

Git사용과 Github 사용

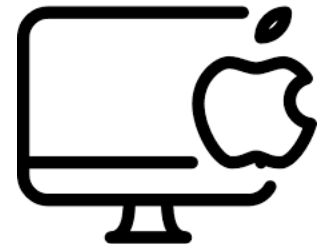
터미널과 기본 리눅스 명령어

- **pwd** : 현재 경로 출력
- **ls** : 현재 폴더의 목록 출력
- **mkdir gitsyudy02** : gitsyudy02 폴더 만들기 (디렉터리)
- **cd gitsyudy02** : gitsyudy02 위치/폴더로 이동
- **.** -> 현재 폴더
- **..** -> 상위 폴더

(cmd는 리눅스 표준 명령어를 안따르므로 안쓰겠습니다)



윈도우: 터미널/ powershell



```
jodonggit03 — jodonghyeon@jodonghyeon-ui-MacBookAir — ../jodonggi...
Last login: Wed Apr 13 16:12:11 on ttys000
[~ ~ git:(master) * ls
#include <stdio.h>                                     Parallels
AndroidStudioProjects                                Pictures
Applications                                           Public
Applications (Parallels)                               Untitled-1.html
Desktop                                                  Untitled.ipynb
Downloads                                               eclipse
IdeaProjects                                             eclipse-workspace
Library                                                  opt
Movies                                                  quiz 복사본.txt
Music                                                    quiz.txt

[~ ~ git:(master) * cd Documents/jjodonggit03
[~ jjodonggit03 git:(master) ls
[~ jjodonggit03 git:(master) git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
[~ jjodonggit03 git:(master)
```

맥 : terminal / iTerm


GIT : VCS의 일종

- VCS: version control system (코드의 변화를 관리하고 추적하는 SW)
- 코드를 변경, 수정, 테스트 할때 코드는 안정/불안정 상태를 반복
- 시험삼아 작성한 코드가 잘못됐을때 코드를 복귀지점이 있으면 더 안정적으로 개발 가능

➤ 버전 관리 시스템

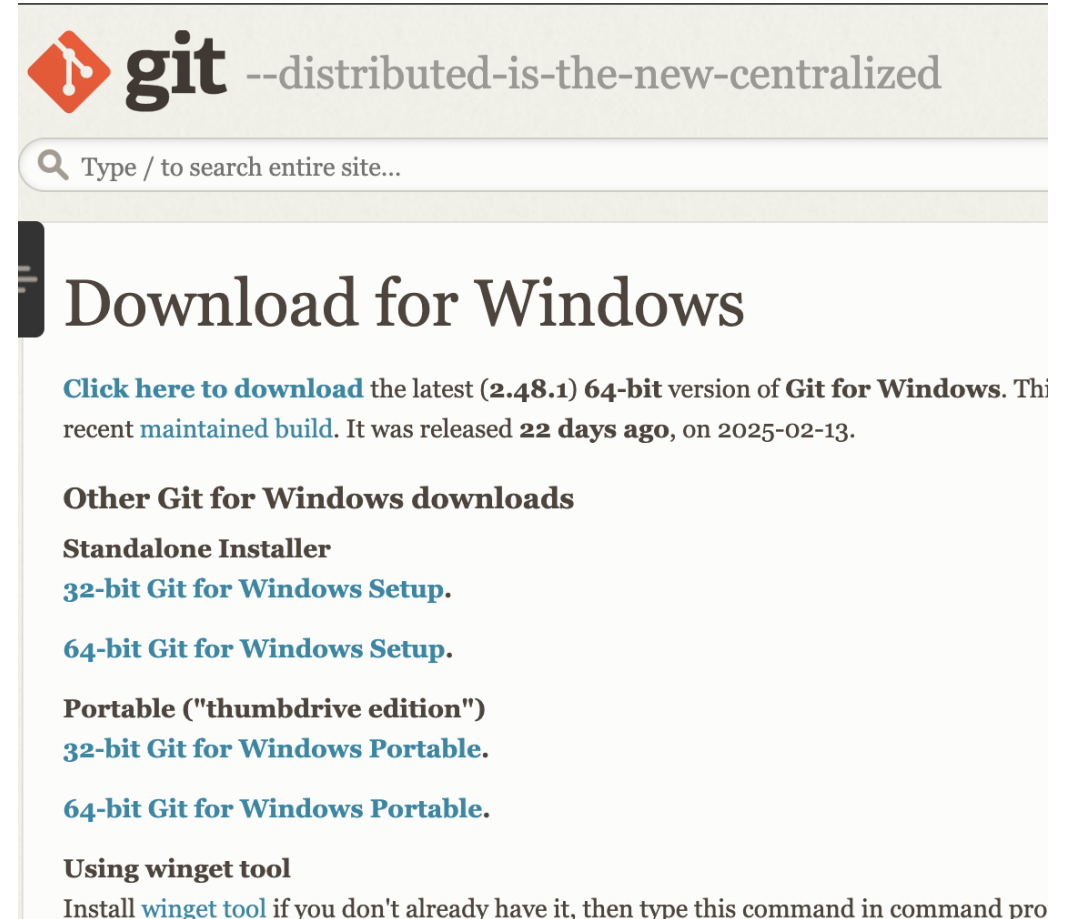
- 다음과 같이 통일성 없이 파일 이름을 만들어 저장했다면 버전을 알아보기 힘들

▼ 그림 1-2 새 파일로 계속 저장

 hello.txt	2019-06-02 오후...	텍스트 문서
 hello_1.txt	2019-06-02 오후...	텍스트 문서
 hello_2 재수정.txt	2019-06-02 오후...	텍스트 문서
 hello_2.txt	2019-06-02 오후...	텍스트 문서
 hello_3.txt	2019-06-02 오후...	텍스트 문서
 hello4.txt	2019-06-02 오후...	텍스트 문서

깃 설치하기 (윈도우)

- <https://git-scm.com/>



The screenshot shows the Git website's "Download for Windows" page. At the top, the Git logo is followed by the tagline "--distributed-is-the-new-centralized". Below this is a search bar with the placeholder text "Type / to search entire site...". The main heading is "Download for Windows". The text below the heading states: "Click here to download the latest (2.48.1) 64-bit version of Git for Windows. This recent maintained build. It was released 22 days ago, on 2025-02-13." Under the heading "Other Git for Windows downloads", there are three sections: "Standalone Installer" with links for "32-bit Git for Windows Setup." and "64-bit Git for Windows Setup."; "Portable ('thumbdrive edition')" with links for "32-bit Git for Windows Portable." and "64-bit Git for Windows Portable."; and "Using winget tool" with the instruction "Install winget tool if you don't already have it, then type this command in command pro".

git --distributed-is-the-new-centralized

Type / to search entire site...

