OSS 7-1 ~ 7-2

OSS (Open Source Software)
- 누구나 특별한 제한 없이 그 코드를 보고 사용할 수 있는 오픈소스 라이선스를 만족하는 SW
    - 소스코드를 공개, 공개 소스의 복제 가능

OSI (Open Source Initiative)
- 공개 소스 정의(OSD)의 관리 및 촉진을 담당하는 비영리 조합
- OSI가 인증하는 공개 소스 소프트웨어(OSS) 인증 마크
    ▶ 소프트웨어가 실제로 공개 소스라는 것을 증명

▶ 의미 : 소프트웨어의 소스코드를 자유롭게 읽고, 수정 및 재배포가 가능

자유 소프트웨어 (Free Software)
- 리처드 스톨먼, 자유 소프트웨어 재단 설립
- GNU 프로젝트와 관련된 소프트웨어에서 자유를 중시
- 제작자에게 저작권은 인정하면서 어느 누구나 소프트웨어를 복제해 사용할 수 있는 권리를 주어야 한다는 개념
- 공개된 무료 유닉스 운형체제 Free BSD (매킨토시, 아이폰 등의 OS로 발전되어 사용)

오픈소스 지원 관리 서버 : 소스코드를 통해 여러 개발자가 협업하고 공유, 이를 지원하는 방안을 마련
    - github, gitlab, bitbucket
    - 오픈소스 커뮤니티 내의 사고 및 협업 양식 / 커뮤니티에서 개발된 아이디어와 소프트웨어를 교환

오픈소스 소프트웨어 장단점
장점 : 쉽게 연구, 새로운 프로그래밍 기술 개발, 광범위한 오픈소스 커뮤니티와 공유
    커스터마이징과 혁신 지원 (요구사항에 맞는 커스터마이징 가능)
단점 : 공개의 의무, 품질보증 및 유지보수, 보안 등의 어려움

▶ 대부분의 웹을 지원하는 서비스 스택 모델 :LAMP
1. Linux : 오픈소스 운영체제(OS)이자 세계 최대 규모의 오픈소스 프로젝트
2. Apache : 초기 웹에서 핵심 역할을 한 오픈소스 크로스 플랫폼 웹 서버
3. MySQL : 대부분의 데이터베이스 기반 웹 어플리케이션에서 사용하는 오픈소스 관계형 데이터베이스 관리시스템
4. PHP : 소프트웨어 개발에 사용되는 범용 스크립팅 언어

오픈소스 라이선스 저작권
- 오픈소스 SW 개발자가 만들어놓은 저작권의 범위에 따라 해당 소프트웨어를 사용

소스코드 반환의무
- GPL, AGPL, LGPL, MPL, EPL 등
- 링크되거나 코드가 포함된 SW 소스코드를 공개, 저작권 고지 의무

1. GPL (GNU - General Public License)
- 자유 소프트웨어재단 (FSS)에서 만든 라이선스 / 리처드 스톨만 제작
- 카피레프트에 속함, 프로그램은 목적이나 형태의 제한 없이 사용 가능
- 프로그램을 이후 수정하고 배포하는 모든 경우에 무조건 GPL로 공개를 해야 함

2. AGPL (GUN - Affero GPL)
 - GPL을 기반으로 만든 라이선스
 - 네트워크로 상호 작용하는 소프트웨어의 소스코드도 공개해야 한다는 조항을 추가한 라이선스
 - 소스코드의 공유 불가 현상을 해결하기 위해 만들어짐

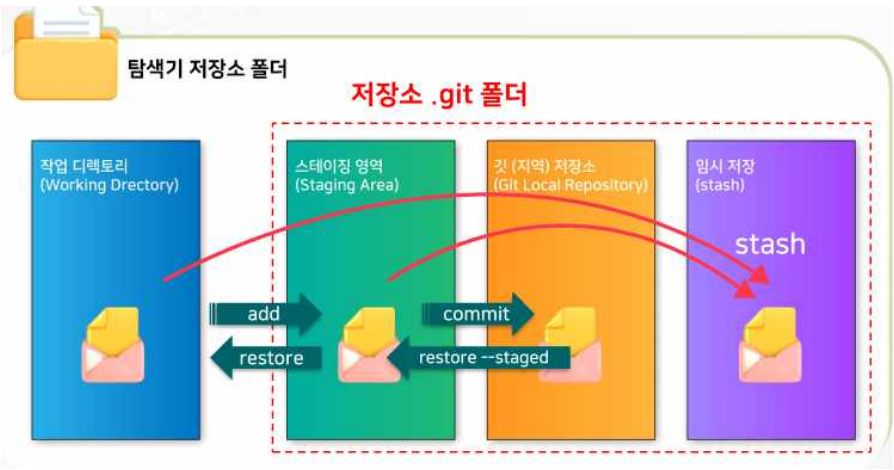
3. LGPL (GUN - Lesser GPL)
 - 보다 완화된 GPL 라이선스
 - GPL 라이선스를 사용할 때 반드시 소스코드를 다시 GPL로 공개해야 하는 부담 때문에 실무에서 사용되기 어려운 점을 보완하기 위해 만들어진 라이선스
 - 전체 소스코드를 공개하지 않고 사용된 오픈소스 라이브러이에 대한 소스코드만 공개

