

협업을 위한 Git

# Session 3

---

NEXT X LIKELION 강동혁

**모두 HTML&CSS 마스터가  
되신걸 축하드립니다!**

## 5분 넥코톡

개발자용 폰트: D2 Coding

### 1. 제품 개발 프로세스 살펴보기

다운로드: <https://github.com/naver/d2codingfont/>

구분	폰트명	숫자	영문자	한글	특수기호
1자 형태	굴림체	1			
	Consolas	1	1		
	D2 Coding	1	l		
0자 형태	굴림체	0	0	o	
	Consolas	0	0	o	
	D2 Coding	0	0	o	
코드 기호	굴림체	( ) { } [ ] " ' " : ; , . , . -- ++ == _ _ /			
	Consolas	( ) { } [ ] " ' " : ; , . , . -- ++ == _ _ /			
	D2 Coding	( ) { } [ ] " ' " : ; , . , . -- ++ == _ _ /			

# 목차

## 1. 제품 개발 프로세스 살펴보기

제품 개발 프로세스

제품 개발 프로세스와 Git의 관계

제품 개발 협업 시 역할과 책임 나누기

## 2. CLI & Git 복습

Git 개념 복습

CLI 명령어 정리

지금까지 배운 Git 명령어 복습

## 3. Git 심화

Git Reset & Git Revert

Git Branch

Git Pull-Request

## 4. Git 팀별 실습

Git branch 관리 방법 3가지

하나의 브랜치에서 Git 팀별 실습

여러 브랜치에서 Git 팀별 실습

# 1. 제품 개발 프로세스 살펴보기

---

제품 개발 프로세스

제품 개발 프로세스와 Git의 관계

제품 개발 협업 시 역할과 책임 나누기

# 제품 개발 프로세스

제품 개발 프로세스

## 1. 제품 개발 프로세스 살펴보기



# 제품 개발 프로세스

제품 개발 프로세스

## 1. 제품 개발 프로세스 살펴보기

아이디어 선정 + 대략적인 UI/UX 제작(기획문서 작성) +  
요구사항 정의 + 개발 환경 세팅 →  
페이지별 or 기능별 역할 분배 →  
개발 + QA + 배포

# 제품 개발 프로세스와 Git의 관계

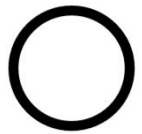
이상적인 해커톤 협업 과정

## 1. 제품 개발 프로세스 살펴보기

혼자서 캐리 **X**



정확한 역할 분배를 통한  
협업

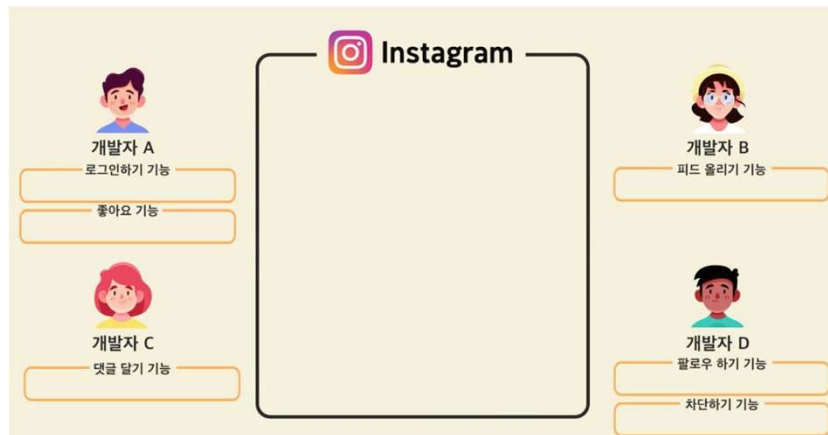




# 제품 개발 프로세스와 Git의 관계

정확한 역할 분배와 협업을 위해서 Git이 필요

## 역할 분배를 통한 협업



## 1. 제품 개발 프로세스 살펴보기

# 협업 시 역할과 책임 나누기

역할 분배 하는 법

## 1. 제품 개발 프로세스 살펴보기

- 프론트엔드와 백엔드로 나눠서 역할 분배
- 페이지 단위로 역할 분배
- 독립적인 개발 단위로 역할 분배
- 기능 단위 역할 분배

# 협업 시 역할과 책임 나누기

프론트엔드와 백엔드로 나눠서 역할 분배

프론트엔드와 백엔드로 나눠서 역할 분배

## 프론트엔드

- Django Templates의 html, css, javascript를 다룬다.
- html에 어떤 변수들이 어디에 들어갈지를 구상한다.
- css로 스타일을 입힌다.
- Javascript로 동적처리 + 데이터 전송을 처리한다.

## 1. 제품 개발 프로세스 살펴보기

## 백엔드

- models.py, urls.py, views.py를 다룬다.
- 서비스를 구성하기에 적합한 model을 짤다.
- urls.py에서 페이지별 경로와 특정 동작을 수행하는 경로를 제작한다.
- views.py에서 각 함수마다의 로직을 짤다.

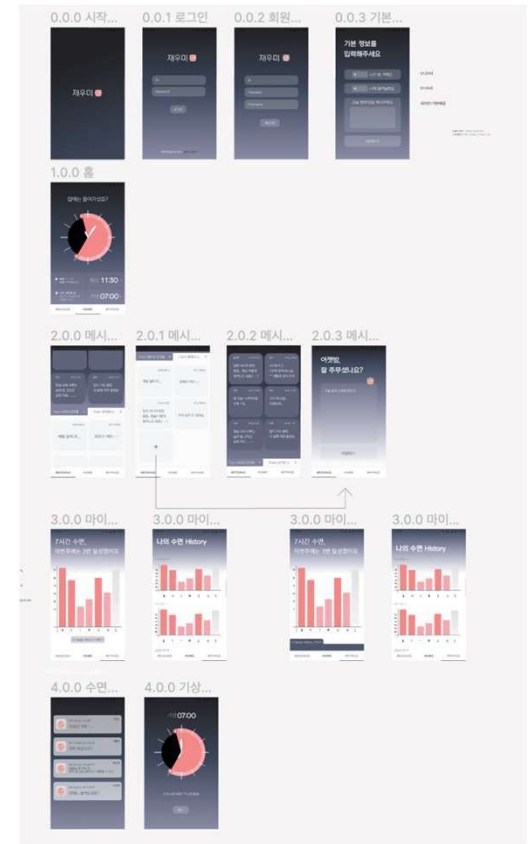
# 협업 시 역할과 책임 나누기

페이지 단위 역할 분배

## 페이지 단위별로 역할 분배

- A = 시작 페이지, B = 홈, C = 메시지 페이지, D = 마이페이지
- 각 페이지마다 필요한 기능과 요구사항들을 정리하기
- 각 페이지마다 필요한 백엔드와 프론트엔드 구현하기



## 1. 제품 개발 프로세스 살펴보기



# 협업 시 역할과 책임 나누기

기능 단위 역할 분배

## 독립 개발 단위로 역할 분배

 accounts
 blog

- 향후 Django에서 배울 app 분리를 이용
- A는 account를, B는 blog를 맡아서 모두다 개발하는 방법

## 1. 제품 개발 프로세스 살펴보기

# 협업 시 역할과 책임 나누기

기능 단위 역할 분배

## 기능 단위별로 역할 분배

재우미 기능 명세서

태표 태표 +

상태	기능분류	type	담당자	기능명	기능설명	중요도
Done	-2.페이지 나누	프론트		-2.0.0react페이지 나누기	react-router-dom 6v를 활용하여 페이지를 나눈다.	필수
Done	-3컴포넌트					
Done	0.시작	프론트		0.0.0 시작 홈 화면 구성	처음시작 화면	필수
Done	0.시작	프론트		0.0.1 로그인	1.로그인 화면 2. firebase를 통한 로그인 연동 3.로그인 후 홈화면으로 이동	필수
Done	0.시작	프론트		0.0.2 회원가입	회원가입 정보 입력 → 기본정보 입력으로 이동	필수
Done	0.시작	프론트		0.0.3 기본 정보 입력	기본 정보 입력, db 저장	필수
Done	1.홈	프론트		1.0.0홈	1. 유저 데이터 가져오기 시간별 메세지 데이터 가져오기 2. 유저 데이터 중 취침 시간 데이터, 기상 시간 데이터 보여주기 3. 메세지 데이터 시간별로 보여주기	필수
Done	1.홈	프론트		1.0.0홈	시간 보여주는 UI 매 시간이 갈수록 보여주기	필수
Done	1.홈	프론트		1.0.0홈	백엔드 홈화면 수정	
Done	2.메세지 페이지	프론트		2.0.0 메세지 홈	프론트 메세지 화면 구성 메세지 db 요청 api 전송하기 지금까지 작성한 나의 메세지, user메세지 보여주기	중
Done	2.메세지 페이지	프론트	민기 백	2.0.1 메세지 개인	개인 메세지 db 불러오기 개인 메세지 보여주기	중
Done	2.메세지 페이지	프론트	민기 백	2.0.2 메세지 모두	유저 메세지 db 불러오기 유저 메세지 보여주기	중
Done	2.메세지 페이지	프론트	민기 백	2.0.2 메세지 작성하기	메세지 작성 db 저장	중

