3. 파일 추적 및 무시 설정

`.gitignore` 작성법

✅ 1. `.gitignore`이란?

> Git 저장소에서 **추적하지 않아야 할 파일이나 디렉토리**를 명시하는 파일

- 예: 로그 파일, 컴파일 결과물, 환경 변수 파일, IDE 설정, OS 임시 파일 등
- Git은 `.gitignore`에 명시된 파일을 **커밋 대상으로 포함시키지 않음**

✅ 2. 기본 문법

문법	의미	예시
`파일명`	해당 파일 무시	`secret.env`
`디렉토리/`	폴더 전체 무시	`node_modules/`
`*.확장자`	모든 확장자 무시	`*.log`, `*.class`
`**/패턴`	모든 하위 디렉토리에 적용	`**/tmp/`
`!`	**무시 예외 처리**	`!important.log`
`/경로`	루트 기준 경로 지정	`/build/`

✅ 3. 예제

```
# OS 및 에디터 관련
.DS_Store
Thumbs.db
.idea/
.vscode/

# 로그 파일
*.log
npm-debug.log*

# 환경 설정
.env
.env.local

# 의존성 및 빌드 산출물
node_modules/
dist/
build/
out/
```

```
# 테스트 및 커버리지
coverage/
*.test.js

# 컴파일러 중간 파일
*.o
*.pyc
__pycache__/

# 예외 처리
!important.config.js
```

✅ 4. 예외 처리 문법 (`!`)

`.gitignore`에 포함된 디렉토리나 패턴에서도,
특정 파일만 추적하고 싶다면 `!`로 예외를 걸 수 있다.

```
# 모든 .env 파일 무시
*.env

# 단, .env.example은 예외
!.env.example
```

✅ 5. 디렉토리 무시 예시

```
# build 폴더 전체 무시
build/

# 하위 모든 tmp 폴더 무시
**/tmp/
```

✅ 6. 전역 `.gitignore` 설정 (개인 컴퓨터용)

자신의 모든 Git 프로젝트에서 특정 파일을 항상 무시하고 싶을 때:

설정 방법:

```
git config --global core.excludesfile ~/.gitignore_global
```

예시 파일 내용:

```
# macOS
.DS_Store

# Windows
Thumbs.db

# IDE
.idea/
*.iml
```

✅ 7. 이미 추적된 파일은 무시되지 않음

중요 주의사항
`.gitignore`은 **Git이 아직 추적하지 않은 파일**에만 적용된다.

이미 커밋한 파일을 무시하려면:

```
git rm --cached filename
```

예:

```
git rm --cached .env
git commit -m "Remove .env from tracking"
```

✅ 8. `.gitignore` 자동 생성 도구

- https://www.toptal.com/developers/gitignore ← 👍 강력 추천
- https://gitignore.io

```
# 예: Node.js + macOS + VSCode 용
curl -L -s https://www.toptal.com/developers/gitignore/api/node,macos,vscode -o
.gitignore
```

✅ 9. 프로젝트별 `.gitignore` 예시

Node.js 프로젝트

```
node_modules/
dist/
.env
*.log
```

Python 프로젝트

```
__pycache__/
*.pyc
.env
venv/
```

Java + IntelliJ

```
*.class
*.jar
target/
.idea/
*.iml
```

✅ 마무리 요약

기능	예시
특정 확장자 무시	`*.log`
폴더 전체 무시	`build/`
특정 파일만 포함	`!keep.txt`
루트 경로 기준	`/dist/`
하위 경로 전부 적용	`**/tmp/`

`.gitkeep`의 의미

✅ 1. Git은 빈 디렉토리를 추적하지 않는다

- Git은 내부적으로 **파일 단위**로만 버전 관리를 함
- **디렉토리 자체는 버전 관리 대상이 아님**
- 즉, 폴더만 있고 그 안에 **파일이 아무것도 없으면**, Git은 **그 폴더를 저장소에 포함하지 않음**

✅ 2. 그래서 `.gitkeep`이 필요하다

> `.gitkeep`**은 비어 있는 디렉토리를 Git에 포함시키기 위해 사용하는** *사실상 아무 의미 없는*** 파일이다.
> 이름은 `git`이 요구하는 건 아니며, 그냥 **관례적인 이름일 뿐**이다.

