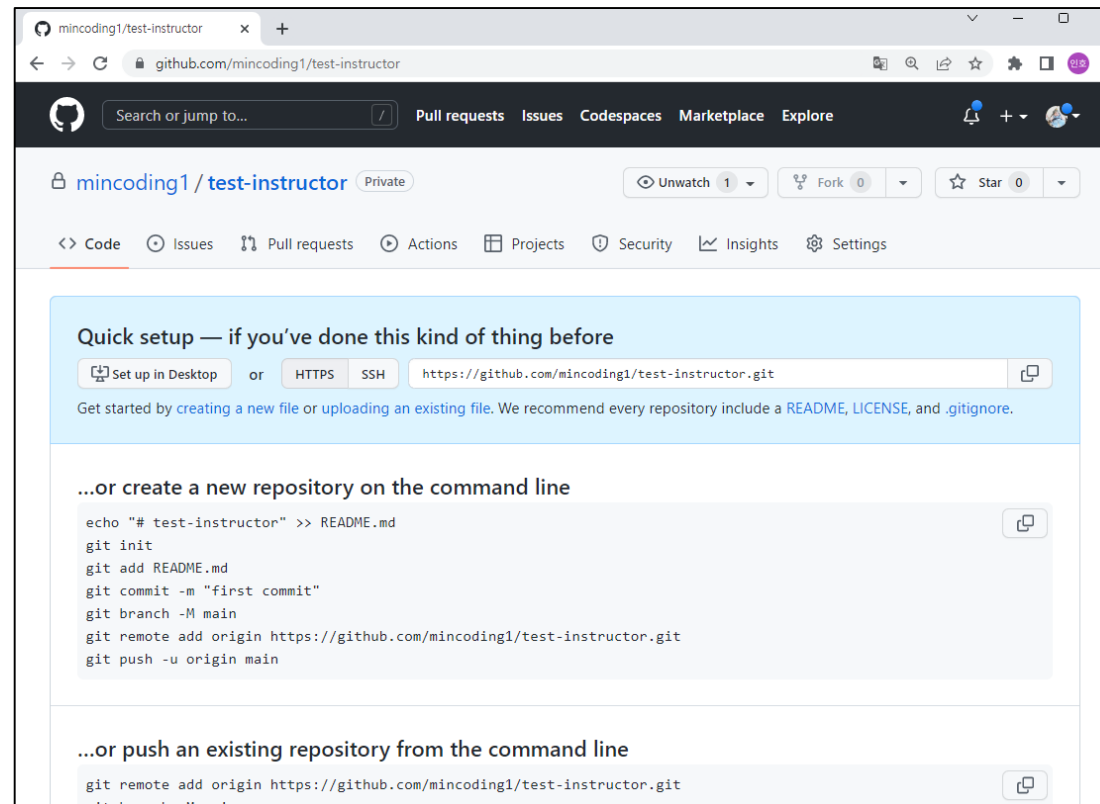
This will set `master` as the default branch. Change the default name in CRA23C3's [settings](#).

Create repository

repo 생성 후 첫 화면

Quick Start

git과 github을 연결하기 위한 방법들이 소개되어 있음

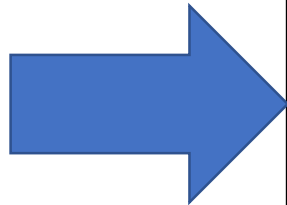


[참고] 연결 방법 - 해석 1

branch -M main

M 옵션 : 현재 branch 이름을 main 으로 변경함

github에서는 Master(주인님)이 노예와 연상되어 **main이라는 이름 권장**



...or create a new repository on the command line

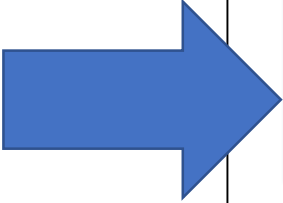
```
echo "# test-instructor" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/mincoding1/test-instructor.git
git push -u origin main
```

[참고] 연결 방법 - 해석 2

git remote add [원격 저장소 이름] [원격 저장소 주소]

origin이라는 키워드는, 원격 저장소를 지칭하는 기본 명칭으로 사용되곤 함.

...or create a new repository on the command line



```
echo "# test-instructor" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/mincoding1/test-instructor.git
git push -u origin main
```

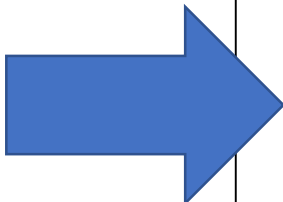
[참고] 연결 방법 - 해석 3

원래 push 명령어 : `git push [origin] [main]`

특정 저장소(origin)에 main branch 내용을 push함

`git push -u origin master`를 한 번 수행 이후에는,
"git push" 라고만 입력해도 push가 진행됨

...or create a new repository on the command line

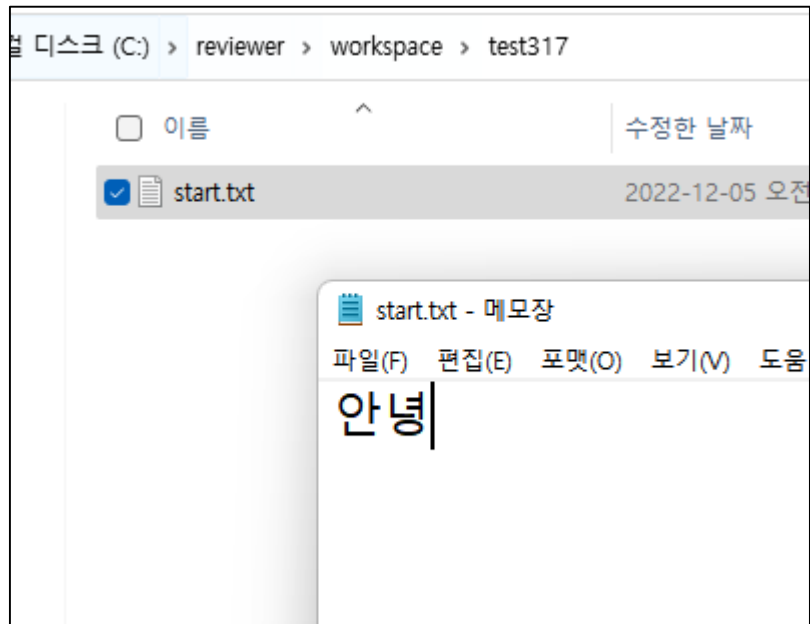


```
echo "# test-instructor" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/mincoding1/test-instructor.git
git push -u origin main
```

작업 폴더 만들기

폴더명 : test317

git init / 첫 Commit까지 완료



```
MINGW64:/c/reviewer/workspace/test317

minco@DESKTOP-T4FIK
$ git init
Initialized empty G

minco@DESKTOP-T4FIK
$
```

git remote add 하기

원격 저장소 추가

git remote add [origin] [주소]

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test317 (master)
$ git remote add origin https://github.com/mincoding1/test-123456.git
```

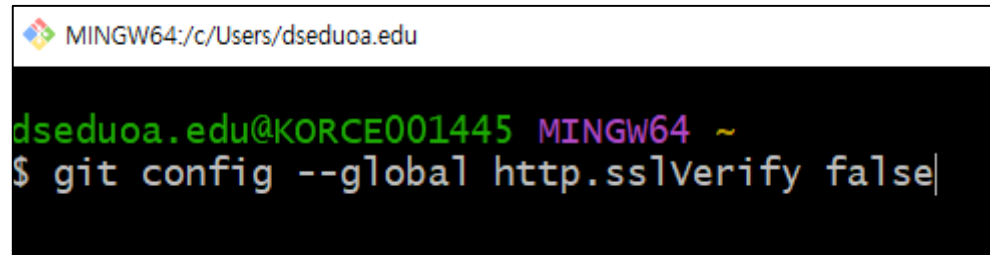
원격 저장소 리스트 보기

git remote

```
minco@DESKTOP-T4FIKKV
$ git remote
origin
```

[중요] git push 를 위한 사전 세팅 1

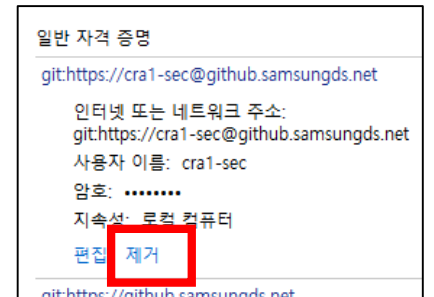
- 사업장에서 SSL 인증이 막혀있기에,
SSL 인증 False로 설정하기



```
MINGW64:/c/Users/dseduoa.edu  
dseduoa.edu@KORCE001445 MINGW64 ~  
$ git config --global http.sslVerify false|
```


[중요] git push 를 위한 사전 세팅 2

- PC에 저장된 git 자동 로그인 ID / Password 삭제를 위해 자격 증명에서 정보 지우고 다시 시도해야 함



[Trouble Shooting] push 도중 403 에러 발생시

- 403 에러 발생시 다음과 같이 해보기

1. 자격증명을 모두 삭제한다.
2. 다시 `git push origin master`를 수행한다. → 30초 후, 로그인 창이 뜬다
3. 새로운 토큰을 발급받아서, 로그인을 시도한다.

```
Admin@DESKTOP-SLNDDHO MINGW64 /d/Web/phanvanlinh.github.io/phanv
(master)
$ git push origin master
remote: Permission to PhanVanLinh/phanvanlinh.github.io.git denied
fatal: unable to access 'https://github.com/PhanVanLinh/phanvan
t/': The requested URL returned error: 403
```

git push

Branch 단위로,
원격 저장소에 소스코드 & 이력을 업로드 함

예시

```
git push origin master
```

```
git push origin feature/printer
```

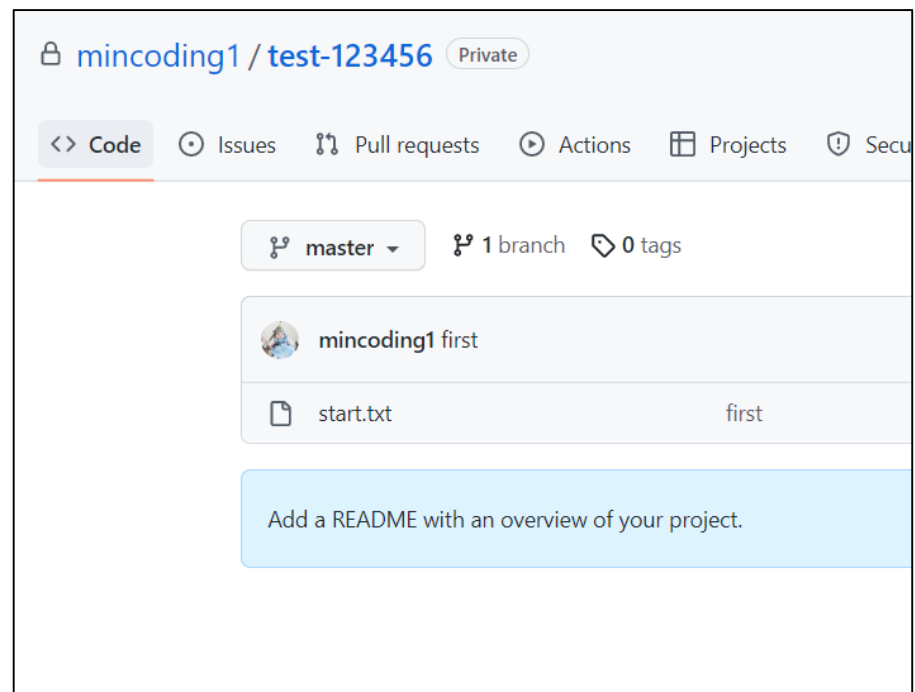
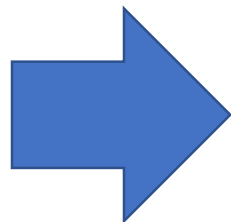
git push 해보기

git push origin master

1. 10초간 대기
2. 윈도우 로그인 창이 뜨면, 로그인을 해주어야 함

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test317 (master)
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 213 bytes | 213.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/mincoding1/test-123456.git
 * [new branch]      master -> master
```

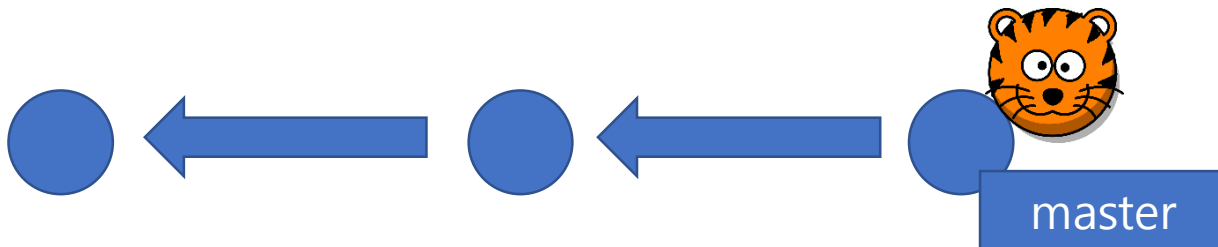
로그인 성공 후, 결과 화면



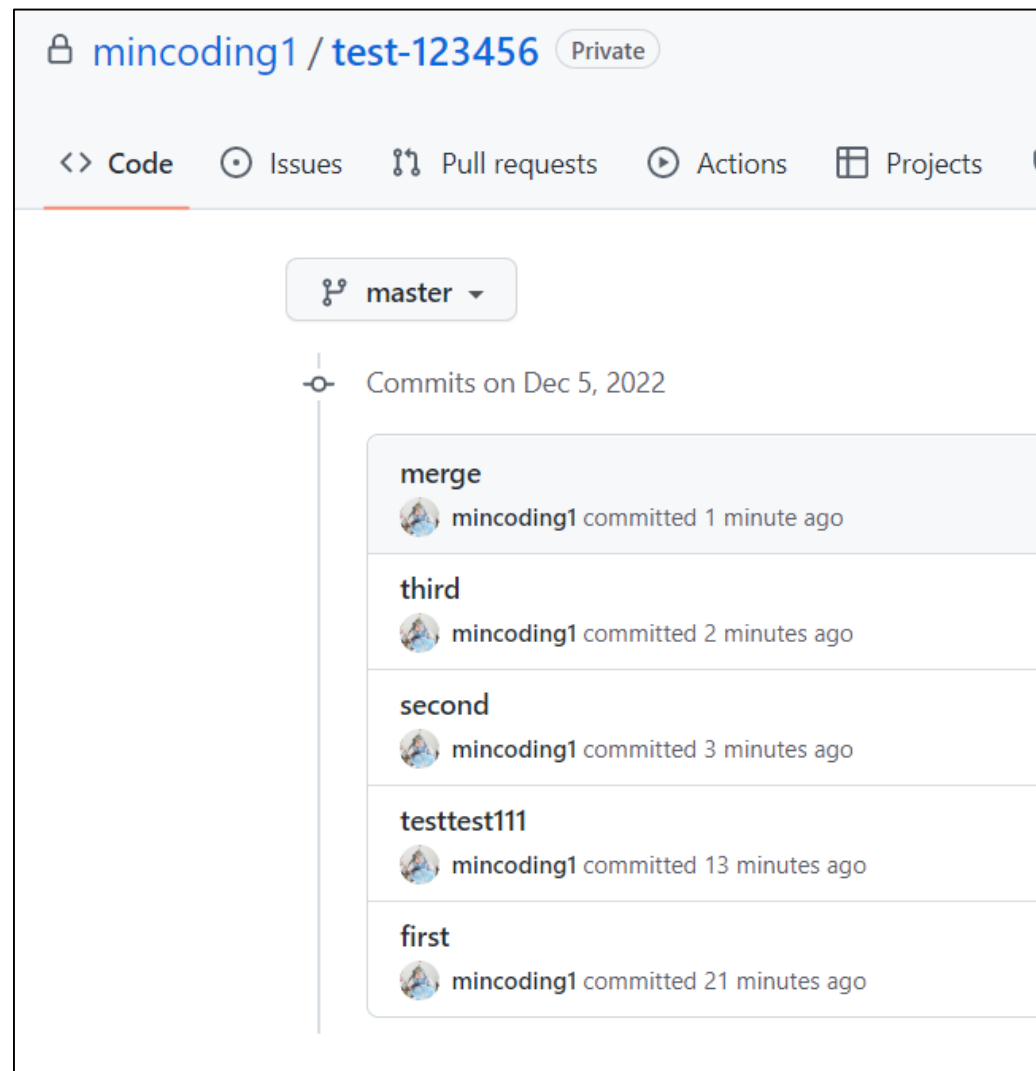
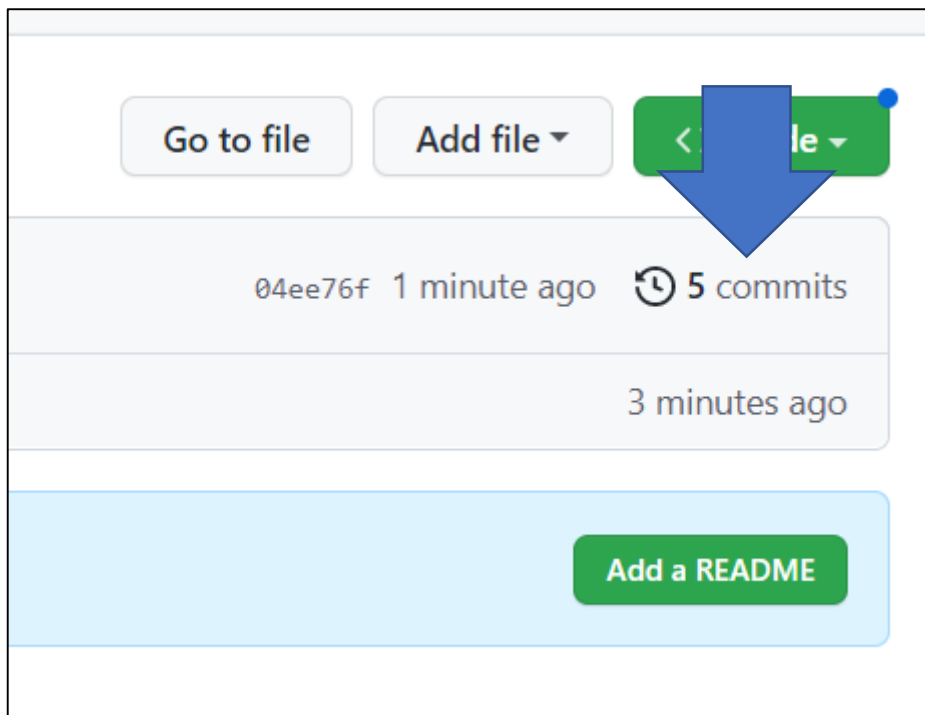
Commit 후 push 해보기

