오픈소스sw의이해

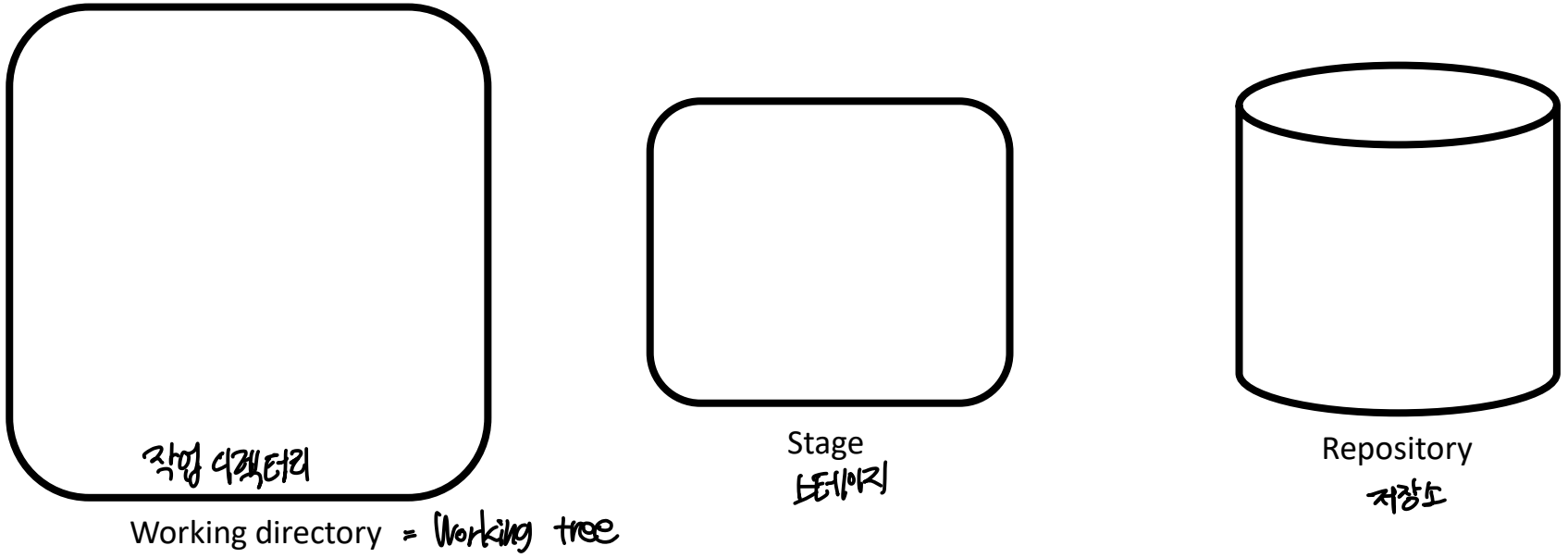
Lecture #3: Version management I

Software Engineering laboratory
Yeungnam University

영남대학교 Yeungnam University

git을 사용한 버전 관리

- 버전 관리를 위한 기초 개념
  - git이 관리하는 세 개의 공간
    - 작업 디렉터리(working directory) or 작업 트리 (working tree),
      - .git 숨김 폴더가 놓여 있는 곳이 버전 관리 대상이 위치하는 공간
    - 스테이지(stage),
    - 저장소(repository)



git을 사용한 버전 관리

- 초기 working directory의 예제
  - 프로젝트 진행을 위해 새로운 파일 또는 폴더 생성
  - 이후 기존 파일 또는 폴더를 수정하거나 삭제 진행
    - 즉, working directory에 변경 발생