🔸 주요 사용 예시:

- `logs/`, `uploads/`, `temp/`, `data/` 폴더는 앱에서는 필요하지만 **초기에는 비어 있음**
- 이 디렉토리를 Git 저장소에 포함시키고 싶을 때:

```
mkdir uploads
touch uploads/.gitkeep
git add uploads/.gitkeep
git commit -m "chore: keep uploads directory"
```

✅ 3. `.gitkeep` vs `.gitignore`

항목	`.gitkeep`	`.gitignore`
목적	**빈 디렉토리를 강제로 추적**	**특정 파일/디렉토리를 추적하지 않도록 설정**
내용	대부분 빈 파일	무시할 패턴을 나열한 설정 파일
Git 공식 여부	❌ 비공식, 커뮤니티 관례	✅ Git 공식 기능
대체 가능 여부	`README.md` 등 다른 빈 파일로 대체 가능	고유 기능이라 대체 불가

✅ 4. 다른 이름도 가능하다?

- 사실 `.gitkeep`은 **필수 이름이 아님**
- 그냥 `keep.txt`나 `README.md`처럼 아무 파일이나 넣어도 Git은 폴더를 추적함
- 하지만 커뮤니티에서는 `.gitkeep`을 **암묵적 표준**으로 받아들이고 있다.

✅ 5. 실무에서 사용하는 디렉토리 예시

디렉토리	설명
`logs/`	로그 파일 저장 위치
`uploads/`	사용자 업로드 파일 저장소
`tmp/`, `cache/`	임시 데이터 저장소
`data/`, `backups/`	내부 DB 또는 설정 파일 위치

→ 이 폴더들에는 **초기에는 비어 있지만**,
앱 실행 시 필요하므로 Git에 포함시키기 위해 `.gitkeep`을 사용

✅ 마무리 요약

항목	설명
`.gitkeep` 역할	**비어 있는 폴더를 Git이 추적하게 하기 위한 더미 파일**
파일 내용	보통 비어 있음
공식 기능 여부	❌ Git이 직접 제공하지 않음 (커뮤니티 관례)
대체 가능	`README.md`, `placeholder.txt` 등 가능하지만 `.gitkeep` 권장
사용 시기	빈 `logs/`, `uploads/` 폴더 등을 커밋해야 할 때

`git clean`, `git rm`, `git mv`

✅ 1. `.gitkeep`의 의미 다시 정리

항목	내용
`.gitkeep`	**빈 디렉토리를 Git에 포함시키기 위한 더미 파일**
이유	Git은 기본적으로 **빈 폴더를 추적하지 않기 때문**
공식 기능?	❌ Git에서 지정한 건 아님 (커뮤니티 관례)
대체 가능	`placeholder.txt`, `README.md` 등 다른 빈 파일도 가능

예시:

```
mkdir uploads
touch uploads/.gitkeep
git add uploads/.gitkeep
git commit -m "chore: keep uploads directory"
```

✅ 2. `git clean` – 추적되지 않은 파일/폴더 제거

> Git이 **추적하고 있지 않은 파일**(untracked)을 완전히 삭제하는 명령어
> 즉, `git status`에 나오는 `Untracked files:`을 정리하는 용도

기본 사용법

```
git clean -n     # 무엇이 삭제될지 미리 보기 (safe!)
git clean -f     # 실제로 삭제
```

폴더까지 삭제

```
git clean -fd
```

옵션	의미
`-n`	시뮬레이션 (삭제 전 확인)
`-f`	실제 삭제 수행
`-d`	디렉토리 포함 삭제

예시

```
git clean -fd
```

→ `node_modules/`, `*.log`, `tmp/` 등 **.gitignore에 있는 파일도 삭제 가능**

> ⚠️ 되돌릴 수 없기 때문에 항상 `-n`으로 먼저 확인할 것

✅ 3. `git rm` – Git이 추적 중인 파일 삭제

> Git이 이미 추적하고 있는 파일을 **버전 관리에서 제거**할 때 사용
> (작업 디렉토리에서 삭제할지 여부는 옵션으로 조절)

기본 사용법

```
git rm file.txt         # 파일을 Git에서도, 디스크에서도 삭제
git rm --cached file.txt  # Git에서만 제거, 파일은 로컬에 남김
```

옵션	설명
`--cached`	인덱스(Git 저장소)에서만 제거, 실제 파일은 유지
`-r`	디렉토리도 제거 (재귀)

예시

```
git rm --cached .env
git commit -m "Remove .env from tracking"
```

> `.gitignore`에 추가할 경우, 추적을 중지하고 다시 커밋되지 않게 막을 수 있다.

✅ 4. `git mv` – 파일 이름 변경 또는 디렉토리 이동

> 파일이나 디렉토리의 **이름을 변경하거나 위치를 옮길 때 사용**

기본 사용법

```
git mv old-name.txt new-name.txt
```

> 이 명령은 실제 파일을 이동시키고, 동시에 Git에게 **"이건 새 파일이 아니라 이동이야"** 라고 알려준다.