## 1. 제품 개발 프로세스 살펴보기

- 요구사항을 정확하게 정리하여 기능 단위별로 역할을 분배한다.
- 장점: 요구사항별 역할분배 가능
- 단점: 요구사항을 정리하고 분배하기 번거로움

기능명세서 템플릿 : <https://ember-novel-440.notion.site/643e86ba0c0c488a9adc5ff51e2ad5a3?v=ebfac6292906417db52523248312b960>

NEXT X LIKELION

## 2. CLI & Git 복습 🤔

---

Git 개념 복습

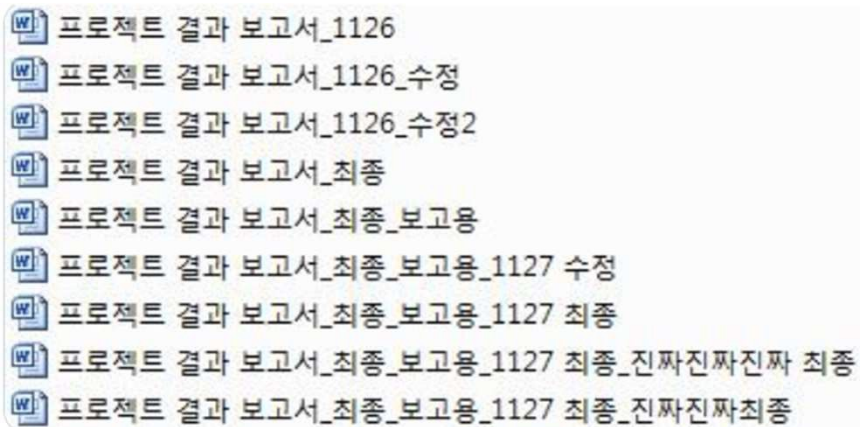
CLI 명령어 정리

지금까지 배운 Git 명령어 복습

# Git 개념 복습

Git이란 무엇일까?

Git은 버전 관리를 위한 도구



이런 방식의 파일의 버전 관리의 문제점

- 큰 용량과 관리의 어려움
- 지난 과정을 확인하기 어려움
- 이전 버전으로 돌아가기 힘들



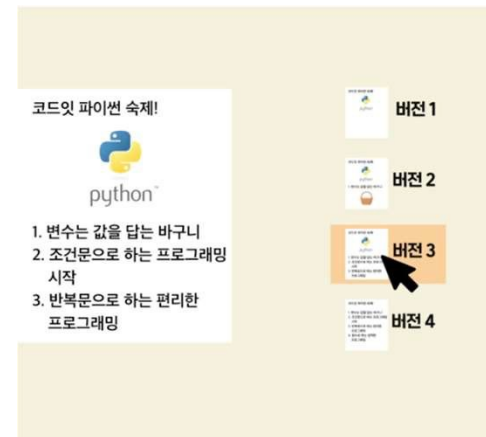
# Git 개념 복습

Git이란 무엇일까?

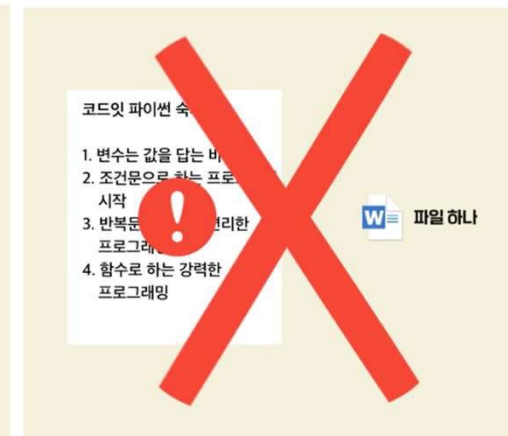
## 2.CLI & Git 복습

### Git 버전 관리의 장점

- 프로젝트 전체를 이전 상태로 되돌릴 수 있고,
- 시간에 따른 변경 사항을 비교해 볼 수 있고,
- 누가 문제를 일으켰는지 추적할 수 있고,
- 누가 언제 만들어낸 이슈인지도 알 수 있고,
- 파일을 잃어버리거나, 잘못된 부분을 쉽게 복구 가능



Git을 사용



파일 하나로 관리

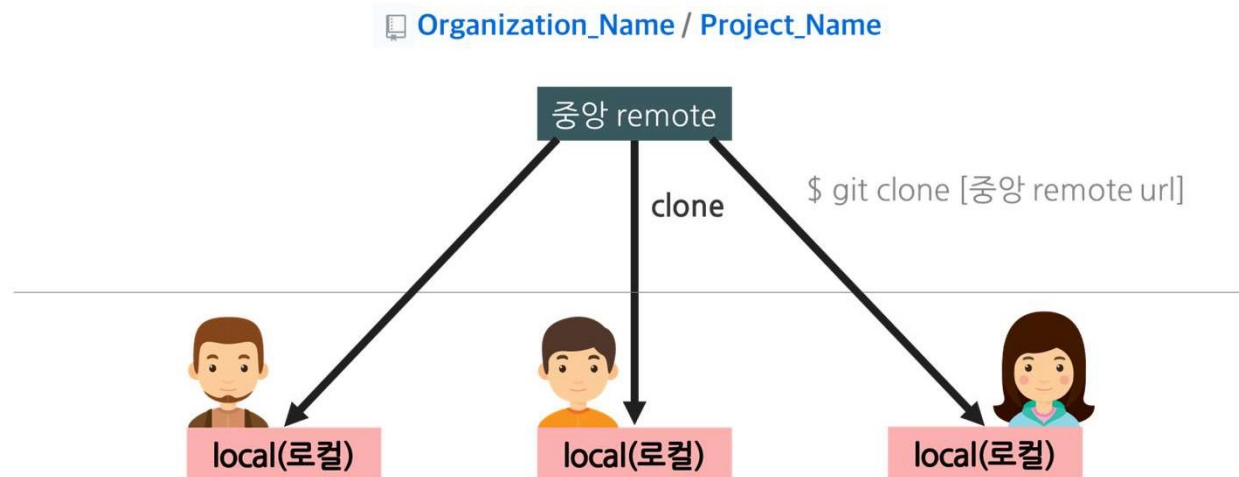
Git 장점:<https://www.boostcourse.org/web344/lecture/35137?isDesc=false> NEXT X LIKELION

# Git 개념 복습

Git이란 무엇일까?

Git은 협업을 위한 도구

2.CLI & Git 복습



# CLI 명령어 복습

CLI 명령어 복습

2.CLI & Git 복습

[경로 탐색] **pwd**: print (current) working directory

[경로 이동] **cd**: change directory

[파일 확인] **ls**: list segments  
**ls -a**: list segments of all  
**ls -l**: list directory with long listing  
**ls -al**: list directory of all with long listing

[파일 생성] **touch**: create a file  
**mkdir**: make a directory

[파일 제거] **rm**: remove a file  
**rm -rf**: remove forcibly including directory and the files in it

NEXT X LIKELION

# 지금까지 배운 Git 명령어 복습

실습을 위한 Repository 만들기

## 2.CLI & Git 복습

### <git 초기 설정>

[전역 사용자명/이메일 구성하기] `git config --global user.name ${user.name}`  
`git config --global user.email ${user.email}`  
`git config --list`