Download for Windows

Click here to download the latest (2.48.1) 64-bit version of Git for Windows. This recent maintained build. It was released 22 days ago, on 2025-02-13.

Other Git for Windows downloads

Standalone Installer

- 32-bit Git for Windows Setup.
- 64-bit Git for Windows Setup.

Portable ("thumbdrive edition")

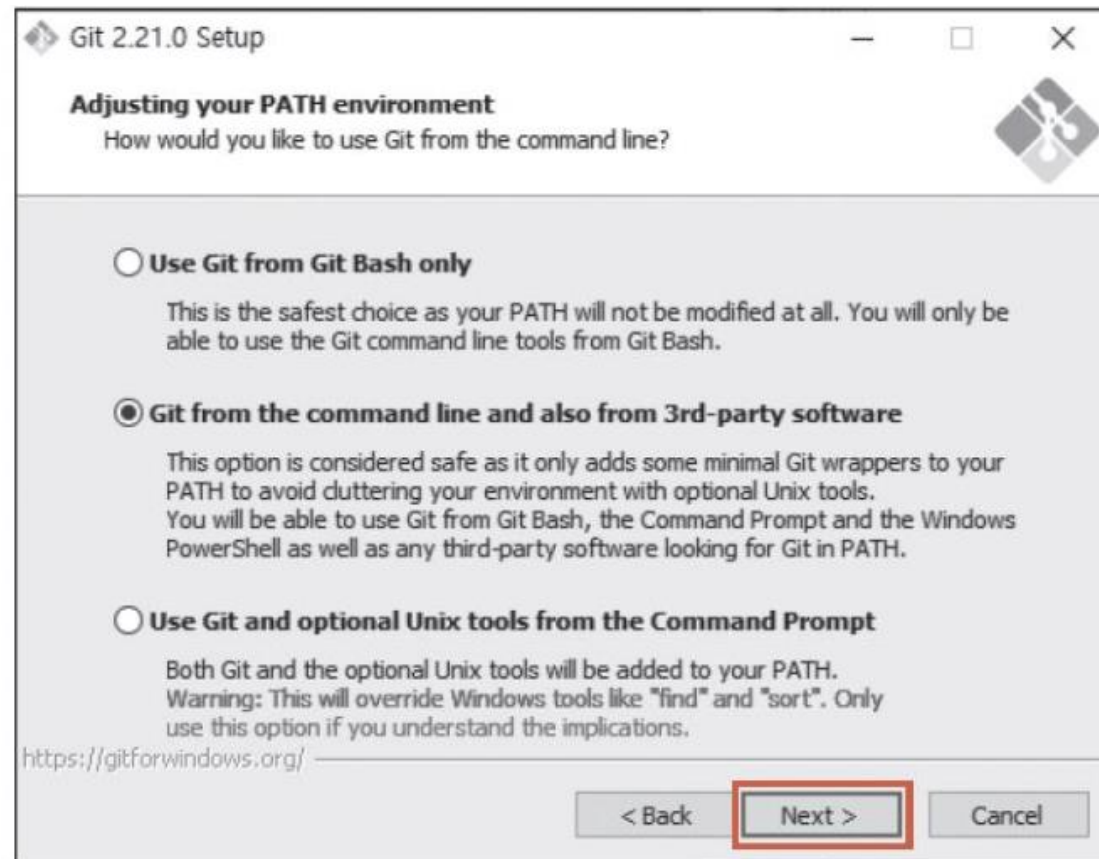
- 32-bit Git for Windows Portable.
- 64-bit Git for Windows Portable.

Using winget tool

Install winget tool if you don't already have it, then type this command in command pro

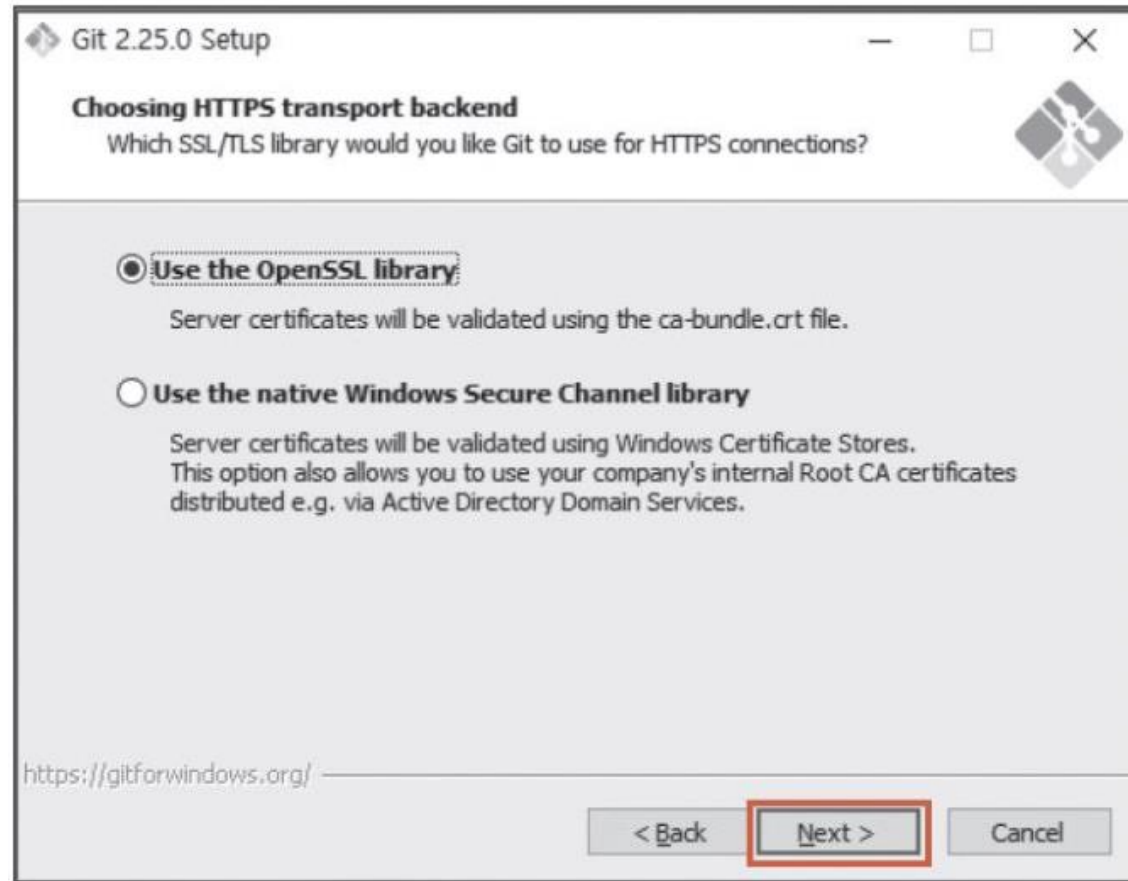
깃 설치하기 (윈도우)