4. Apache License
 - 소스코드 공개에 대한 의무사항은 라이선스에 포함되어 있지 않음

5. MIT License
 - MIT에서 학생들을 지원하기 위해서 만든 라이선스
 - 라이선스와 저작권 관련 명시 의무
 - 가장 느슨한 조건을 가지고 있어 많은 사람들이 사용하기 용이
 - 사용자가 최소의 제한사항으로 원래 코드를 수정할 수 있도록 허용하는 무료 소프트웨어 라이선스

깃 4영역과 임시저장 개요

깃 3영역 + stash 영역



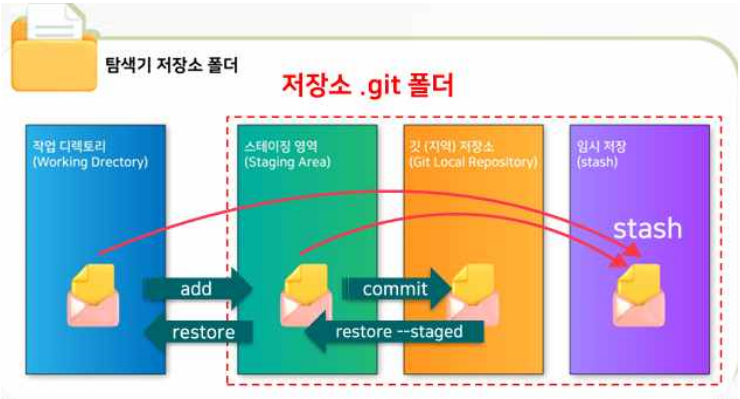
깃 영역의 가장 단순한 상태 : working tree is clean

Git Stash
- 커밋할 필요 없이 파일의 변경 사항을 숨기거나 비밀리에 저장할 수 있는 강력한 도구
- 작업 디렉토리와 스테이징 영역의 원래 내용을 stash 스택에 저장
- 작업 디렉토리와 스테이징 영역의 값을 깃 저장소의 마지막 커밋 내용으로 복사
- 따로 안전한 곳에 저장했다가 다시 가져올 수 있는 기능

Stash 저장 구조
- 깃에서 임시저장소 (스택구조)에 저장 / Last In - First Out

Git stash 필요성
- 작업 디렉토리 내용과 스테이징 영역 내용이 stash에 저장, 최신 커밋 자료로 남음

작업 폴더와 스테이징 영역을 stash에 저장
작업폴더를 정리
- $ git stash
- $ git stash -m '메세지'
- $ git stash save
- $ git stash save '메세지'



기본으로 작업 디렉토리 내용만 다시 복사해 활용
- $ git stash apply
스테이지 영역도 함께 복사하기 위해 옵션 사용
- $ git stash apply --index

주요 옵션
- 옵션 -k | --keep-index : 스테이징 저장 X, checkout 못함
- 옵션 -u | --include-untracked : Untracked 파일도 포함해 저장
- 옵션 -p | --patch : 변경된 모든 사항들을 저장하는 것 X, 지정 가능

특정 stash 확인 : 커밋 자료와 최신 stash 항목 간의 차이로 표시
- $ git stash show
    1 file changed, 2 insertions(+) ← 변화된 파일과 변화된 수만 표시

임시 저장된 stash 반영 /특정 stash 삭제와 모든 stash 삭제

최근 or 지정된 임시저장소 내용을 가져와 반영 후 삭제	$ git stash pop $ git stash pop stash@ {n}
내용을 가져와 반영, 작업 디렉토리만 반영, stash 목록은 그대로	$ git stash apply $ git stash apply@ {n}
내용을 가져와 반영, 작업 디렉토리와 스테이징 영역도 반영, stash 목록은 그대로	$ git stash apply --index $ git stash apply --ndex stash@ {n}
최근 임시저장 내용을 삭제	$ git stash drop
지정된 임시저장 내용을 삭제	$ git stash drop stash@ {n}
모든 stash 목록을 모두 제거	$ git stash clear

Untracked 파일 삭제
- $ git clean : 바로 삭제되진 않음
- $ git clean -f : 무조건 삭제