Local에 여러번 Commit 후
Remote에 Push 해보기

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/t
$ git push origin master
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (7/7), 573 bytes | 573.00 KiB/s
Total 7 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/mincoding1/test-123456.git
a4b420c..04ee76f master -> master
```



Commit Log 확인 가능



원격 저장소의 변경내용을
내 저장소에 가져오기

협업시 pull 하는 이유

타인이 작업 후, push를 한 경우가 있기 때문

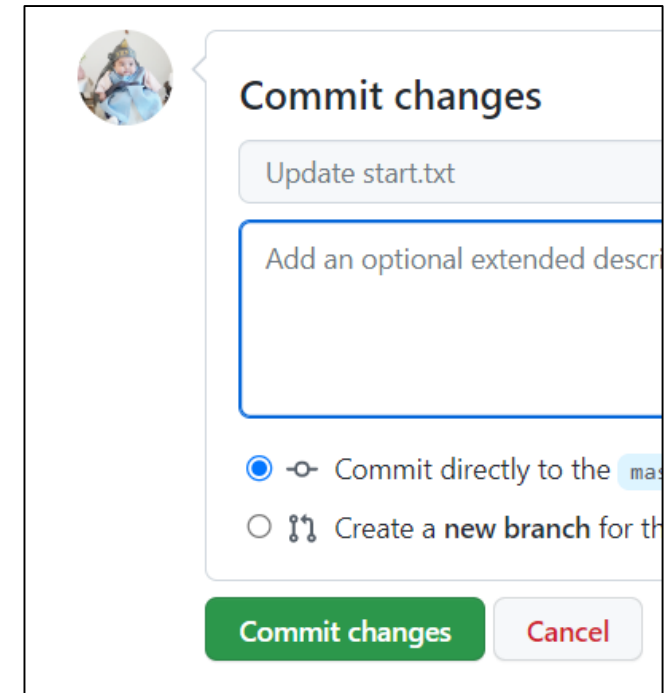
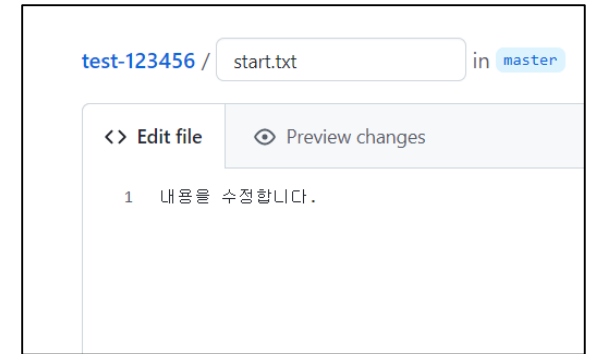
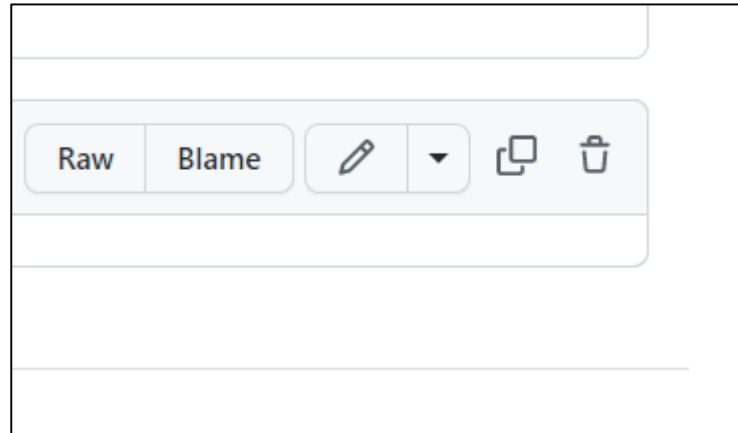
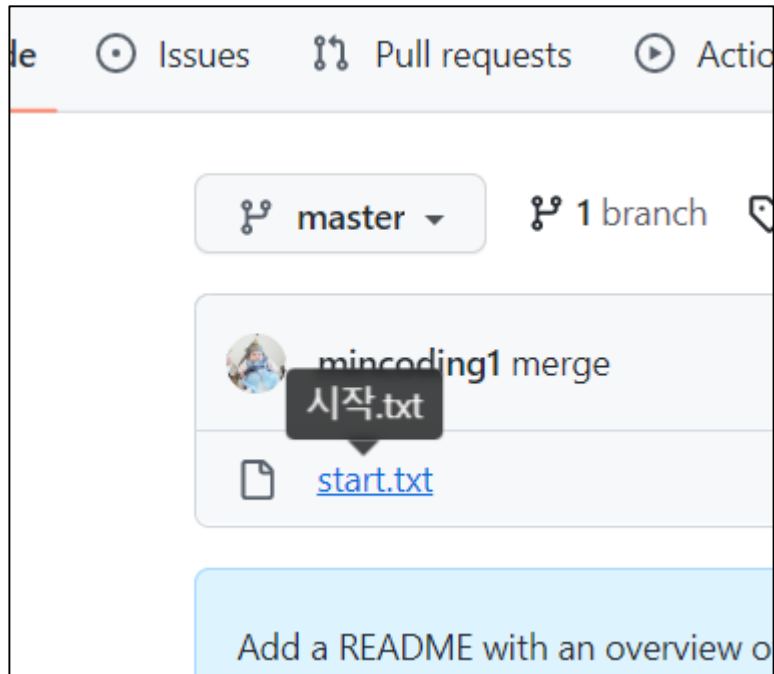
개발을 시작하기 전, pull을 해야
충돌이 없거나, 충돌로 인한 Merge 작업 양이 줄어든다.

```
minco@DESKTOP-T4FIKKV MINGW
$ git pull
Already up to date.

minco@DESKTOP-T4FIKKV MINGW
$
```

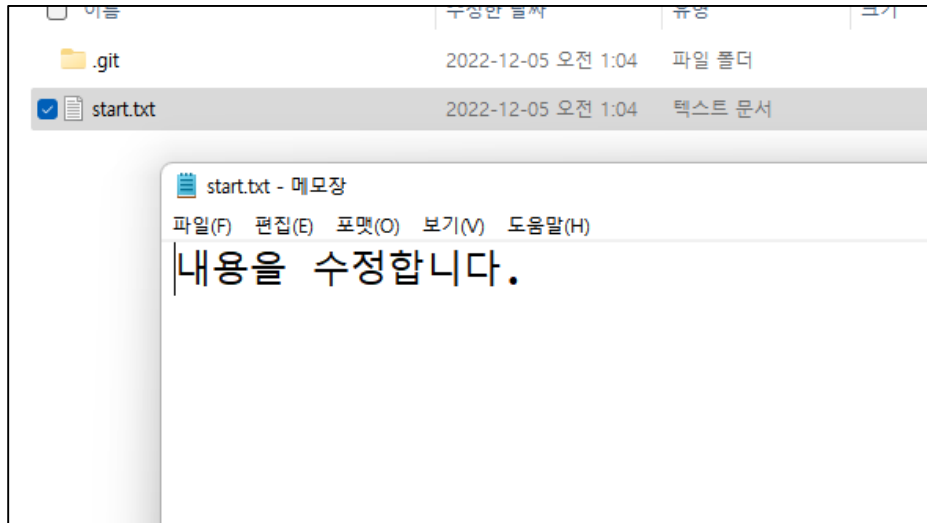
Remote에서 직접 수정하기

파일을 선택 후
연필 버튼을 누르고 내용을 수정, Commit 한다.



git pull 하기

pull하여 내용변경 확인 후,
git log를 확인한다.

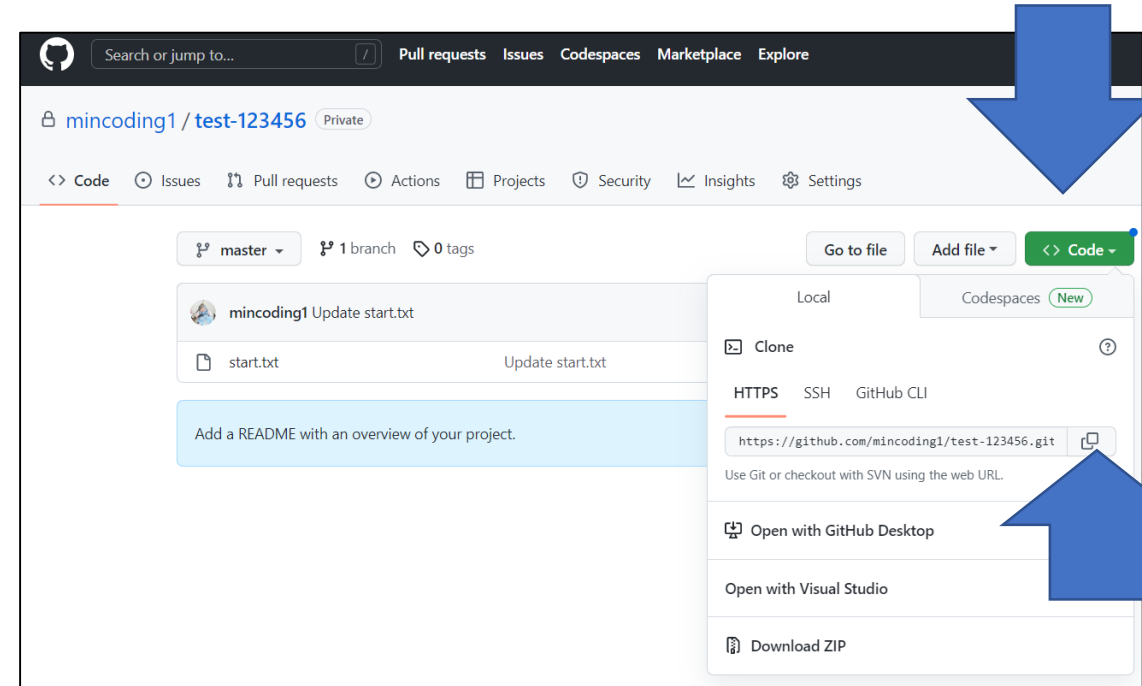


```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/test317 (master)
$ git log --oneline
ebff437 (HEAD -> master, origin/master) Update start.txt
04ee76f merge
e423a4b third
bd466da second
```

git clone

원격 저장소의 내용들을
로컬 저장소를 생성 후 복사를 해주는 명령어

협업 개발을 시작하는 중요한 명령어

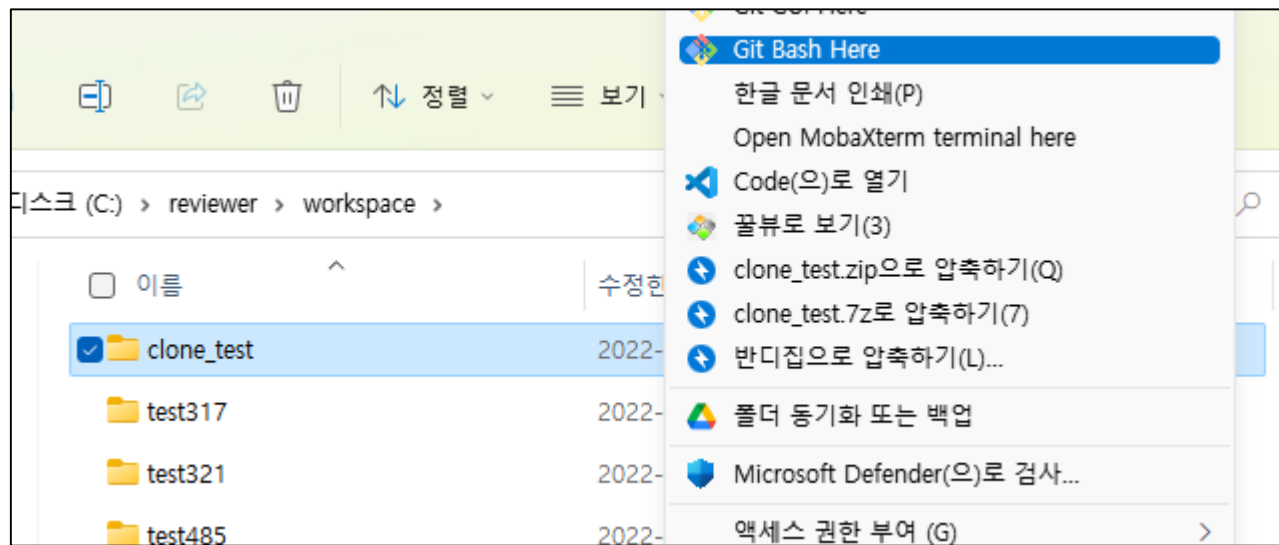


새로운 폴더 생성

clone_test 폴더를 생성한다.

해당 폴더 내부에

원격 Repo를 Local Repo로 복제 할 것이다.



git clone 하기

repo 이름과 동일한 폴더가 생성 / 내용이 복제된다.

.git 파일 자동 생성됨

remote와 연결되어 있음

MINGW64:/c/reviewer/workspace/clone_test

```
minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer/workspace/clone_test
$ git clone https://github.com/mincoding1/test-123456.git
Cloning into 'test-123456'...
```

로컬 디스크 (C:) > reviewer > workspace > clone_test >

☐ 이름

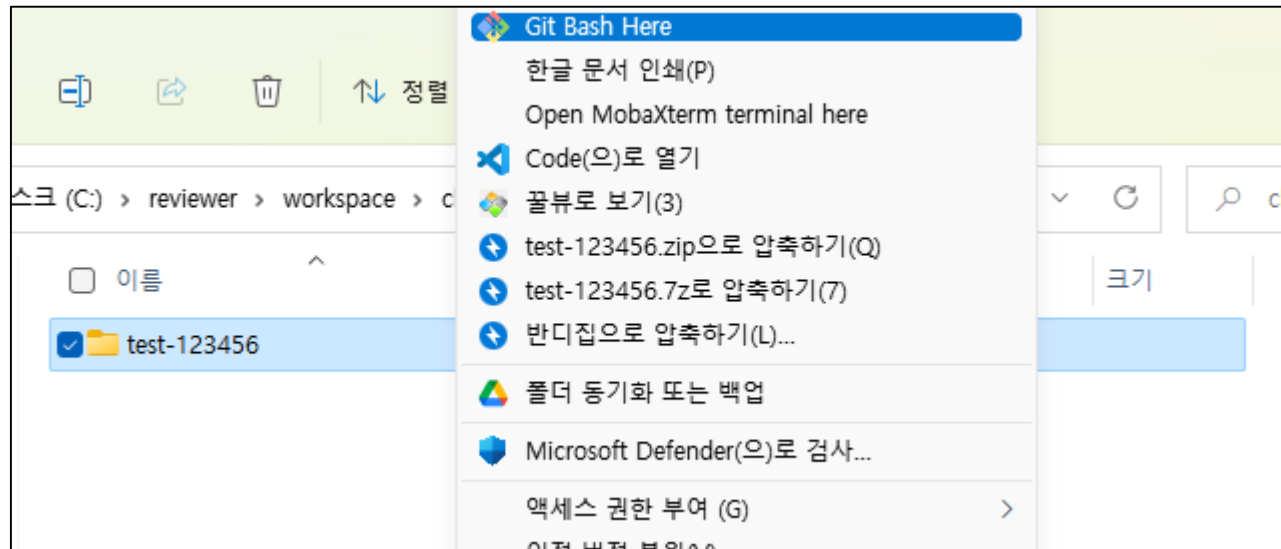
수정한 날짜

test-123456

2022-12-05

git log 확인해보기

bash로 들어가서
log를 확인해본다.