- <https://git-scm.com/>



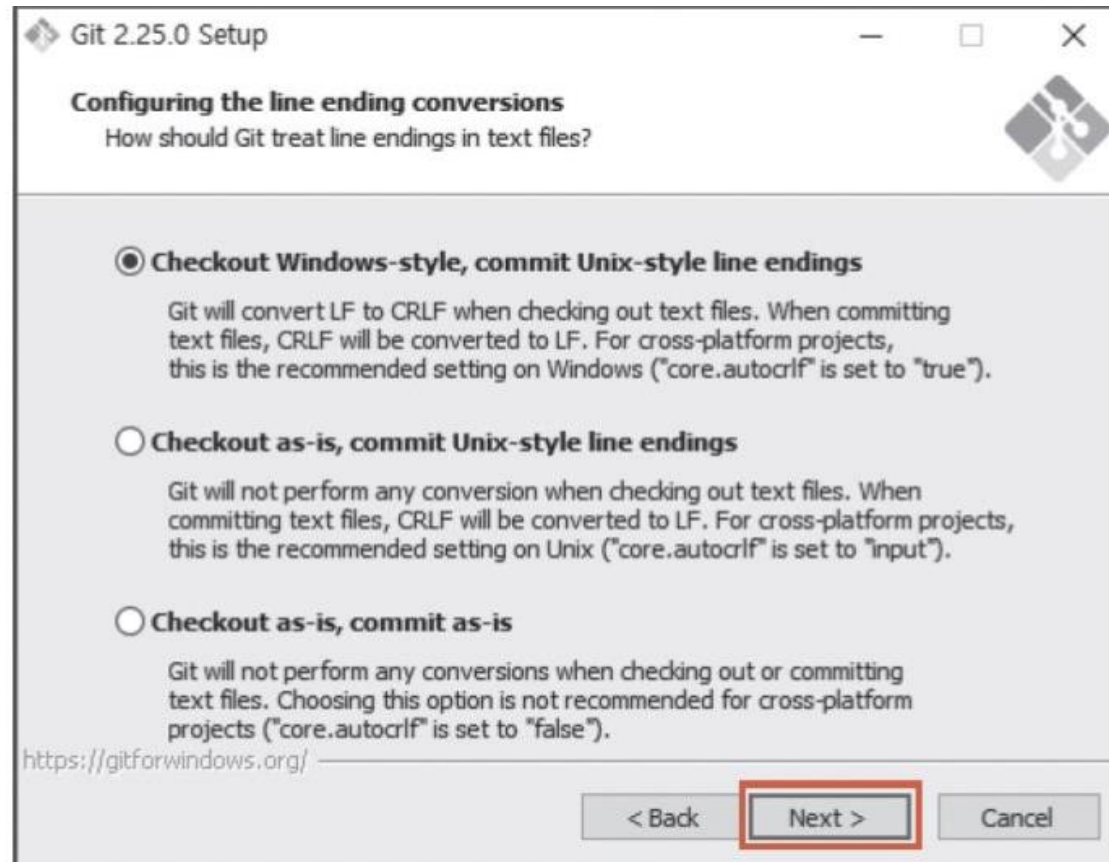
깃 설치하기 (윈도우)