[새로운 저장소 초기화] `git init`

[저장소 복제] `git clone ${깃헙 주소}`

[Local & Remote 저장소 연결] `git remote add origin ${본인 레포 주소}`  
`git remote -v`

### <git 파일 스테이징 & 푸시>

[git 스테이징 영역에 새로운 파일 추가] `git add ${file_name}`  
`git status`

[git 스테이징 영역에 존재하는 파일 커밋하기] `git commit -m "${commit message}"`  
`git log`

[Github 레포에 코드 올리기] `git push origin ${branch name}`

# 지금까지 배운 Git 명령어 복습

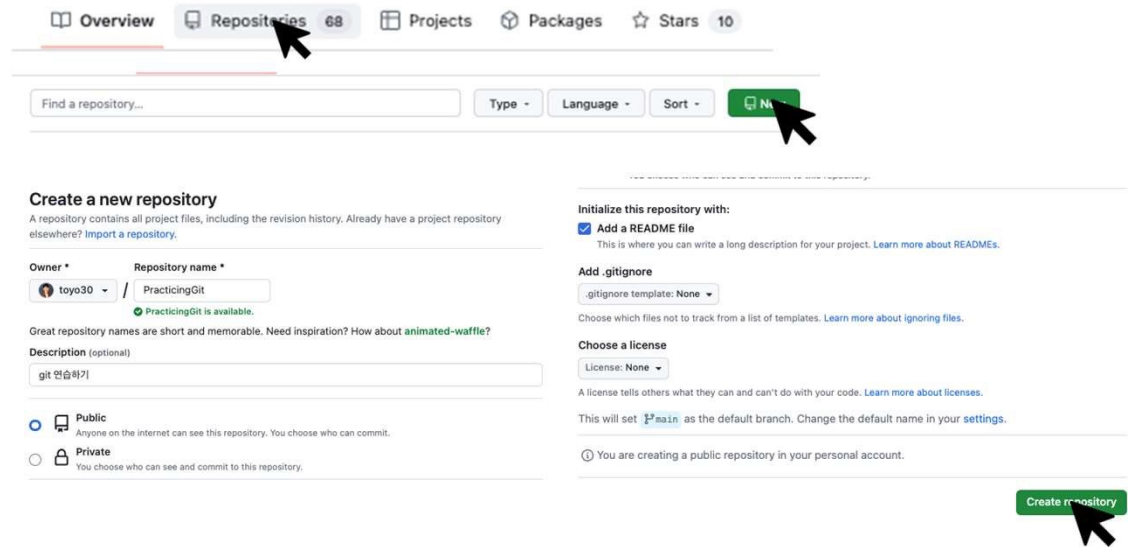
실습을 위한 Repository 만들기

## 2.CLI & Git 복습

실습을 위한 remote Repository를 만들어보자.

원격 저장소, github에서 Repository만드는 것을 말한다. 내 컴퓨터 저장소는 local repository라고 한다.

1. Github 접속 및 로그인
2. Repositories 클릭 및 접속
3. New 클릭 및 접속
4. 이름 설정 "PracticingGit"+ 설명 "git 연습하기"
5. 리드미 추가하기(Add a ReadeMe file 체크)
6. Create Repository 생성하기



# 지금까지 배운 Git 명령어 복습

Remote Repository와 Local Repository 연결하기

2.CLI & Git 복습

## Remote Repository와 Local Repository를 연결하기

원격 저장소(Github에 있는 저장소)와 로컬 저장소(내 컴퓨터 저장소)를 연결해보자.

### 1. git clone \${remote Repository 주소}

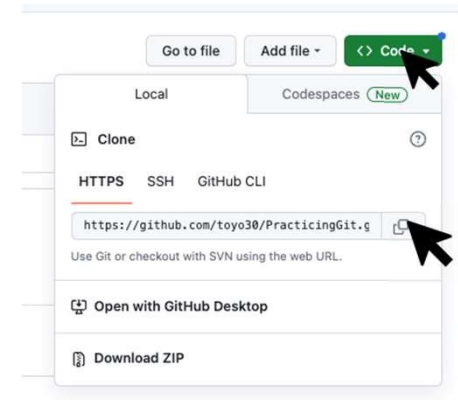
상위에 git이 하나도 없는 폴더여야 해. Desktop 폴더(바탕화면)나, 사용자 폴더에 만들어야 해

### 2. cd PracticingGit

👉 여기까지 했다면, 연결 성공

git remote -v로 연결된 remote repository 확인 가능  
code . 으로 해당 폴더 열기 가능

```
→ PracticingGit git:(main) git remote -v
origin https://github.com/toyo30/PracticingGit.git (fetch)
origin https://github.com/toyo30/PracticingGit.git (push)
→ PracticingGit git:(main) code .
```



```
→ ~ git clone https://github.com/toyo30/PracticingGit.git
'PracticingGit'에 복제합니다...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
오브젝트를 받는 중: 100% (3/3), 완료.
→ ~ cd PracticingGit
→ PracticingGit git:(main)
```

NEXT X LIKELION

# 지금까지 배운 Git 명령어 복습

git clone과 git remote add origin

둘 다 처음 연결할 때 하는 것 같은데...

👉 잠깐 git clone과 git remote add origin의 차이가 뭘까?

## git clone

- Remote repository의 모든 데이터를 복사하여 Local repository에 새로운 git 저장소를 생성
- Remote repository의 모든 커밋 히스토리, 브랜치, 태그 등이 포함 됨.
- 자동으로 Local에 Remote repository가 origin이라는 이름으로 등록됨

## git remote add origin

- Local repository에 Remote repository를 연결함
- origin 은 remote repository의 별칭(별명), origin을 통해서 local repository와 remote repository의 통신이 가능해짐

👉 git clone은 복제, git remote add origin 연결 이구나

복제하면 연결까지 미리 다 되는구나!

# 지금까지 배운 Git 명령어 복습

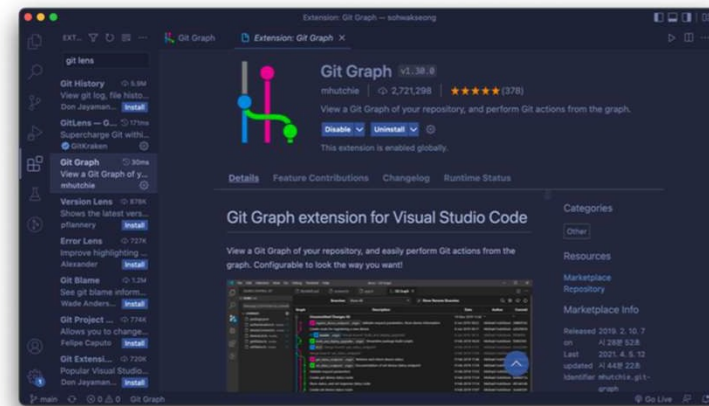
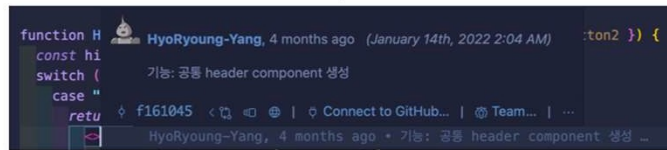
Git에서 자주 사용되는 VS extension

## 2.CLI & Git 복습

Git Lens & Git Graph 설치하기



Ex)



Ex)



Git Lens: <https://usingu.co.kr/frontend/vscode/vscode-gitlens-%EC%82%AC%EC%9A%A9/>

Git Graph: <https://jhyeok.com/vscode-git-graph/>



## 잠깐! Push Error

push 할 때 가장 많이 에러가 발생하는 두가지 케이스

### 1. 원격 저장소와 로컬 저장소 간의 동기화 문제

⇒ git pull 명령어로 원격 저장소의 버전을 가져온 후 다시 push!

### 2. 브랜치 이름 문제

=> main이 아니라 master일 수도 있고, master가 아니라 main일 수 있어요!

**다 해봤는데도 에러가 난다면 .git 폴더 삭제 후 다시 처음부터 push!**

Git push -f origin main. (강제 push)는 정말 최후의 보루...

# 잠깐! 커밋 메시지 쓰는 법

Commit 메시지 7규칙

- 1.제목과 본문을 한 줄 띄어 구분
- 2.제목은 50자 이내
- 3.제목 첫 글자는 대문자
- 4.제목 끝에 마침표 x
- 5.제목은 명령문으로, 과거형 x
- 6.본문의 각 행은 72자 이내 (줄바꿈 사용)
- 7.본문은 어떻게 보다 무엇을, 왜에 대하여 설명

## 잠깐! 커밋 메시지 쓰는 법

기본 포맷

```
<type>: <subject> //필수  
(한 줄 비우기)  
<body> //생략 가능  
(한 줄 비우기)  
<footer> //생략 가능
```

fix: Safari에서 모달을 띄웠을 때 스크롤 이슈 수정

모바일 사파리에서 Carousel 모달을 띄웠을 때,  
모달 밖의 상하 스크롤이 움직이는 이슈 수정.

resolves: #1137

# 잠깐! 커밋 메시지 쓰는 법

## 커밋 타입 분류

저는 보통 feat, fix, refactor만 써요..