```
MINGW64:/c/reviewer/workspace/clone_test/test-123456

minco@DESKTOP-T4FIKKV MINGW64 /c/reviewer)
$ git log --oneline
ebff437 (HEAD -> master, origin/master)
04ee76f merge
e423a4b third
bd466da second
a4b420c testtest111
4dd3361 first
```

[도전] git repo 만들어 push / pull 하기 – 10분

Local Repo

새롭게 Working Dir.을 만든다.
hello.txt 파일을 생성 후 commit 한다.
파일 내용을 수정 후 commit 한다.
원격 repo에 push 한다.

Remote Repo

Commit Log를 확인한다.
파일내용을 Github에서 수정한다.
Local Repo에서 pull을 한다.

Clone Repo

새로운 폴더를 하나 생성한다.
만들어진 Repo를 Clone 해본다.
파일 내용을 수정 후, Commit / push 해본다.

새로운 git repo를 생성 후 시작
repo 명 : test2-번호
description : 이름

지금까지 한 내용

1단계 : Git Init부터 Commit까지

2단계 : checkout과 git log view

3단계 : git branch와 merge

4단계 : github과 push, pull, clone

남은 Git 내용

Pull Request를 이용한 Code Review



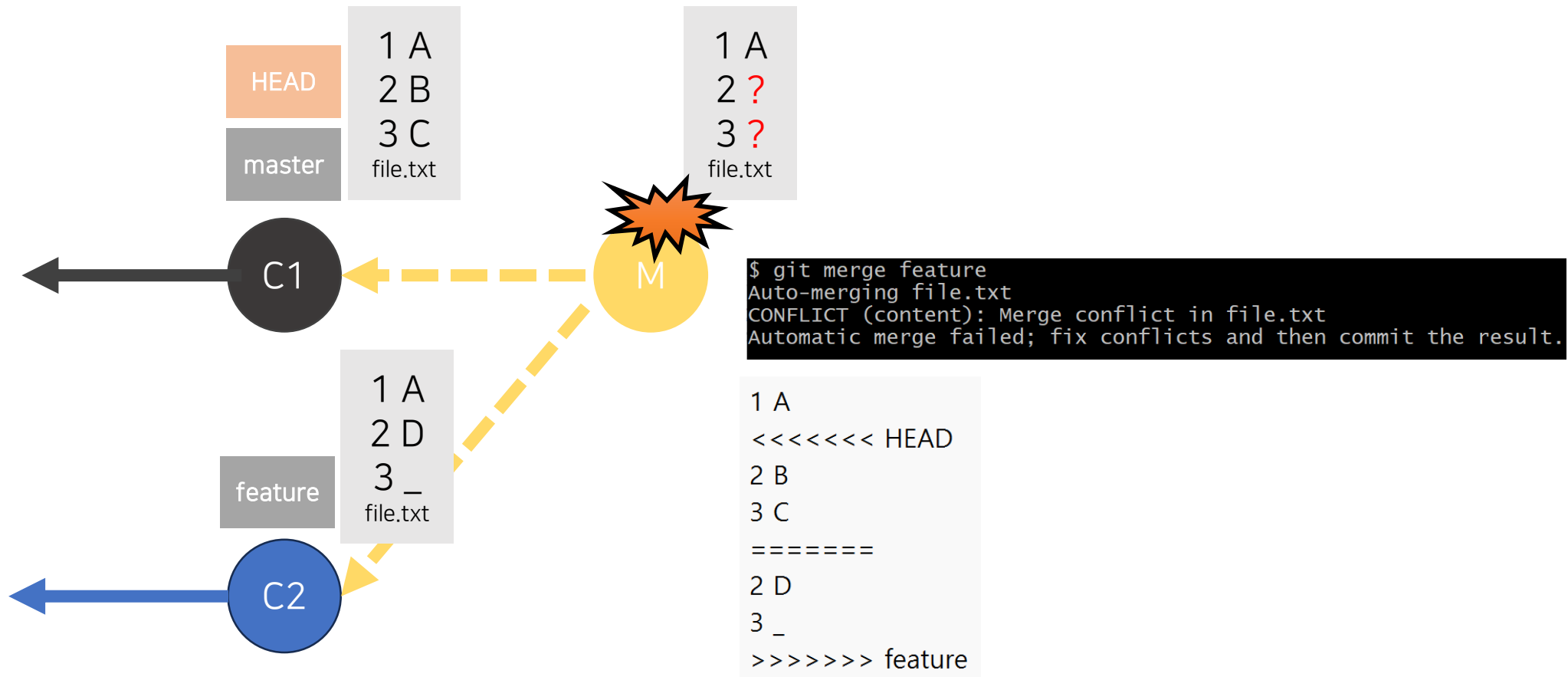
Git conflict 해결

merge 를 하는 과정에서 발생하는 문제들을 해결한다

Merge 할 때 충돌

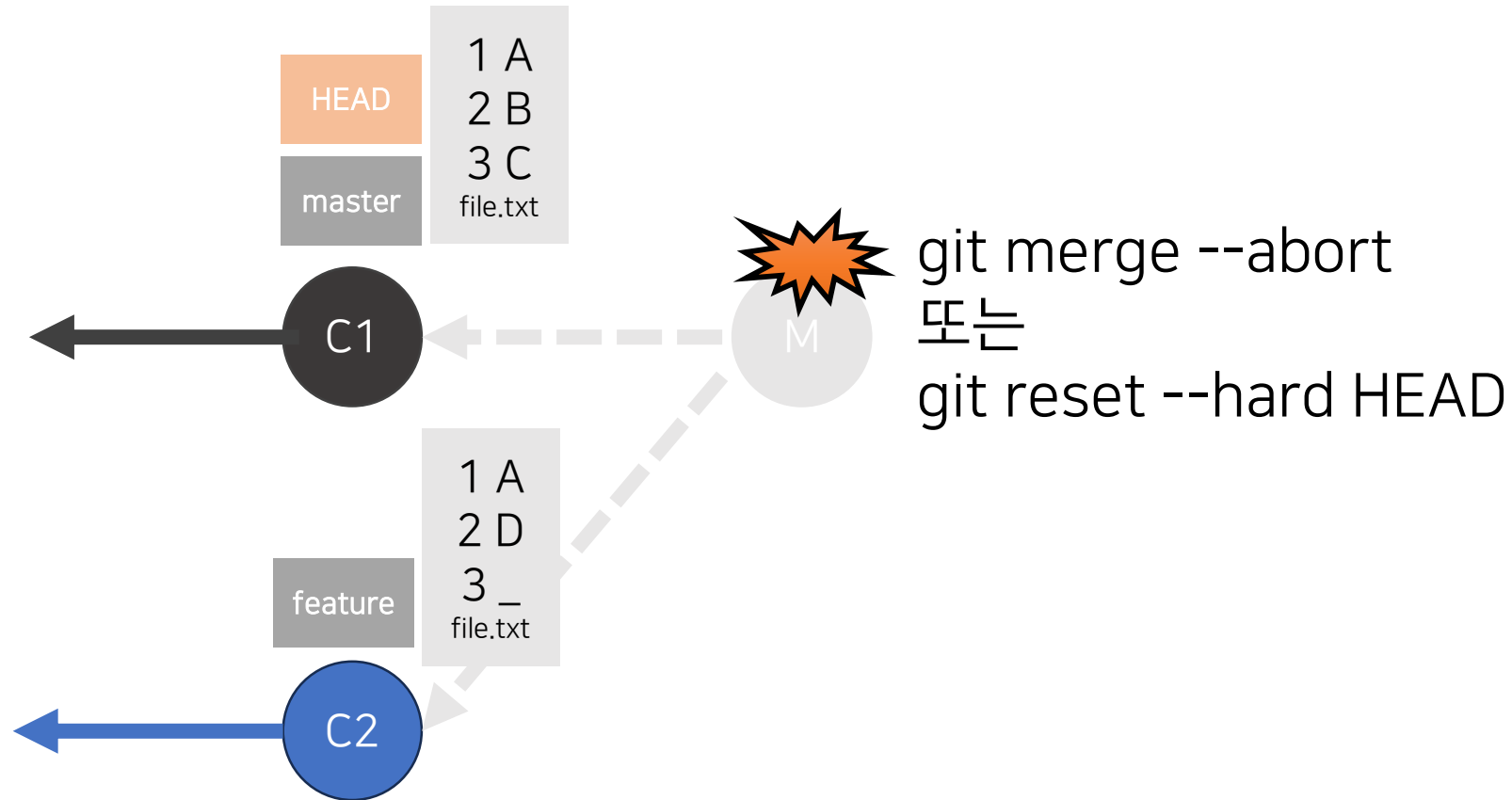
두 브랜치를 병합하는 커밋을 만들 때, 같은 파일의 변경사항에 대해서 충돌이 일어날 수 있다.

충돌이 발생하면 충돌을 해결한 뒤, merge 커밋을 다시 만들어야 한다.



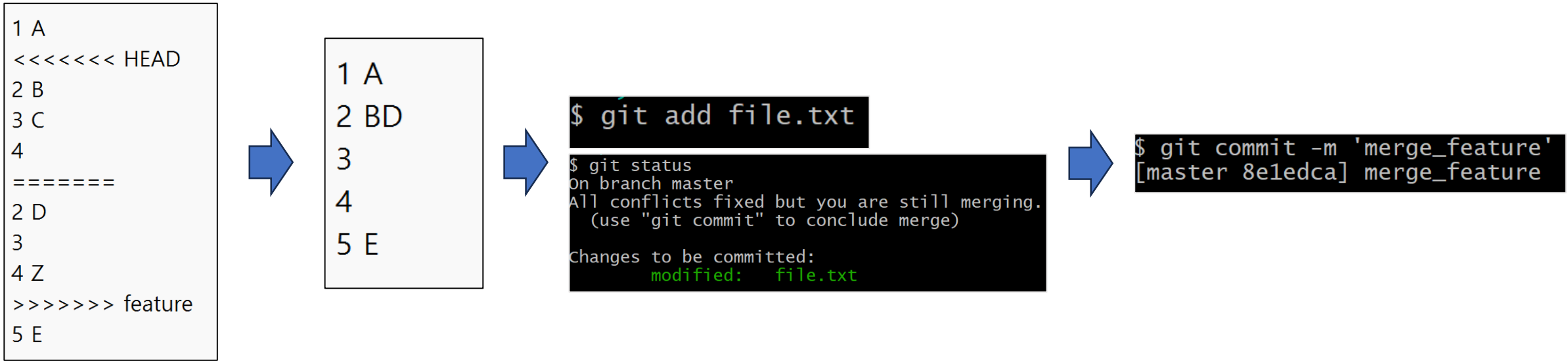
가장 단순한 해법

애초에 Merge 하지 않은 상태로 돌아간 뒤 서로 충돌 없는 커밋을 만들고 merge하는 것도 하나의 방법이다.



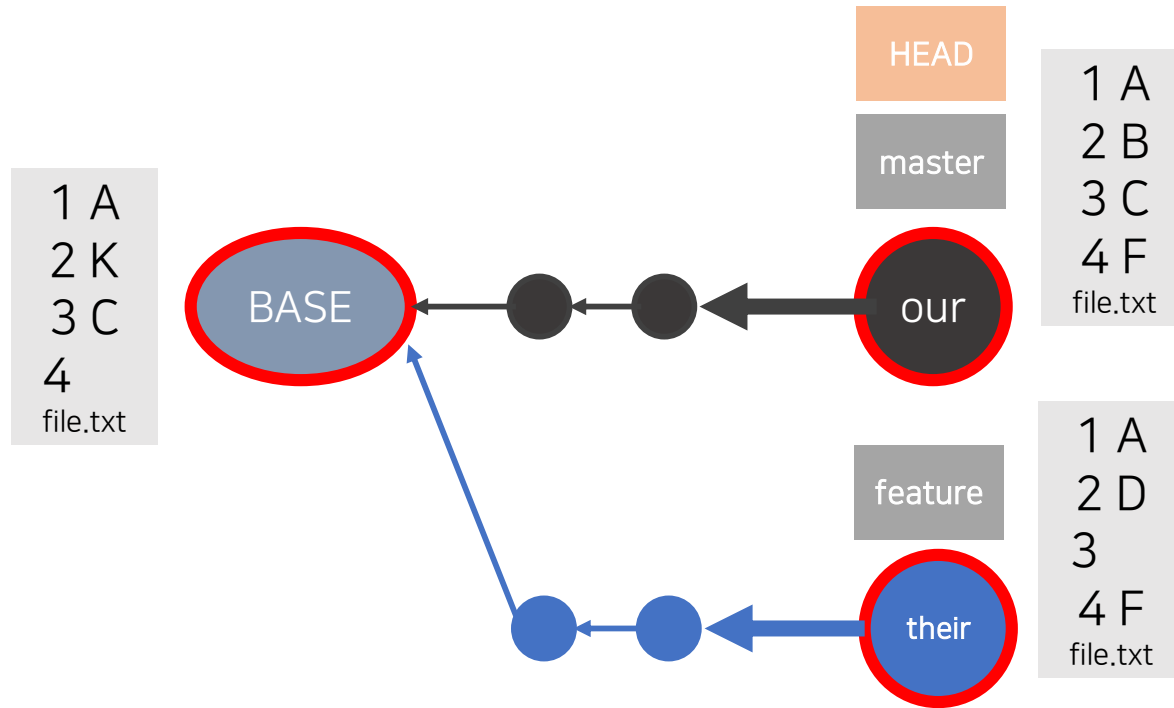
충돌 직접 해결

충돌이 발생한 파일을 연 뒤, 직접 충돌을 해결 후 다시 add, commit 하여 충돌이 해결된 merge 커밋을 만든다.



3-Way Merge

기존에 merge를 할 때 이용하는 방식을 Three-Way Merge 라고 한다
Three-Way merge 는 3개의 커밋을 가지고 merge 커밋을 만들게 되는데,
각 커밋을 Base, Our, Their 라고 해보자. 여기서 Base 는 두 브랜치의 공통조상
이다



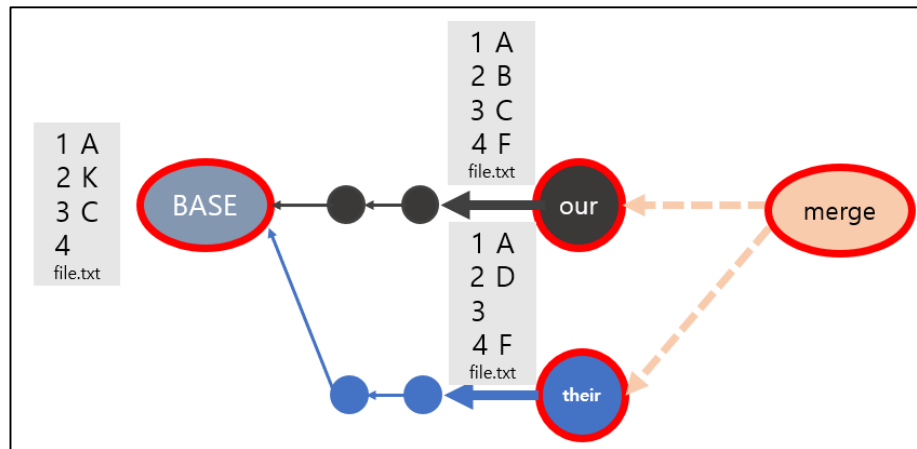
기본적으로 두 브랜치를 merge 하는 경우, Base 는 공통조상인 커밋이 된다.

3-Way Merge 통합 방법

Base 는 기준으로 해당 기준으로 Our version의 변경, Their version 의 변경이 발생했다.

Base를 기준삼아, Our 와 Their 의 각 변경들을 적용해본다.

Our version	Base -> merge commit	Their version
A	A -> ①	A
B	K -> ②	D
C	C -> ③	
F	-> ④	F



정답:
A, 충돌, , F

[Quiz] 어떤 코드에서 충돌이 발생할까?

어떤 부분에서 충돌이 발생하고 충돌이 없는 부분은 어떻게 될지 맞춰본다.

1.

Our version	Base version	Their version
X	X	X
Y	Y	B
A	Z	Z
W	W	V

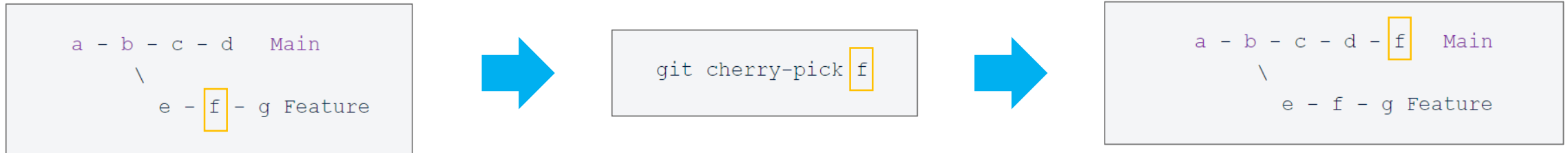
2.

Our version	Base version	Their version
P	P	P
A	Q	Q
R	R	B
S	S	
V	T	T
U	U	X

cherry-pick이란?

특정 커밋에서 변경사항을 가져오는 기능이다.

- 사전적 의미의 "체리픽" : 가장 좋은 것을 선택하는 것

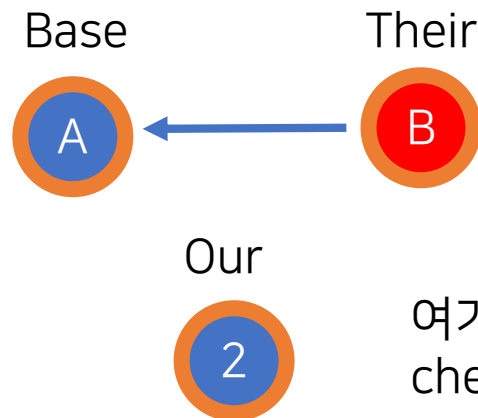
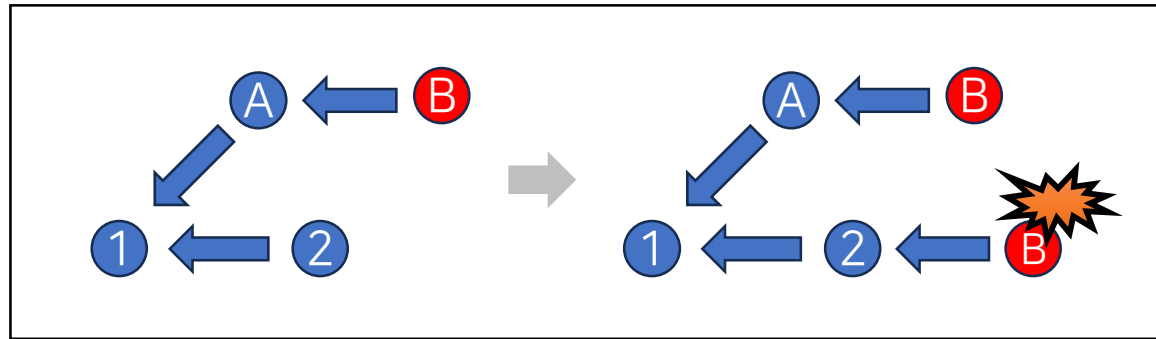


e에서 f 커밋으로의 변경사항을 Main 브랜치에 적용할 수 있다

cherry-pick 에서 충돌

cherry-pick을 적용하려고 할때 충돌이 발생할 수 있다.

cherry-pick 또한 기본적으로 3-Way Merge 방식으로 동작한다.



여기서 Base는 공통조상이 아닌,
cherry-pick 하고자하는 커밋의 부모 커밋이다.

Commit 실수를 했을 때,

두 가지 해결 방법이 있다.

1. Reset
2. Revert



Reset과 Revert 차이

두 가지 해결 방법이 있다.

1. Reset : 이력을 남기지 않는다.
2. Revert : 이력을 남기는 커밋을 만든다.



Reset
(이력삭제)



Revert
(이력남김)

언제 사용하면 될까?

두 가지 해결 방법이 있다.

1. Reset : 이력을 남기지 않는다. ← Local에서 작업할 때 혼자 증거를 없앤다.
2. Revert : 이력을 남긴다. ← 실수 Commit을 Push를 해버릴 경우, 증거를 없애면 안된다.



Reset
(이력삭제 : Local 전용)



Revert
(이력남김 : Remote 전용)

Commit 3회 진행 후, Revert 하기

1. git revert "취소할 Commit ID" 입력
2. vi 에디터가 뜨면 그대로 저장 후 종료 → :wq!

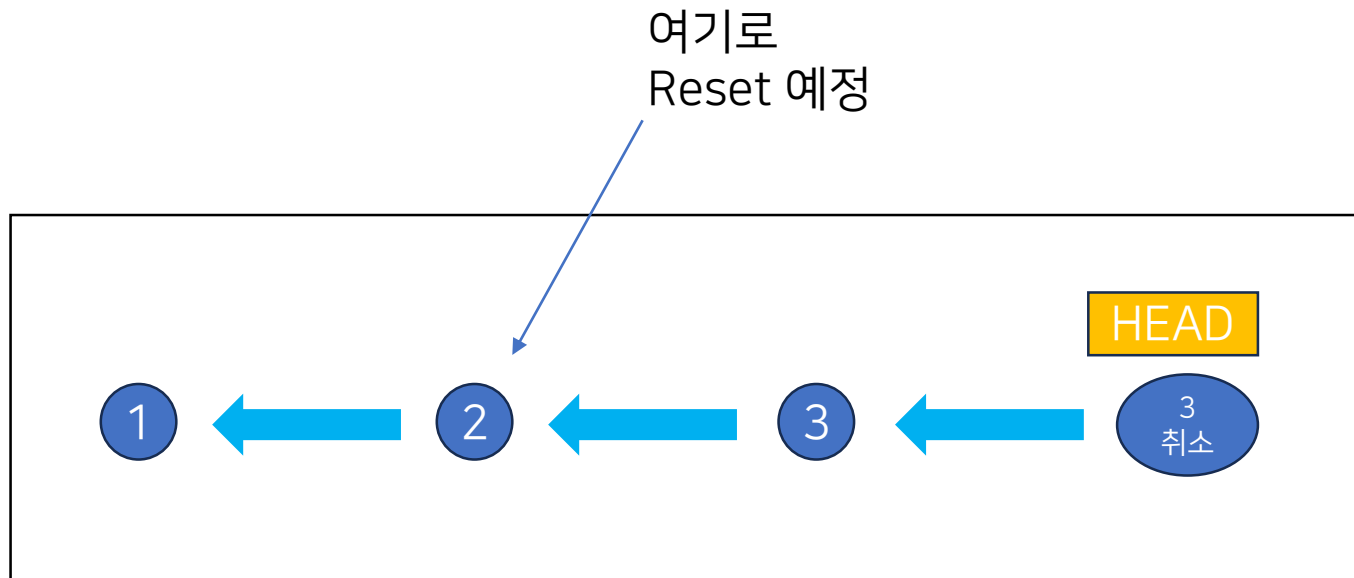
[illegible]

Revert (이력남김)

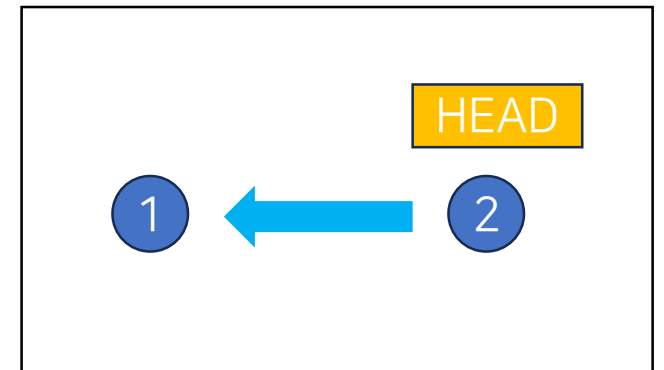
Reset 실습

기존 실습 내용에서 2번까지 Reset하기

`git reset --hard` "Commit ID" 입력



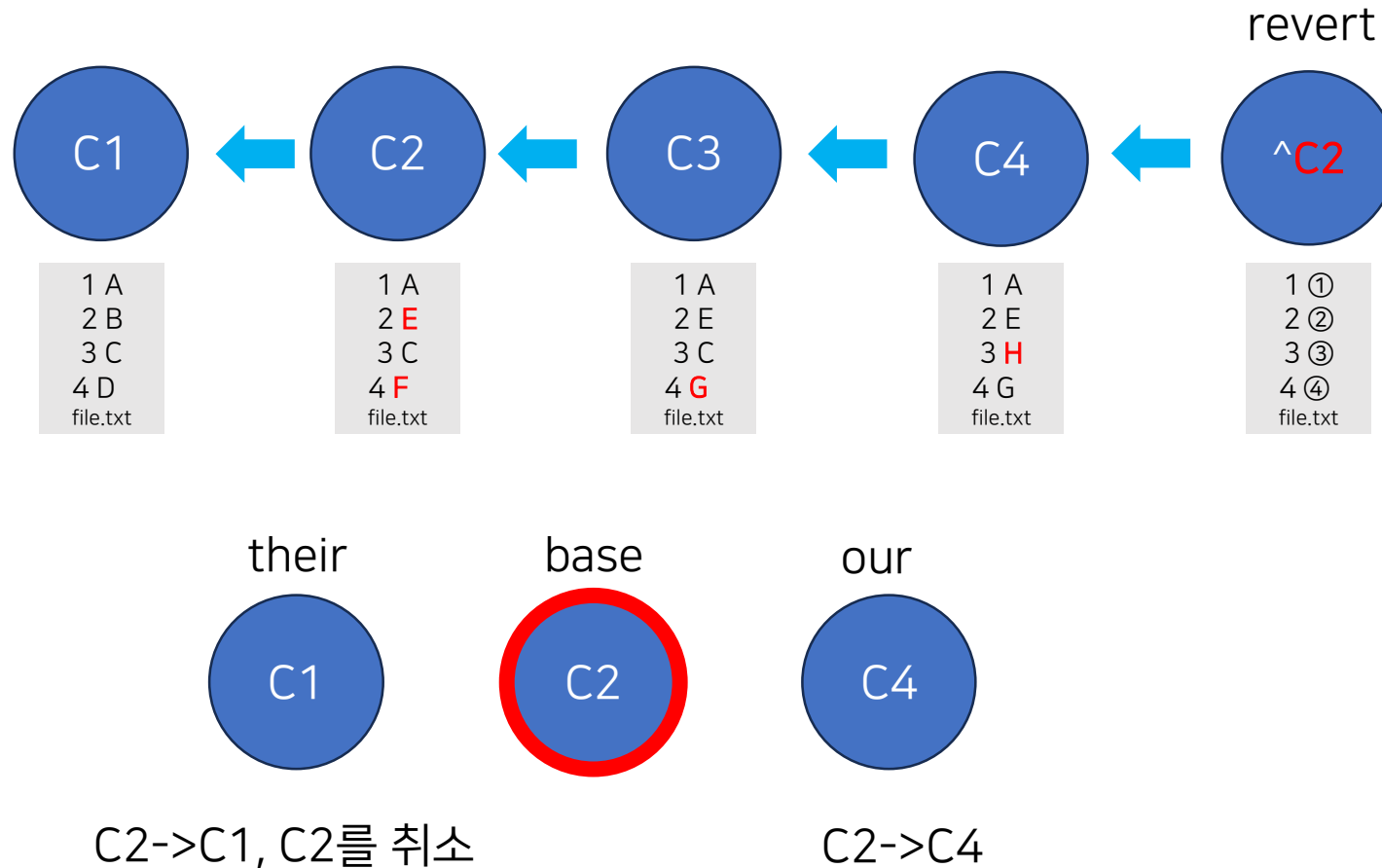
Reset 전



Reset 후

Revert 할 때 충돌

revert는 바로 직전의 커밋을 취소하는 경우 충돌이 발생하지는 않는다.
중간에 다른 커밋이 있는 경우 충돌 발생할 수 있다.



Chapter6

Markdown

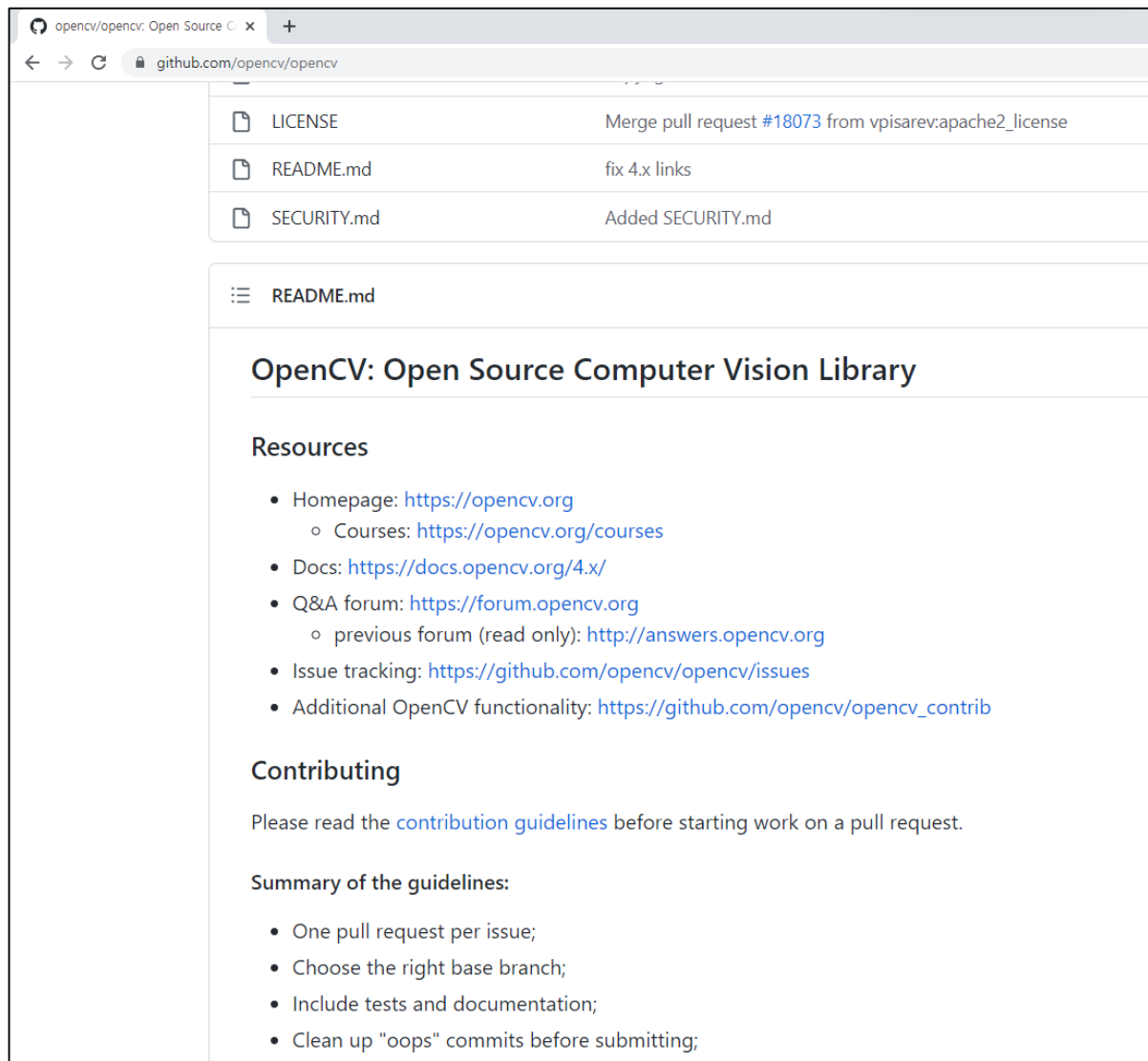
코드 리뷰어를 위한 쉬어가는 페이지

Markdown 파일

Text만으로

서식문서를 만드는 문법

html로 자동 변환 후 출력됨



Github에서 Markdown으로
Readme 문서를 제공하곤 한다.


새로운 파일 생성


readme.md 파일을 만들어보자.

🔗 master ▾

🔗 1 branch

🏷️ 0 tags

 mincoding1 Update start.txt

 start.txt

Update start.txt

Go to file

Add file ▾

Create new file

Upload files

<> Code ▾

🕒 6 commits

32 minutes ago

Add a README with an overview of your project.

Add a README

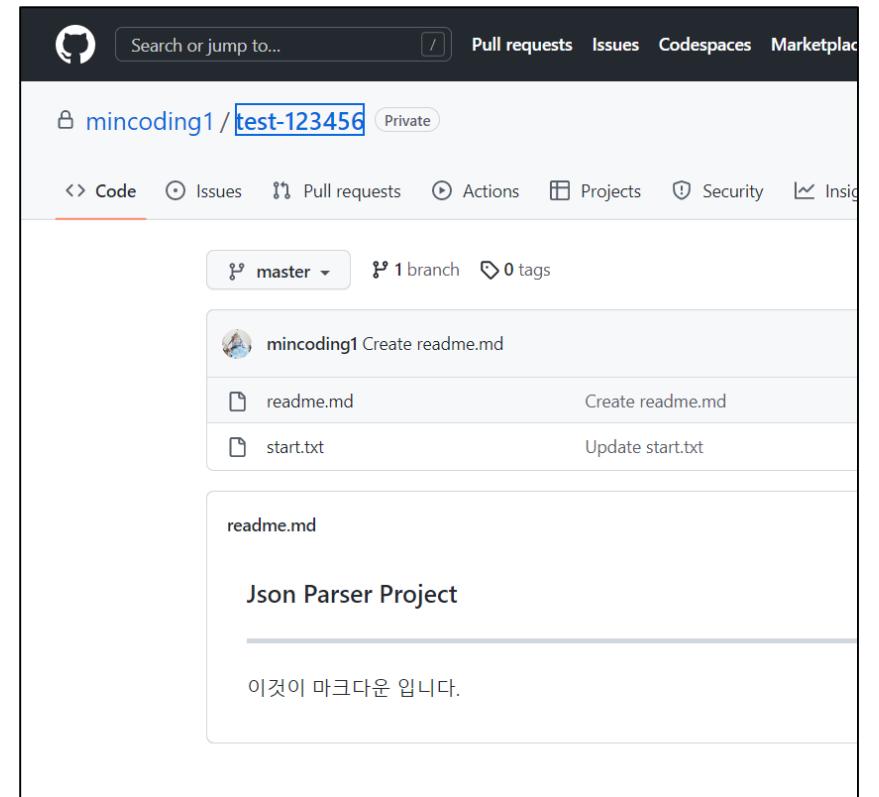
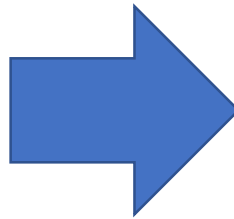
Commit 해보기

readme.md 파일 작성시
repo에서 첫 번째로 보여주는 페이지가 된다.

<> Edit new file

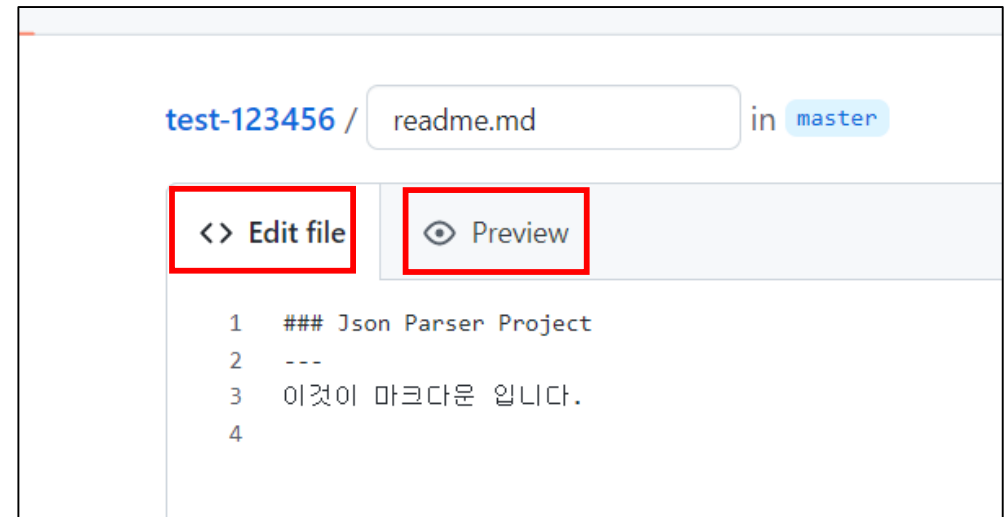
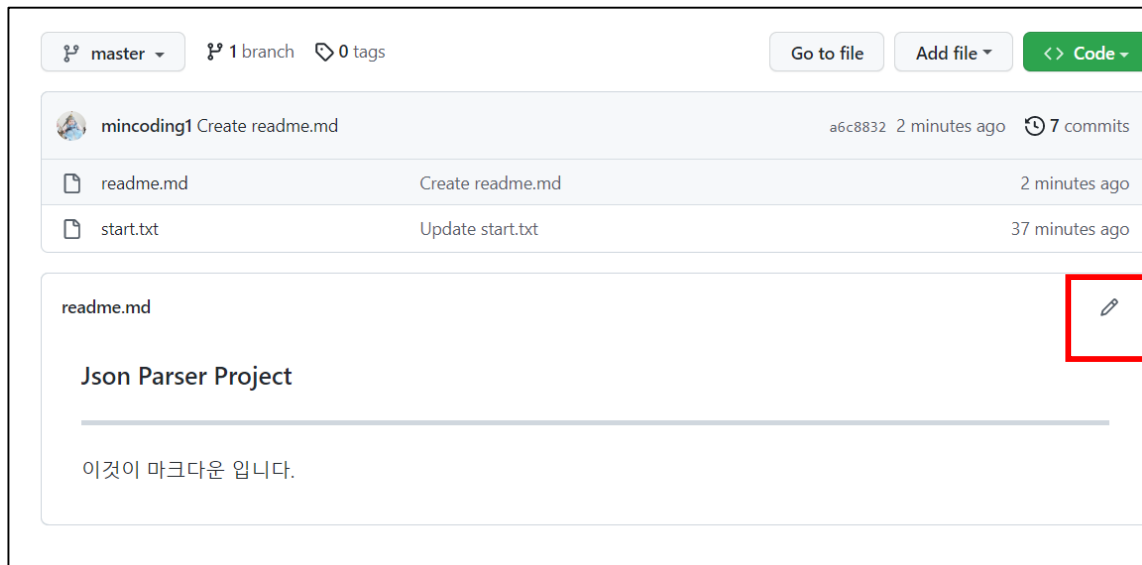
Preview

