git으로 시작하는 기초부터 실무 협업

Ryan | 2023.04.17

버전 관리와 Git

Git

- 컴퓨터에서 파일을 관리하는 방법 중 하나
- 시간을 되돌리게 해줌
- 여러 명이 서로 겹치지 않게 파일을 수정하게 해줌

버전 관리란?

버전 관리란?

- 소프트웨어 변경사항을 기록 및 추적하는 과정
- 개발자가 이전 상태로 돌아갈 수 있도록 함
- 다양한 변경사항을 병합하여 코드 충돌 방지

버전 관리를 왜 해야 할까요?

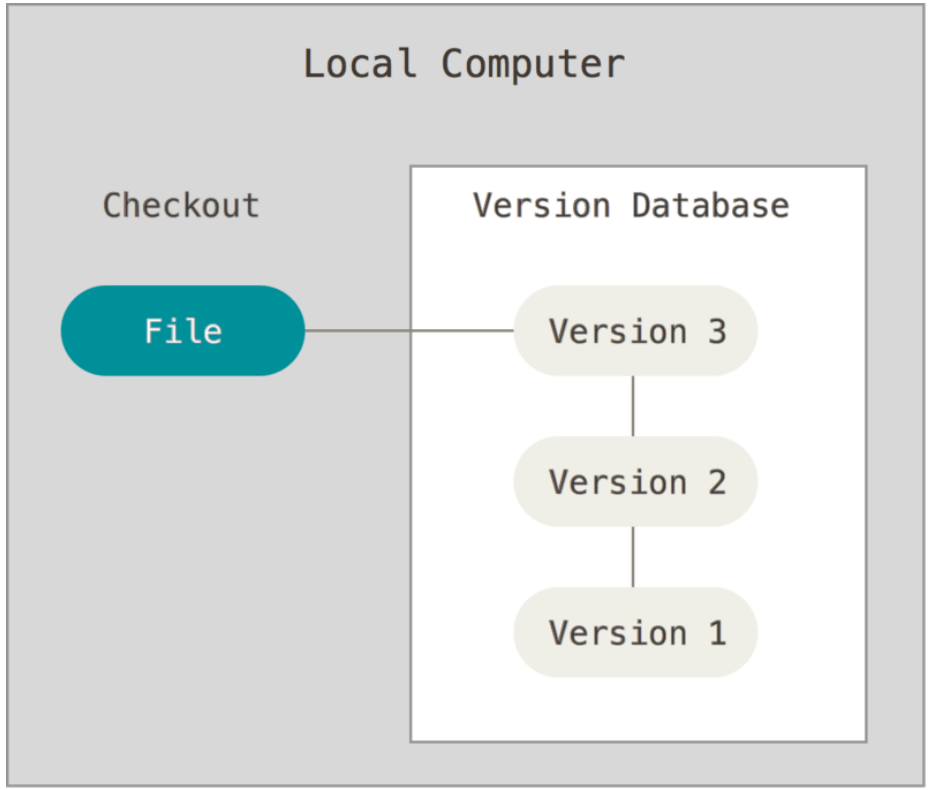
- 협업을 용이하게 함
- 코드 변경사항을 효과적으로 추적할 수 있음
- 프로젝트의 안정성과 신뢰성 확보

