**git 기초 명령**

# **git의 기본**

# **시작하기**

소스코드 파일의 버전 백업

소스코드를 수정하다가 수정 전 상태로 되돌리고 싶을 때 방법

1. 수정을 시작하기 전에 소스코드 파일들을 따로 백업해두기

문제점) 소스코드 파일을 편집할 때 마다 매번 백업하는 일이 번거롭다

백업을 잊어버린경우 낭패를 본다

백업이 많아지면 구체적인 내용이 기억나지 않아 혼란스럽다

1. 프로젝트 공동작업

같이 프로젝트를 진행할 때 구현한 코드를 다른 팀원이 망치는 경우

팀원들이 각자 구현한 버전들을 전부 취합할 때 어려움

1. 버전관리 시스템

위와 같은 문제들을 해결하기 위해 만들어진 것이 git과 같은 버전 관리 시스템이다.

git에서는 소스 코드가 변경된 이력을 쉽게 확인할 수 있고, 특정 시점에 저장된 버전과 비교하거나 특정 시점의 버전으로 되돌아갈 수도 있다.

또 내가 팀 공동 repository에 올리려는 파일이 팀원 누군가 편집한 내용과 충돌한다면, 공동 repository에 올릴 때 경고 메시지가 발생하고 업로드는 취소된다. 그래서 누군가 애써 편집한 내용을 날려버리는 실수는 걱정하지 않아도 된다.

# **git repository**

git repository에는 소스코드 파일의 수정 과정에서 만들어졌던 전체 버전들이 전부 저장된다.

## **remote repository와 local repository**

git repository는 local repository와 remote repository가 있다.

## **remote repository**

팀원들이 공유하기 위한 repository다. 팀원들 모두 접근할 수 있는 remote서버에 만들어진다.

무료로 이용할 수 있는 대표적인 git remote repository 서버는 github 이다.

## **local repository**

개발자 개인의 PC에 생성되는 개인 전용 repository이다.

평소에는 팀원 각자 개인 PC의 local repository에 버전을 저장하며 작업하다가, 그동안 작업한 내용을 팀원들에게 공개하고 싶을 때, local repository에 저장된 내용을 remote repository에 업로드 한다.

누군가 remote repository에 작업한 내용을 업로드하면, 다른 팀원들은 그 내용을 각자의 개인 PC의 local repository에 다운로드 할 수 있다.

## **local repository 만들기**

개인의 PC에 local repository를 만드는 방법은 두 가지가 있다.

1. 비어있는 local repository를 새로 만든다.
2. 이미 만들어져 있는 remote repository를 다운로드 해서 local repository를 새로 만든다.

# **변경을 기록하는 commit**

local repository에 새 버전을 등록하는 작업을 commit이라고 한다.

commit을 할 때만 local repository에 버전이 등록된다.

어느정도 의미있는 소스 코드 수정 작업을 마쳤을 때마다,

그 작업을 마친 버전을 local repository에 commit하는 것이 바람직하다.

예를 들어, 어떤 기능 구현을 완료했다거나, 어떤 버그를 해결했다면,

그 작업 결과 버전을 local repository에 commit한다.

commit은 소스코드 변경 이력을 저장소에 기록하는 중요한 작업이다.

## **commit 메시지**

local repository에 commit을 등록할 때, commit 메시지를 필수로 입력해야 한다.

작업 내용을 commit 메시지에 요약해서 입력하면 된다.

작업 내용을 팀원들이 쉽게 이해할 수 있도록 입력하는 것이 좋다.

ommit 메시지가 여러 줄인 경우 다음과 같은 형식을 주로 쓴다.

첫째 줄을 commit 메시지 제목을 적는다.

둘째 줄은 비워둔다.

셋째 줄부터 commit 메시지 본문을 적는다.

## **commit hash**

각 commit에는 알파벳과 숫자로 이루어진 40자리의 hash가 자동으로 부여된다.

commit hash는그 commit을 식별하기 위한 기본키(primary key)에 해당한다.

# **working tree와 staging area**

git 버전 관리 대상인 소스코드 폴더를 working tree라고 부른다. (working directory)

commit을 실행하기 전 local repository와 working tree 사이에 staging area 라고 불리는 가상의 공간이 있다.

git의 commit 작업은 working tree에 있는 변경 내용을 전부 repository에 바로 기록하는 것이 아니라, staging area에 등록된 파일들의 변경 내용만 다음 번 commit에서 기록된다.