- <https://git-scm.com/>



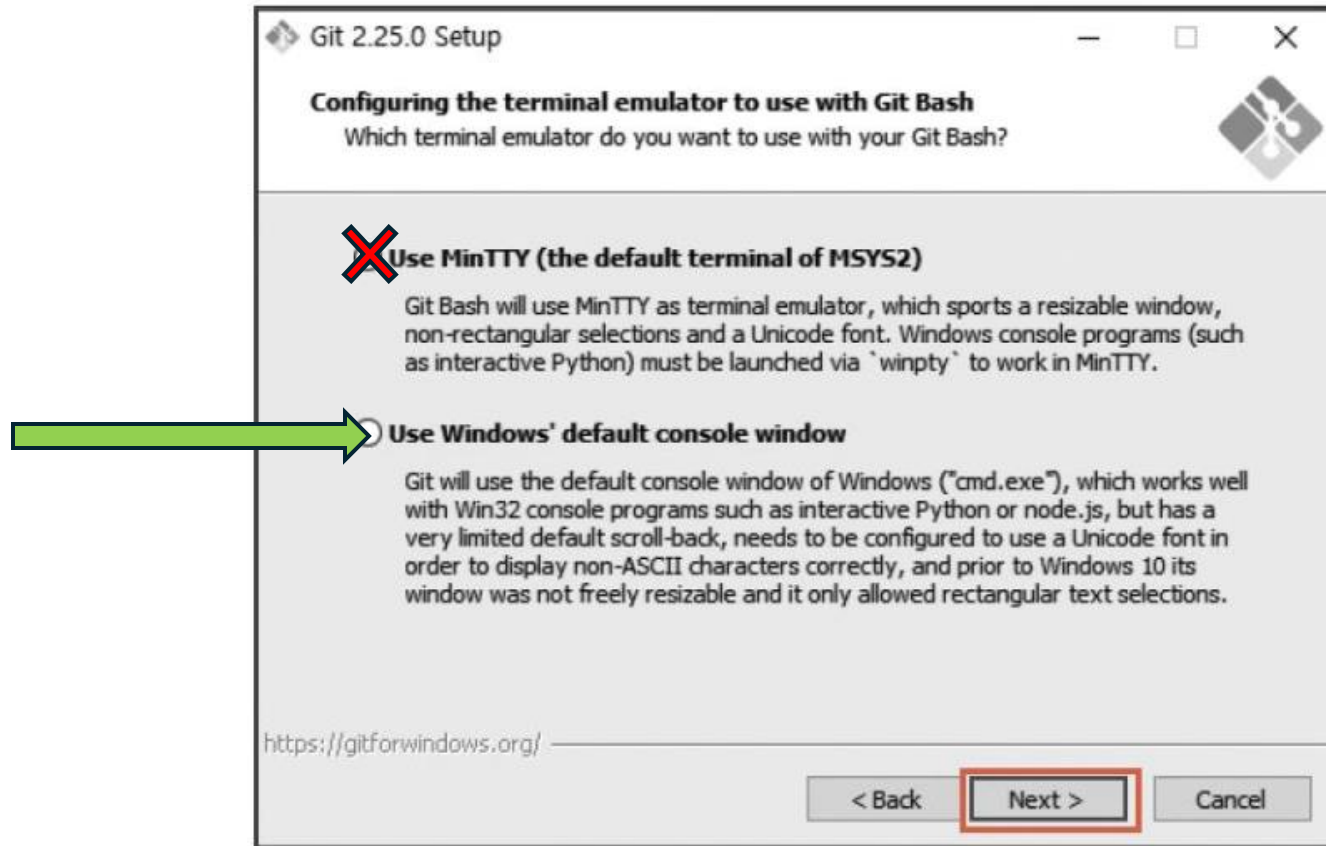
깃 설치하기 (윈도우)

- <https://git-scm.com/>



깃 설치하기 (윈도우)

- <https://git-scm.com/>



깃 설치하기 (MAC)

- Xcode 설치돼있는 경우 설치 필요X
- **Homebrew** : macOS 및 Linux에서 패키지 관리를 쉽게 할 수 있도록 도와주는 오픈 소스 패키지 관리
- 명령어 작성
\$ brew install git



사용자 환경설정

- 사용자 정보를 환경변수에 등록 해줘야 git 사용 시작이 가능하다.
- 협업할때 " 이 코드 누가 작성/수정 했음?"이라고 따지기가 문의하기 위해서 이용한다.
- 파일 단위가 아니라 그 파일에 코드 한줄한줄 누가 어느시점에 수정했고 어떤줄을 어떻게 다르게 수정했고 등등을 전부 조회 가능하다.

- 명령어 입력

git config —global user.name "사용자이름"

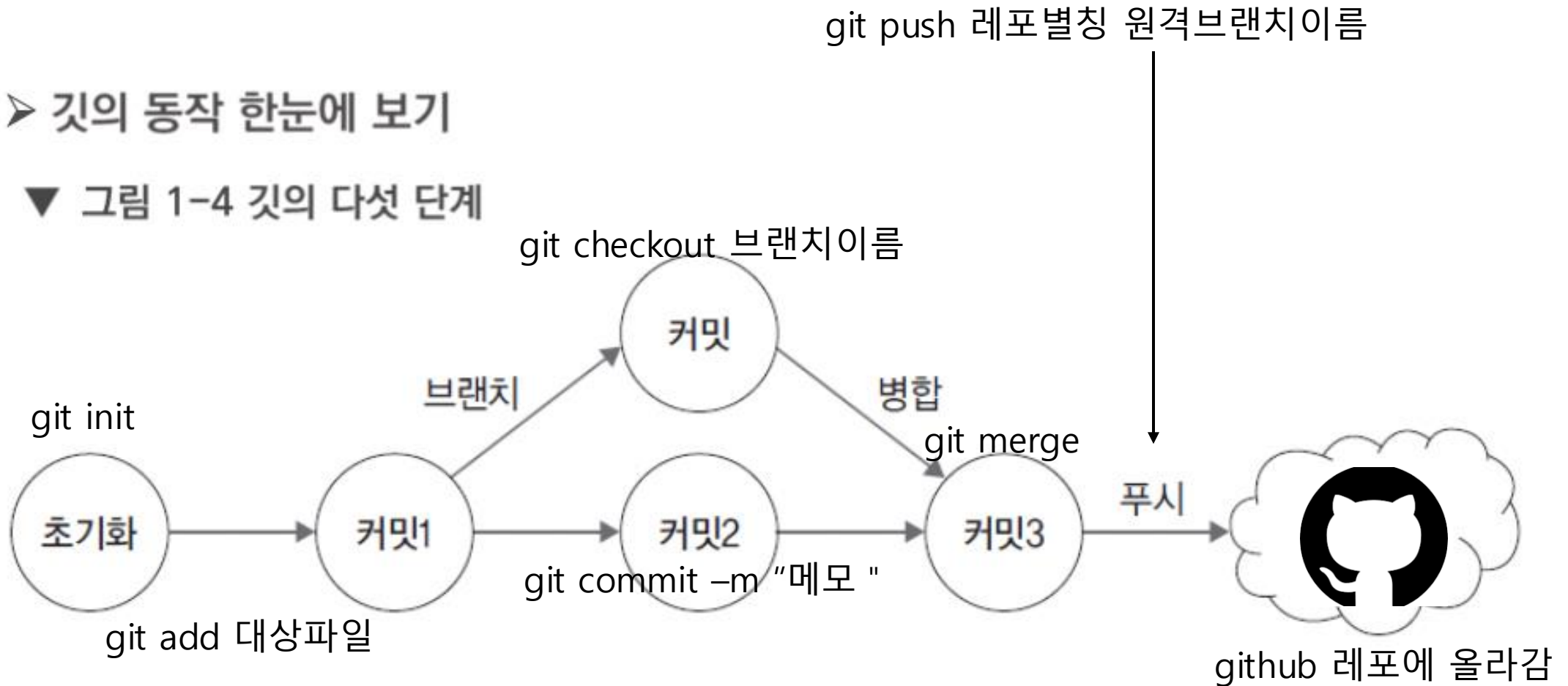
git config —global user.email "이메일주소"