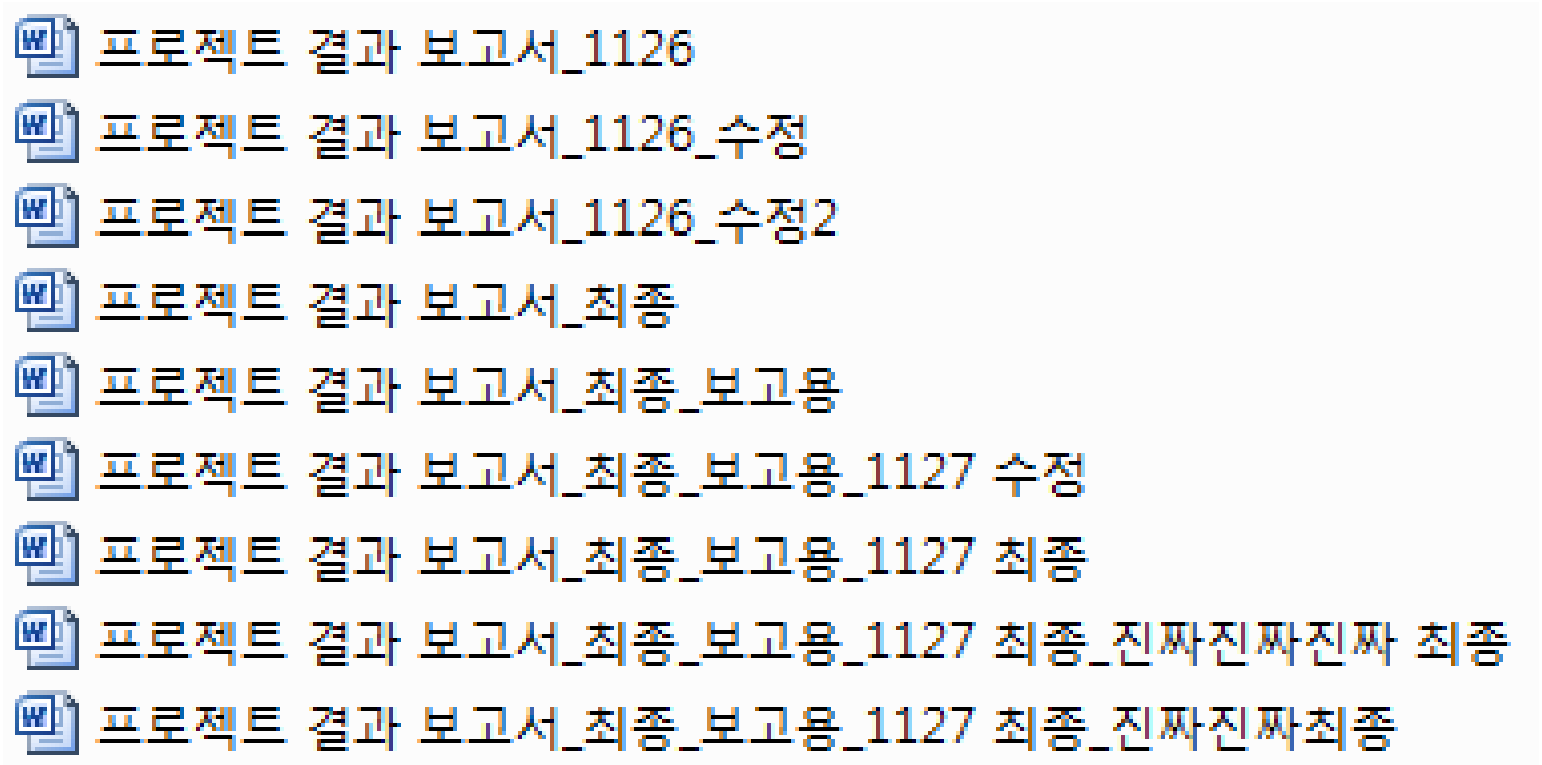
버전 관리의 간략한 역사

- 초기 버전 관리 시스템(VCS): RCS 등
- 중앙 집중식 버전 관리 시스템(CVCS): SVN, Perforce 등
- 분산형 버전 관리 시스템(DVCS): Git 등

우리가 이미 하고 있는 버전 관리

- 문서 툴 제공 : Google 문서, 드롭박스, 노션 등
- 온라인 편집 : 위키피디아, 나무위키 등
- ~~파일명으로~~




프로젝트 결과 보고서_1126
프로젝트 결과 보고서_1126_수정
프로젝트 결과 보고서_1126_수정2
프로젝트 결과 보고서_최종
프로젝트 결과 보고서_최종_보고용
프로젝트 결과 보고서_최종_보고용_1127 수정
프로젝트 결과 보고서_최종_보고용_1127 최종
프로젝트 결과 보고서_최종_보고용_1127 최종_진짜진짜진짜 최종
프로젝트 결과 보고서_최종_보고용_1127 최종_진짜진짜최종

