OpenSource

오픈소스란?

소스코드에 대한 자유로운 접근, 재배포, 파생 저작물의 작성, 제한없는 사용 등을 허용하는 라이선스와 함께 배포되는 SW

OSS라고 불림

오픈소스 핵심 정의 요소 중 하나는 라이센스이며, 오픈소스는 이들 라이센스의 중요 요구사항들을 만족시켜야 함

독접 SW

Closed source SW혹은 독점 SW가 있음.

이는 소스코드에 대한 독접적인 소유권을 유지함.

ex) Windows

OSS와 독점SW의 차이점

1. 신뢰성
   독점 소프트웨어는 단일 조직 또는 개발자를 통해 코드를 업데이트함

오픈소스는 더 넓은 커뮤니티에서 관리

따라서 오픈소스의 경우가 더 세밀하고 신뢰성이 높은 경우가 많음

보는 눈이 많다.

2. 보안
   모든 소스코드는 보안에 취약한 결함이 있다.

오픈 소스는 수정이 더 빠름 커뮤니티 구성원이 보고하면 하루내에 업데이트됨

반대로 독점은 작업자가 적거나, 재정적 고려사항, 한번에 업데이트 선호등의 이유로 업데이트 주기가 김

3. 라이센스
   회사는 보통 독접 라이센스에 따라 판매함

누구도 허가 없이 보거나 편집할 수 없음

오픈소스 현황

2021년 기준으로 영향력이 더 커짐 시장 규모도 매년 증가

오픈소스 성립 조건

1. 무료 재배포
   - 오픈소스를 누구나 자유롭게 받아서 사용하고 재배포할 수 있음

- 라이센스는 SW를 판매하거나 제공하는 당사자를 제한해서는 안됨
- 라이센스는 해당 판매에 대한 로열티나 기타 수수료를 요구해서는 안 됨

2. 소스 코드

- 오픈소스의 소스코드를 누구나 볼 수 있도록 공개해야함
- 프로그램 소스코드를 포함, 형식까지 배포할 수 있어야 함

3. 파생 작품

- 라이센스는 수정 및 파생된 저작물을 허용해야함
- 원본 소프트웨어의 라이센스와 동일한 조건에 따라 배포될 수 있어야 함

4. 저자 소스코드의 무결성

- 오픈소스 소스코드의 무결성을 보장해야함
- 다른 개발자가 소스코드를 수정하거나 개선할 때, 그 수정이나 개선이 원래의 소스코드와 다르게 나타나지 않도록 보장해야함
- 수정된 소스코드는 반드시 원작자가 명시한 라이선스와 조건을 따라야 함

5. 개인 또는 집단에 대한 차별 금지

- 어떤 개인이나 집단을 차별해서는 안 됨

6. 노력 분야에 대한 차별 금지

- 특정 분야에서 프로그램을 사용하는 사람을 제한해서는 안 됨

7. 라이센스 배포

- 프로그램에 첨부된 권한은 해당 당사자가 추가 라이센스를 실행할 필요 없이 프로그램을 재배포하는 모든 사람에게 적용되어야 함
- 오픈소스 SW를 사용하거나 배포할 때는 반드시 해당 소프트웨어의 라이센스를 함께 제공해야함

8. 라이센스는 제품에 한정되지 않아야 함

- 라이센스는 특정 제품을 한정되지 않아야 함
- 프로그램에 첨부된 권리는 프로그램이 특정 소프트웨어 배포의 일부 인지에 따라 달라져서는 안 됨

9. 라이센스는 다른 소프트웨어를 제한해서는 안 됨

- 라이센스는 라이센스된 소프트웨어와 함께 배포되는 다른 소프트웨어에 재한을 두어서는 안 됨
- 예를 들어, 라이센스는 동일한 매체에 배포된 다른 모든 프로그램이 오픈소스 소프트웨어야한다고 주장해서는 안 됨

10. 라이센스는 기술 중립적이어야 함

- 개별 기술 또는 인터페이스 스타일에 대한 라이센스 조항은 제공하지 않음

오픈소스 성립 조건 필요성

모든 공개되어있는 소스가 다 유용한 소스 코드는 아님*특히 산업계*

어떤 오픈소스 제품에 넣었을 때 나중에 오픈소스 코드가 없어지거나 하면 난감해짐

없어진 오픈소스를 다시 만들어 오픈소스로 유지관리를 하든지 혹은 내부 코드로 자체적으로 개발을 해야함

제품 당 평균 수백개 정도의 오픈소스가 들어있다. 이를 생각하면 오픈소스를 이용하는 것은 위험한 소프트웨어 개발임

따라서 10개의 기준을 만족하는 것만 사용함

이것이 오픈소스 성립 조건 10가지

오픈소스 활용

오픈소스는 소스코드, 라이브러리, 유틸리티, 툴, 완제품수준에 이르기까지 다양한 방법으로 활용될 수 있다.

