

# AI 개발 실무

## 9. SW 형상 관리

김 윤 기    교수



09<sub>week</sub>

A I   개 발   실 무   |   김 윤 기

# SW 형상 관리



- » SW 형상 관리의 개념을 설명할 수 있다.
- » Git을 설정하고, 적용할 수 있다.
- » Git을 이용해 SW 형상 관리를 할 수 있다.

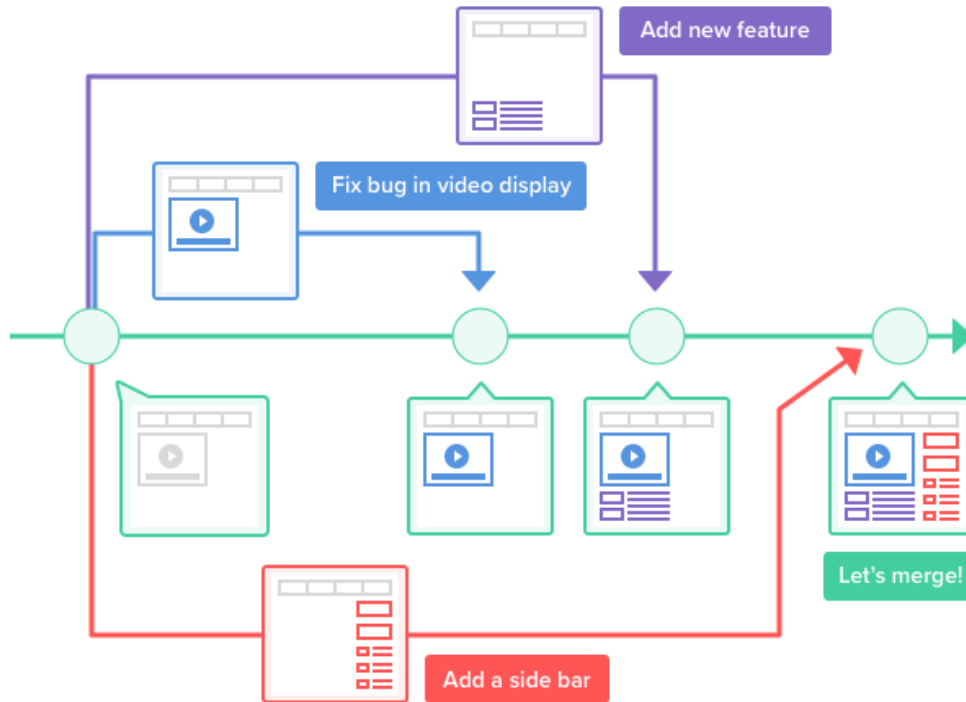
## ① SW 형상 관리 개념

---

## ② Git을 활용한 버전 관리 실습

---

## SW 형상 관리가 왜 필요한가?



SW는 계속적으로 업데이트 되며 버전이 추가됨  
다른 사람들과의 **협업**을 통해 **버전을 확정**

---

CHAPTER

01

# SW 형상 관리

# 1. SW 형상 관리란?

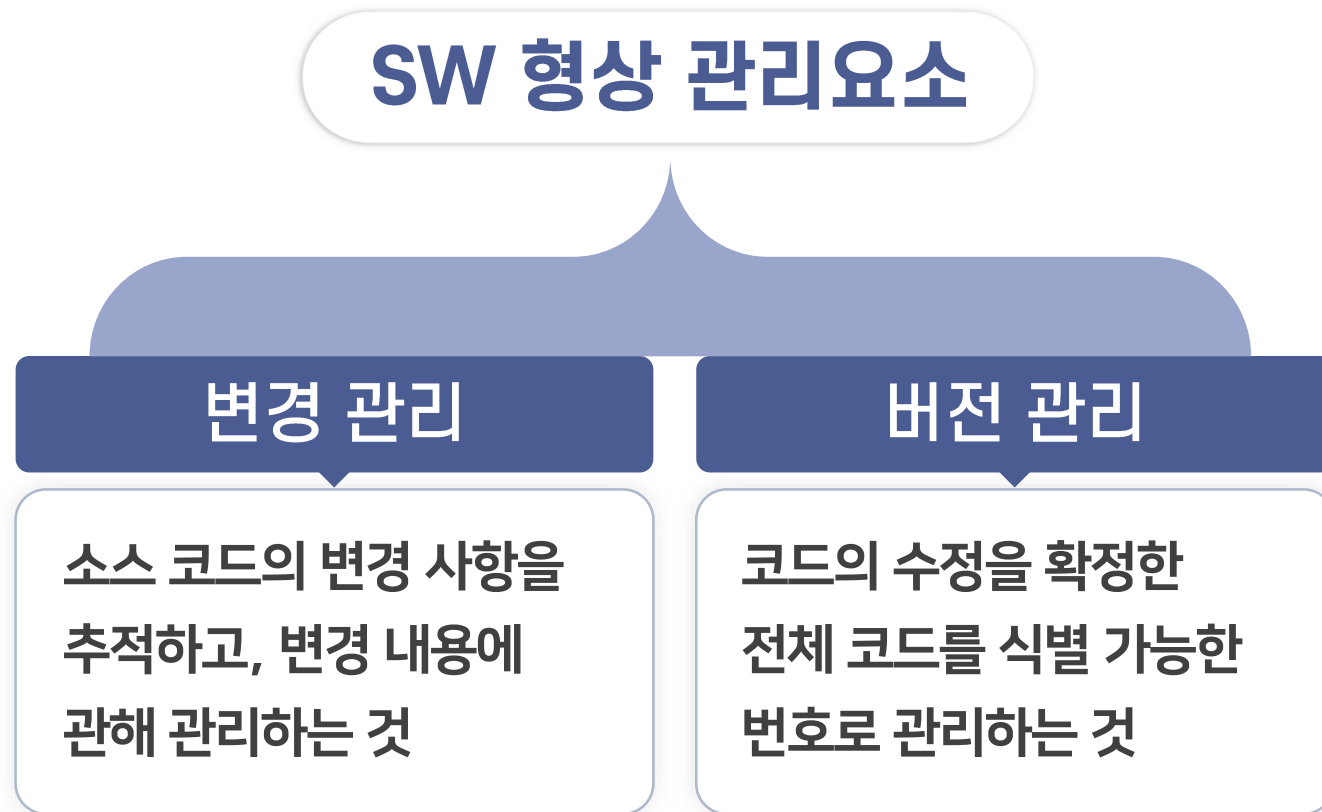
## 소프트웨어의 변경사항을 체계적으로 추적하고 통제 하는 것

SW 개발 생명주기 동안 소프트웨어 변경사항을 추적하여,  
이를 효율적으로 통제하여 개발 프로세스의 안정성을 확보

협업을 통해 공동으로 SW를 개발 및 관리하는 것이 가능하며,  
여러 버전의 충돌을 방지

개발 - 테스트 - 배포 - 유지보수 등의 과정을 효율적으로  
수행 가능

## 2. SW 형상 관리의 요소





## 3. SW 형상 관리의 역할

- » 소프트웨어 수정에 사용되는 리소스와 시간 절약
  - 개발자들이 변경 사항을 추적하기에 용이함
  - 이전 버전을 보존함으로써, 현 버전에서 문제 발생시 롤백이 가능함
- » 대규모 팀의 협업을 가능하게 함
  - 코드의 충돌을 방지
  - 소프트웨어의 안정성을 보장함
  - 개발에 참여하는 구성원이 프로젝트를 신뢰할 수 있음

## 4. SW 형상 관리 도구

### ① Git

- » 가장 많이 사용되는 형상 관리 도구
- » 다양한 무료 저장소(Github, BitBucket 등)가 활성화 되어 있음
- » 분산형 버전 관리 시스템을 제공
- » 브랜치 관리, 병합 및 충돌 해결과 같은 협업 기능을 제공
- » 많은 오픈소스들이 Git을 통해 소스 코드를 제공하며, 이슈 관리를 함