- feat : 새로운 기능 추가, 기존의 기능을 요구 사항에 맞추어 수정
- fix : 기능에 대한 버그 수정
- build : 빌드 관련 수정
- chore : 패키지 매니저 수정, 그 외 기타 수정 ex) .gitignore
- ci : CI 관련 설정 수정
- docs : 문서(주석) 수정
- style : 코드 스타일, 포매팅에 대한 수정
- refactor : 기능의 변화가 아닌 코드 리팩터링 ex) 변수 이름 변경
- test : 테스트 코드 추가/수정
- release : 버전 릴리즈

# 지금까지 배운 Git 명령어 복습

Git Add, Git Commit, Git Push, Git Pull 연습해보기

다음과 같은 상황에서 지금까지 배운 Git 명령어를 연습해보자!



이번에 나 인턴으로 인스타그램에 합류했어. 과제로 인스타그램을 클론해보래.

인스타그램에는 다음과 같은 기능이 있어.

“로그인하기 기능”, “좋아요 기능”, 피드 올리기 기능”, “

팔로우 하기 기능”, “차단하기 기능”, “댓글 달기 기능”

다음과 같은 기능을 제작하고,

기능 단위로 **add, commit, push** 해보자!



👉 기능단위로 커밋한다면, 효율적으로 코드를 관리할 수 있을 거야!

# 실습1. 지금까지 배운 Git 명령어 복습

## 2.CLI & Git 복습

Git Add, Git Commit, Git Push, Git Pull 연습해보기

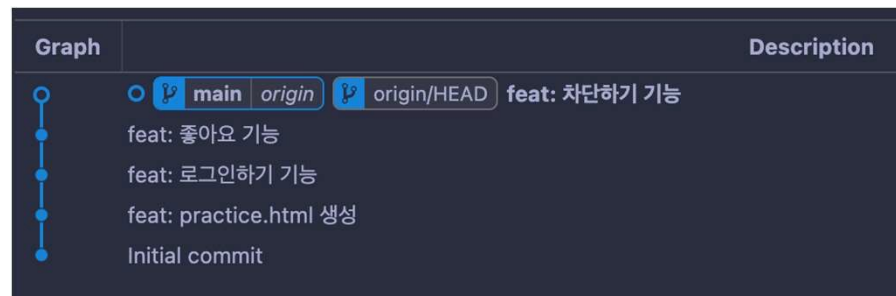
### 기능별로 push를 연습해보자

1. Practice.html만 생성하고, add, commit, push해보자.
2. <h1>로그인하기 기능</h1>만 생성하고, add, commit, push
3. <h1>좋아요 기능</h1>만 생성하고, add, commit, push
4. <h1>차단하기 기능</h1>만 생성하고, add, commit, push

HTML 기본 형식은 ! 입력하고 enter 누르면 자동 생성!

```
practice.html U x
practice.html > ...
1 <!DOCTYPE html>
2 <html lang="ko">
3 <head>
4   <meta charset="UTF-8" />
5   <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7   <title>Document</title>
8 </head>
9 <body></body>
10 </html>
11
```

```
9+ <body> You, 1 minute ago • Uncommitted changes
10+ <h1>로그인하기 기능</h1>
11+ </body>
```



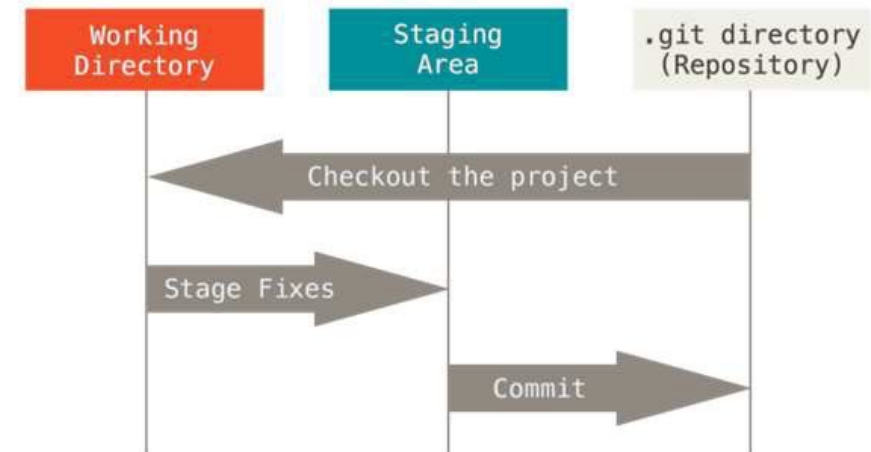
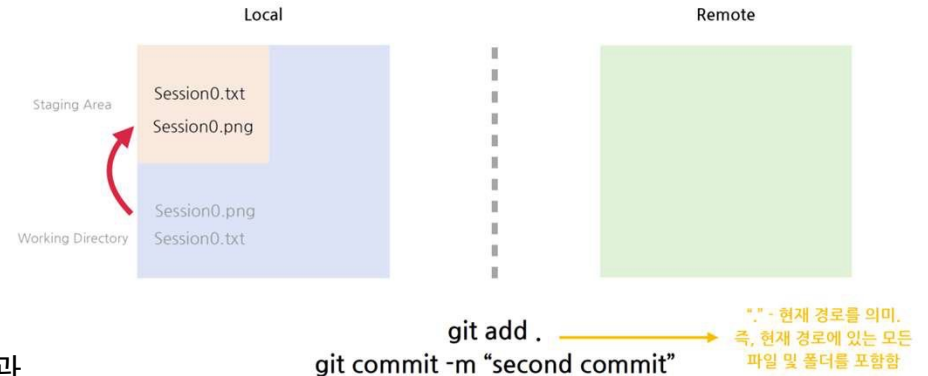
# 지금까지 배운 Git 명령어 복습

Git Add, Git Commit, Git Push, Git Pull 연습해보기

## git 상태 및 공간 변화 알아보기

- `git add .`  
working Directory에 있는 녀석들 전부가 staging Area에서 관리됨.
- `git commit -m ${메세지}`  
staging Area에 있는 녀석들이 local repository로 옮겨짐  
포장 패키징 = 메세지라는 박스에 보관됨
- `git push`  
Local repository에 있는 녀석들이 Remote Repository로 옮겨짐

## 2.CLI & Git 복습



# 3. Git 심화 🙌

---

Git Reset & Git Revert

Git Branch

Git Pull-Request



# HEAD란?

최종 커밋 상태

커밋을 중첩적으로 했을 경우 마지막 상태!

Push 했을 때 github에 반영되는 가장 최종 상태라고 이해하면 좋음

```
● PS C:\Users\kdhyeok\Desktop\PracticingGit\PraticingGit> git log
commit 445e763cd39a6f650fbe46c50d67ff01d26b1336 (HEAD -> main)
Author: cucumber5252 <87911068+cucumber5252@users.noreply.github.com>
Date: Sat Mar 9 23:41:47 2024 +0900

Commit 1
```

# Git Reset & Revert

Git Reset & Revert의 필요성

## Reset과 Revert는 왜 필요할까?

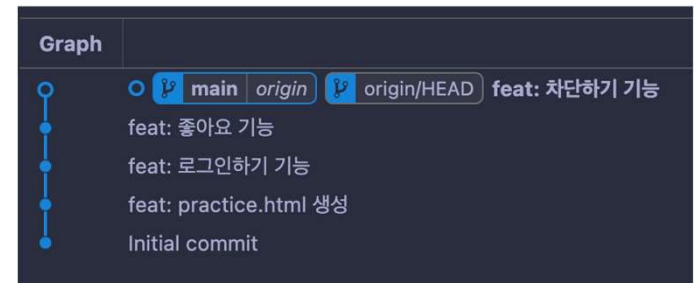


인스타그램을 열심히 만들고 있었더니, 인스타그램의 CEO 아담모세리가 말한다.

**“User가 차단하기 기능을 필요로 하는지 모르겠어.**

**다음 배포에서 차단하기 기능을 빼서 배포해”**

아니, 열심히 만들었는데, 당장 빼라고? 미친 거 아니야?



개발을 하다보면, 위와 같은 상황이 생긴다.