<붙임> 1. 코로나19 격리입원치료비 지원 및 본인부담 사전고지서
2. 단계적 일상회복 1차 개편 주요 방역 수칙
3. 사회적 거리두기 개편안 관련 질의 답변
4. 재택치료 관련 주요 질의 답변
5. 감염병 보도준칙

· 첨부파일 : [보도참고자료]_코로나19_사회적_거리두기_강화_추진(진짜최종).hwp 바로보기

버전 관리의 장점

- 프로젝트 변경사항의 추적 및 이해
- 코드 충돌의 최소화
- 실수로 인한 손상된 코드의 복구 가능성
- 팀원들 간의 원활한 코드 공유
- 각 버전에 대한 변경 로그 확인 가능

Git

- Linux 커널 개발자인 Linus Torvalds가 2005년에 개발
- 다양한 프로젝트에서 사용되며, 오픈 소스 생태계에서 표준으로 채택
- 특징 : 분산 저장 시스템, 델타 기반 기록 등

Git 시작하기

Git 설치

- Mac
  - homebrew 설치
  - `brew install git`
- Windows (택 1)
  i. cmder 설치 (git 포함, linux 명령어 사용)
  ii. git 홈페이지에서 배포판 받아 설치
- `git --version` 으로 확인

Sourcetree 설치

`https://www.sourcetreeapp.com/`

`git init`

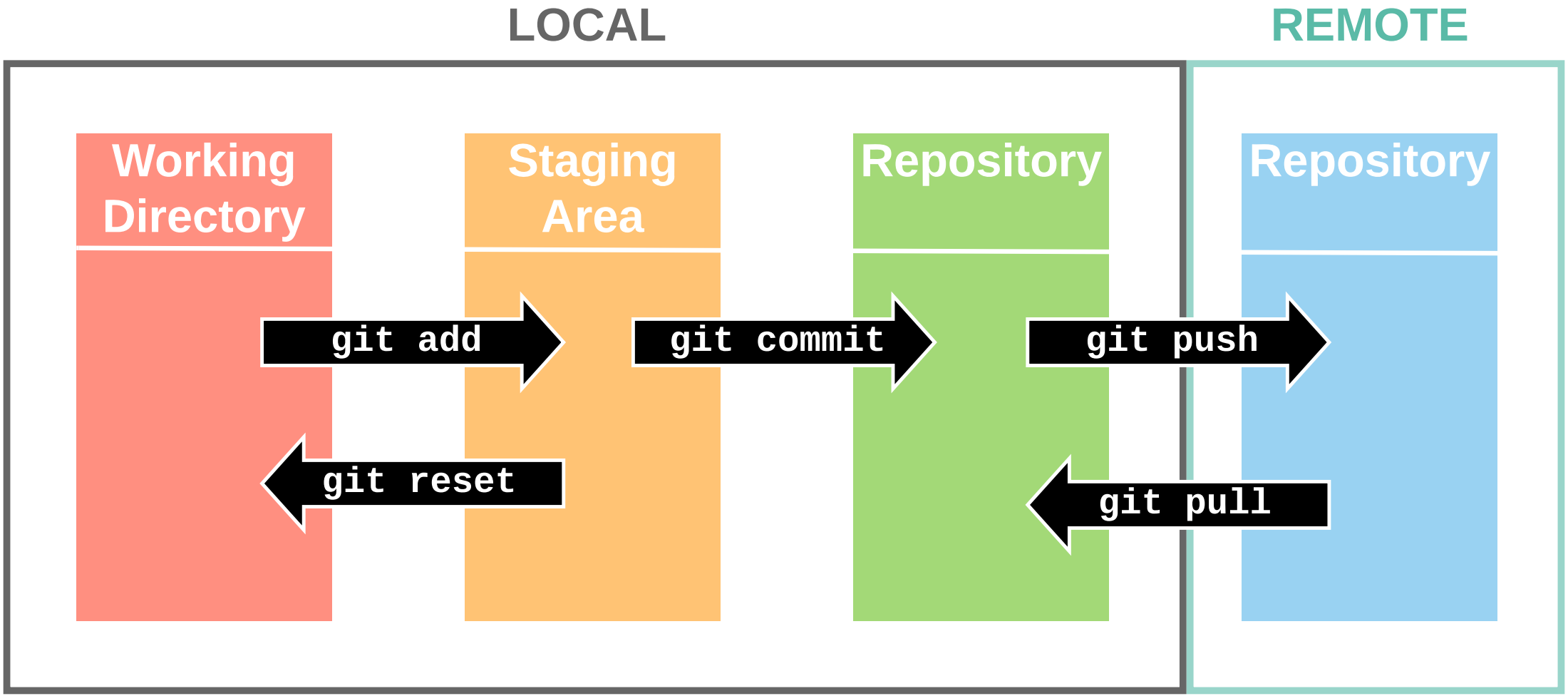
```
mkdir repo
cd repo
git config --global user.name "<username>"
git config --global user.email "<email>"
git init
git status
```