예를 들어, 10개의 파일을 수정했지만 그 중에 7개 파일의 변경 사항만 repository에 기록하고 싶다면, 그 7개 파일을 staging area에 등록한 후 commit 해야 한다.

이렇게 staging area에 등록된 파일의 변경 사항만 repository에 기록되기 때문에,

working tree에 이는 파일들 중 내가 원하는 변경 사항만 repository에 commit할 수 있다.

## **working tree 스냅샷(snapshot)**

commit을 하면, working tree의 현재 상태가 local repository에 기록된다.

그래서 나중에 그 스냅샷 상태로 working tree를 되돌리는 것이 가능하다.

단 staging area에 등록된 파일들만 local repository에 기록된다.

# **working tree 내부의 파일의 상태**

working tree 내부의 파일 상태는 다음 상태 중 하나이다.

## **untracked**

한 번도 commit되지 않았고, 아직 staing area에 등록되지도 않은 새 파일의 상태이다.

## **modified**

변경 내용이 commit된 이후 수정되었지만, staging area에 다시 등록되지 않아서, 다음 commit 대상이 아닌 파일의 상태는 dirty / modified 이다.

## **staged**

1. 아직 한 번도 commit되지 않은 새 파일이고, 처음으로 commit되기 위해,

staging area에 등록된 파일의 상태는 staged이다. (untreaked => staged) new file

1. 변경 내용이 commit된 이후 수정되었고, 그 수정 내용이 다시 commit 되기 위해,

staging aea에 등록된 파일의 상태는 staged 이다. (modified => staged)

## **unmodified**

변경 내용이 commit 된 이후로 수정되지 않은 파일들의 상태는 unmodified 이다.

# **working tree 내부의 파일의 상태 변화**

## **상태 변화**

working tree에 새 파일을 생성하면 그 파일은 untracked 상태이다.

untracked 파일을 staging area에 등록하면 staged 상태가 된다.

staged 파일을 commit 하면 unmodified 상태가 된다.

unmodified 파일을 수정하면 modified 상태가 된다.

modified 파일을 staging area에 등록하면 staged 상태가 된다.

staged 파일을 commit 하면 unmodified 상태가 된다.

## **staing area 등록 대상 파일**

untracked 파일

modified 파일

# **git 기초 명령**

# **local repository 명령**

## **local repository 새로 생성하기**

현재 디렉토리에 local repository 새로 생성하기

git init

## **working tree의 파일 상태 확인**

git status

# **staging area 등록/취소 명령**

## **파일을 staging area에 등록하기**

git add 파일명

예:

git add a.txt => 현재 디렉토리의 a.txt 파일을 staging area에 등록한다.

git add \*.txt => 현재 디렉토리의 \*.txt 패턴의 파일 중 staging 대상 파일을 staging area에 등록한다.

git add “\*.txt” => working tree의 \*.txt 패턴의 파일 중 staging 대상 파일을 staing area에 등록한다.

git add \* => 현재 디렉토리의 모든 staging 대상 파일들을 staging area에 등록한다.

git add . => working tree의 모든 staging 대상 파일들을 staging area에 등록한다.

## **staging area에 등록 취소하기**

git reset 파일명

예:

git reset a.txt => a.txt파일을 staing area에 등록을 취소한다.

git reset => staging area의 모든파일을 등록 취소한다.

git rm --cashed 파일명 => staging area에 등록 취소 명령이다.

# **commit 명령**

## **commit 하기**

git commit

위 명령을 실행하면, commit 메시지를 입력하기 위한 폅집기가 자동을 열린다.

commit 메시지를 입력하고 저장한 후 편집기를 닫으면, commit 명령이 완료된다.

저장하지 않고 편집기를 닫으면, commit 명령이 취소된다.