결국 소프트웨어는 유저를 만족시키는 것이기 때문에 계획과 다른 요구사항을 만족시켜야 한다.

이 상황에서 우리는 어떻게 특정 기능만 뺄 수 있을까?

# Git Reset & Revert

Git Reset & Revert 개념

## Reset



- 현재의 HEAD를 특정 커밋으로 이동시키는 명령어
- 이전의 커밋 상태로 코드를 되돌릴 수 있음
- 완전히 되돌아가는 것이기에 commit 이력에 남지 않음
- 주로 로컬에서만 사용(되돌려버리는 건 위험)
- --soft, --mixed, --hard 3가지 옵션이 있음

## Revert



- 특정 커밋의 변경사항을 취소함
- 특정 커밋의 내용만 쏙 빼서 변경함
- commit 이력에 남음
- 이전에 commit한 내역도 변경가능

# Git Reset & Revert

## Git Reset

### 3. Git 심화

## Reset

- `git reset -${option} ${커밋id}`
- C4의 상황에서 C2의 커밋으로 Head를 변경함

주의사항: Local Repository에서만 사용 권고  
다른 코드를 삭제하거나 다른 코드와 충돌할 수 있음  
즉, Push하기 이전에만 사용하는게 좋음



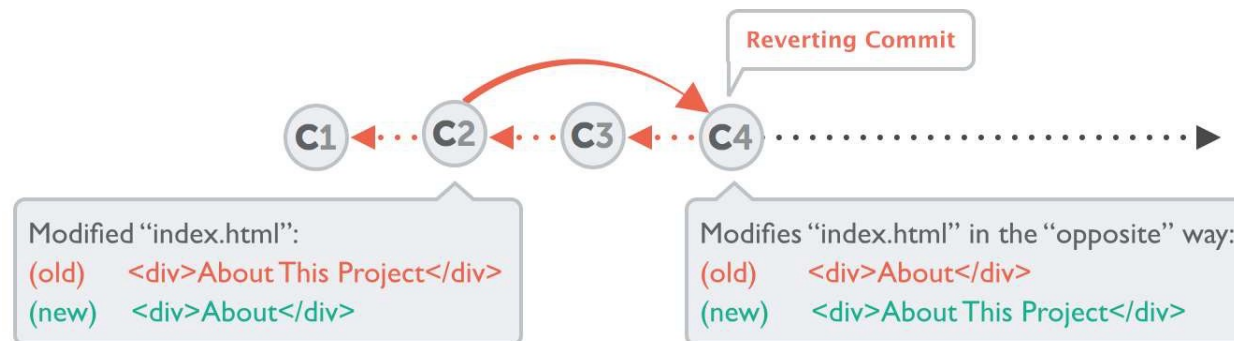
# Git Reset & Revert

## Git Revert

### 3.Git 심화

## Revert

- `git revert ${커밋id}`
- C2에 해당하는 변경사항을 되돌리고, C4라는 새로운 commit 을 남김



# Git Reset & Revert

Git Reset 명령어

--soft 또는 --mixed 또는 --hard

## Reset 명령어 정리

**git reset \${원하는 옵션} \${commit Id 또는 HEAD 형식}**

reset --soft: **git reset --soft \${commit ID}**

reset --mixed: **git reset --mixed \${commit ID}**  

 = git reset \${commit ID}

reset --hard: **git reset --hard \${commit ID}**

reset HEAD~숫자: **git reset HEAD~10**

reset HEAD^: **git reset HEAD^**

reset HEAD: **git reset HEAD**

soft → commit 취소하기

Commit한 파일들을 staging area로 돌린다.

mixed → (default값) commit 취소 + add 취소

Commit한 파일들을 working directory로 돌린다.

hard → commit 취소 + add 취소 + 변경된 파일 삭제

Commit한 파일들을 working directory에서 삭제한다.

Head~\${취소할 커밋 수} → 현재로부터 숫자만큼 commit 취소

HEAD^ → 가장 최근 커밋이 취소된다.

HEAD → 변경된 파일들을 unstaged 상태로 되돌린다.

# Git Reset & Revert

Git Reset 상황별 명령어

## Reset 상황별 명령어 정리

--mixed는 default 값이기 때문에 생략

git add 취소하기 → : git reset \${원하는 파일, 전체 파일일 경우 공란으로}

git commit 취소하기 →: git reset \${원하는 옵션} \${commit Id 또는 HEAD}

git push 취소하기 →: git reset --hard \${commit Id 또는 HEAD}

git 다시 commit 후에

git push origin \${레포지토리 이름} -f

Commit 취소한 후에 강제 push 한다고  
생각하시면 편합니다!

# Git Reset & Revert

## Git Reset 실습

### 실습2. 차단하기 기능을 reset해보자

- `git reset --hard ${좋아요 기능 commit id}`  
HEAD를 어디로 돌릴 것인가
- 현재 local repository에서만 reset, remote repository에 올리려면 안 올라감
- `git push -f origin main` Remote repository가 앞서 있어도, Local repository로 맞춰서 강제로 올리기

Graph	Description	Date	Author	Commit
○ main origin	origin/HEAD feat: 차단하기 기능	14 May 2023 20:14	Minky Baek	25961f57
●	feat: 좋아요 기능	14 May 2023 20:00	Minky Baek	29f55de5
●	feat: 로그인하기 기능	14 May 2023 19:55	Minky Baek	4c4b2bce
●	feat: practice.html 생성	14 May 2023 19:48	Minky Baek	5f73c582
●	Initial commit	14 May 2023 19:28	Minky Baek	9d0e0dd1

Graph	Description
○ origin/HEAD origin/main	feat: 차단하기 기능
○ main	feat: 좋아요 기능
●	feat: 로그인하기 기능
●	feat: practice.html 생성
●	Initial commit

Graph	Description
○ main origin	origin/HEAD feat: 좋아요 기능
●	feat: 로그인하기 기능
●	feat: practice.html 생성
●	Initial commit



# Git Reset & Revert

## Git Revert 실습

## 3.Git 심화

### 실습3. 로그인하기 기능을 revert해보자

- `git revert ${로그인하기 기능 commit id}`  
무엇을 뺄 것인가?
- 독립적인 기능이 아닌 경우, conflict가 생김
- 내가 빼고 싶은 기능만 새롭게 만들어서 commit을 하면 됨
- Accept Current Changes 클릭 후,  
<h1>좋아요 기능</h1>을 수동으로 삭제하고,  
Add => commit => push

Graph	Description	Date	Author	Commit
o main origin	origin/HEAD feat: 좋아요 기능	14 May 2023 20:00	Minky Baek	20f5d45
o	feat: 로그인하기 기능	14 May 2023 19:55	Minky Baek	4c462bce
o	feat: practice.html 생성	14 May 2023 19:48	Minky Baek	5173c582
o	Initial commit	14 May 2023 19:28	Minky Baek	9d0e0dd1

```
* PracticingGit git:(main) git revert 4c462bce
자동 병합: practice.html
충돌 (내용): practice.html에 병합 충돌
error: 다음을 되돌릴 (revert) 수 없습니다: 4c462bce... feat: 로그인하기 기능
힌트: After resolving the conflicts, mark them with
힌트: "git add/rm <paths>", then run
힌트: "git revert --continue".
힌트: You can instead skip this commit with "git revert --skip".
힌트: To abort and get back to the state before "git revert",
힌트: run "git revert --abort".
```

```
</head>
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<----- HEAD (Current Change)
<body>
  <h1>로그인하기 기능</h1>
  <h1>좋아요 기능</h1>
</body>
=====
<body></body>
>>>>>> parent of 4c462bce (feat: 로그인하기 기능) (Incoming Change)
</html>
```

## + Git Conflict 해결하기

Git Conflict git 충돌

### Git conflict 이전 버전과 지금 버전의 충돌

- Accept Current Changes 현재 변경사항  
누르면 현재 변경사항만 남음
- Accept Incoming Changes 다가오는 변경사항  
merge를 요청했거나, revert를 요청한 코드에서 새롭게 적용될 변경사항  
누르면 새롭게 적용될 변경사항만 뜸
- Accept Both Changes 두 변경사항을 동시에 남김
- Compare Changes 비교하여 하나하나 선택가능

```
</head>
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<< HEAD (Current Change)
<body>
  <h1>로그인하기 기능</h1>
  <h1>좋아요 기능</h1>
</body>
=====
<body></body>
>>>>>> parent of 4c462bc (feat: 로그인하기 기능) (Incoming Change)
</html>
```