- `git config`
  - user.name : 커밋할 때 사용할 사용자 이름 설정
  - user.email : 커밋할 때 사용할 이메일 주소 설정 (github account와 동일 설정)
- `git status`
  - git 저장소 상태 확인

로컬 저장소 만들기

- 로컬 저장소와 리모트 저장소
  - 로컬 저장소: 개발자 개인의 컴퓨터에 있는 저장소
  - 리모트 저장소: 인터넷 상에 공유되는 저장소 (예: GitHub, GitLab)
- `git init`
  - 해당 디렉토리를 Git 저장소로 설정

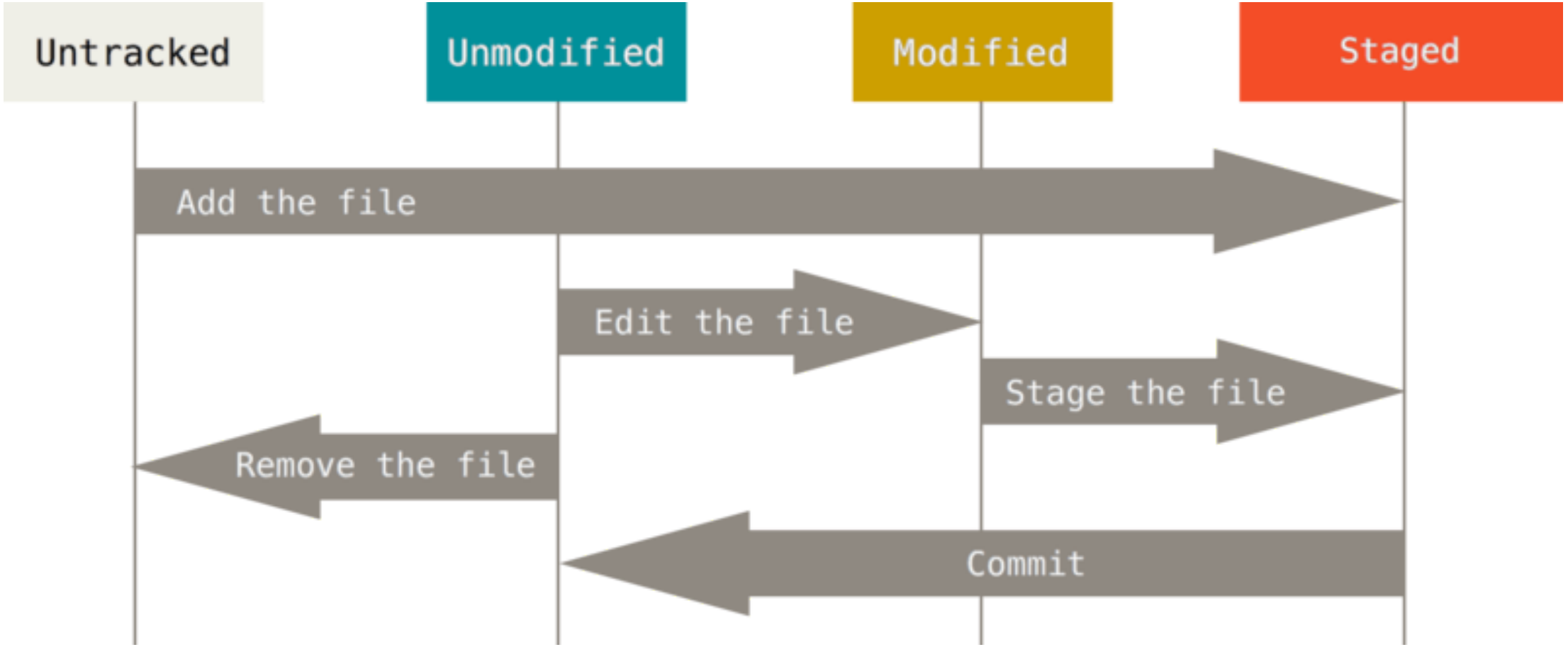
git 각 영역들


LOCAL
REMOTE
Working Directory
Staging Area
Repository
Repository
git add
git commit
git push
git reset
git pull

- **working directory (작업 디렉토리)**
  - 프로젝트 파일들이 위치하는 공간으로 파일의 변경 작업이 이루어지는 곳
  - 파일 상태
    - untracked: Git이 추적하지 않는, 아직 버전 관리에 포함되지 않은 파일
    - tracked: Git이 추적하는 상태
      - unmodified: 변경되지 않은 파일
      - modified: 수정된 파일
- **staging area (스테이징 영역, 인덱스)**
  - 작업 디렉토리에서 변경한 내용을 `commit` 하기 전 준비하는 공간
  - `git add` 명령어로 변경 사항을 스테이징 영역에 추가
  - 파일 상태
    - staged: 변경된 파일이 스테이징 영역에 추가된 상태

- **local repository (로컬 저장소)**
  - 컴퓨터의 로컬 환경에 위치한 Git 저장소
  - 변경 내역의 이력과 메타데이터가 저장되는 공간
  - `git commit` 명령어로 스테이징 영역의 변경 사항을 로컬 저장소에 기록
- **remote repository (원격 저장소)**
  - 인터넷 상의 원격 서버에 위치한 Git 저장소
  - 팀원 간의 협업과 코드 공유가 가능한 공간