Working directory

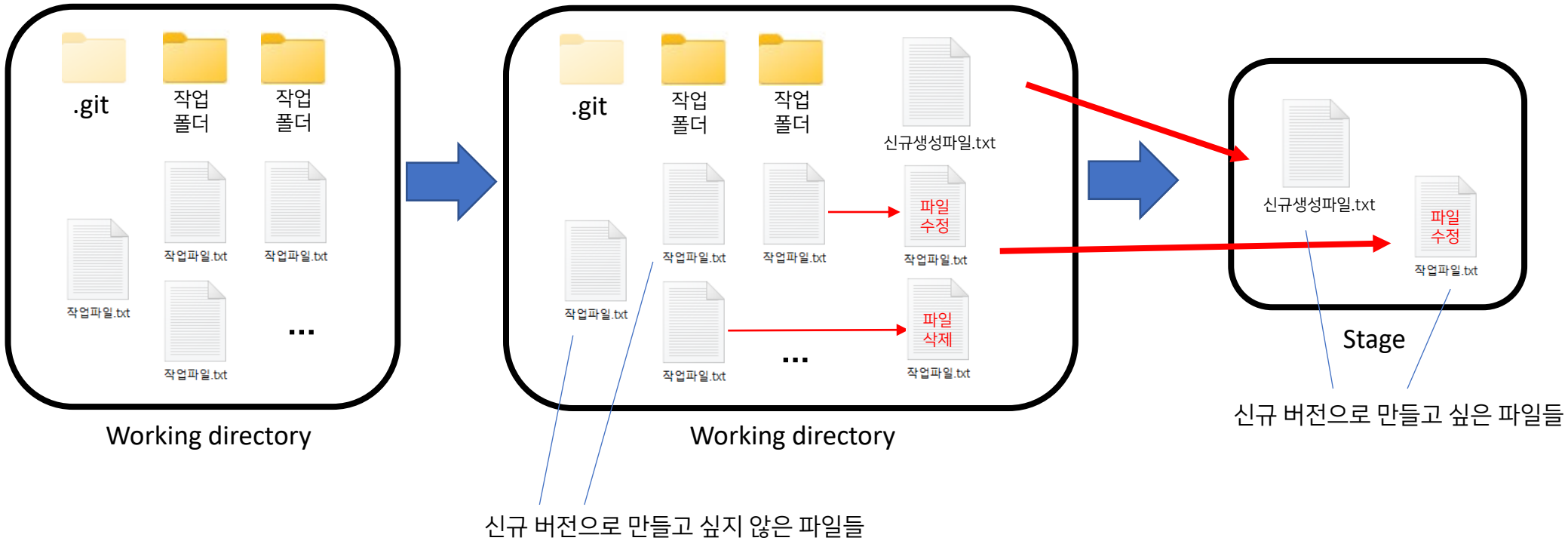
Working directory

git을 사용한 버전 관리

- Stage (staging area, index)
  - Git으로 버전을 만들 때는 working directory 내에서 변경된 파일들 중에서 새로운 버전으로 만들려는 파일들을 선별해 특별한 공간으로 옮기는 작업을 거치게 되는데 이 특별한 공간이 바로 stage.
    - 작업 디렉터리에 파일이 1,000개 있고 이 중 100개가 변경됐을 때, 100개 중 새로운 버전이 될 파일을 선별하는 작업.
    - 스테이지는 변경 사항이 있는 파일 중 다음 버전이 될 후보가 올라가는 공간인 셈.
  - Workging directory는 프로젝트가 위치한 공간이라 눈으로 직접 볼 수 있는 반면, stage는 명시적으로 보이지 않음.
  - '버전을 만든다'는 말은 '특정 순간의 변경 사항을 기억한다'는 말과 같음.
    - 작업 디렉터리에 있는 프로젝트에 변경 사항이 생기는 순간 새로운 버전을 만들 수 있게 됨.

git을 사용한 버전 관리

- Stage (staging area, index)