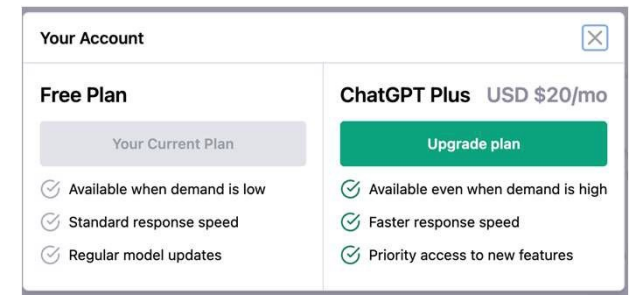
# Git Branch

Git Branch의 필요성

## Branch는 왜 필요할까?

- 병렬적인 작업, 코드의 안정성, 코드 리뷰
- 하나의 프로젝트에서 독립적인 여러 가지 코드 관리가 필요하거나, 코드를 효율적이고, 안정적으로 관리하기 위해서 필요하다
- Ex) Production/develop/testing 분리, A/B 테스트 등등

3. Git 심화



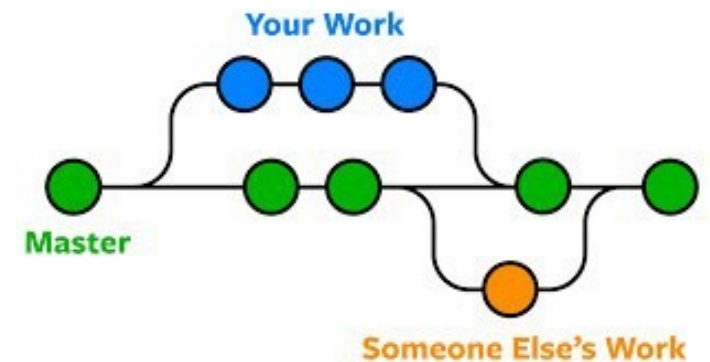
NEXT X LIKELION

# Git Branch

Git Branch 개념

## Git Branch

- 같은 프로젝트를 독립적으로 개발할 수 있는 저장소(작업영역)
- 커밋과 커밋 사이를 가볍게 이동할 수 있는 포인터 같은 개념
- Branch = 나뭇가지: 독립적인 작업을 위해 줄기를 만드는 것
- 하나의 코드 관리 흐름



3. Git 심화

# Git Branch

Git Branch 명령어

## Branch 명령어 정리

Branch 생성: **git branch \${브랜치 이름}**

Branch 이동: **git checkout \${브랜치 이름}**

Branch 생성과 이동: **git checkout -b \${브랜치 이름}**

Branch 삭제: **git branch -d \${브랜치 이름}**

현재 Branch가 무엇인지 확인: **git branch('Q'로 나가기)**

모든 Branch가 확인: **git branch -r('Q'로 나가기)**

Branch 합치기: **git merge \${브랜치 이름}**

현재 브랜치에서 합치고 싶은 브랜치 이름 입력하여 병합

## 3. Git 심화

최초 repository 생성 시 브랜치 이름은

→ 'master' or 'main'

대부분의 상황에서 새로운 브랜치를 생성할 때는

→ git checkout -b \${브랜치 이름}를 사용

기존 것을 똑같이 복사한 채로 이동하기 때문이야.

(git branch \${브랜치 이름}은 생성만 해줌)

git checkout을 통해 브랜치 이동하여 독립적인 작업가능

🔥 이동하기 전 working directory에 있는 내역은 사라지고, 바뀐 브랜치의 최신 커밋이

working directory에 반영됨.

# Git Branch

Git Branch 실습

## Branch 실습 상황 설명



“아니, 도대체 사람들이 “차단기능”을 좋아하는지 안 좋아하는지 모르겠어. 그래서 User 100명에게는 “차단기능”을 배포하고, 또 다른 User 100명에게는 차단하기 기능을 빼서 배포해야겠어.

before

??? XXXX!!! 아까는 다 빼라며?!

왜 이래라저래라래야! 🤔 😊 🤔 🤔 🤔

After

오키오키!! Branch를 활용해서  
차단기능이 있는 브랜치와 아닌 브랜치를  
나눠서 관리하면 되겠다 👍

# Git Branch

## Git Branch 실습

### 실습4. Branch block을 만들어보자

- `git checkout -b block` (`git checkout -b ${브랜치 이름}`)
- 차단하기 기능 추가
- add, commit 하고, branch 확인
- 차단하기 기능이 있는 block과 차단하기 기능이 없는 main이 나뉘어져있음.
- `git checkout main`과, `git checkout block`을 왔다갔다하면, 유무가 확연하게 차이남

```
PracticingGit git:(block)
```

Graph	Description
	Revert "feat: 로그인하기 기능" feat: 좋아요 기능 feat: 로그인하기 기능 feat: practice.html 생성 Initial commit

```
10 <h1>로그인하기 기능</h1>
11 <h1>차단하기 기능</h1>
```

Graph	Description
	feat: 차단하기 기능 Revert "feat: 로그인하기 기능" feat: 좋아요 기능 feat: 로그인하기 기능 feat: practice.html 생성 Initial commit

# Git Branch

Git Merge 실습

## Branch Merge 실습 상황 설명



이거 A/B 테스트를 해보니까, “차단기능”있는게 유저반응이 더 좋은데?  
다른 유저들 모두에게도 “차단기능”을 넣자!

before

잉?! 아까 다 넣었으면 되잖아!!!

왜 나눈거야? 🤔 😂 🤔 🤔 🤔

After

오! A/B 테스트의 결과가 그렇게 나왔군.

Merge를 통해서 두 개의 브랜치를 합쳐서 관리하면 되겠다. 👍



# Git Branch

## Git Merge 실습

### 3. Git 심화

## 실습5. branch block을 branch main에 Merge

- 추가하고 싶은 branch로 이동

git checkout main

- main branch에 block branch를 병합

git merge block

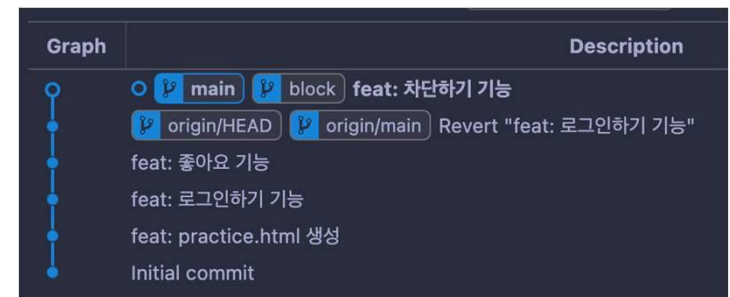
- Branch main과 Branch block이 같은 head를 가르킴

- main 브랜치에서

git push origin main

```
→ PracticingGit git:(block) git checkout main
'main' 브랜치로 전환합니다
브랜치가 'origin/main'에 맞게 업데이트된 상태입니다.
→ PracticingGit git:(main) git merge block
업데이트 중 7e9fbec..2c7c8c6
Fast-forward
 practice.html | 1 +
 1 file changed, 1 insertion(+)
→ PracticingGit git:(main)
```

```
10     <h1>로그인하기 기능</h1>
11     <h1>차단하기 기능</h1>
```



# Git Pull-Request

Git Pull-Request 필요성

## Git Pull-Request는 왜 필요할까?



“나인턴” 열심히 개발했더니, 클론말고 진짜 기능을 만들어보래!

인스타그램에서 정직원으로 합류한거야! 하하하!!

아까 만들었던, 차단기능있잖아, 요즘엔 광고가 너무 많아서 광고계정이 소유하거나, 만드는 다른 계정을 다 차단하는 슈퍼 차단이 필요해.

그래서 내가 만들었더니, 고민이 돼. 이 기능을 내가 실제 제품에 합쳐도 되는거야? 내가 Merge한 거 때문에 인스타그램 실제 production에 문제가 생기는 거 아니겠지?

👉 그러면 내가 코드를 합치지 말고, 합쳐 달라고 제안해보는거야!