오픈소스 관련 단체

- FSF: Free Software Foundation
    - 리처드 스톨만에 의해 1985년 만들어진 비영리 단체
    - GNU 프로젝트를 운영하고 free SW를 배포 및 관리함
- OSI: Open Source initiative
    - 오픈소스 정의 및 관련 표준을 관리함
- SFLC : Software Freedom Law Center
    - 오픈소스 SW의 법률적인 문제를 해결하는 비영리 단체
- GPL Violations
    - GPL 위반 사례를 조사하고, 이를 해결하는 비영리 단체
- 기타
    - Linux Foundation, FOSS, OIN등

오픈소스 역사

- 초기
    - 컴퓨터 초기에는 SW는 독접적이지 않았음
    - 70, 80년 초반대부터 비로소 사람들이 자신 소프트웨어를 감추고 다른 사람이 볼 수 없게 함. 그 당시 컴퓨터를 사면 독접적인 운영체제를 사야했고 변경할 수 없었다..
- GNU 프로젝트
    - 리처드 스톨만은 소프트웨어 상업화에 반대하고 SW 개발 초기의 상호협력적인 문화로 돌아갈 것을 주장하며 1984년 GNU 프로젝트를 시작함
    - 이듬해 1985년에 FSF를 설립
    - 다음과 같은 GNU 선언문을 제정하고 이를 위해 CopyLeft운동을 주장
        - *소프트웨어는 공유되어야 하고 프로그래머는 SW로 돈을 벌어서는 안된다.*

- 이름 변경
    - 원래 자유소프트웨어라는 이름에서 오픈소스 소프트웨어로 영어 변경
    - 자유란 용어가 일반이이 무료로 인식하고 GPL조항 엄격성 때문에 SW개발이 용이하지 않다는 점을 탈피하기 위해서
- OSI결성
    - 1998년 오픈소스 SW를 인증하는 OSI가 에릭 레이몬드 등에 의해 결성되면서 오픈소스 SW운동은 궤도에 오르게 됨

GNU 프로젝트

- GNU 프로젝트
    - 1991년 리누스 토르발스는 유닉스 호환의 리눅스 커널을 작성하여 GPL 라이센스 아래 배포했고 다른 여러 프로그래머들에 의해 리눅스는 더 발전함. 1992년 리눅스는 GNU 시스템과 통합되었고 이로써 완전한 공개 운영 체제가 탄생됨
- 리눅스는 수많은 사용자가 있고 수많은 프로그래머가 인터넷을 공동으로 개발하고 있는 가장 대표적인 오픈소스 운영체제이다.

오픈소스 동기부여

왜 사람들은 팔지도 못하는데 만들까..? 자원봉사?

- 대가
    - 처음에 리눅스는 선의로 만들어집 보수가 없어도 좋고 주말에 SW 개발에 참여하고 이 결과물을 다른 기업 등이 이득을 봤으면 좋겠다는 마음으로 시작함
    - 그러나 최근에는 오픈소스 개발이 실제 상업적인 이익이 되며 대부분 상업적인 이유로 오픈소스에 참여하게 됨
    - 수많은 개발자가 개인의 이익을 위해 오픈소스에 기여하며 회사는 이런 개발자들에게 대가를 지불함
- 호환
    - 단순히 좋은 오픈소스를 찾아 잘 쓰기만 하고 참여는 안 한다?
    - 어떤 오픈소스를 제품에 사용한 이후에 업데이트되면서 기능이 많이 달라지면서 제품에 호환이 안 되는 경우 발생
    - 그 때마다 안 맞는 부분을 다 조정하는 것은 불가능. 수백개의 오픈소스를 사용하기 때문
    - 따라서 기업에서 자신이 가져다 쓴 부분에 대해 다음 오픈소스 버전에 적극적으로 기여함, 기여해 놓으면 그 기여한 부분이 반영된 버전을 그대로 가져다 쓸 수 있음
- 취업
    - 개발자에게 중요한 것은 실력
    - 실력을 정량화하기 어려움. 면접을 통해 실력이 있는지 알기 어려움
    - 회사에서의 작업물은 외부에 보여주기도 어렵고 말로 설명 쉽지않음
    - 오픈소스 참여는 취업에 도움됨. 특히 경력없는 신입 개발자가 이력을 보여주면 차별화를 둠
- 리뷰
    - 보통 오픈소스 committer들이 코드 수정을 그냥 받아주지 않고 유지관리를 함
    - 코드 리뷰를 많이 하고 이상한 것이 있으면 질문도 많이 함
    - 새로운 지식 학습이 가능하며, 무료로 자신의 코드를 리뷰 받을 수 있음
    - 좋은 코드 작성법
        - 요구 사항 잘 이해