```
1  ### Json Parser Project
2  ---
3  이것이 마크다운 입니다.
```



수정하기

연필버튼을 눌러,
Preview로 수정화면 확인 가능



[중요] Sample 1

<> Edit file

```
1  # H1
2  가장 큰 제목입니다. <br/>
3
4  ## H2
5  그 다음 큰 제목입니다 <br/>
6
7  ### H3
8  그 다음 큰 제목입니다 <br/>
9
10 그리고 **굵은 글씨**는 이렇게 합니다.
11
12 엔터는 줄바꿈이 되지 않습니다.<br/>
13 엔터를 하는 방법은 다음과 같습니다. <br/>
14 1. 엔터 두 번 누르기
15 2 br/ 태그 사용하기
```

H1

가장 큰 제목입니다.

H2

그 다음 큰 제목입니다

H3

그 다음 큰 제목입니다

그리고 **굵은 글씨**는 이렇게 합니다.

엔터는 줄바꿈이 되지 않습니다.

엔터를 하는 방법은 다음과 같습니다.

1. 엔터 두 번 누르기
- 2 br/ 태그 사용하기

[중요] Sample 2

pixabay.com에서 원하는 이미지 링크를 복사하자
그리고 다음과 같이 입력한다.

![제목](https://링크.jpg)

<> Edit file Preview

```
1  ## 좋아하는 사이트
2  다음은 제가 가장 즐겨찾는 곳이지요
3  - 구글 : [google](https://google.com)
4  - 네이버 : [naver](https://naver.com)
5
6
7  ## 좋아하는 사진
8  제가 좋아하는 사진입니다.
9  ![미소](https://user-images.githubusercontent.com/74457149/205504537-2a3794e3-82e8-437e-9d0c-bb71c27c2de2.jpg)
10
```

<> Edit file Preview

좋아하는 사이트

다음은 제가 가장 즐겨찾는 곳이지요

- 구글 : [google](https://google.com)
- 네이버 : [naver](https://naver.com)

좋아하는 사진



[중요] Sample 3

```cpp

소스코드

```

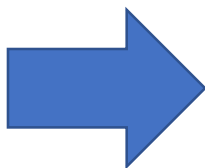
```
```cpp
#include <iostream>
#include <iomanip>
using namespace std;

int main() {

 ios_base::sync_with_stdio(false);
 cin.tie();
 cout.tie();

 double a = 312.14623663125;
 cout << fixed << setprecision(2) << a; // iomanip 필요
 //cout << round(a * 100) / 100.0;

}
```
```



```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {

    ios_base::sync_with_stdio(false);
    cin.tie();
    cout.tie();

    double a = 312.14623663125;
    cout << fixed << setprecision(2) << a; // iomanip 필요
    //cout << round(a * 100) / 100.0;