  - `git clone`, `git fetch`, `git pull`, `git push` 명령어로 원격 저장소와 로컬 저장소 간 동기화

파일 상태 변화


Untracked
Unmodified
Modified
Staged
Add the file
Edit the file
Stage the file
Remove the file
Commit

working directory -> staging area

- `git diff <file>`
  - 파일의 변경 사항 확인
- `git add` : 스테이징 영역에 추가
  - 파일 추가 : `git add a.txt b.txt`
  - 패턴으로 추가 : `git add *.txt`
  - 현 디렉토리 모든 파일 추가 : `git add .`

실습

- `git diff`로 변경점 확인 후 `git add`를 사용하여 staging area에 파일 올리기

local repository

- **`git commit -m <message>`**
  - 커밋 메시지와 함께 local repository에 커밋
- **`git log`**
  - 커밋 히스토리 확인
  - 커밋 로그 한 줄 요약: `git log --oneline`
  - 커밋 로그 그래프: `git log --graph`
- **`git checkout <commit>`**
  - 해당 커밋으로 되돌리기
  - 이후 `git checkout -` 으로 돌아옴

실습 : `git commit` 으로 로컬 저장소에 커밋을 여러 번 하고, `git log` 로 확인 후 원하는 commit으로 `git checkout` 하여 `cat <filename>` 으로 확인

`git init` 심화

- **git init시 동작**
  - .git 디렉토리 생성
  - .git 디렉토리에 저장소 설정 파일 생성
- **새 디렉토리에서 init vs 기존 디렉토리에서 init**
  - 새 디렉토리에서 init
    - 새로운 프로젝트를 시작하는 경우 사용
    - 빈 저장소가 생성됨
  - 기존 디렉토리에서 init
    - 기존 프로젝트를 Git으로 관리하려는 경우 사용
    - 현재 디렉토리의 파일들이 추적되지 않은 상태로 시작