        - 코드 작성
        - 리뷰 받음
        - 리뷰 반영

오픈소스 성공 사례

- stable diffusion
- tensorflow, pytorch
- flutter

소프트웨어 유형

- freeware
    - 다운은 자유롭지만 소스코드는 공개하지 않음
- shareware
    - 일정 기간 사용하고 돈을 지불하면 계속 사용할 수 있음
- Proprietary software
    - 소스코드를 공개하지 않음(회사)

소프트웨어 지식재산권

- 현재 SW는 다음과 같이 저작권, 특허권, 상표권 영업비밀 등의 지식재산권에 의해 보호받고 있다.
- 저작권
    - 창작물에 대하여 창작자가 취득하는 권리로서 창작의 결과물을 보호하며, 창작과 동시에 권리가 발생함
    - 따라서 어떤 프로그래머가 특정 SW를 개발하면 저작권이 자동 발생하며 그 권리는 프로그래머 또는 그가 속한 회사에 부여됨
    - 저작권이 있는 저작물의 경우 누구도 저작권자의 허락없이는 해당 저작물을 쓸 수 없다.
    - Copyleft: 저작권 없음이 아니라 저작권을 주장하지 않겠다는 의미(GNU 프로젝트의 기본 이념)
- 특허권
    - 발명에 관하여 발생하는 독점적/배타적 지배권으로 법에 정해진 절차에 의해 출원을 해야하며, 심사를 통해 부여되는 권리임
    - 특허기술을 사용하기 위해서는 반드시 특허권자의 허락을 얻어야 함
    - 특허 받은 방식을 구현하는 SW라면 프로그래밍 언어나 소스 코드와 상관없이 특허권자의 명시적인 허락을 받아야함
- 상표권
    - 상표권자가 지정상품에 관하여 그 등록상표를 사용할 독점적인 권리로서 일정한 절차에 따라 등록하여야 효력이 발생
    - 이러한 상표를 사용하기 위해서는 반드시 상표권자의 허락을 얻어야 하며 허락받지 않고 상표를 사용할 경우 처벌을 받게 됨
    - 상표권을 취득한 SW의 경우 상표를 사용하려면 상표권자의 명시적인 허락을 받아야함
- 엽어비밀
    - 공개되지 않은 SW의 경우 엽업비밀로서 보호를 받을 수 있으며, 공개된 SW라 하더라도 아이디어에 대한 부분은 영업비밀로 보호를 받을 수 있는 가능성 있음
    - 단, 엽업비밀로서의 SW보호는 널리 공개되어 유통되는 경우에는 보호되기 어렵고, 이를 알지 못하고 사용한 제3자에게 법적으로 문제를 삼을 수 없음

라이센스

- 라이센스
    - SW는 지식재산권에 의해 보호받으며 저작권자만이 쓸 수 있지만, 권리자가 다른 사람에게 일정 조건으로 특정 행위를 할 수 있는 권한을 부여할 수 있음
    - 이 권한을 보통 라이센스라고 함
    - ex)Windows
        - windows를 구입하면 SW권리자인 MS로부터 windows을 컴퓨터 한대에 설치하여 이용할 수 있는 라이센스를 받은 것에 불과함
        - windows정품을 구입했다고 하더라도 다른 사람에게 빌려주거나 복제하여 팔 수 없다.