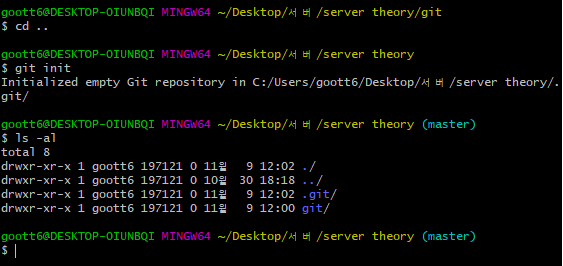
## **commit 목록 보기**

git log

git log --stat => commit에 기록된 파일 목록과 삽입/삭제된 줄 수를 볼 수 있다.

# **실습**

# **local repository 만들기 (git init)**



## **git init**

현재 디렉토리에 local repository를 만든다.

## **ls -al**

현재 디렉토리의 파일 목록을 출력한다.

## **local repository**

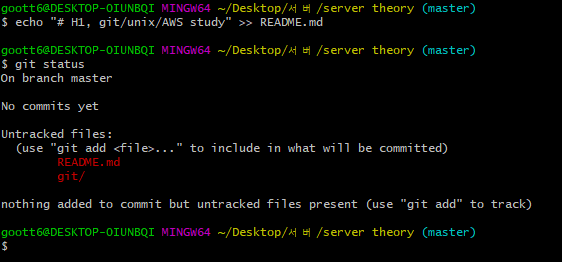
.git 디렉토리가 생성되었다.

이 디렉토리가 현재 프로젝트의 local repository 이다.

# **파일 상태 확인 (git status)**

파일 상태 확인 명령은 다음과 같다.

git status



## **No commits yet**

아직 commit 이 없음

## **untracked files**

README.md

git/

위 두 파일은 untracked 상태이다.

## **nothing added to commit but untracked files present**

수정된 파일 중 tracked 상태의 파일이 없어서 commit 할 파일이 없다.

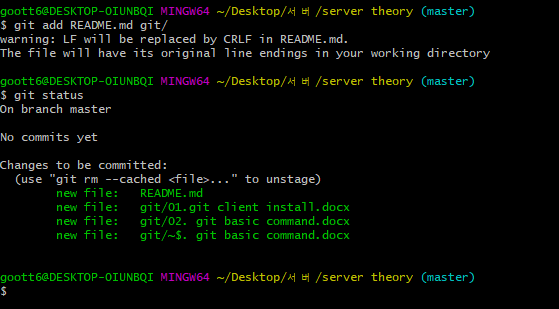
그러나 untraked 상태의 파일이 있다.

tracked 상태의 파일만 commit 될 수 있기 때문이다.

# **새 파일을 tracked 상태로 바꾸기 (git add)**

새 파일을 tracked 상태로 바꾸는 명령

git add 파일명



## **On branch master**

현재 작업 브랜치는 master이다.

## **No commits yes**

아직 commit 없다.

## **changes to be committed**

수정된 파일 중에서 commit 대상은 다음과 같다. (tracked 상태의 파일만 commit 가능하다)

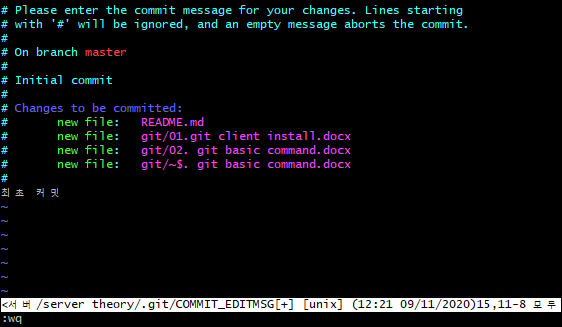
# **수정 내용 등록하기 (git commit)**

파일 수정 내용을 local repository에 등록하는 명령

git commit

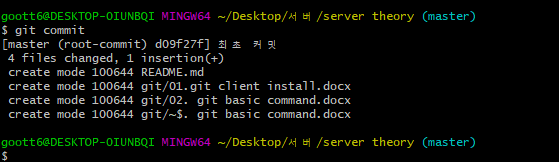
위 명령을 실행하면, commit 메시지를 입력하기 위한 편집기가 자동으로 열릴 것이다.

git 클라이언트 설치할 때, 선택한 텍스트 편집기가 열린다.



#으로 시작되는 라인은 메시지에 포함되지 않고 무시된다.

메시지를 입력학 편집기를 종료하면, git commit 명령이 완료된다.