## 4. SW 형상 관리 도구

### ② Subversion(SVN)



- » 중앙 집중식 버전 관리 시스템
- » 파일을 중앙 저장소에서 관리하며, 각 개발자들이 중앙 저장소에서 파일을 가져와 작업을 수행
- » 파일 버전을 중앙 저장소에서 관리하기 때문에, 파일 버전을 관리하기가 비교적 간단하고 편리
- » 중앙 저장소의 장애로 인한 작업 중단 가능성이 있음
- » 모든 버전 관리 동작이 적어도 한번 중앙 서버를 거쳐야 함으로 속도가 느림

## 4. SW 형상 관리 도구

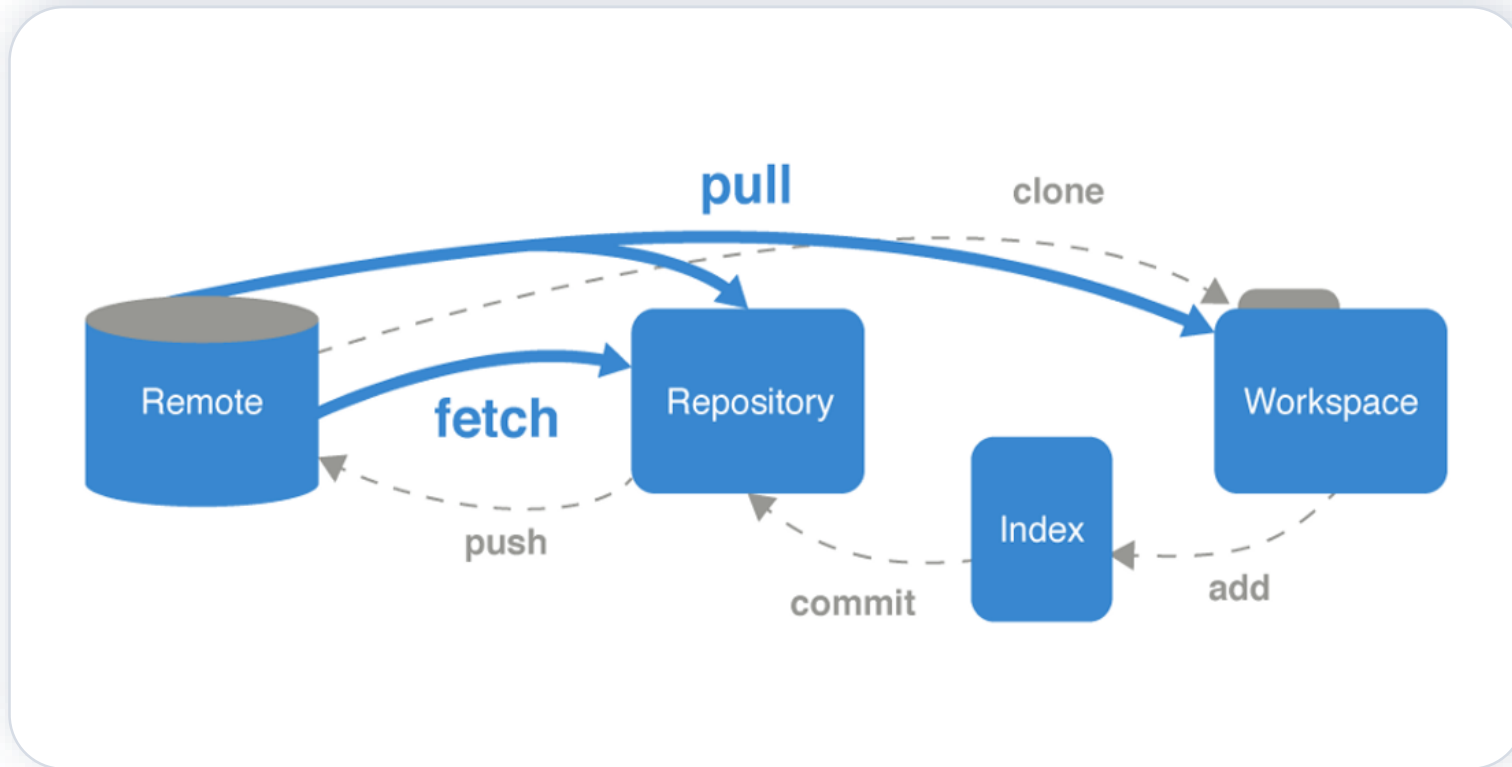
### 3 Mercurial

mercurial

- » Git과 유사한 분산형 버전 관리 시스템
- » 파일 이름의 대소문자 구분을 하지 않음
- » 사용방법이 비교적 간단하나, 버전 관리를 세심하게 제어하지 못함
- » 소규모 프로젝트에 적합