예시

```
git mv src/Login.js src/pages/LoginPage.js
git commit -m "refactor: move Login.js to pages/"
```

> Git은 자동으로 rename을 감지하지만, `git mv`를 쓰면 명시적으로 관리 가능하고 히스토리 추적에 유리하다.

✅ 세 명령어 요약 비교

명령어	대상	역할	주의사항
`git clean`	추적되지 않은 파일	완전 삭제 (되돌릴 수 없음)	`-n`으로 먼저 확인 필수
`git rm`	추적 중인 파일	Git 저장소에서 제거	`--cached`로 로컬 보존 가능
`git mv`	추적 중인 파일	이름 변경 + 이동	`git add` + `rm` 조합보다 안전

✅ 실무 예시 흐름

```
# 1. 추적되지 않은 파일 정리 (주의!)
git clean -n
git clean -fd

# 2. 잘못 커밋한 민감한 파일 제거
git rm --cached secret.env
echo "secret.env" >> .gitignore
git commit -m "remove tracked secret file and ignore it"

# 3. 파일 리팩토링
git mv old.js src/components/NewComponent.js
git commit -m "refactor: rename old.js to NewComponent.js"
```

✅ 마무리 요약

목적	명령어	설명
빈 폴더 유지	`.gitkeep`	Git이 폴더를 추적하게 만듦
쓰레기 청소	`git clean`	추적되지 않은 파일 완전 제거
추적된 파일 삭제	`git rm`	Git 저장소에서 제거
파일/디렉토리 이름 변경	`git mv`	리네이밍/이동 추적용

대소문자 변경 추적 이슈 (core.ignorecase)

✅ 1. 문제 상황: Git이 대소문자 변경을 감지하지 않음

예시:

```
mv logo.png Logo.png
git status
```

> ❗ 아무 변경도 안 뜸
> Git은 **"logo.png"와 "Logo.png"가 같은 파일**이라고 판단함
> → **대소문자만 다른 변경은 무시**해버리는 문제가 발생

✅ 2. 원인: `core.ignorecase = true`

- `core.ignorecase`는 Git의 내부 설정으로,
  파일 이름의 대소문자를 무시할지 여부를 결정하는 옵션이다.
- 기본값:
  - **Windows, macOS(HFS+)** → `true`
  - **Linux(ext4 등)** → `false`

> ❗ Git은 로컬 파일시스템이 대소문자를 구분하는지에 따라 이 값을 자동 설정함

✅ 3. 설정 확인 방법

```
git config core.ignorecase
```

출력 예:

```
true
```

✅ 4. 해결 방법 1: 설정 변경

```
git config core.ignorecase false
```

> 이후에는 대소문자 변경도 `git status`로 감지됨

✅ 5. 해결 방법 2: 우회 처리 (실제 변경 유도)

```
# 임시 이름으로 한번 rename
git mv logo.png tmp.png

# 다시 원하는 이름으로 변경
git mv tmp.png Logo.png

git commit -m "chore: rename logo.png to Logo.png"
```

> Git은 같은 이름이 아니라고 생각해야 commit 가능

✅ 6. 주의사항

항목	설명
Windows/macOS	대소문자를 구분하지 않는 파일시스템 → Git이 추적 어려움
GitHub 리눅스 서버	실제로는 대소문자 구분됨 → 리포지토리에선 이름 충돌 가능
협업 시 혼란	`Logo.png`와 `logo.png`가 공존하게 되면 충돌 위험 있음

✅ 7. 실제 사례

- 한 팀원이 macOS에서 `logo.png → Logo.png`로 변경하고 push함
- 다른 팀원이 Windows에서 pull 했더니 Git은 둘을 동일하게 인식해서 충돌 발생
- 이 문제는 주로 **이미지, JS 파일, 컴포넌트 이름 변경 등**에서 자주 발생함

✅ 8. 안전한 처리 순서 (실무 권장)

```
git config core.ignorecase false

# 또는 확실한 rename 처리
git mv SomeFile.js tmp.js
git mv tmp.js someFile.js
git commit -m "rename SomeFile.js to someFile.js"
```

> 팀 전체가 같은 규칙을 사용하도록 `.gitattributes` 또는 문서화하는 것이 좋음

✅ 마무리 요약

항목	설명
`core.ignorecase`	대소문자 무시 여부 설정
기본값	Windows/macOS: true, Linux: false
문제점	`file.js → File.js` 같은 변경 감지 안 됨
해결 방법	`mv → commit`, 또는 `core.ignorecase = false` 설정