## **[master (root-commit) d09f27f] 최초 커밋**

master 브랜치에 커밋 되었음.

커밋의 ID : d09f27f

커밋 메시지 : 최초 커밋

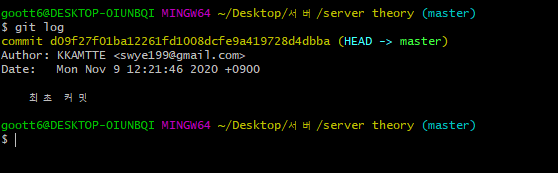
4 files changed, 1 insertion(+)

수정된 파일은 4개이고, 1줄이 추가됨.

# **커밋 목록 보기 (git log)**

커밋 목록을 보는 명령

git log



## **commit d09f27f01ba12261fd1008dcfe9a419728d4dbba**

commit ID는 d09f27f…이다

ID가 이렇게 길지만, 보통 ID의 앞 7자만 사용한다.

## **HEAD -> master**

현재 작업하고 있는 브랜치의 선두 버전이, master 브랜치의 이 커밋이다.

# **remote repository 명령**

# **remote reository에 연결**

현재 디렉토리의 local repository를 remote repository에 연결하기 위한 명령은 다음과 같다.

git remote add 별칭 URL

별칭 => remote repository의 짧은 이름

URL => remote repository URL

예:

git remote add origin <https://github.com/KKamtte/ServerStudy.git>

위 명령에서 remote repository URL은 다음과 같다.

[**https://github.com/KKamtte/ServerStudy.git**](https://github.com/KKamtte/ServerStudy.git)

위 remote repository에 **origin** 이라는 별칭(별명)을 부여한다.

# **remote repository에 업로드**

## **최초 업로드**

git push --set-upstream 원격저장소\_별칭 업로드할\_브랜치

local repository의 내용을 remote repository에 최초 업로드할 때는 위 명령을 실행한다.

원격저장소\_별칭 : remote repository의 별칭을 말한다.

앞에서 remoe repository URL을 연결할 때 사용한 별칭이다. (git remote add)

이 별칭은 **origin**이다.

업로드할\_브랜치 : 업로드할 브랜치 이름이다.

현재 작업 브랜치 이름은 **master** 이다.

예:

git push --set-upstream origin master

## **최초 업로드 이후**

최초 업로드 이후에는 다음 명령을 실행하면 된다.

git push

# **remote repository에 다운로드**

remote repository를 다운로드 하여 local repository를 새로 만드는 명령은 다음과 같다.