## 5. Git의 기본 개념

### ① 저장소(Repository)와 워크스페이스(Workspace)



## 5. Git의 기본 개념

### ① 저장소(Repository)와 워크스페이스(Workspace)

#### 저장소 (Repository)

- 버전 관리를 위한 모든 데이터를 저장하는 곳
- 로컬 컴퓨터나 원격 서버 등 어디서든 생성 가능
- 변경 사항이 확정(commit)된 상태의 파일들을 저장

#### 워크페이스 (Workspace)

- 저장소에 있는 파일을 로컬 컴퓨터에서 작업하기 위해 복제된 디렉토리
- 저장소에서 가져온 파일들을 수정하고, 새로운 파일을 추가하거나 삭제
- 워크스페이스의 변경 내용을 확정할 수 있음

## 5. Git의 기본 개념

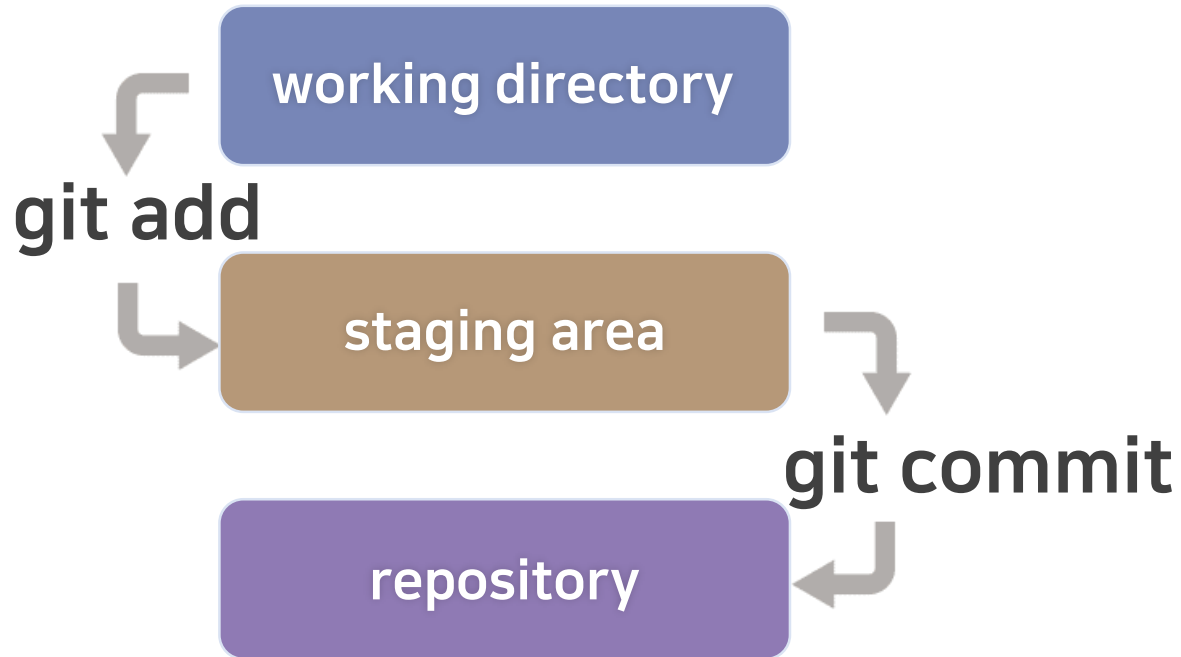
### ① 저장소(Repository)와 워크스페이스(Workspace)

#### ➔ 주요 명령어

- **git init**: 새로운 저장소를 생성
- **git clone**: 저장소를 로컬 컴퓨터로 복제하여 워크스페이스에 저장
- **git add**: 워크스페이스에서 변경된 파일을 스테이징 영역에 추가
- **git commit**: 스테이징 영역에 추가된 변경 내용을 확정하여 저장소에 커밋
- **git push**: 로컬 저장소의 커밋 내용을 원격 저장소로 전송
- **git pull**: 원격 저장소의 변경 내용을 로컬 저장소로 가져와 병합