Git 동작 원리

원격저장소 = github 레포지토리(repository)

➤ 깃의 동작 한눈에 보기

▼ 그림 1-4 깃의 다섯 단계

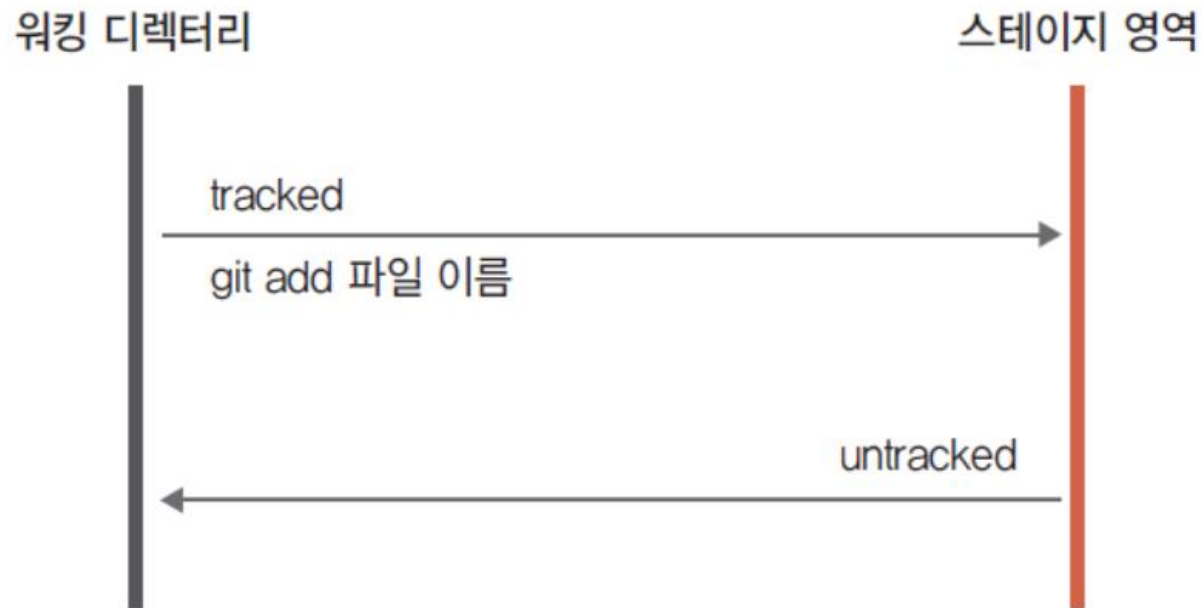


Git 저장공간 분류

- **워킹 디렉터리** : 작업을 하는 공간 - 그냥 코딩하면 여기에 저장됩니다!
 - **스테이지** : 임시로 저장하는 공간(add)
 - **repository** : 실제로 저장하고 기록하는 공간 (복귀지점. Commit)
-
- 이렇게 논리적으로 공간을 분리하는 것은 git의 동작과 이력을 더 효율적으로 처리하기 위한것입니다.
 - **워킹 디렉터리**는 말 그대로 로컬 저장소에 접근하는곳으로 실제로 파일을 수정하고 저장하는 공간입니다.
-
- 해당 디렉토리를 git 저장소로 만들기(git 시작할때 필수)
 - **git init**