# Git Pull-Request

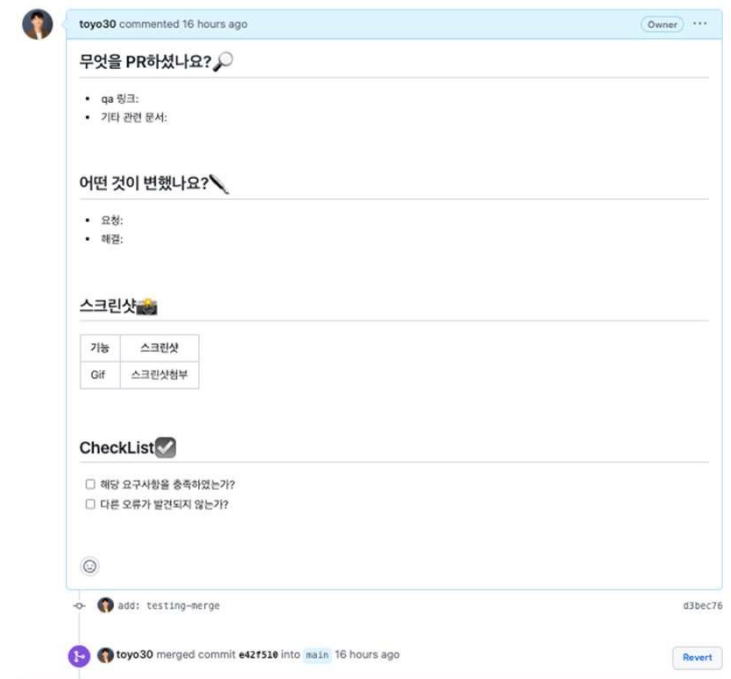
Git Pull-Request 개념

3. Git 심화

## Git Pull Request

- 코드 변경 사항을 다른 사람에게 알리고 그들의 검토와 승인을 받는 방법
- 이 기능은 주로 GitHub 같은 원격 저장소에서 활용되며, 협업을 하는 개발 환경에서 사용됨
- Pull을 요청(Request)하다

git pull은 다른 저장소 혹은 다른 branch의 코드를 가져오는 git fetch와 코드를 합치는 git merge의 조합이다.



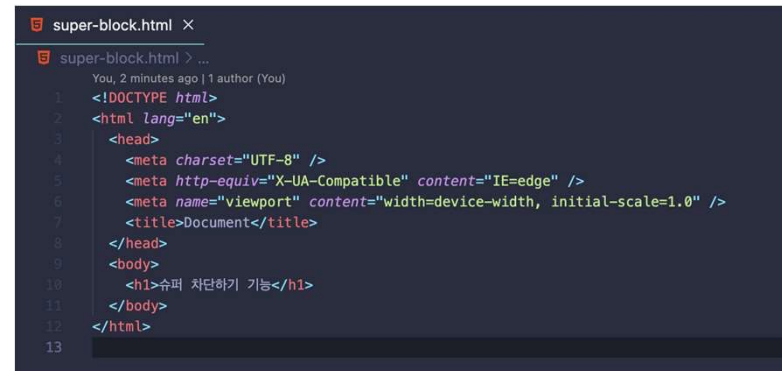
NEXT X LIKELION

# Git Pull-Request

Git Pull-Request 실습

## 실습6-1. branch block에서 super-block.html을 만들어보자.

- git checkout block (git checkout \${브랜치 이름})
- super-block.html 생성
- <h1>슈퍼 차단하기 기능</h1> 추가
- git add, commit, push origin block



```
super-block.html x
super-block.html > ...
You, 2 minutes ago | 1 author (You)
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7     <title>Document</title>
8   </head>
9   <body>
10    <h1>슈퍼 차단하기 기능</h1>
11  </body>
12 </html>
13
```

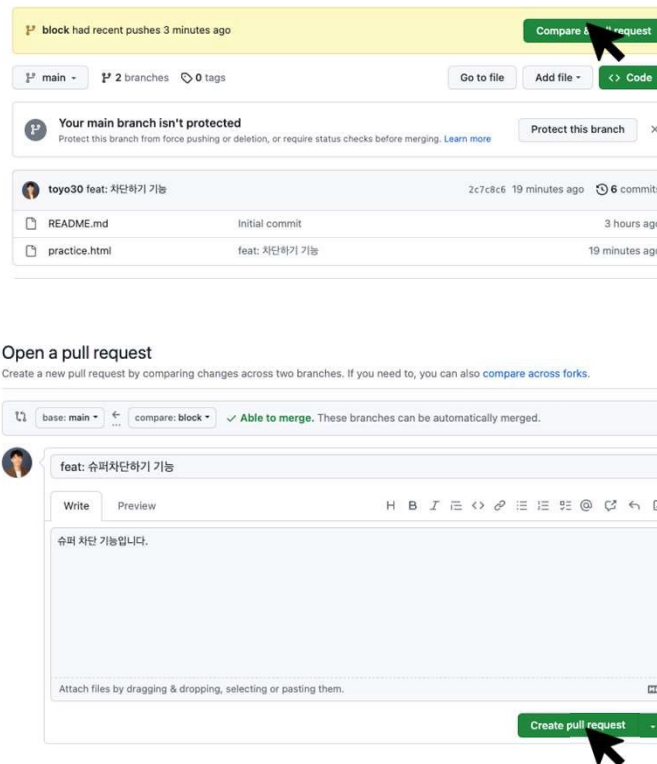
# Git Pull-Request

## Git Pull-Request 실습

### 3. Git 심화

## 실습6-2. block branch에서 main branch로 pull-request를 해보자

- 내 레포지토리 접속하기
- Pull-request 생성하기
- Pull-request 설명적기
- Create pull-request



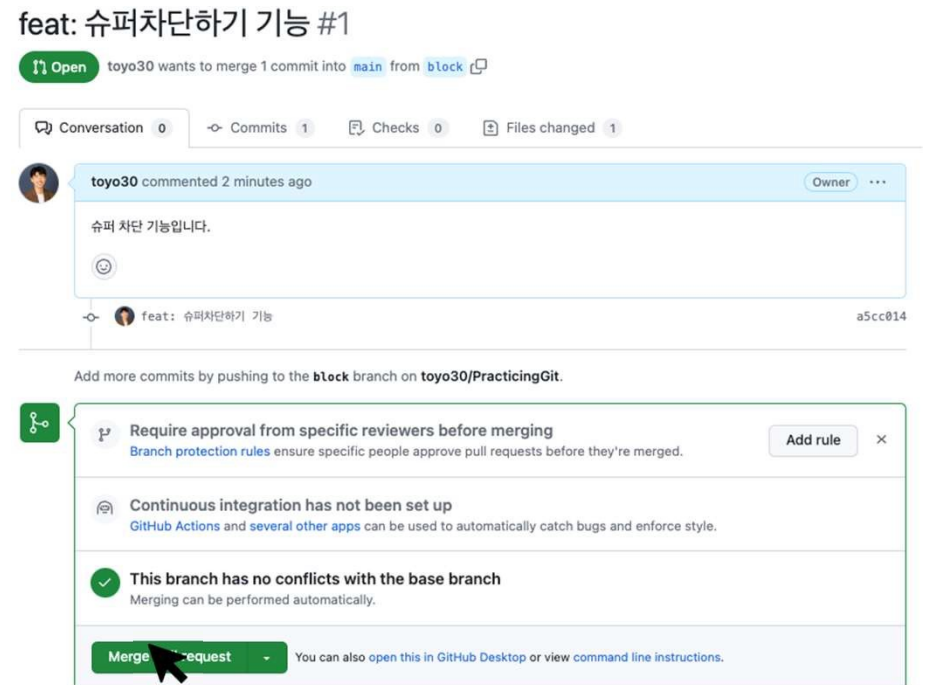
# Git Pull-Request

Git Pull-Request 실습

3. Git 심화

## 실습6-3. 코드를 검토하고, 관리자 입장에서 Merge를 해보자.

- commit 확인하기
- 리뷰 달기
- 코드 검토 후 Merge Pull-request



NEXT X LIKELION

# Git Pull-Request

Git Pull-Request 실습

3. Git 심화

## Commits

main	All users	All time
Commits on Mar 10, 2024		
좋아요 기능 추가 cucumber5252 committed 17 minutes ago	f6bb33b	<>
로그인하기 기능 추가 cucumber5252 committed 18 minutes ago	5656261	<>
Initial commit cucumber5252 committed 20 minutes ago	Verified c572544	<>