## 5. Git의 기본 개념

### ② 스테이지 영역(staging area)와 커밋(commit)





## 5. Git의 기본 개념

### ② 스테이지 영역(staging area)와 커밋(commit)

#### 스테이지 영역 (staging area)

- 스테이지 영역은 변경된 파일 중 커밋할 파일을 선택할 수 있는 중간 영역
- Git에서는 스테이지 영역을 이용하여 변경 내용을 세분화하고, 커밋할 내용을 선택적으로 추가할 수 있음
- 스테이지 영역을 사용하면 작은 단위로 변경 내용을 커밋할 수 있어서, 이후 변경 내용을 추적하거나 롤백(roll-back)하는데 용이

#### 커밋 (commit)

- 스테이지 영역에 추가된 변경내용을 저장소(repository)에 반영하는 것 의미
- 커밋은 변경 내용에 대한 설명을 함께 추가할 수 있음
- Git에서 커밋은 버전 관리의 핵심 단위로, 커밋마다 고유한 식별자(commit hash)가 생성

## 5. Git의 기본 개념

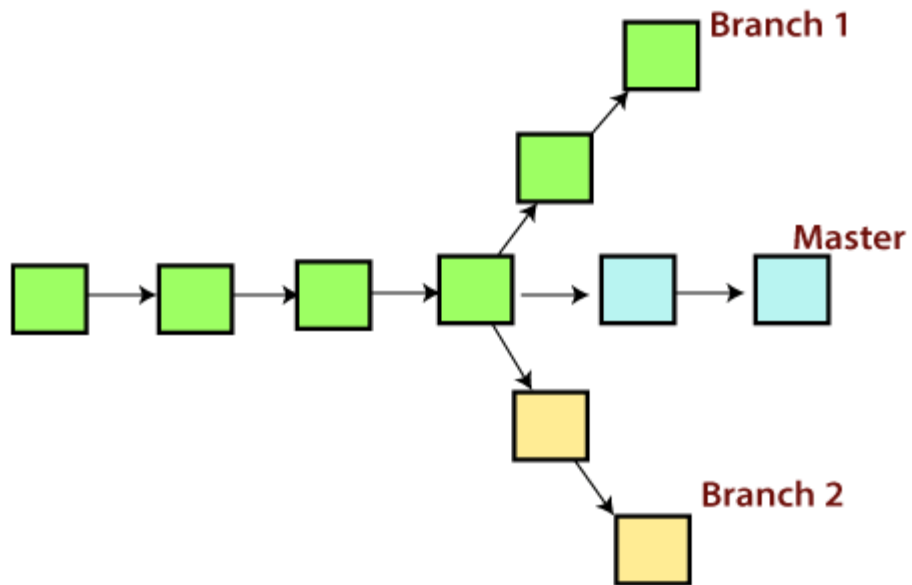
### ② 스테이지 영역(staging area)와 커밋(commit)

#### ➔ 주요 명령어

- **git add**: 워크스페이스에서 변경된 파일을 스테이지 영역에 추가
- **git reset**: 스테이지 영역에서 파일을 제거
- **git commit**: 스테이지 영역에 추가된 변경 내용을 커밋
- **git log**: 저장소에 커밋된 내용을 조회
- **git diff**: 변경된 내용을 비교

## 5. Git의 기본 개념

### ③ 브랜치(branch)와 마스터(master)



## 5. Git의 기본 개념

### ③ 브랜치(branch)와 마스터(master)

#### 브랜치 (branch)

- 브랜치는 기존 커밋을 기준으로 새로운 작업 공간을 생성
- 새로운 브랜치에서 변경된 내용은 원래 브랜치에 영향을 주지 않음
- 여러 개의 브랜치를 이용하면 동시에 여러 작업을 수행하고, 변경 내용을 병합할 수 있음
- 브랜치는 생성, 삭제, 병합 등을 자유롭게 수행