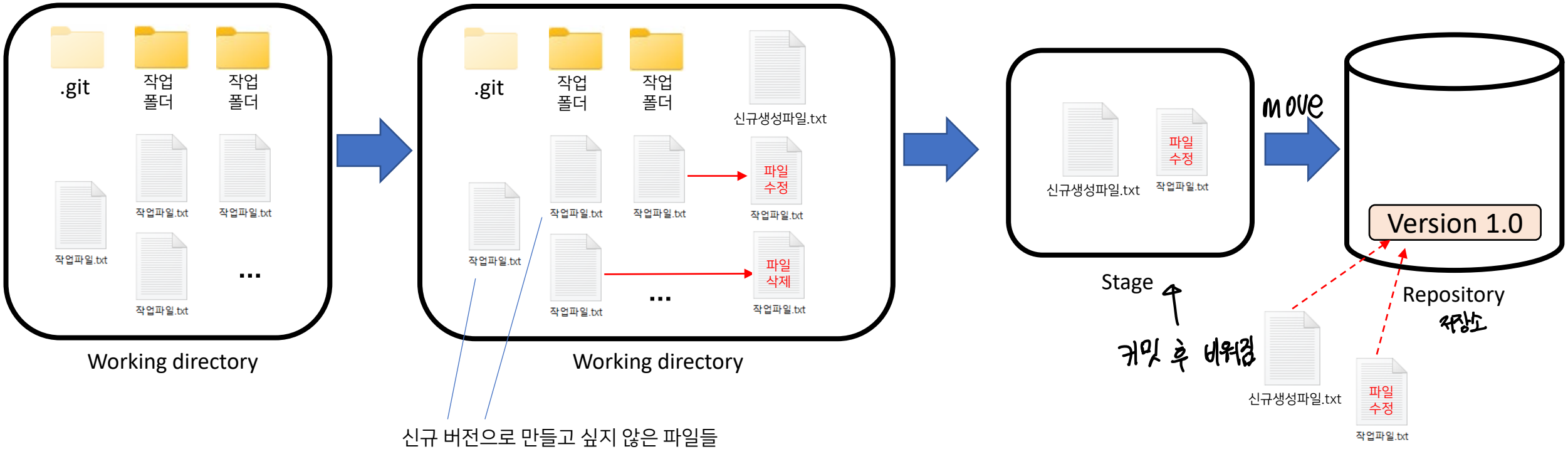


git을 사용한 버전 관리

- 저장소
  - Staging된 파일들을 바탕으로 만들어진 새로운 버전이 저장되는 공간.
    - 작업 디렉터리에서부터 만들어진 모든 버전들의 내역이 저장소에 있음.
  - 저장소는 버전이 만들어지고 관리되는 공간.
  - 저장소는 스테이지와 마찬가지로 사용자에게는 명시적으로 보이지 않음.
  - Stage에 올라온 파일을 토대로 새로운 버전을 만들면 새로운 버전이 될 후보가 더 존재하지 않으니 stage는 깨끗하게 비워짐.