오픈소스 라이센스

- 오픈소스 라이센스
    - 오픈소스는 소스코드가 공개되어 사용, 복제, 배포, 수정을 할 수 있지만 원 저작자의 라이센스 규칙에 따라야 함
    - 오픈소스 라이센스란 오픈소스 개발자와 이용자 간의 사용 방법 및 조건의 범위를 명시한 계약임 오픈소스 저작권자가 소스코드를 공개했을 뿐 지적재산권을 보호받고 있으며, 라이센스 종류에 따라 다양한 의무사항을 포함한다.
    - 오픈소스를 이용하려면 오픈소스 개발자가 만든 조건의 범위에 따라 해당 SW를 사용해야하며, 이를 위반할 경우 라이센스 위반 및 저작권 침해로 법적 책임을 져야함
- 오픈소스 라이센스 사용 분포
    - OSI에 따르면 현재 약 80개의 라이센스가 있지만 실제로 많이 사용되는 라이센스 개수는 한정되어 있다.
    - GPL,LGPL이 약 72%

오픈소스 라이센스 상세

- 공통적 준수 사항(의무사항)
    - 저작권 관련 문구 유지
        - 저작권은 저작물의 창작과 함께 자동적으로 부여됨. SW의 경우 소스코드에 프로그램의 이름과 개발자, 버전, 연락처등을 포함하는 경우가 많고, 이들은 저작인격권으로 보호받음
        - 오픈소스는 거의 대부분 소스코드 상단에 개발자 정보와 연락처등이 기록되어 있으며 개발자 정보를 임의로 수정하거나 삭제해서는 안됨
    - 제품명 중복 방지
        - SW의 제품명은 상표권으로 보호받음 오픈소스의 경우에 이와 동일한 이름을 제품명이나 서비스명으로 사용하면 상표권 침해에 문제가 생기게 됨. 특히 유명한 오픈소스일수록 해당 오픈소스의 이름이 상표로 등록되어 있는 경우가 많기 때문에 더욱 조심해야 함
    - 서로 다른 라이센스의 SW 조함 시 조합 가능 여부 확인
        - SW를 작성하고자 할 경우 기존에 만들어진 코드를 재사용하거나 결합하는 경우 많음, 이 때 결합되는 각 코드의 라이센스가 서로 상충되는 경우가 있다.
        - 서로 다른 라이센스로 배포된 오픈소스를 결합하는 경우 반드시 2개의 라이센스가 서로 호환되는지 확인해야함
- 선택적 준수 사항(라이센스에 따라 다름)
    - 이용 여부 명
        - 많은 오픈소스 라이센스들은 SW를 이용할 때 해당 오프소스가 이용되었음을 명시핟록 함, 즉 "이 SW는 오픈소스인 무엇 무엇을 이용하였습니다" 라는 식으로 이용 여부를 명확히 기

술하라고 함
    - 소스코드 공개
        - 오픈소스는 라이센스에 따라 수정하거나 추가한 부분이 있을 때 해당 부분이 소스코드로 공개하여야 한다고 명시하는 경우가 있음(GPL)
        - 정확한 공개 범위는 각 라이센스에서 정하고 있는 범위와 SW를 개발하는 방법에 따라 달라질 수 있음
    - 특허
        - 어떤 기술이 특허로 보호될 경우 해당 기술을 구현할 때 반드시 특허권자의 허락을 받아야 한다.
        - 특히 최근SW특허가 급격히 증가하면서 새롭게 만들어지는 오픈소스 라이센스들에서는 특허 관련 조항을 포함하는 경우가 많음

대표적인 라이센스 (GPL 2.0)