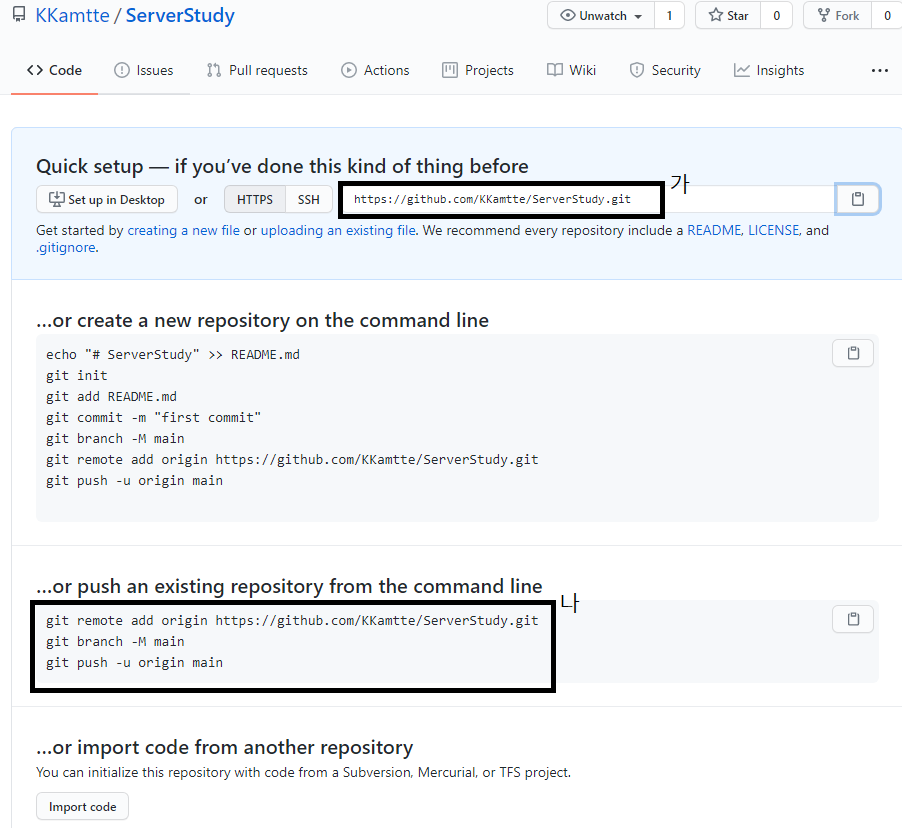
git clone URL

예:

git clone <https://github.com/KKamtte/ServerStudy.git>

# 실습

# remote repository 생성



## [**https://github.com/KKamtte/ServerStudy.git**](https://github.com/KKamtte/ServerStudy.git)

생성된 remote repository의 URL 이다.

git remote add origin <https://github.com/KKamtte/ServerStudy.git>

git branch -M main

git push -u origin main

local repository를 remote repository에 연결하고 업로드 하기 위한 git 명령이다.

# **remote repository에 연결**

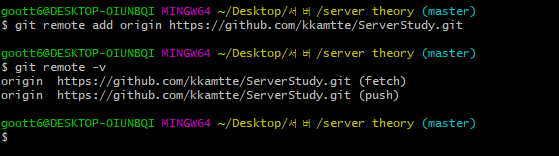
현재 디렉토리의 local repository를 remote repository에 연결하기 위한 명령은 다음과 같다.

git remote add origin <https://github.com/kkamtte/ServerStudy.git>

위 명령에서 remote repository URL은 다음과 같다.

<https://github.com/kkamtte/ServerStudy.git>

위 remote repository에 **origin** 이라는 별칭(별명)을 부여한다.



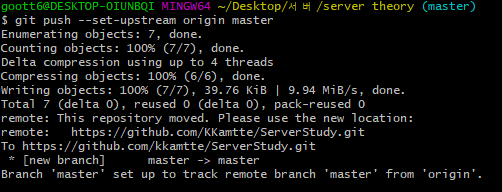
git remote -v

연결된 remote repository URL 확인

# **작업 내용 업로드**

## **최초 업로드**

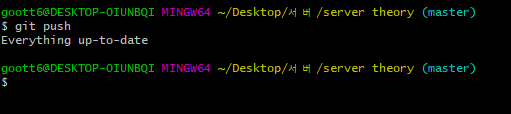
git push --set-upstream origin master



## **최초 업로드 이후**

최초 업로드 이후에는 다음 명령을 실행하면 된다

git push



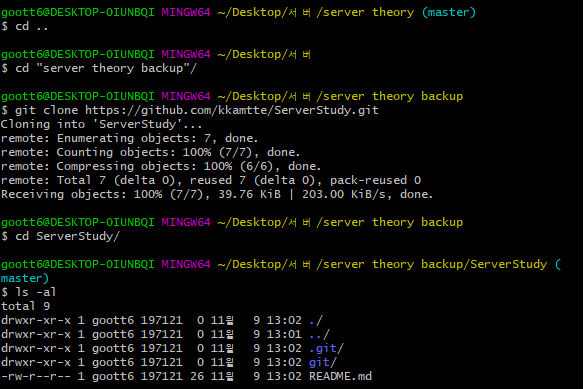
이전 업로드 이후 수정된 내용이 없으면, 위와 같이 화면이 출력된다.

# **remote repository 다운로드**

## remote repository 다운로드 (clone)

remote repository를 다운로드 하는 명령은 다음과 같다.

git clone URL



git clone <https://github.com/kkamtte/ServerStudy.git>

현재 디렉토리 아래에 ServerStudy 디렉토리를 생성하고 remote repository를 다운로드 한다.

디렉토리 목록을 보면

local repository(.git)가 생성되었고, 파일도 전부 다운로드 되어음을 확인할 수 있다.

즉 server theory에서 업로드한 작업 공간이 전부 server theory backup에 복제되었다.