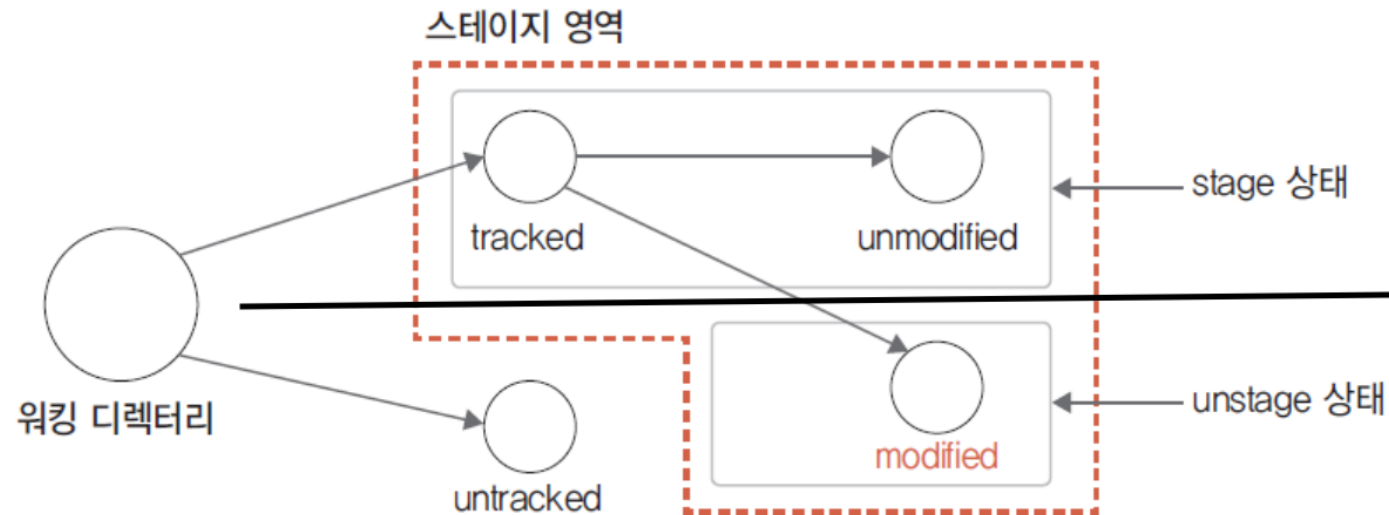
파일의 untracked 상태와 tracked 상태

- 깃은 워킹 디렉터리에 있는 파일들을 '추적됨'과 '추적되지 않음' 상태로 구분합니다.
- git에게 이 파일의 변경사항등을 감시할지, push 할때 github에 올릴지 명령하는것입니다.
- **git add 파일이름** 이나 **git add .** (이 폴더에 있는 파일 전체)를 해서 깃에게 추적시킬수 있습니다.



Git 스테이지

- 스테이지는 워킹디렉토리와 리포지토리 사이에 있는 임시 영역입니다.
- 깃은 워킹 디렉터리에서 작업이 끝난 파일을 스테이지로 잠시 복사합니다.
- 파일들의 스테이지 상태는 `git status` 명령어로 확인 할 수 있습니다.
- 워킹에서 자신이 저장한 파일들이 스테이지에 제대로 올라갔는지 체크 해줘야됩니다.



파일의 modified상태와 unmodified상태

- 워킹에서 등록(git add 파일이름)하면 스테이지에 등록되어 tracked 상태가 돼서 파일을 수정하지 않으면 계속 스테이지 상태에 머무릅니다.
- 워킹에서 파일이 수정되면 modified 상태가 되고, 스테이지에 떨어져나와 unstaged 상태가 됩니다.
(스테이지에서 tracked(추적중) 상태이지만 스테이지에 내용이 제대로 업로드가 안된 unstage 상태인거죠)

이때는 수정한 내용을 다시 git add를 이용해 stage에 올려줘야합니다.

(git commit -am "커밋 메세지도" 가능하다)

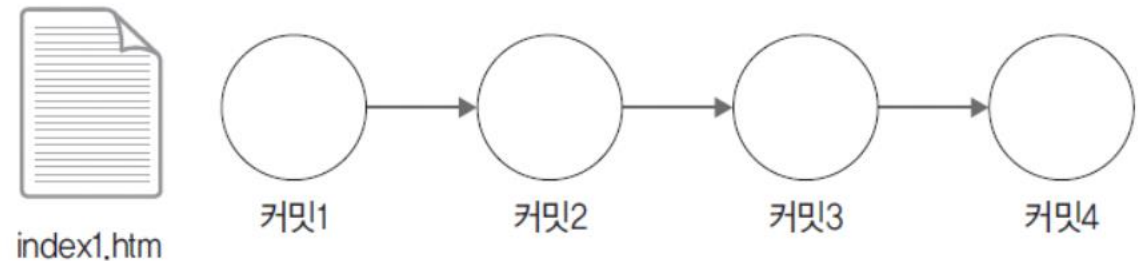
git add -> track, stage해줌

git commit -a -> stage하고 그 시점을 체크포인트로 찍는다. (새파일은 untracked 상태이므로 당연히 등록이 안된다.)

초기에 파일 add해서 tracked와 stage동시에 시켜줌 -> commit을 통해서 체크포인트 찍음 -> 추후에 파일을 수정하고 체크포인트를 찍고 싶음 -> 수정된 파일은 unstaged 되지만 tracked 상태이므로 git은 변화를 추적하고 있다. -> 다시 stage만 시켜주고 commit 하면된다.