}
```

표 추가하기

| <> Edit file | Preview |
|--------------|----------------|
| 1 | 제목 내용 설명 |
| 2 | ----- --- --- |
| 3 | 테스트1 테스트2 테스트3 |
| 4 | 테스트1 테스트2 테스트3 |
| 5 | 테스트1 테스트2 테스트3 |

<> Edit file

Preview

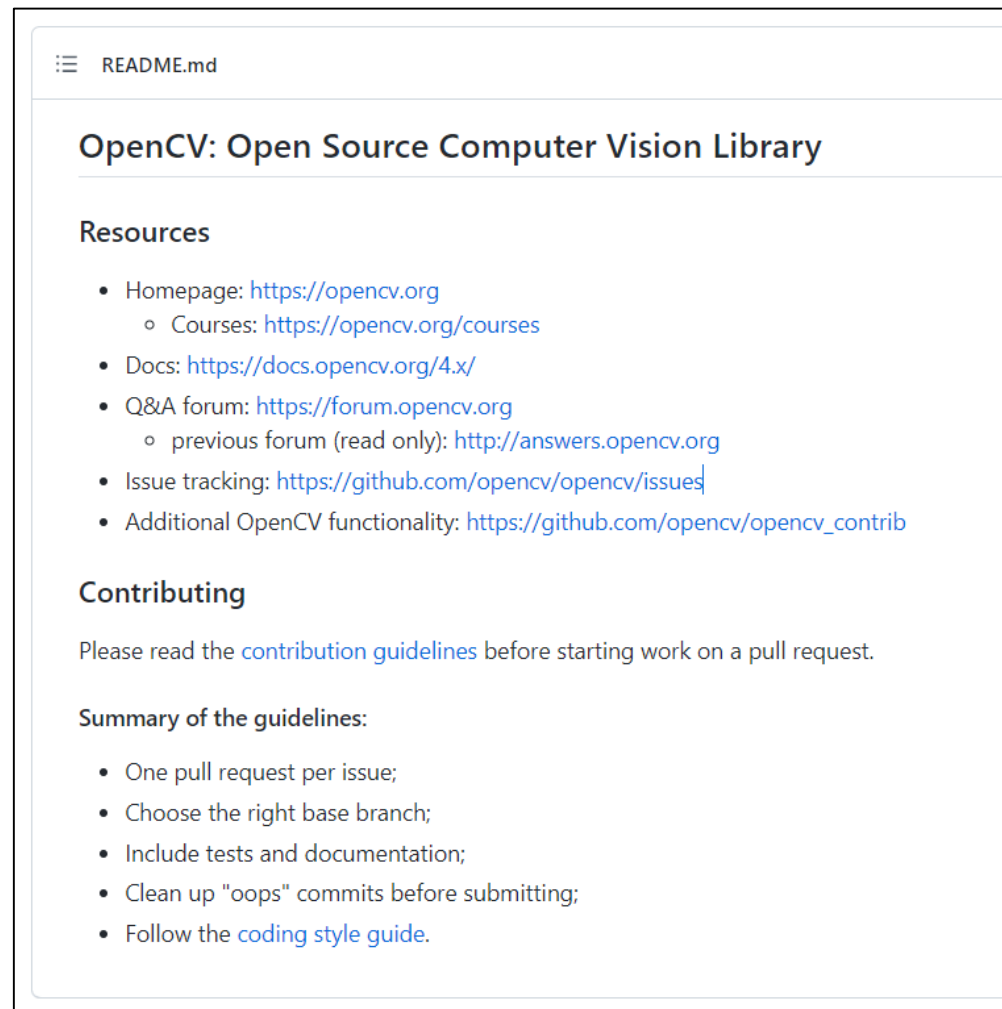
| 제목 | 내용 | 설명 |
|------|------|------|
| 테스트1 | 테스트2 | 테스트3 |
| 테스트1 | 테스트2 | 테스트3 |
| 테스트1 | 테스트2 | 테스트3 |

[도전] Markdown 비슷하게 만들어보기

opencv

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

readme 문서와 비슷한 형태로
markdown을 직접 작성해보자.



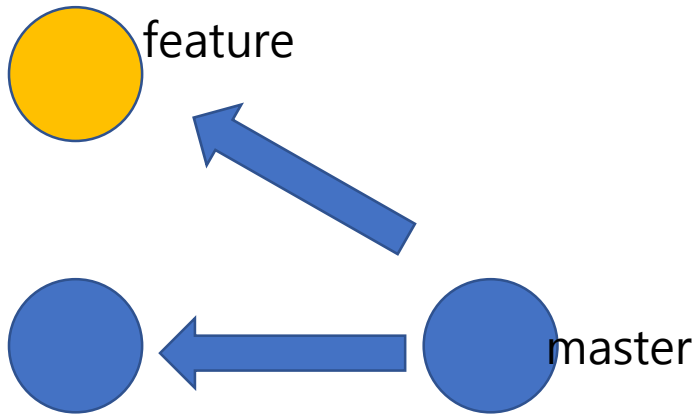
Chapter7

git 5단계

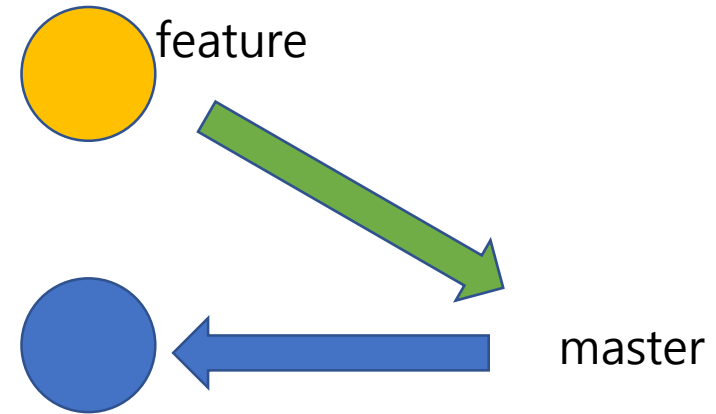
Pull Request

Merge 대신 Pull Request

- github은 코드 리뷰가 가능하기 때문에 merge 대신 PR 을 해야 함.



merge
master에서 feature를 합쳐 commit 하기



Pull Request (PR)

master branch님, 절 받아주시겠어요? (pull)
→ 리뷰어들의 코드 리뷰가 시작된다.

Repository 생성

- 신규 Repo 생성

Repo 이름 : prtest-번호

Description : 이름

Internal 속성 선택


README 파일 생성

Create a new repository


A repository contains all project files, including the revision history. Already have a project? [Import a repository.](#)

Owner *

Repository name *


 mincoding1


 /

prtest-123456 

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

Description (optional)

☐  Public

☒  Internal 선택

☐ Public Anyone on the internet can see this repository. You choose who can commit.

☒ Internal You choose who can see and commit to this repository.

Initialize this repository with:

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

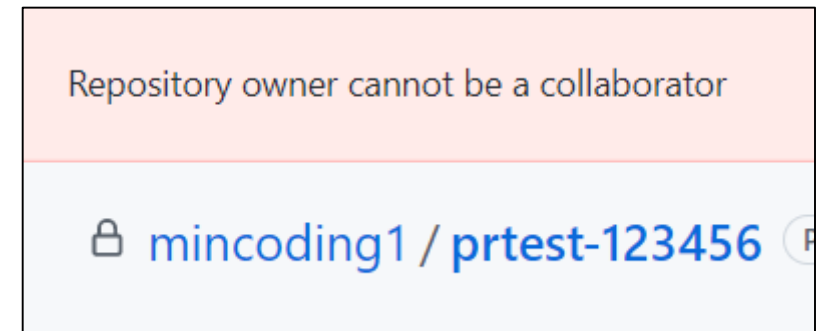
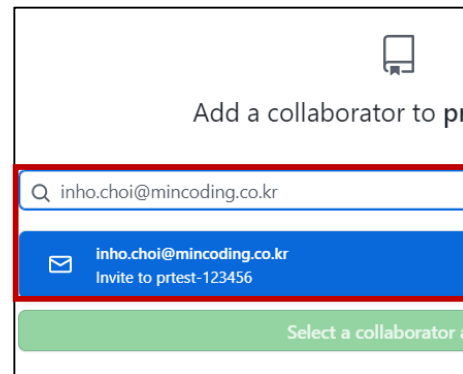
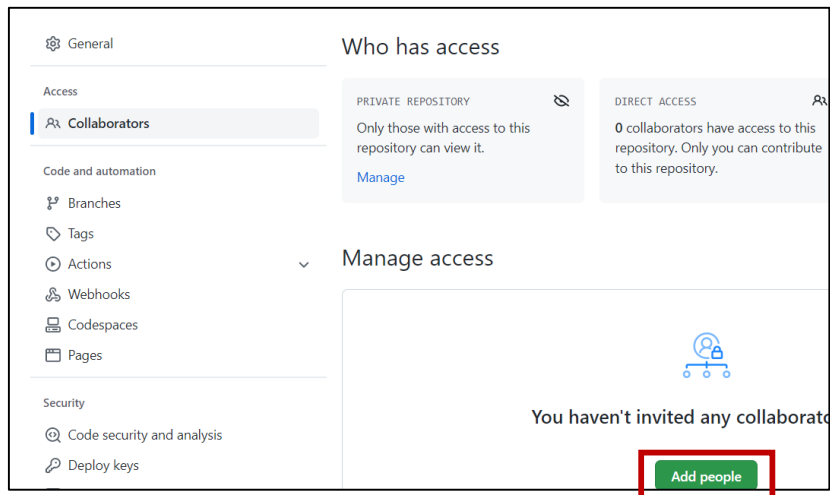
☒ Add a README file

This is where you can write a long description for your project. [Learn more.](#)

팀원 추가 (collaborator)

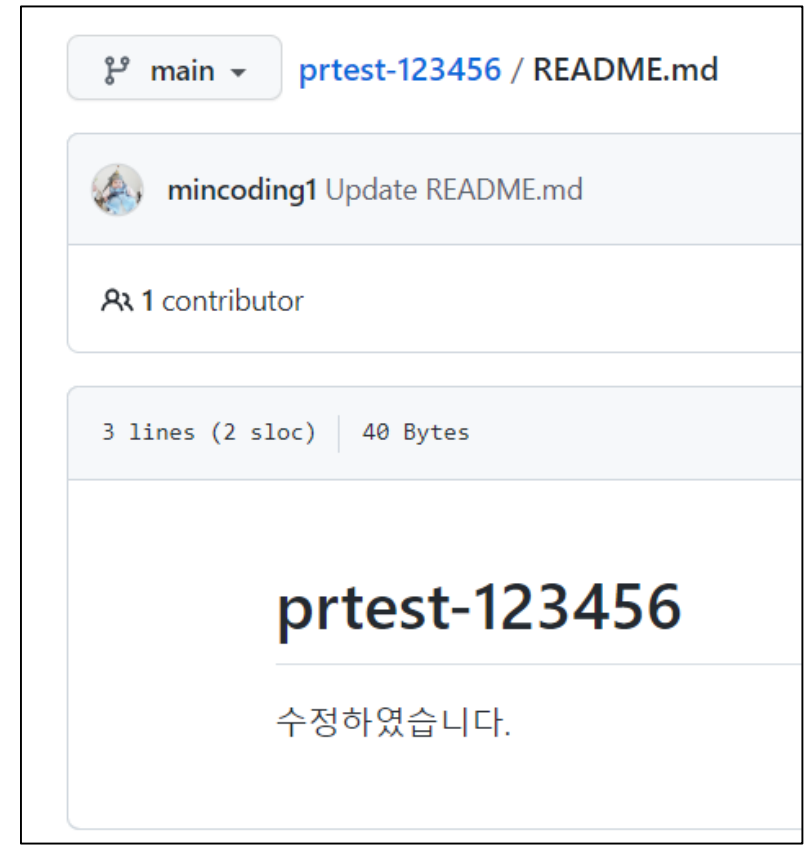
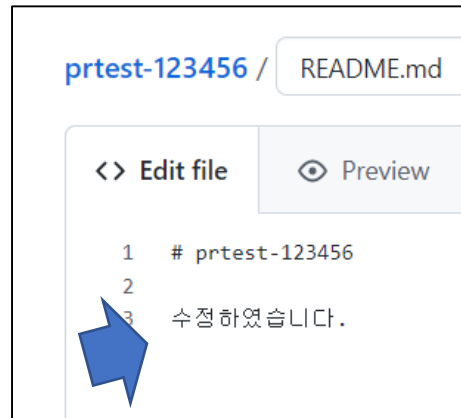
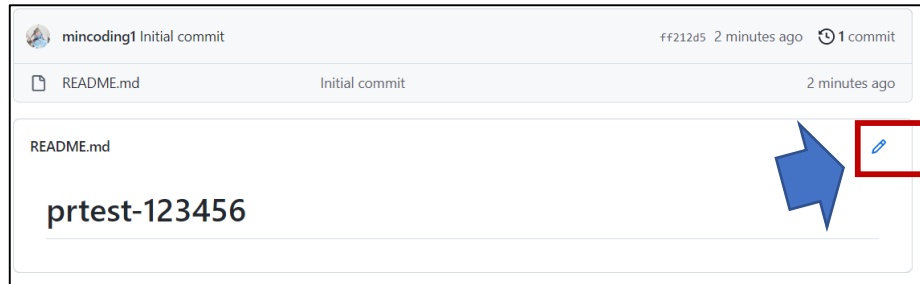
- Setting > Collaborators (& Team)

자기 자신을 추가해보고, 에러메세지가 뜸을 확인한다.
팀원들을 기입한다. (항상 Write 권한으로 부여할 것)
강사를 추가한다.



Commit 추가

- README.md 파일 수정하여 Commit



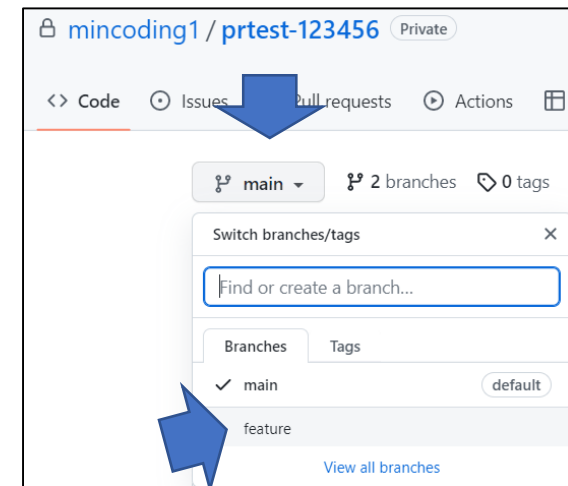
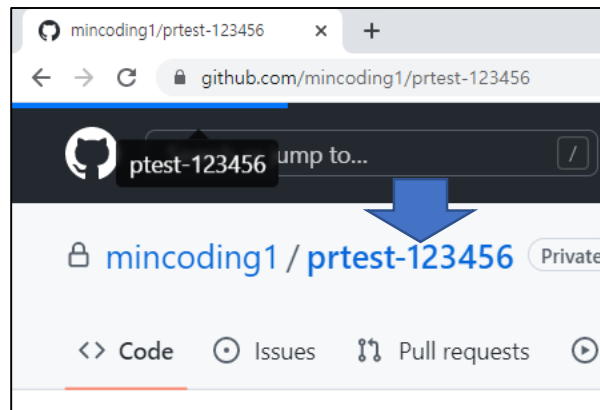
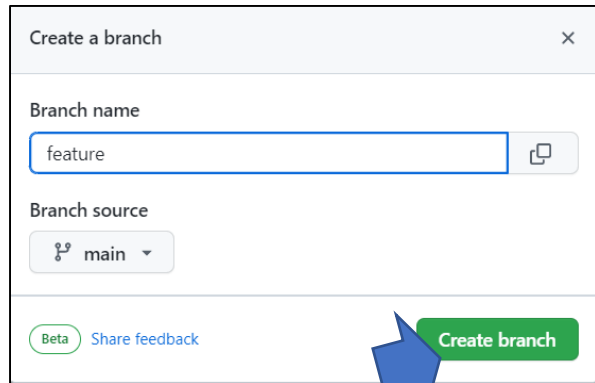
branch 생성 방법 1

- 1 branch 클릭하자.



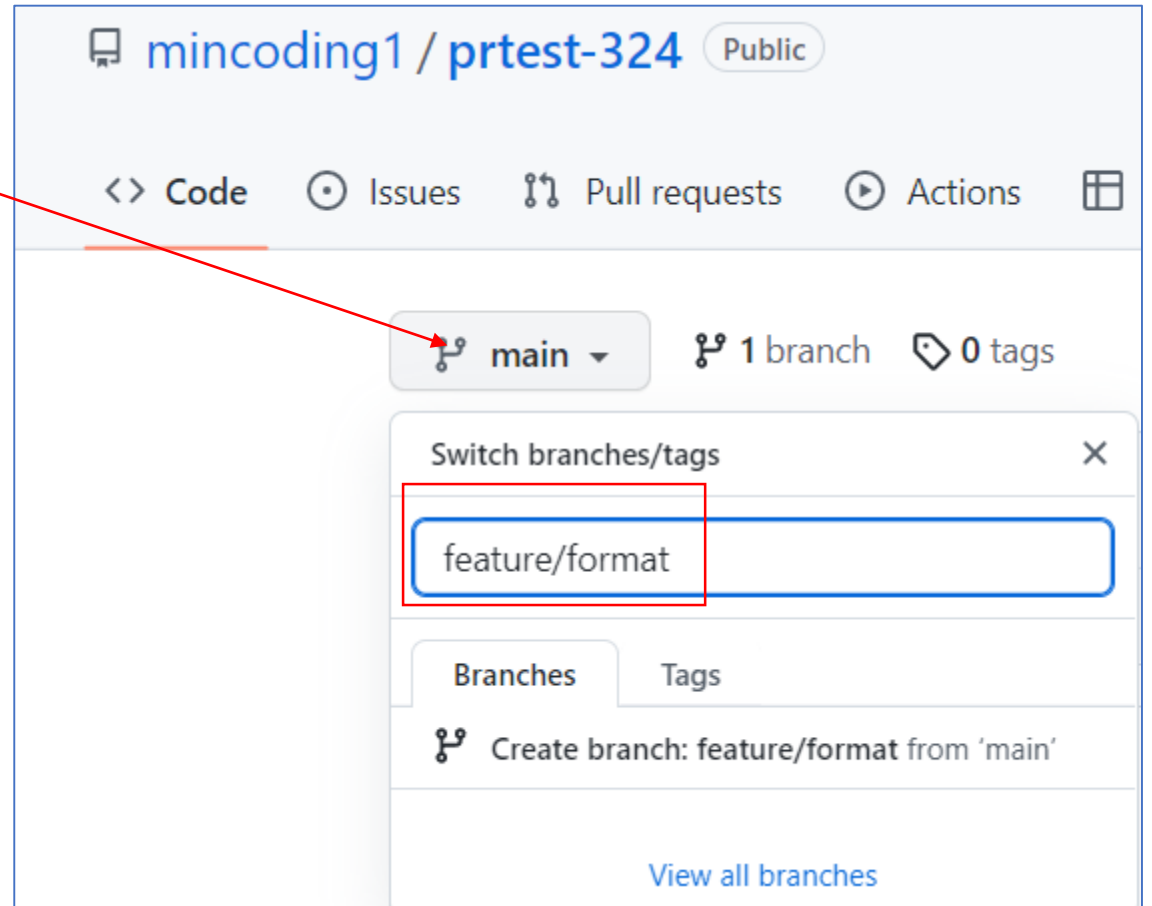
branch 생성 후 checkout

- 새로운 branch를 생성한다.



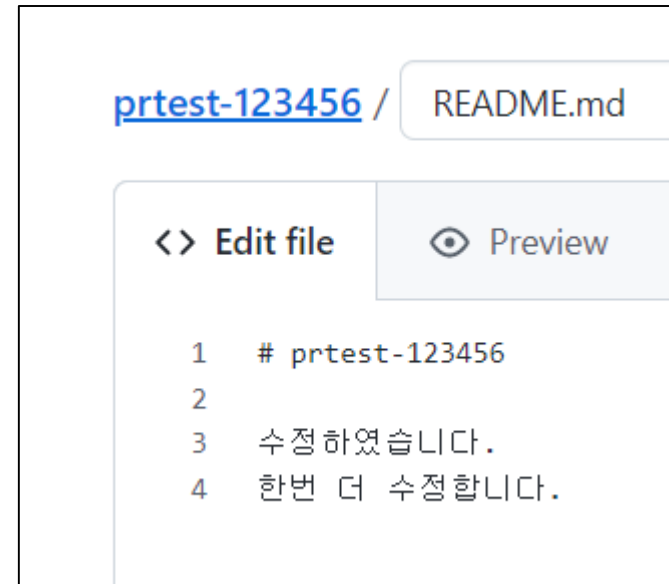
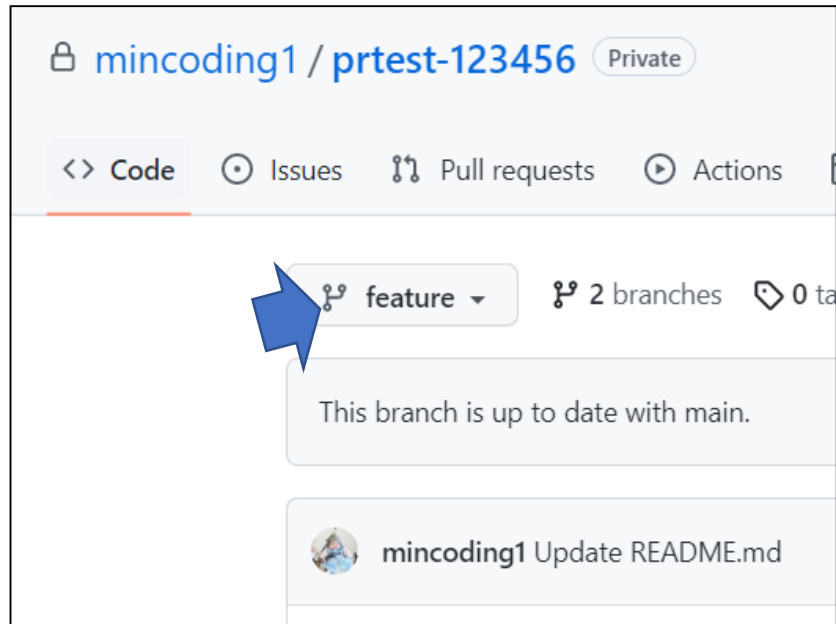
branch 생성 방법 2

“main” 클릭 후
branch명 입력 > create 버튼



내용 한번 더 수정 후 commit

- feature Branch를 선택한다.
- README.md 파일을 한번 더 수정 후 commit 한다.

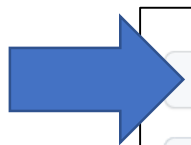



비교하기


main branch로 checkout


vs

feature branch로 checkout



 main ▾ prtest-123456 / README.md

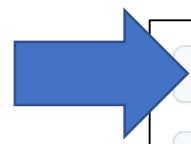
 mincoding1 Update README.md


 1 contributor


3 lines (2 sloc) | 40 Bytes


prtest-123456

수정하였습니다.



 feature ▾ prtest-123456 / README.md

 mincoding1 Update README.md

 1 contributor

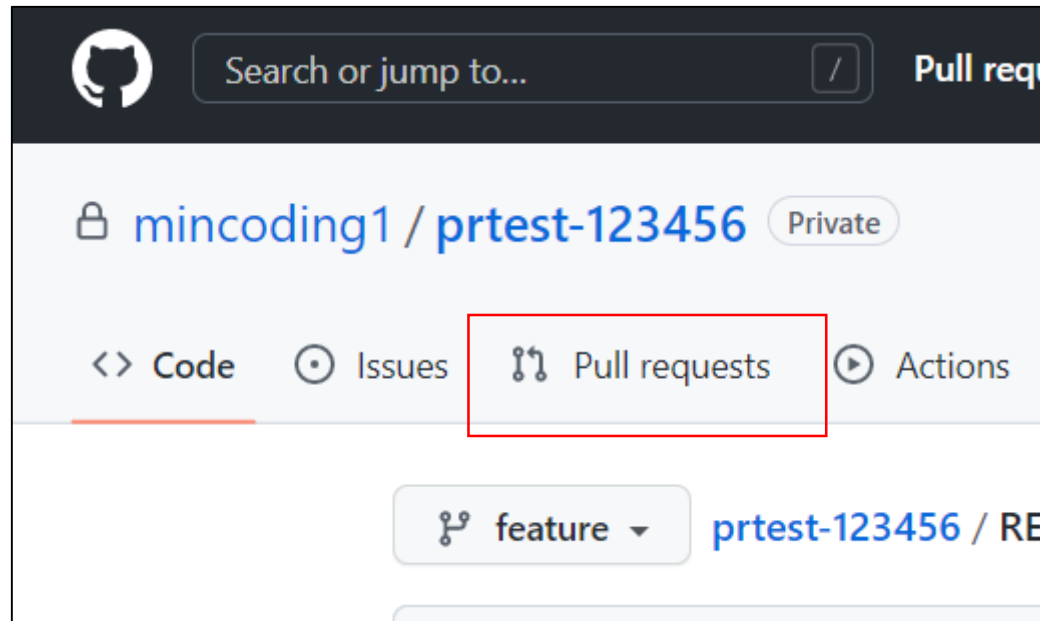
4 lines (3 sloc) | 68 Bytes

prtest-123456

수정하였습니다. 한번 더 수정합니다.

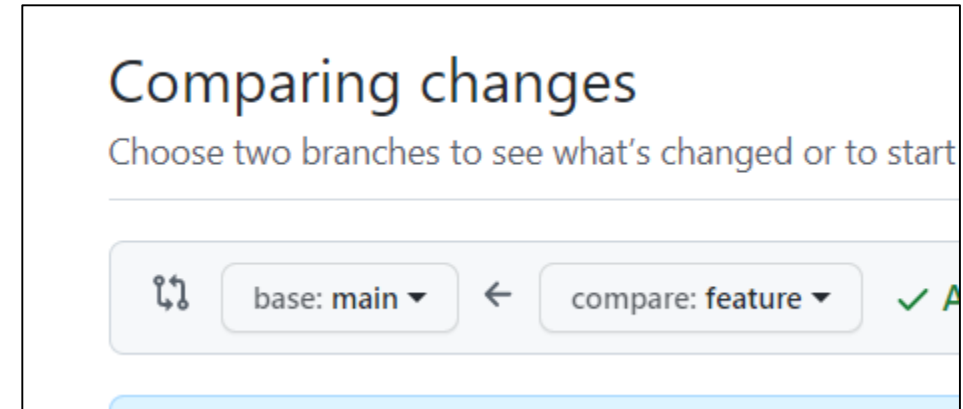
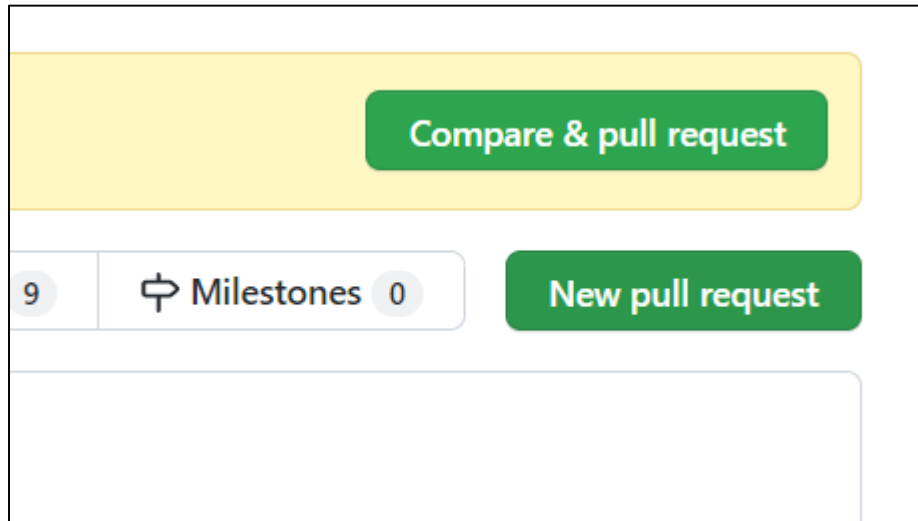
Merge 대신 Pull Request

- feature branch로 checkout을 한다.
- Merge를 하기 위해, Pull Request를 선택한다.



new PR 클릭

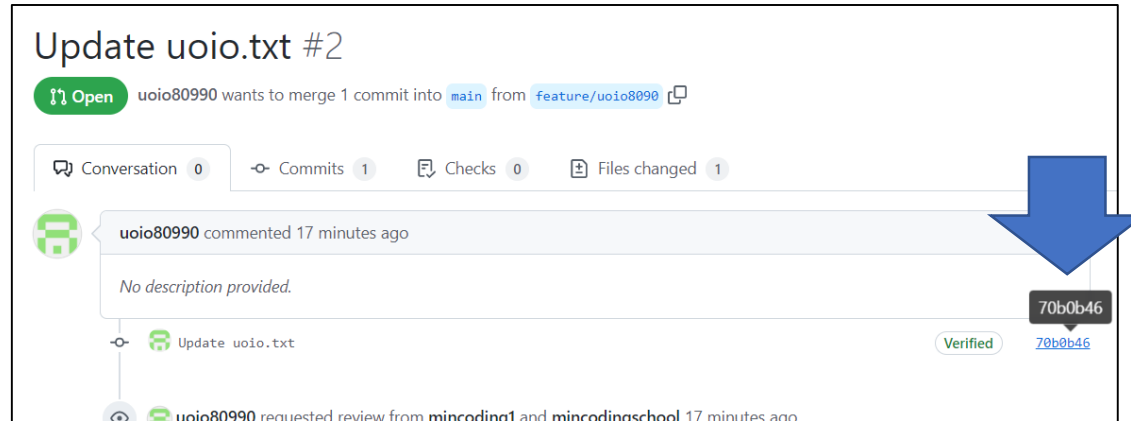