#### 마스터 (Master)

- 마스터는 기본 브랜치로서, 보통은 최종 릴리즈 버전을 관리
- 새로운 기능 개발이나 버그 수정 등의 작업은 브랜치를 생성하여 진행하며, 이후 마스터 브랜치에 병합

## 5. Git의 기본 개념

### ③ 브랜치(branch)와 마스터(master)

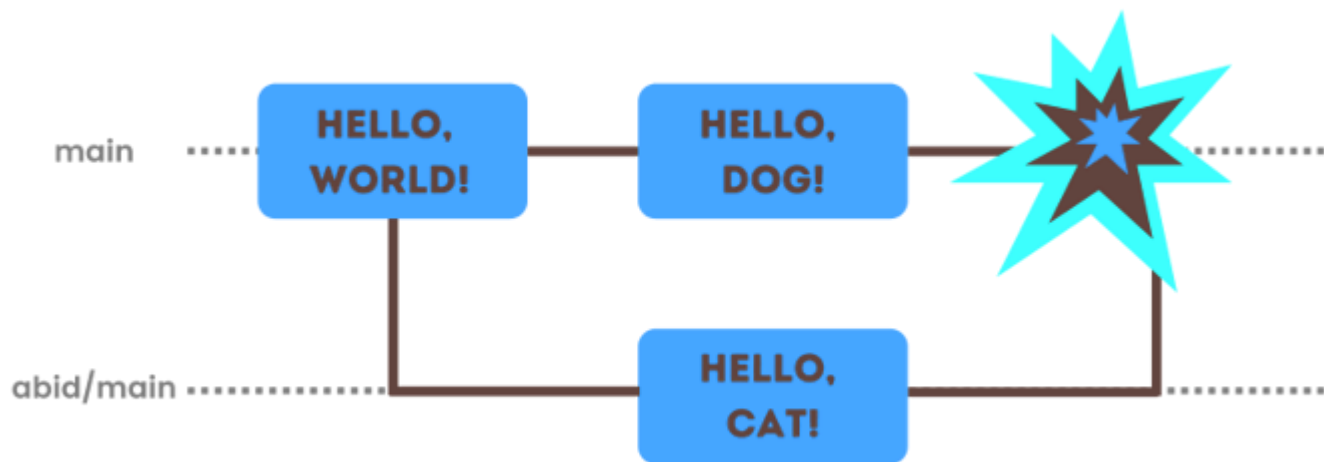
#### ➔ 주요 명령어

- **git branch**: 브랜치를 생성, 삭제, 조회
- **git checkout**: 특정 브랜치로 이동
- **git merge**: 브랜치에서 변경된 내용을 마스터에 병합
- **git push**: 로컬 저장소의 변경 내용을 원격 저장소에 업로드

## 5. Git의 기본 개념

### ④ 충돌과 해결

» 여러 개발자가 동시에 같은 파일을 수정할 때 발생



## 5. Git의 기본 개념

### ④ 충돌과 해결

- » 충돌이 발생한 부분을 직접 수정하거나, 한쪽 변경 사항을 선택하여 충돌을 해결할 수 있음
- » 충돌이 발생한 파일을 수정한 뒤, 커밋을 수행하여 충돌 해결을 마무리
- » 충돌이 해결된 뒤 브랜치 병합이 가능



# 개발 실무 실습하기

P r a c t i c a l P r o g r a m m i n g f o r A I

## Git 활용 실습



## ① SW 형상 관리란?

소프트웨어의 변경사항을 체계적으로  
추적하고 통제 하는 것

## ② 형상 관리의 요소

1. 변경 관리

2. 버전 관리

### ③ SW 형상 관리의 역할

소프트웨어 수정에 사용되는  
리소스와 시간 절약

대규모 팀의 협업을 가능하게 함

### ④ SW 형상 관리 도구

Git

Subversion

Mercurial

## ⑤ Git의 기본 개념

저장소(Repository)와 워크스페이스(Workspace)

스테이지 영역(staging area)와 커밋(commit)

브랜치(branch)와 마스터(master)