.git 디렉토리

- HEAD: 현재 브랜치에 대한 참조
- config: `git config`으로 설정한 config 파일
- hooks: 기본적인 Git 동작 전후에 실행되는 스크립트 모음
- info: 프로젝트의 관련 정보가 저장되는 디렉토리
- objects: 모든 데이터 (커밋, 트리, 블롭)가 저장되는 디렉토리
- refs: 브랜치, 태그 등 참조 정보를 담은 디렉토리
- COMMIT_EDITMSG : 마지막으로 커밋한 메시지
- **`rm -rf .git`**
  - .git 디렉토리와 모든 버전 관리 정보가 삭제
  - Git 저장소에서 제거된 상태로 롤백

Github 시작하기

Github 시작하기

- **Github 계정 만들기**
  - 사용자 이름, 이메일, 비밀번호 설정
  - 이메일 확인을 통한 계정 활성화
- **personal token 복붙**
  - Github -> Settings
  - 개발자 설정에서 Personal access tokens 클릭 (classic)
  - 새 토큰 생성, 필요한 권한 선택 및 토큰 발급 (repo 모두 체크)
  - 발급된 토큰 복사하여 사용
    - mac terminal : username 입력 후 password 자리에 token 입력
    - windows cmder : auth manager

repository 만들기

- GitHub에서 New repository 클릭
- 저장소 이름, 설명, 공개 여부 설정
- 초기 README.md 파일 생성 및 라이선스 선택

remote repository 설정

- `git remote add`
  - 로컬 저장소와 원격 저장소 연결
  - 명령어 입력: **`git remote add origin <repository_url>`**
  - 'origin'은 remote repository 의미 (by convention)
  - `git config` 확인

remote repository로 push

- `git push`
  - 로컬 저장소에서 변경 사항 스테이징: `git add`
  - 변경 사항 커밋: `git commit`
  - 커밋한 내용을 원격 저장소로 전송: `git push origin <branch_name>`
- `.gitignore`
  - 깃에서 추적하지 않을 파일 목록을 지정하는 파일
  - 프로젝트 루트 디렉토리에 생성
  - 각 줄에 무시할 파일 패턴 작성

저장소 받아오기

- `git pull`
  - 원격 저장소의 변경 사항을 로컬로 가져와 병합하기: `git pull origin <branch_name>`
- `git fetch`
  - 원격 저장소의 변경 사항을 로컬로 가져오기 (병합은 수동으로): `git fetch origin <branch_name>`
- `git clone`
  - 원격 저장소의 프로젝트 전체를 로컬로 복사하기: `git clone <repository_url>`

Repository 탐색

- 주요 기능
  - 코드 보기
  - 커밋 이력 조회
  - 브랜치 관리
- Issue 생성
  - 이슈 탭에서 New issue 클릭
  - 이슈 제목과 설명 작성
  - 라벨, 담당자, 마일스톤 지정
- Collaborators
  - 저장소의 설정 탭에서 Collaborators 메뉴 접속
  - 협업할 사용자를 검색 및 초대

협업을 위한 GitHub

협업을 위한 브랜치 전략

- 브랜치란?
  - 프로젝트에서 독립적으로 작업할 수 있는 단위
  - 각각의 브랜치에서 개발하거나 수정한 내용은 다른 브랜치에 영향을 주지 않음
  - 신기능 개발, 버그 픽스, 배포 생성 등

- Git Flow
  - master: 안정화된 배포 버전의 코드가 저장
  - develop: 개발 중인 버전의 코드가 저장
  - feature: 기능 개발을 위한 브랜치
  - release: 배포를 준비하기 위한 브랜치
  - hotfix: 긴급 수정을 위한 브랜치
- GitHub Flow
  - master: 항상 배포 가능한 상태를 유지
  - 각 기능별 브랜치: master 브랜치로부터 분기
  - 풀 리퀘스트를 통한 코드리뷰 및 병합