Main 브랜치를 확인해보면

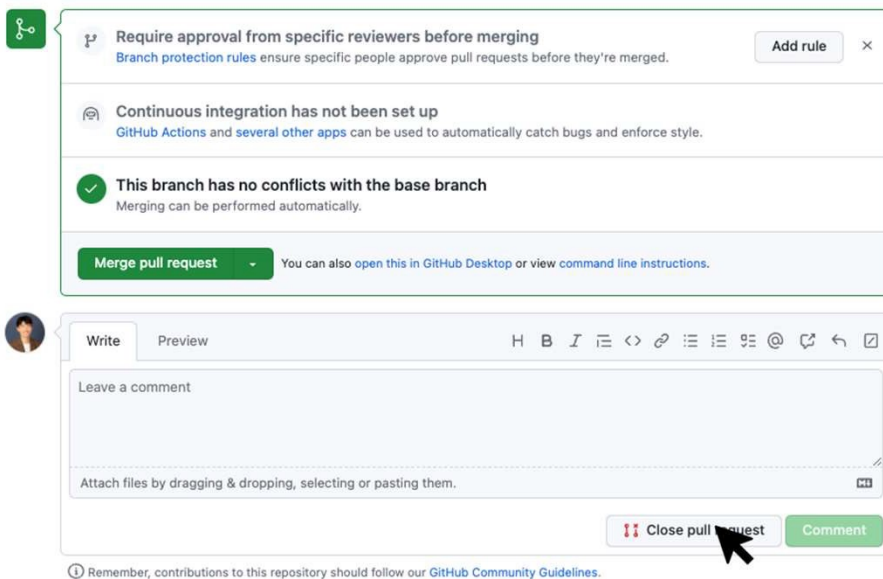
변경사항이 반영된 것을 알 수 있음

NEXT X LIKELION

# Git Pull-Request

Git Pull-Request 실습

3.Git 심화



코드가 마음에 안 들면, close pull request로  
Pull-request를 반영하지 않을 수 있다

NEXT X LIKELION

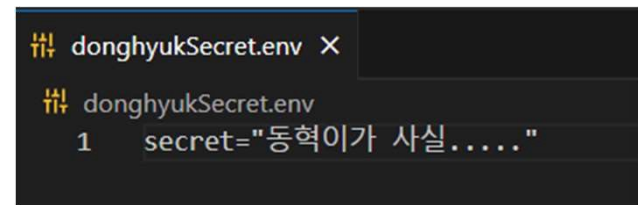


# +추가 git 명령어

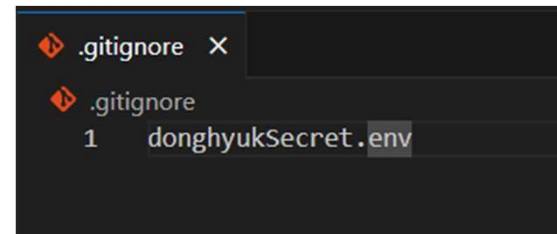
## Git ignore

### Git ignore

- git 파일 상태 중 하나
- Git에 의해서 관리하지 않겠다는 의미
- 변경사항을 추적할 필요 없는 것들을 Ignore 처리하면 된다.
- .gitignore을 생성한 후에 파일이름이나, 경로 + 이름을 설정하면 git에 의해서 관리되지 않는다.



```
donghyukSecret.env X
donghyukSecret.env
1 secret="동혁이가 사실....."
```



```
.gitignore X
.gitignore
1 donghyukSecret.env
```

프로젝트에 맞는 gitignore을 만들어주는 사이트

<https://www.toptal.com/developers/gitignore>

NEXT X LIKELION

## 자리배치

**팀별 실습을 위해서  
팀별로 자리에 착석해주세요!**

## 4. Git 팀별 실습

---

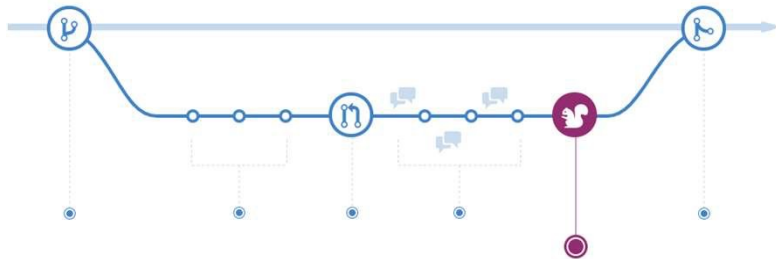
Git branch 관리 방법 3가지 하  
나의 브랜치에서 Git 팀별 실습  
여러 브랜치에서 Git 팀별 실습

# Git branch 관리 방법 3가지

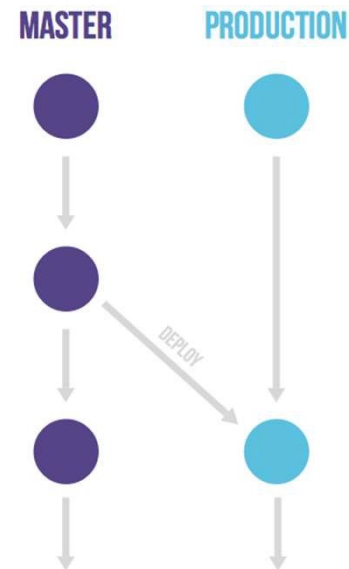
## Git branch 전략

### Git으로 어떻게 협업을 할까?

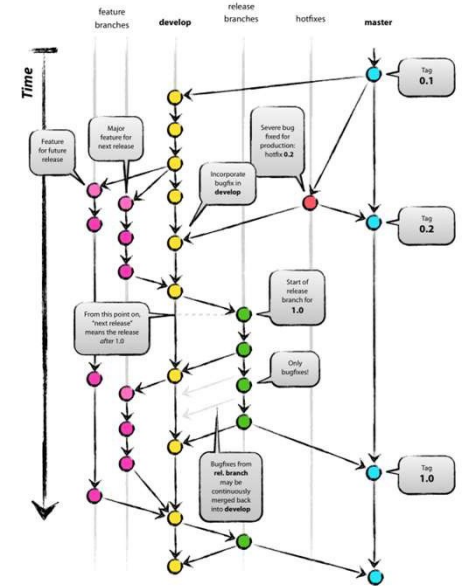
- 다양한 브랜치 관리 방법이 존재함.
- 3가지 예시 중 Git Flow 방식과 GitLab Flow 방식을 연습해보자



Github flow



Gitlab flow



Git flow

브랜치 관리 전략에 대해: <https://tecoble.techcourse.co.kr/post/2021-07-15-git-branch/>

# 과제1. 하나의 브랜치에서 Git 팀별 실습

## Git 팀별 실습

Github flow 전략(하나의 브랜치에서 협업하기)

1. 팀장이 하나의 레포지토리를 만든다.  
(public, readme 파일 추가)
2. 레포지토리 Collaborators에 팀원을 추가한다.
3. 팀장은 practice.html 파일 기본 구조를 만들어 push한다.
4. 각 팀원은 초대된 repository를 clone한다.
5. 각 팀원은 우측의 기능 중 하나를 맡아서 push한다.

Push전 다른 팀원의 커밋 내역을 pull해야 함



## 과제2. 여러 브랜치에서 Git 팀별 실습

### Git 팀별 실습

#### Git flow 전략(여러 브랜치에서 협업하기)

1. 같은 브랜치에서 각 팀원들은 “자기의 이름” or “기능 ” 을 기반으로 한 새로운 브랜치를 만든다.
2. 기능마다 \${새로운 기능}.html 파일을 만들고 git push origin \${브랜치 이름}을 해준다.
3. 각 팀원들은 Remote repository에서 Pull-request를 보낸다.
4. 각 팀원들은 Pull-request에 comment를 1개이상씩 달고, 팀장은 merge를 한다.

#### 인스타 추가기능

메모 기능

스토리 기능

DM 기능

저장 기능

광고 기능

# 개발 과제

시간 내 팀별 실습을 끝내지 못했다면 과제!

## 과제 내용

Gitflow 전략으로 팀원들의 브랜치 병합해보기  
Github flow 전략으로 팀원들의 변경사항 push해보기

## 제출 방법

1. 팀원 이름 및 역할
2. 팀별 레포지토리 링크

## 제출 기한

다음 세션까지 => 3월 14일 (목)

**+ 파이썬 설치도 잊지 말고 꼭!**

## 창업 과제

본인 자기소개 노션 만들기

### 과제 내용

노션에 마련되어있는 템플릿을 사용하여 자유롭게  
본인 자기소개 자료를 마련해주세요!

### 제출 방법

12기 WORKSPACE 노션 템플릿 이용하여 제출

### 제출 기한

창업 세션까지 => 3월 16일 (토)