- New pull request를 선택한다.



feature branch 내용을
main branch로 Merge 요청한다는 의미.

어떤 내용인지 확인

commit 해쉬 클릭하면,
어떤 수정사항이 있는지 내용 확인 가능




리뷰어 추가

OK 받을 사람들 (리뷰어) 추가 진행

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).




base: main ▾

←

compare: feature ▾

✓ Able to merge. These branches can be automatically merged.



Update README.md

Write

Preview

H B I ≡ <> 🔗 ☰ ≡ ⚙️ @ ↗ ↶

Leave a comment

Reviewers

No reviews


Assignees

No one—assign yourself

Labels

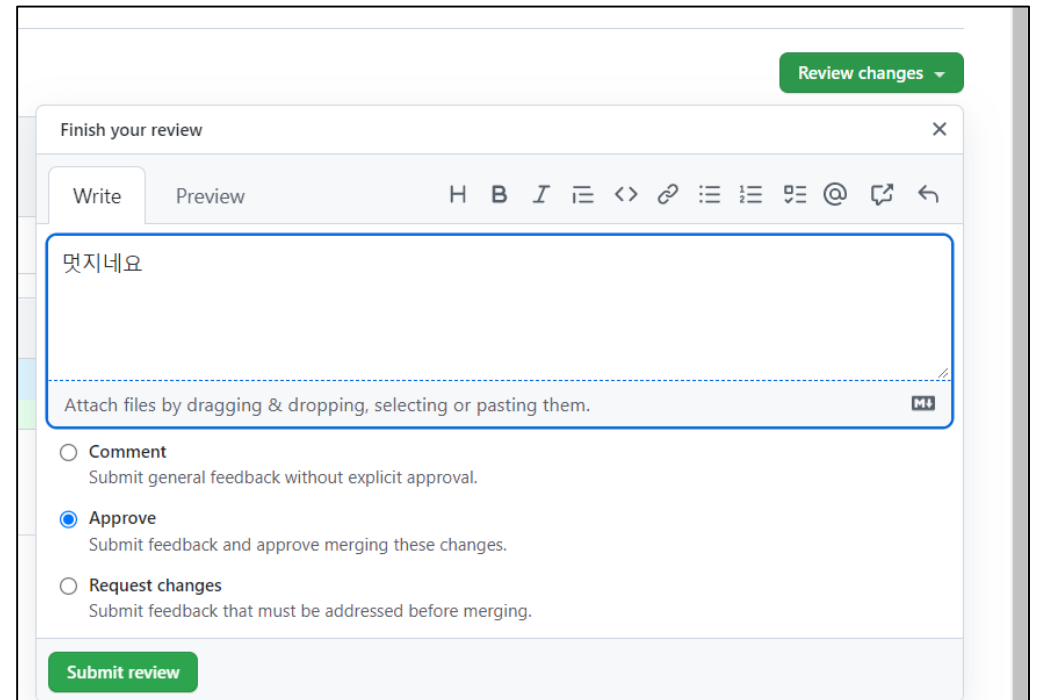
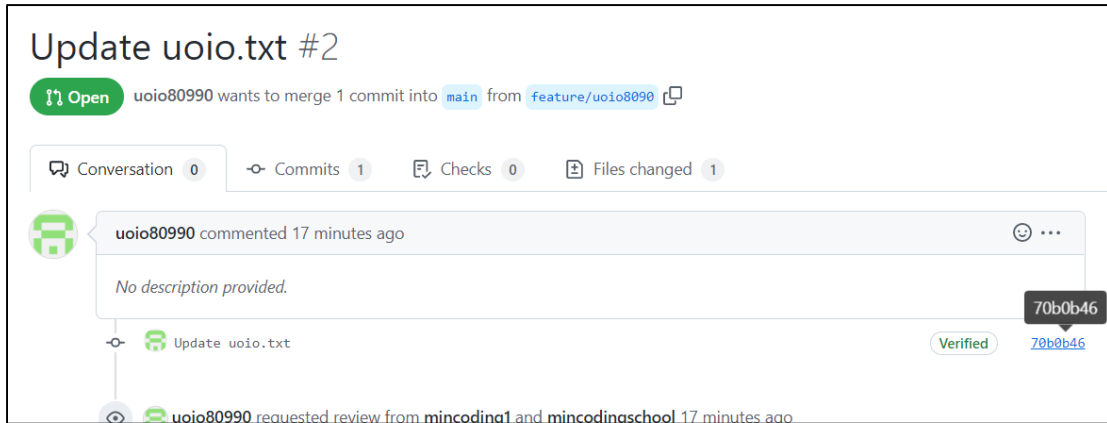
None yet

Assignees



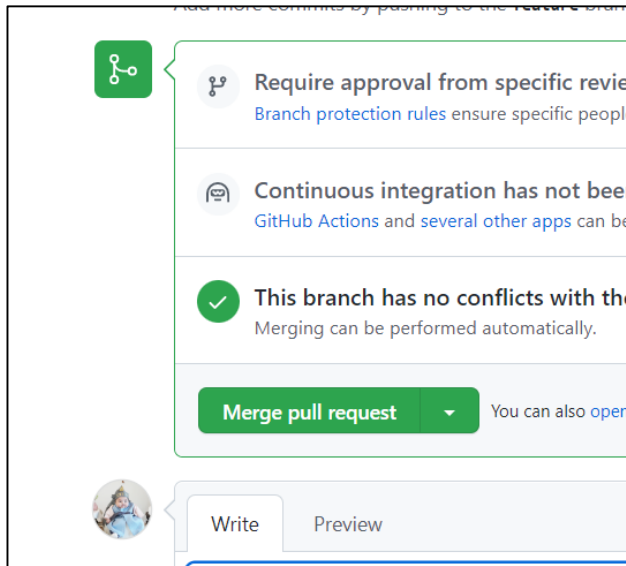
승인 방법

- Comment : 결정 없는 댓글
- Approve : 승인
- Request changes : 거절

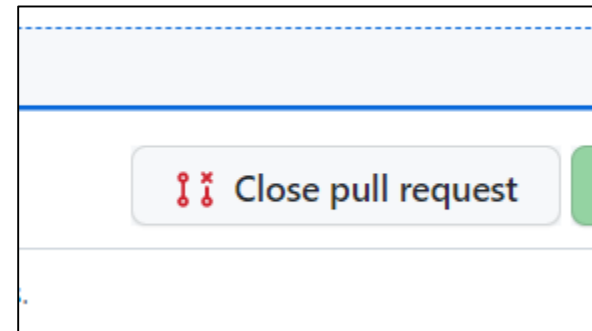


Merge PR

PR에 대해 Merge를 수행하거나,
Close PR로 PR 요청을 취소시킬 수 있다.



최종 Merge 하기



PR Close 하기 (취소)

PR 취소

Revert 하면

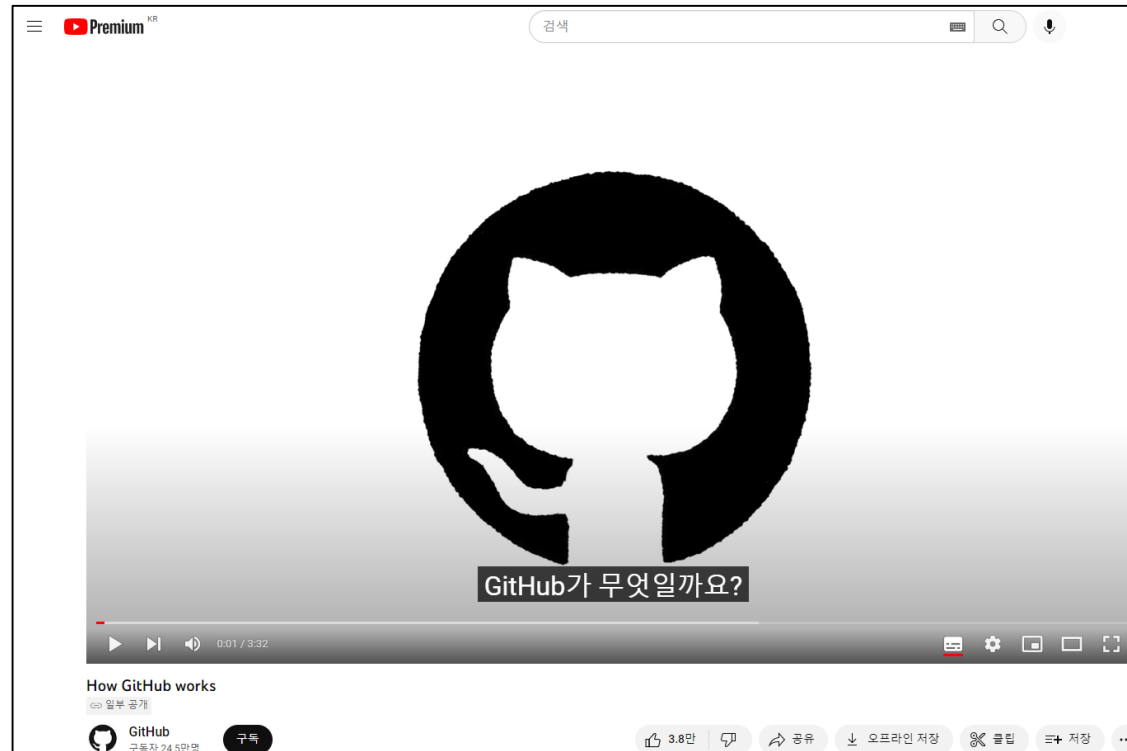
Merge 취소되고 다시 새로운 PR을 생성된다.

[도전] 직접 해보기

1. repo 생성 (readme.md 파일 추가)
2. Branch 생성
3. readme 수정
4. readme 한번 더 수정
5. PR
6. 어떤 내용인지 확인해보고, comment 남기고
7. Merge

GitHub 소개자료

- <https://www.youtube.com/watch?v=w3jLJU7DT5E>



TDD 수업, 임시 팀장님 선정

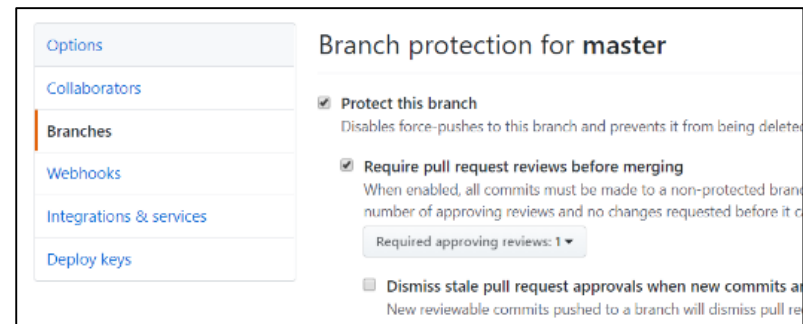
1. 임시팀장님 기준 1 : 우측 상단에 앉으신 분 기준
2. 만약 Git 사용이 어려우시다면, 다른 분께 양도

임시 팀장님 역할, 두 가지

1. 메신저 단톡방 초대
2. Repo 생성 & 세팅 후 Collaborators 추가하기
3. 메신저 단톡방에 Repo 링크 공유

[도전] PR 하기 -팀장님만 진행할 것.

1. 팀장님 Repository 생성 (+ 팀원들 설정)
 - readme 파일 생성
2. Collaborators : 팀원 / 강사 등록하기 (Write 권한)
3. PR 요청에 대해, 팀원 모두의 Approve를 받는 경우 Merge 가능하도록 하기
 - Setting > Branches > Protect this branch 설정
 - 팀원 수 만큼 Required approve reviews 설정하기



[도전 1 단계] PR 하기 – 팀장 / 팀원 모두 진행

IDE / git bash에서가 아닌, **Github**에서만 진행한다.

1. **Github**에서 각자, branch 생성한다. : feature/영어이름1
2. 개인 txt 파일 생성 후, 본인 branch로 commit & push
 - txt 파일 내용 : 내가 언젠가 희망하는 취미생활 or 하고 있는 취미생활을 적는다.
3. **Github**에서 본인 branch를 master로 PR 요청을 한다. (메신저로 "PR 요청했습니다. 코드리뷰 부탁드립니다~")
 - reviewer 등록은 팀원 전원으로 한다.
4. 각자의 PR을 보며 approve를 한다.
5. 모두의 approve를 받은 경우, PR작성자 merge pull request를 수행한다. (이후 Branch 삭제)

햇갈리는 부분은
팀원분들에게 도움을 요청해주세요.

[도전 2 단계] PR 하기 – 팀장 / 팀원 모두 진행

팀원 전원 1 단계 마무리 이후, **git bash**에서 진행을 시작한다.

1. **Git bash**에서 현재 Master branch를 Pull을 한다.
2. 가장 최근 Commit에서 각자의 branch를 생성 (feature/영어이름2)
3. 개인 txt 파일 생성 후, 본인 branch로 commit & push
 - txt 파일 내용 : 좋아하는 음식이나, 먹고 싶은 음식을 적는다.
4. **Github**에서 본인 branch를 master로 PR 요청을 한다. (메신저로 "PR 요청했습니다. 코드리뷰 부탁드립니다~")
 - reviewer 등록은 팀원 전원으로 한다.
5. 각자의 PR을 보며 approve를 한다.
6. 모두의 approve를 받은 경우, PR작성자 merge pull request를 수행한다. (이후 Branch 삭제)

햇갈리는 부분은
팀원분들에게 도움을 요청해주세요.

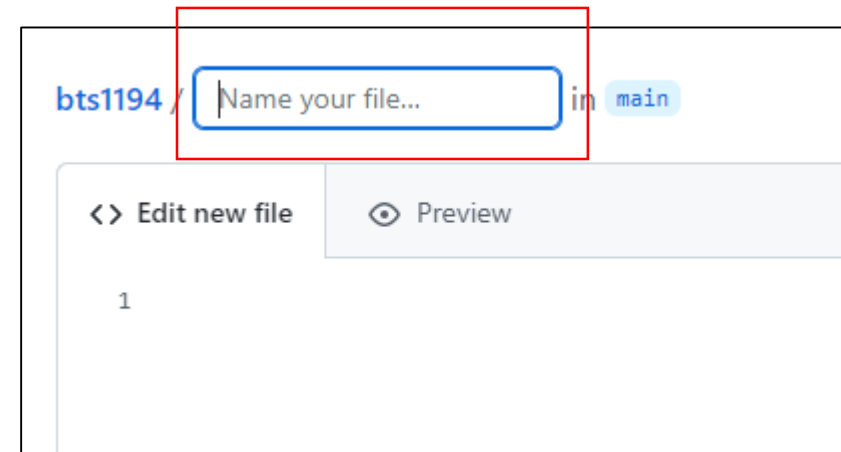
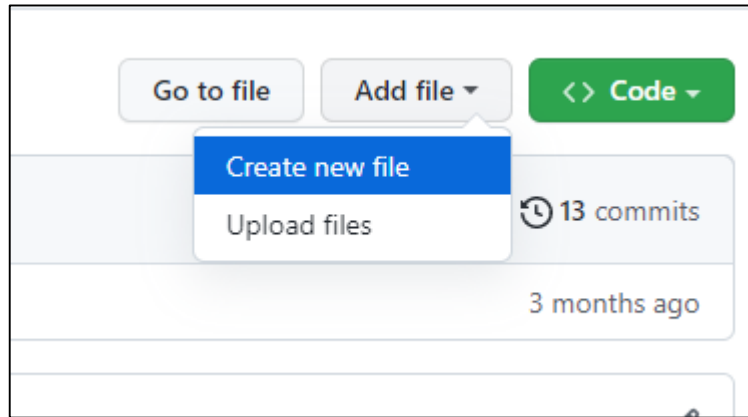
Chapter8

PR Template

PR 작성자를 위한 Check List 만들기

PR Template 작성해보기

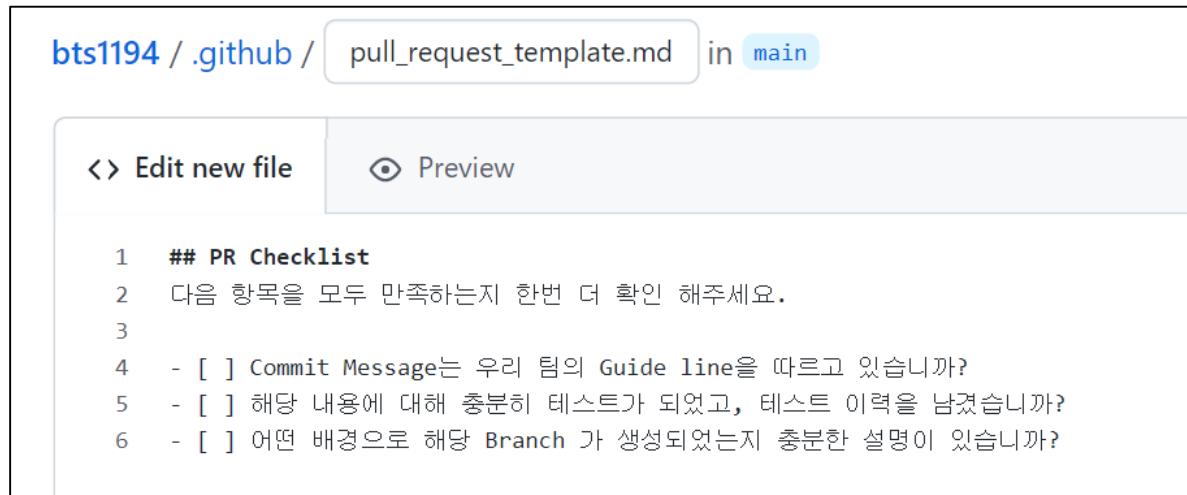
- PR을 작성할 때,



[.github/pull_request_template.md](#)
이름으로 파일명 입력

PR Template 내용 기입

- 체크리스트 항목 내용 기입



The screenshot shows a GitHub interface for editing a file named `pull_request_template.md` in the `main` branch of the repository `bts1194 / .github /`. The interface has two tabs: `<> Edit new file` and `Preview`. The `Edit new file` tab is active, displaying the following content:


```
1  ## PR Checklist
2  다음 항목을 모두 만족하는지 한번 더 확인 해주세요.
3
4  - [ ] Commit Message는 우리 팀의 Guide line을 따르고 있습니까?
5  - [ ] 해당 내용에 대해 충분히 테스트가 되었고, 테스트 이력을 남겼습니까?
6  - [ ] 어떤 배경으로 해당 Branch 가 생성되었는지 충분한 설명이 있습니까?
```


PR 작성시

- Template 내용을 기반으로 내용을 추가하여 PR을 작성한다.
- 콤보박스 내용은
[] 부분에서
[x] 로 수정하면 된다.

Open a pull request







Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

 base: main ← compare: test ✓ Able to merge. These branches can be automatically merged.

 Update README.md

Write

Preview

H B I      @  ↶

PR 요청
이런 내용으로 PR 요청합니다.

PR 세부 사항
이런 세부 사항이 있습니다.

PR Checklist
다음 항목을 모두 만족하는지 한번 더 확인 해주세요.

- [x] Commit Message는 우리 팀의 Guide line을 따르고 있습니까?

- [x] 해당 내용에 대해 충분히 테스트가 되었고, 테스트 이력을 남겼습니까?

- [x] 어떤 배경으로 해당 Branch 가 생성되었는지 충분한 설명이 있습니까?

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Chapter9

코드리뷰 : 마음가짐

더 나은 코드리뷰 문화를 위한 마음가짐

코드 리뷰이 (PR 요청자)

상대방에게는 추가 업무가 될 수 있기에
추가적인 업무 부하를 주지 않도록, 준비를 철저하게 하고 PR을 요청한다.

리뷰를 보며, 마음의 상처를 덜 받는다.

코드 리뷰어

상대가 마음의 상처를 입지 않도록, 정중한 표현을 사용하도록 노력한다.

팀원의 코멘트에 상처를 덜 받자.

극단적인 마인드를 제어하자.

예시 1) 감히 (?) : 직급도, 나이도 어린 것이.. 감히 나에게 이런 지적을?

예시 2) 코멘트 주셔서 감사합니다. 아이고, 네네.. 무조건 맞습니다.

코멘트는 커뮤니케이션의 시작이다.

코멘트에 의문이 있다면,

충분히 분석 / 조사를 해본 후, 구체적인 내용에 대해 논의를 한다.

코멘트 내용을 받아들이지 않아도 된다.

받아들이지 않고 진행하더라도

해당 내용에 대해 충분히 검토한 것이기에, 품질에 도움이 된다.

코드 리뷰이 – PR 작성시

리뷰어를, 돈을 내고 책을 구매한 독자들이라고 생각해야 한다.

읽기 편한 문체와 전문용어 사용

- 리뷰어들이 이해하기 쉽게 써야 한다.

PR의 단위가 적절해야 한다.

너무 많은 파일과 많은 소스코드를 리뷰하라고 하면, 리뷰하지 않는다.

PR 코드가 딱 떨어지는 적은 정도

- 예시) Demo 노래 평가시, 노래를 이해하기 좋은 구절 정도가 PR 단위로 좋다.
- PR 단위 = Branch 단위

코드 리뷰이 – 셀프 리뷰

셀프 리뷰를 먼저 하고, 코드 리뷰를 요청한다.

이 PR을 보면서
내가 리뷰어라면 어떤 Comment를 달 것인가?를 예측하고,
이것에 대해 개선을 먼저 한다.

그리고 나서 다시 PR을 준비한다.

나에게 추가적인 일인 만큼,
상대방에도 추가적인 일이 되므로, PR은 꼼꼼한 대비가 필요하다.

코드 리뷰이 - 테스트 정보 제공

테스트 정보를 제공한다.

테스트를 어느정도 했고, 충분히 검증이 되었음을 명시한다.

코드 리뷰어가 품질에 대한 의심이 없도록 해주어야 한다.

Unit Test 결과

가능하다면, 더 높은 Level의 테스트 까지

가능하다면, 성능 분석까지

Self 코드리뷰 결과까지

코드 리뷰어가, 더 Deep한 숨은 버그 발생 요소 &

코드 개선점에 대해 논의할 수 있도록 테스트 정보를 제공한다.

배경 설명

어떤 History에 의해서, 어떤 코드를 왜 바꿨는지,
미팅에 참석했으면 알 수 있는 정보라도, 다시 적는다.

변경 내용

기존 대비 변경 내용에 대해 내용을 적는다.
코드 보면 알 수 있어도, 이해에 도움이 되도록 적는다.

테스트 이력

어떠한 테스트를 했고, 성능 / 품질에 이상이 없음을 알린다.
정적 분석도구 결과 / 코드 커버리지 결과 등

코드리뷰어 – 도움을 위한 Review

코드 리뷰이와 다른 코드 리뷰어까지
모두가 도움이 될 수 있는 Review를 남긴다.

안 좋은 예시)

이 부분을 수정했으면 좋겠습니다.
요즘 이런 식으로 개발 안합니다.(X)

좋은 예시)

이 부분을 수정했으면 좋겠습니다.
이유는 이렇습니다,
고치는 방법은
1. 어떤 방법
2. 어떤 방법
인데 제 개인적으로는 1번을 더 추천합니다. 이러하기 때문입니다.
관련 자료는 이렇고, 검토를 부탁드립니다.

코드리뷰어 – 명확한 표현

중립적인 표현은 하지 않는다.

Approve인지, Request changes인지 명확하게 의견을 표현한다.
리뷰를 했지만, Comment로 모호한 의견을 남기지 않는다.
수정할 사항이 없으면 **Comment가 아닌 Approve이다.**

좋지 못한 예시)

comment : 좋은 것 같지만, 다른 분 의견이 궁금합니다.
코드 리뷰했지만, 무응답

좋은 예시)

무엇을 왜 바꿔야 하는지가 명확해야 한다.

Approve : 이 코드는 가독성이 있는 Clean한 코드로 생각합니다.
저도 이런 코드 Style로 진행하고자 합니다.

Request changes 이 부분에서 이런 부분은 이러한 risk가 있을 것으로 보입니다. 왜냐하면 ..,

리뷰를 남길 때, 다음 항목을 재검토 해보자.

1. 이 Review가 코드 리뷰어에게, 도움이 되는 내용인지 검토해본다.
2. 이 Review가 코드 리뷰어에게, 부정적인 지적 / 비난으로 들릴 수 있을지 생각해본다.
3. 모든 팀원들이, 내가 작성한 Review처럼 남길 때, 긍정적인 코드 리뷰 문화가 만들어질 수 있을지 생각해본다.

Q. 코드 리뷰를 시작하기 위해서는?

1. Branch + PR 형태 개발 강제화
2. n명 이상 Approve 후 Merge PR 가능 Rule 도입

Q. 성의 없는 리뷰, 무응답으로 코드리뷰 문화 정착이 잘 안된다면?

→ Daily (온/오프라인) 코드 리뷰 미팅

Q. 코드 컨벤션 / 품질에 대한 논의 필요

→ 정적 분석도구 등에 맡기고, 사람이 직접 리뷰할 필요 없음

Q. 코드 리뷰를 Stop 해야 할 때

→ 대화의 핑퐁 (다른 커뮤니케이션 수단을 활용 필요)

[참고자료]

- 구글의 코드리뷰 문화

- https://hanbit.co.kr/channel/category/category_view.html?cms_code=CMS3858769941
- <https://m.post.naver.com/viewer/postView.naver?volumeNo=30978428&memberNo=36733075>

- 카카오 코드리뷰

- <https://tech.kakao.com/2022/03/17/2022-newkrew-onboarding-codereview/>

- 구글 코드 리뷰 가이드

- 영문 : <https://google.github.io/eng-practices/review/>
- 한글번역 : <https://soojin.ro/review/>

Chapter10

계산기 구현과 코드리뷰 실습

소스코드 Base, PR

[도전] 소스코드 base로 PR하기 (with 충돌해결) - 1

각 팀원들은 아래 기능중 하나를 선택 후 개발한다.

메신저로 어떤것을 개발할지 미리 정해주세요.

팀장님, 팀원분들 모두 아래 기능 중 하나를 선택해주세요.

UnitTest - TestCase도 추가해야합니다.

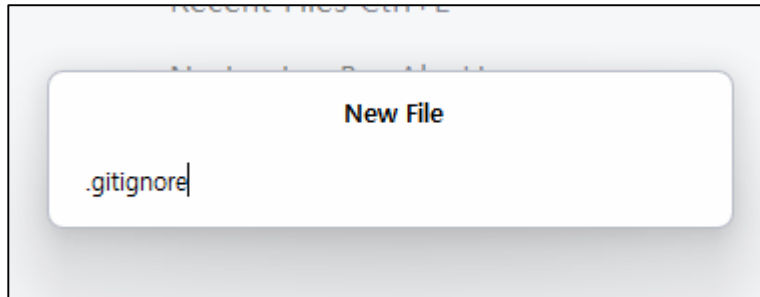
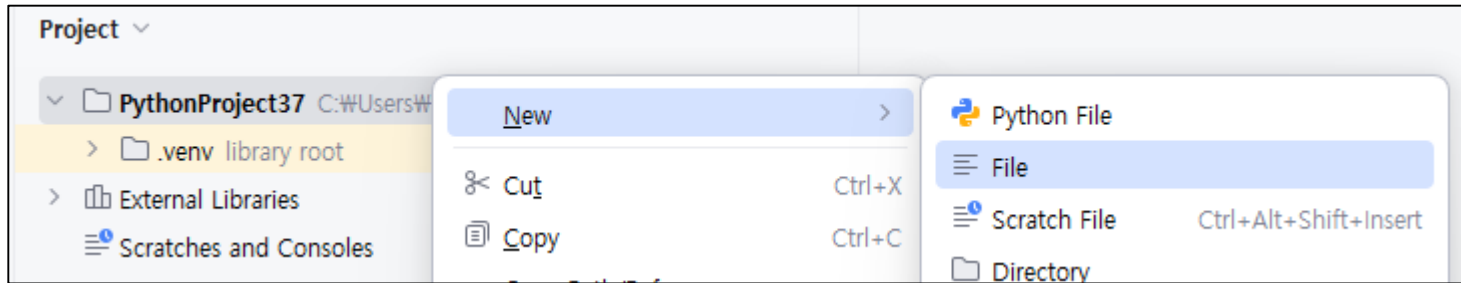
제작할 기능

1. getSum(a, b) : $a + b$ 반환
2. getGop(a, b) : $a * b$ 반환
3. getZegop(a) : $a * a$ 반환
4. getMinus(a, b) : $a - b$ 반환
5. getDivide(a, b) : a / b 반환
6. getSumSum(a, b, c) : $a + b + c$ 반환

팀장님은 README.md 파일 없이, 비어있는 Repository를 생성한다.

스켈레톤 코드 제작

1. 프로젝트 > .gitignore 파일 추가



.gitignore 파일 작성하기

pycharm 용 gitignore 템플릿 내용을 모두 복사 붙여넣기