git을 사용한 버전 관리

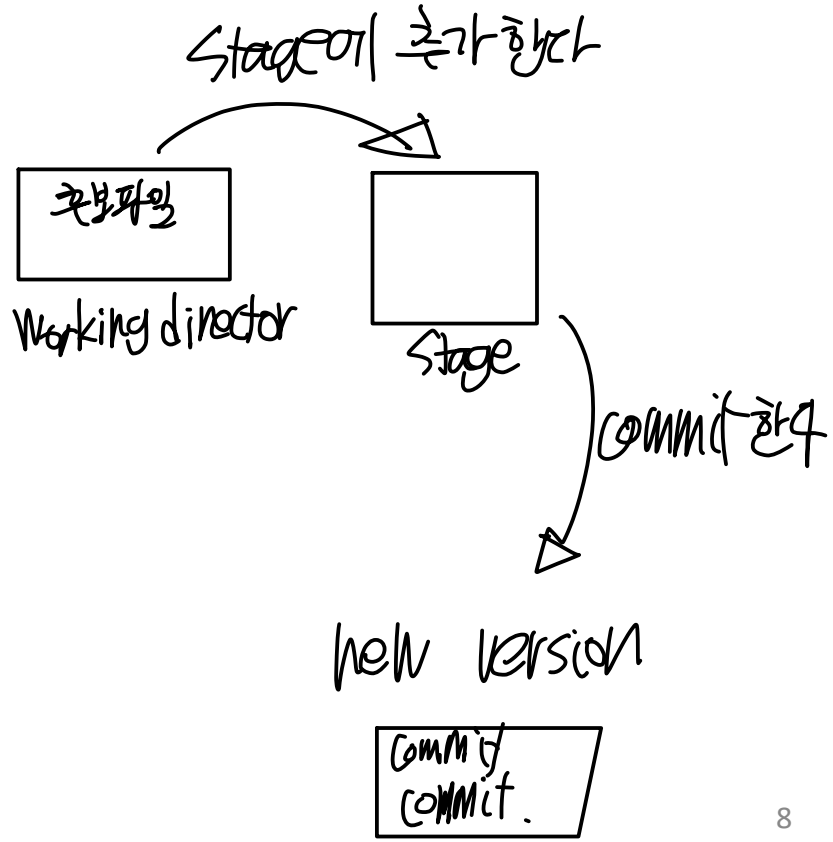
- 저장소



git을 사용한 버전 관리

- 새로운 버전을 만들기 위한 git에서 사용하는 명령어 절차
  - Working director에서의 후보 파일을 stage로 옮기는 것: stage에 추가한다.
  - Staged된 파일들을 바탕으로 저장소에 새로운 버전을 만드는 것: commit 한다.
    - 저장소에 저장된 각각의 버전들을 commit이라 지칭함.
    - 즉, git으로 만든 버전을 편의상 commit으로 지칭함.

Working directory의 파일은
1 | 변경 사항 생성
2 | add
3 | commit
…의 과정을 통해
1 | 작업 디렉터리
2 | 스테이지
3 | 저장소
순으로 이동하며 새로운 version으로 생성됨.



Sourcetree를 활용한 첫 버전 만들기

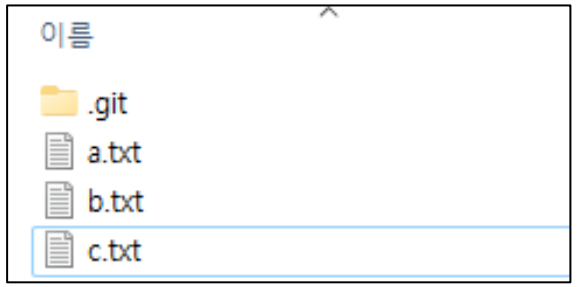
- (1) Local repository 만들기
  - Sourcetree 실행 후 Create 클릭
  - 목적지 경로를 설정
    - 예) c:\git-test
  - 하단에 생성 버튼 클릭
  - 만들어진 Local repository 확인해보기
    - 우측 상단 "탐색기" 클릭.
    - .git 숨김폴더가 보인다면 성공적으로 local repository 생성 완료.
  - 생성한 목적지 경로가 working directory.
    - git으로 버전 관리할 대상을 여기에 위치시키면 됨.

Sourcetree를 활용한 첫 버전 만들기

(2) 버전 관리할 파일 만들기

- Working directory에 다음 처럼 예제 파일 생성해보기
  - a.txt, b.txt, c.txt
  - 각각의 파일 안에 text file a, text file b, text file c를 적은 뒤 저장.

: 지금은 간단한 텍스트 파일로 실습하지만, 이 텍스트 파일들이 여러분의 프로젝트 소스 코드 파일일 수 있다는 점을 염두에 두길 바람.



* 줄 바꿈을 하고 저장 권장함.