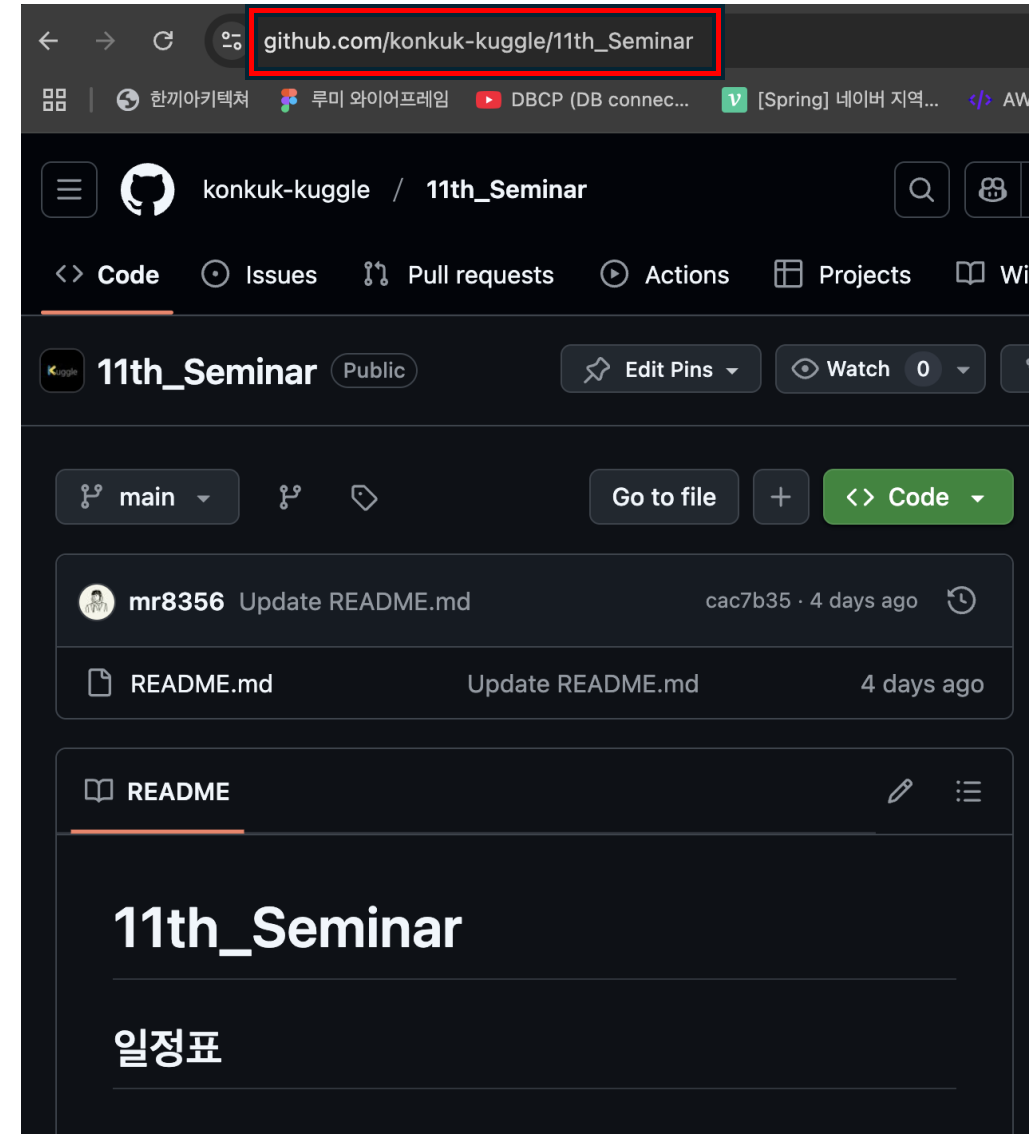
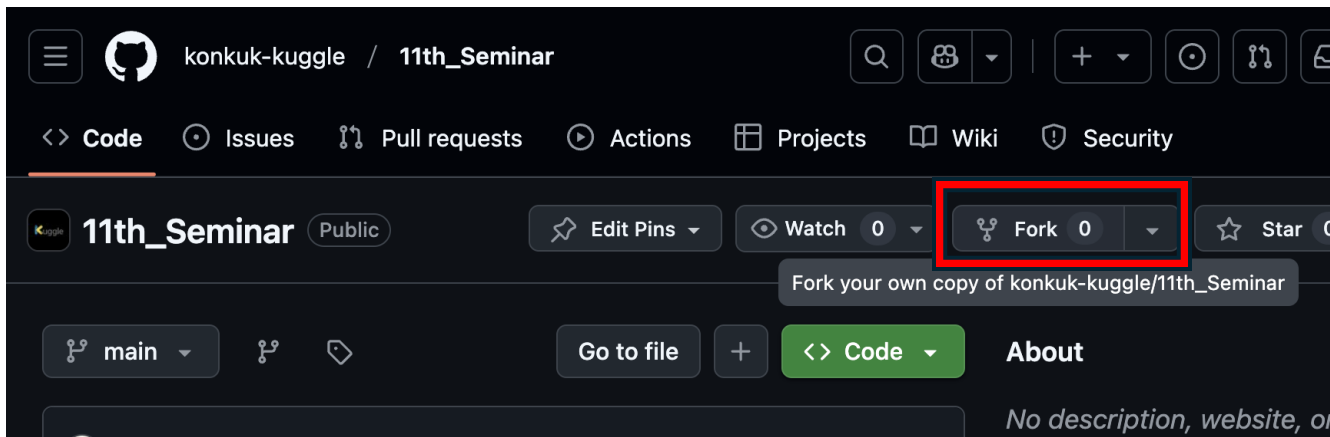
git commit

- 깃의 커밋을 이용하면 새로 변경된 부분만 추출해서 저장합니다.
- **git commit -m "커밋이름"** : stage에 올라간 파일을 커밋
- **git commit -am "커밋이름"** : 수정돼서 unstaged 된 파일을 다시 stage 시켜주면서 동시에 커밋하는 명령어
- 주의 할것은 **git commit -am** 을 한다고 해서 새로 생성된 파일을 add 시켜주진 않습니다. 이미 tracked된 파일만 다시 staged 시켜줍니다.



Github란?

- GitHub는 Git을 기반으로 한 코드 저장소 및 협업 플랫폼이다.
- https://github.com/konkuk-kuggle/11th_Seminar
- 원격저장소(레포지토리) 링크 저장하기!
- 다들 협업 지식 없이 같은 레포를 쓰면 어지러워지니까 Fork를 해서 자신의 계정으로 레포를 복사(연동)후, 자신의 레포로 각자 이용하기!



Github(원격저장소) 연결

- **git remote add** 원격저장소별칭 원격저장소URL
- 예) git remote add origin https://github.com/konkuk-kuggle/11th_Seminar
- user1의 예) git remote add origin https://github.com/user1/11th_Seminar
- **git remote -v** : 원격저장소 목록 확인 (연결 잘 됐는지 확인하기)
- **git push** 원격저장소별칭 브랜치이름 : github에 로컬 깃 저장소에 커밋단위의 내용을 업로드한다.
- 예) git push origin week01
- **git clone** : 원격 저장소에 내용을 자신의 컴퓨터로 내려 받는다. (초기)
- 예) git clone https://github.com/konkuk-kuggle/11th_Seminar
- git pull origin main : 원격저장소에 있는 내용을 한번에 내려받는다. 최신 커밋과 내 로컬파일과 자동 병합한다.
- (원격저장소가 로컬저장소보다 더 최신 커밋이 있을때만 내려받는다.)

요약

- 사실 명령어는 자연스레 외워지는거라 외울필요없고 gpt 물어보세요
- 근데 tracked, stage 상태등등 개념은 알고 있어야지 이용할수가 있어요!