```
.gitignore x
1 # Byte-compiled / optimized / DLL files
2 __pycache__/
3 *.py[cod]
4 *$py.class
5
6 # C extensions
7 *.so
8
9 # Virtual environment
10 venv/
11 env/
12 .venv/
13 .env/
14
15 # PyCharm project files
16 .idea/
17
18 # Distribution / packaging
19 .Python
20 build/
```

<https://gist.github.com/jeonghwan-seo/73ca1003296ee0a170ef722ab2a19c66>

main 코드 작성

충돌이 자주 발생 되도록 하는 목적
팀장님 / 팀원분들은
모두 주석이 달린 부분에 구현을 해야 한다.

```
import pytest

class Calc:
    # 이곳에 코드를 작성
    pass

# 테스트 케이스 작성
def test_sample():
    assert 1 == 1
    pytest.fail()
```

터미널 열고 git push하기

git 명령어를 이용하여 지금까지 코드를 repository에 push한다.

[도전] 소스코드 base로 PR하기 (with 충돌해결) - 2

하나의 소스파일과 하나의 테스트파일로만 개발을 한다.

- 충돌이 자주 발생할 예정이며, 충돌이 발생할 경우, 충돌을 해결한다. (수동 or 자동)

팀장님 역할

- 스켈레톤 코드 작성 (Unit Test 파일 포함)
- push 진행 후, github repo 공유
- 기능 하나 맡고, 개발 Branch 생성 후 개발 시작

팀원 역할

- clone
- 기능 하나 맡고, 개발 Branch 생성 후 개발 시작

감사합니다.