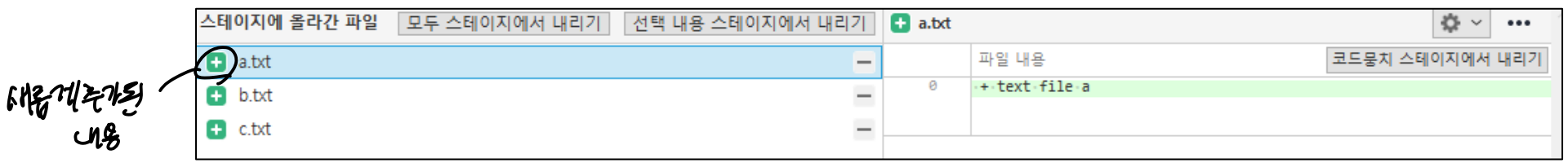
- 각 텍스트 파일에 내용을 적은 후 바로 저장하지 말고, 다음 그림처럼 줄 바꿈을 한 뒤 저장

text file a
| ----→ 커서

- 줄바꿈을 하지 않고 저장하면 스테이지에 올릴때 경고메시지 발생 가능.

Sourcetree를 활용한 첫 버전 만들기

- (3) Sourcetree에서 생성된 파일 확인
  - 텍스트 파일을 작성한 뒤 저장하면 Sourcetree의 파일 상태에서 “스테이지에 올라가지 않은 파일” 항목에 방금 생성한 파일들이 추가됨.
  - 스테이지에 올라가지 않은 파일은 말 그대로 ‘작업 디렉터리 내 변경 사항이지만, 아직 버전이 될 후보로 선정되지는 않은 파일’을 말함.
- (4) Stage에 올리기
  - “모두 스테이지에 올리기” or 각 파일 옆에 + 클릭.
  - 각 파일 클릭해보면 파일 변경 사항 확인 가능.
    - 초록색 표시와 +표시는 각 파일에서 새롭게 추가된 내용을 의미함.

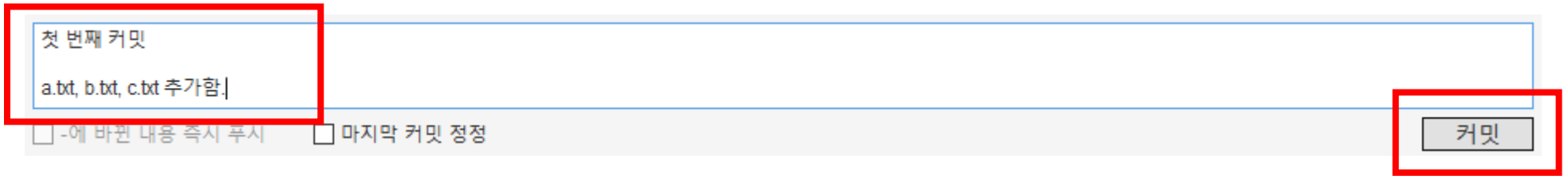


Sourcetree를 활용한 첫 버전 만들기

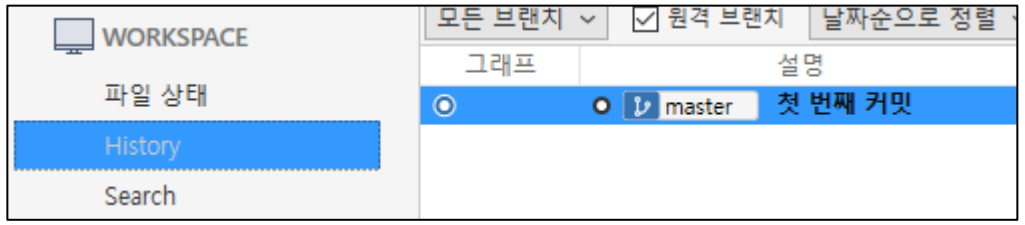
- (5) commit 해보기
  - commit: 스테이지에 올라온 파일들을 새로운 버전으로 만드는 것.
  - 일반적으로 commit 하기 전에 commit message 작성.
    - Commit message: 버전을 설명하는 메시지
      - '내가 지금 어떤 파일을 어떻게 변경했는지, 왜 이렇게 변경했는지' 등의 내용을 담은 일종의 쪽지.
      - 커밋 메시지는 크게 제목과 본문으로 이루어져 있으며 본문은 생략 가능.
    - git을 통해 새로운 버전을 만들 때 반드시 commit message를 적기를 권장
      - 커밋 메시지를 적지 않는다면 프로젝트의 규모가 커지고 수많은 커밋이 쌓였을 때 다른 누군가가(또는 커밋했던 본인조차) 누가, 언제, 어떤 변경 사항을 만들었는지, 이 변경 사항이 의미하는 것은 무엇인지 파악하기 어렵기 때문.
      - 다른 개발자들이 잘 이해할 수 있도록 구체적으로 작성 필수 (에러 메시지나 링크 첨부 가능).