GPT 질문시

- ❌ 나 이거 add 써야돼? commit 해야돼? (틀린 질문)
- ✅ 이 파일이 untracked돼서 등록이 안돼. Tracked 시키는 명령어 알려줘 (좋은 질문)

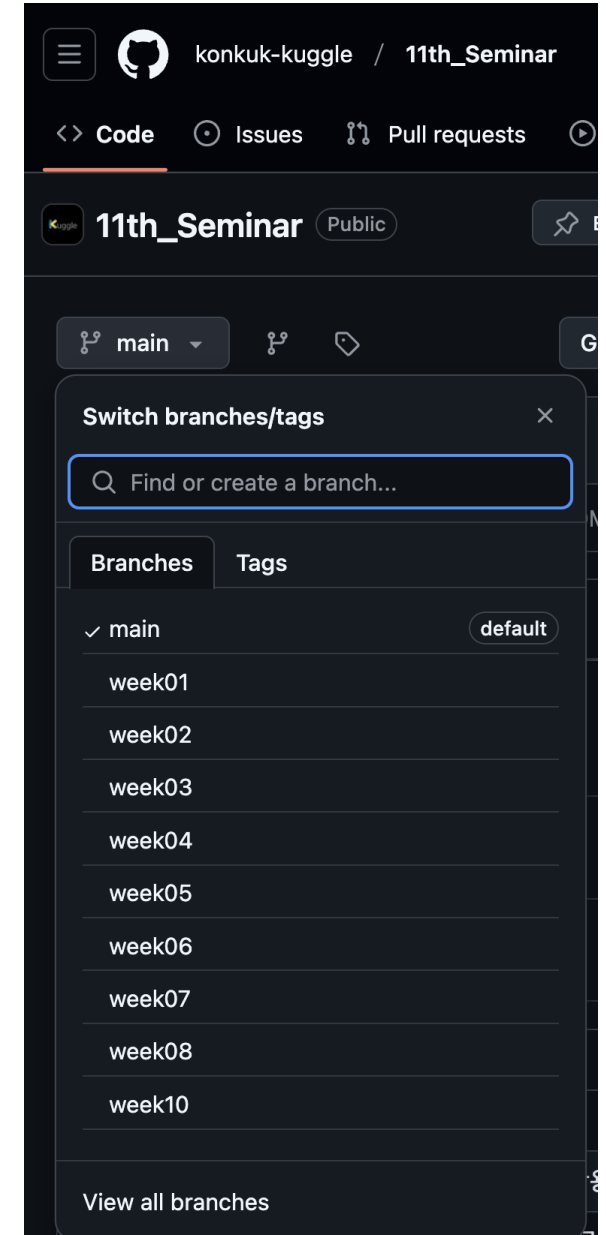
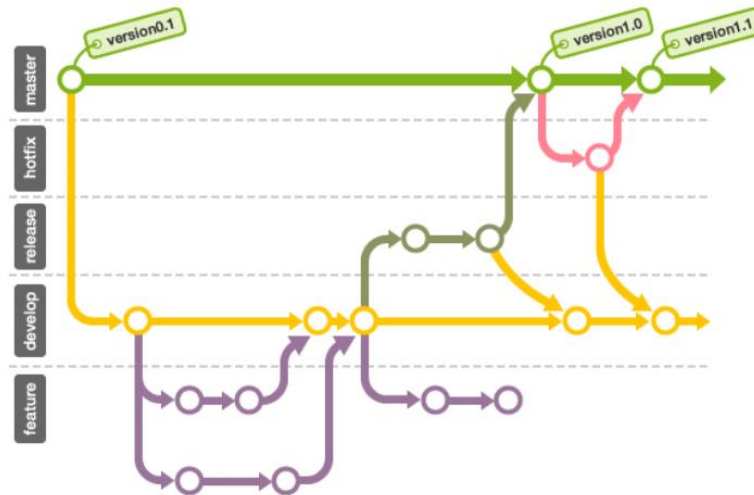
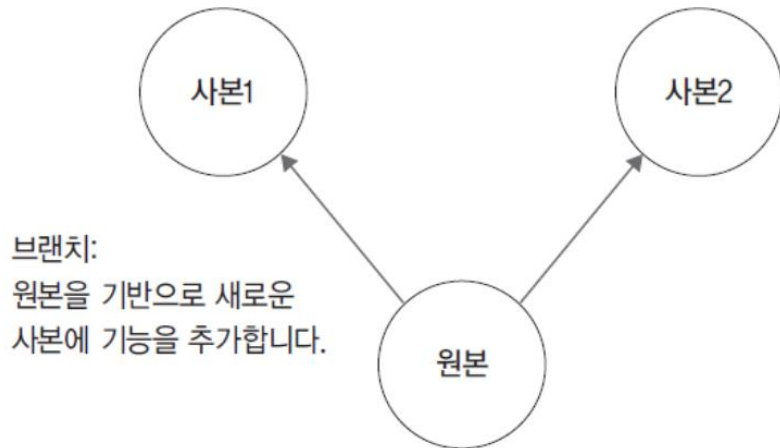
pull ➡ coding ➡ commit ➡ pull ➡ push

▶ 풀과 푸시를 자주 하여 충돌을 최소한으로 줄여 나가면서 작업을 유지함



Git branch 개념

- 브랜치는 나뭇가지, 지사, 분점 등 줄기 하나에서 뻗어 나온 갈림길을 의미합니다
- 큰 나무 줄기에서 작은 줄기가 뻗어나가는 것처럼 저장공간 하나에서 또 다른 가상 저장 공간을 만드는 것입니다.
- 커밋은 파일의 수정 이력을 관리하는데 사용한다면, 브랜치는 프로젝트를 독립적으로 관리하는데 사용합니다. -> **구글은 주차별로 파일을 독립적으로 관리하기 위해 이용**



branch 관련 명령어

- **git branch -v**
 - 가지고 있는 모든 브랜치를 볼 수 있습니다
- **git checkout 브랜치이름**
 - 그 브랜치로 이동 (현재브랜치에서 커밋을 해야 이동할 수 있습니다! git status로 확인)
 - 예) git checkout week02 : 2주차 파일을 받기 위해서 이동
 - 예) git checkout week03 : 3주차 세미나 들을때 이렇게 이동
- **git checkout -b 새로운브랜치이름**
 - 브랜치를 새로 만들고 그 브랜치로 이동(checkout)을 한번에 해줍니다.