- GPL 2.0에서 GPL은 GNU general public license의 약자
- 자유 소프트웨어 제단(FSF)에서 만든 라이센스로서 가장 강력한 라이센스임
- 리처드 스톨만이 GNU 프로젝트로 배포된 프로그램의 라이센스로 사용하기 위해 작성함
- 소스 코드 형태로 재배포하는 경우
    - GPL 2.0 오픈소스를 소스 형태로 재배포시 다음 사항을 준수해야함
    - 먼저 고지의 의무가 있음. 저작권 고지 제공, 보증 부인 제공, 라이센스 사본 제공
    - 즉, 소스 코드 내 명시된 저작권/라이센스 정보를 그대로 유지한 상태로 재배포함
- 소스 코드 일부를 추가/수정하고 재배포하는 경우
    - 추가/수정한 부분에 GPL 2.0을 적용함
    - 수정 사항에 대한 고지를 포함함
- 바이너리 형태로 재배포하는 경우
    - 고지의 의무
        - 먼저 고지의 의무가 있음. 저작권 고지 제공, 보증 부인 제공, 라이센스 사본 제공
    - 소스 코드 제공
        - 바이너리에 해당하는 소스 코드를 제공해야하며 다음을 준수해야함
        - GPL2.0은 파생저작물에 대해서는 GPL2.0을 적용, 소스코드를 공개할 것을 요구
        - GPL2.0파생 저작물 범위
            - 수정 코드
            - GPL 프로그램과 동일한 프로세스에서 동작하는 Module
            - Library
            - Class
        - GPL2.0 파생 저작물 범위에 속하지 않는 것
            - CD와 같은 매체에 함께 존재하지만 GPL프로그램과 전혀 연동되지 않는 독립적인 프로그램
            - GPL 프로그램과는 별도의 프로그램으로써 pipe, Socket, IPC, Command Line Argument로 GPL 프로그램과 통신하는 경우
        - 빌드 환경 제공
            - 바이너리 사용자가 공개된 소스코드로 동일한 바이너리를 만들 수 있는 빌드 환경을 제공함
            - tool chain정보, 빌드 스크립트, 빌드 방법
        - 서면 약정서(소스 코드 대신)
            - 소스 코드 대신 서면 약정서를 제공할 수 있으며 여기에는 아래 진술이 포함
                - 서면 약정서는 3년간 유효

            - 누구에게나 제공
            - 비용을 청구하지 않음
        - 이후 외부로부터 서면 약정서를 근거로 소스 코드 제공을 받을 경우, 바이너리에 해당하는 소스 코드를 제공해야함
        - 따라서 회사는 제품 판매 후 최소 3년간 소스 코드를 보관해야 함
    - 라이센스 호환성
        - 서로 요구하는 의무사항이 상충되는 오픈소스 라이센스는 하나의 프로그램에 동시에 존재해서는 안 됨
        - 충돌하는 라이센스 Apache-1.1, pyton-2.0, FTL, OpenSSL등

대표적인 라이센스 (GPL 3.0)

- FSF에서 2007년 공개한 라이센스로 GPL 2.0과 유사한 의무사항을 갖기만, 추가로 User product배포 시 설치 정보 제공을 요구함
- 바이너리를 User Product와 배포한다면 설치 정보를 제공해야 함
    - user product: 전자 기기 같은
    - 설치 정보

대부분의 user product는 보안상의 이유로 설치 정보를 제공하는 것이 불가능함. 따라서 User product로 배포하는 SW에는 GPL 3.0의 오픈소스를 사용하지 않아야 함

대표적인 라이센스 (AGPL)

- Affero GPL의 줄임말
- 어떤 개발자가 프로그램의 소스코드를 수정해서 사용하고 프로그램을 배포하지 않았을 때 발생할 수 있는 소스 코드의 공유 불가 현상을 해결하기 위해 만들어짐
- 서버에서 프로그램을 실행해서 다른 사용자들과 통신하게 되면 실행되고 있는 프로그램의 소스코드를 사용자들이 다운로드할 수 있도록 해야하는 조항을 포함함
- AGPL 라이센스 적용 예: 몽고 PDF등

대표적인 라이센스 (LGPL)

- Lesser GPL의 줄임말
- GPL이 붙은 라이센스를 사용할 때 반드시 소스코드를 다시 GPL로 공개해야 하는 부담 때문에 실무에서 사용이 어려운 점을 보안하기 위해 만들어진 라이센스
- LGPL을 따르는 프로그램은 전체 소스코드를 공개하지 않고 사용된 오픈소스 라이브러리에 대한 소스코드만 공개하면 됨
- 초기에는 한정된 라이브러리에만 적용된다는 것을 나타내기 우해 Library GPL이라고 불렸으나, 변경됨
- 적용예시 Firefox

대표적인 라이센스 (MIT)

- MIT에서 학생들을 지원하기 위해 만든 라이센스
- 라이센스와 저작권 관련 명시만 지켜주면 되는 등 가장 느슨한 조건을 가지고 있음 많은 사람드이 사용하기 용이하다는 특징이 있음