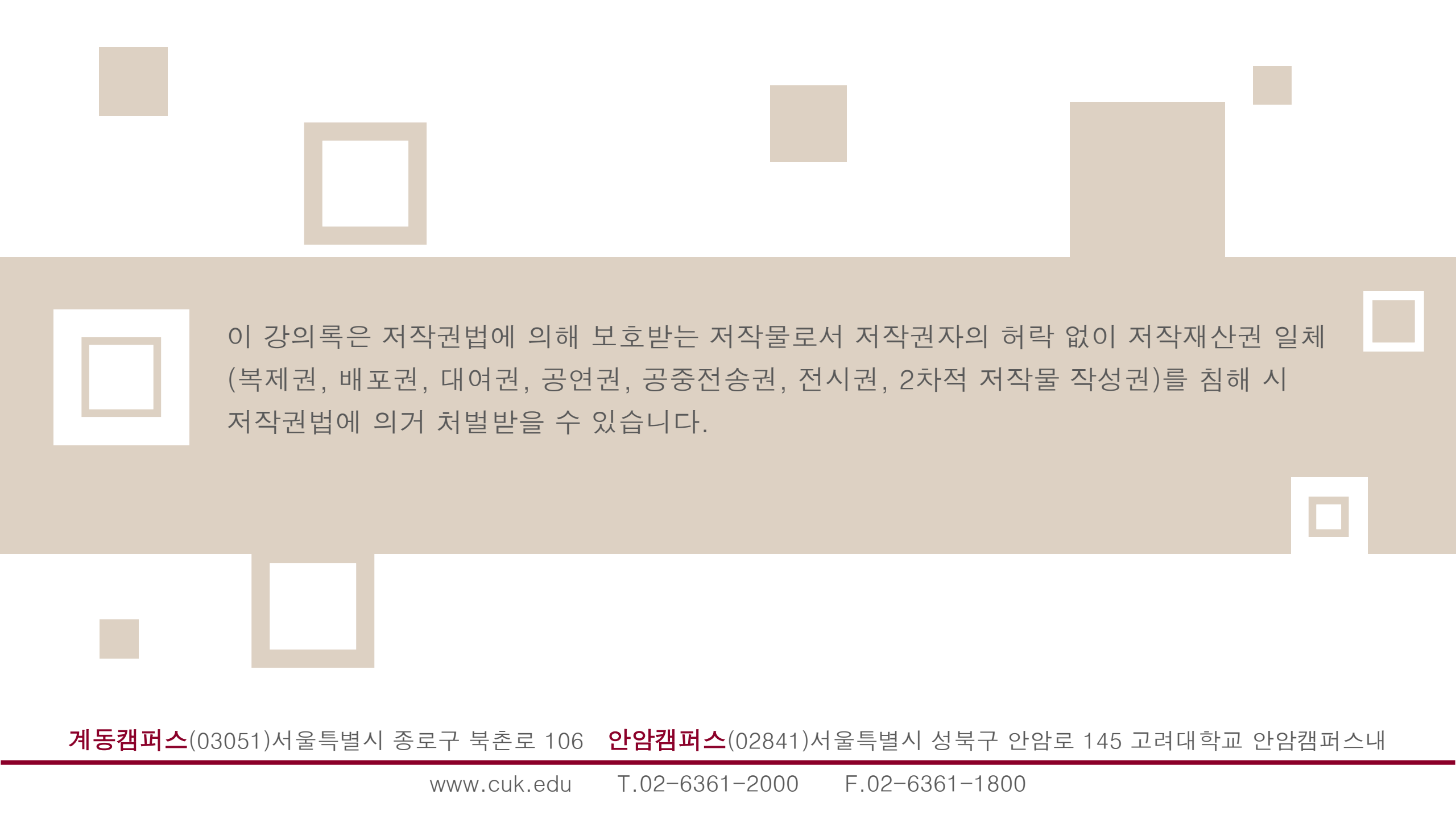
충돌과 해결

## ⑥ Git 실습

Git 활용 실습

## 참고문헌

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



이 강의록은 저작권법에 의해 보호받는 저작물로서 저작권자의 허락 없이 저작권재산권 일체 (복제권, 배포권, 대여권, 공연권, 공중전송권, 전시권, 2차적 저작물 작성권)를 침해 시 저작권법에 의거 처벌받을 수 있습니다.