라이센스 양립성 문제

- 둘 이상의 오프소스 소스코드를 사용하여 새로운 프로그램을 개발할 경우 각 오픈소스의 라이센스의 요구사항이 상충되는 문제
- 새 프로그램에서 각 라이센스 요구사항을 모두 중촉시키지 못하는 경우 이런 문제 발생함
- 둘 이상의 오픈소스를 결합할 때에는 각 라이센스가 서로 양립 가능한지 미리 조사해야함
- 예시
    - 각각 GPL, MPL로 배포되는 오픈소스의 소스코드를 함께 결합하여 사용하는 경우 고려
    - MPL은 MPL로 배포된 오픈소스에 해당하는 소스코드로부터 파생된 부분을 동일한 MPL로 배포하기를 요구하고, GPL은 GPL로 배포된 오픈소스에 해당하는 소스코드로부터 파생된 부분 뿐만 아니라, 그와 링크된 전체의 소스코드를 GPL로 배포하기를 요구하기 때문에 이 두 라이센스 요구를 동시에 충족시키는 것은 불가능함
- 특히 GPL과 관련하여 양립성이 많이 발생. GPL은 사용자들에게 GPL에서 규정하는 것 이외의 제한사항을 추가하지 못하도록 엄격하게 통제하고 있기 때문

오픈소스 검정 도구

- 개념
    - 오픈소스 검증 도구는 어떤 프로그램을 개발했을 때 그 프로그램에서 어떤 오픈소스를 썼는지, 어떤 라이센스 및 저작권이 사용되었는지를 파악해주는 도구
- Cdoe eye, Fossology, oss

GIT

버전관리

- 버전관리
    - 원하는 시점으로 이동

버전관리 시스템 필요성

- 혼자서는 버전관리 시스템 없이도 버전관리 어렵지 않음
- 가령 보고서를 작성할 때 중간중간 버전을 저장해주면 필요할 때 원하는 버전으로 돌아갈 수 있다.
- 하지만 팀 프로젝트의 경우 버전관리가 필요

Git

- 소스코드 버전 관리 시스템
- 자신이 원하는 곳에 깃발을 꽂고, 필요시 꽂힌 시점으로 이동 가능
- 데이터 공간만 있으면 어디서나 git을 사용가능

.git

- .git 폴더는 git으로 생성한 버전들의 정보와 원격저장소 주소등이 들어있는 숨겨진 폴더
- 이 .git 폴더를 로컬저장소라고 부르며 이 폴더에다 버전 관리를 할 수 있음

커밋 개념

- 생성하는 각 버전 혹은 각 버전을 통해 생성된 파일을 커밋이라고 함
- 저장과 같은 개념
- 커밋을 행할 때 마다 새로운 버전을 하나씩 만드는 것
- 커밋에 상세한 설명을 적음

Checkout

- Checkout은 다른 커밋으로 시간여행

원격저장소

로컬만 있어도 버전관리는 가능하지만 다른 개발자와 협업을 위해 원격저장소(github를 사용해야함)

SVN과 GIT의 차이점

SVN은 차이점만을 저장하는 반면 git은 전체를 저장함

바로바로 비교연산만 하기 때문에 빨리 동작함

스테이지 개념

스테이지는 마치 무대에 사진을 찍기 위해 사람을 올리는 것과 같음

커밋에 추가할 파일을 무대로 올린다고 함

Git으로 관리하는 파일의 상태

- Untracked
    - git이 관리하지 않는 파일
    1. 즉, 추적되지 않는 파일
- Tracked 2. 수정 없음 3. 수정됨 4. 스테이지 됨

브랜치 정체

단순 포인터

PR : 브랜치 vs fork

---	의의	편리한 점	불편한 점
브랜치	브랜치는 하나의 원격 저장소에서 분기를 나눈다.	브랜치는 하나의 원본저장소에서 코드 커밋이력을 편하게 볼 수 있다.	다수의 사용자가 다수의 브랜치를 만들면 관리가 어렵다.
포크	포크는 여러 원격저장소로 분기를 나눈다.	원본저장소에 영향을 미치지 않으므로 원격저장소에서 마음껏 코드를 수정할 수 있다.	원본저장소의 이력을 보려면 따로 주소를 추가해야 한다.