Sourcetree를 활용한 첫 버전 만들기

- (5) commit 해보기
  - commit message 작성 후 커밋 버튼 클릭.

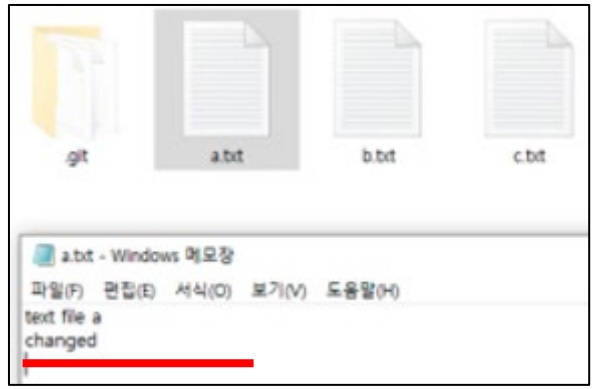


  - 커밋 이후…
    - 좌측 메뉴의 “파일 상태” 에는 커밋할 내용이 없다고 표시.
    - 좌측 메뉴의 “History” 클릭 후, 하단부에서는 커밋들의 커밋 메시지, 만들어진 시간, 작성자 등을 확인 가능.

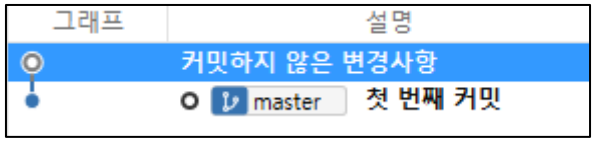


버전 쌓아 올리기

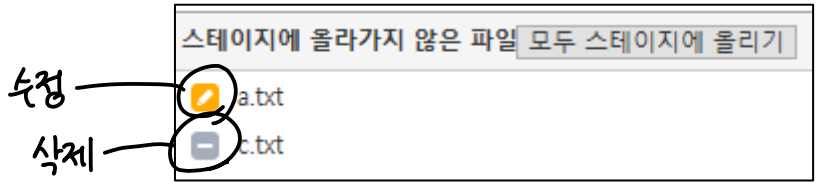
- 방금 만든 첫 번째 버전을 수정한, 또 다른 버전 생성해보기
  - 파일 수정 하기
    - “탐색기”를 열어서 a.txt 파일 안에 새로운 줄로 changed를 추가한 뒤 저장.
    - 또한, c.txt 파일은 삭제하기.



  - History 확인
    - ‘커밋하지 않은 변경 사항’이 추가된 것을 확인할 수 있음

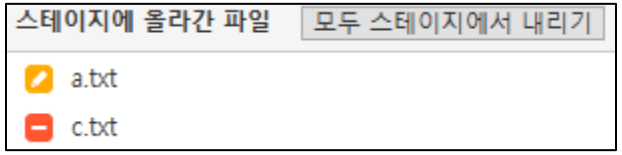


  - 파일 상태 확인
    - “스테이지에 올라가지 않은 파일” 항목에서 a.txt 파일과 c.txt 파일 확인 가능.

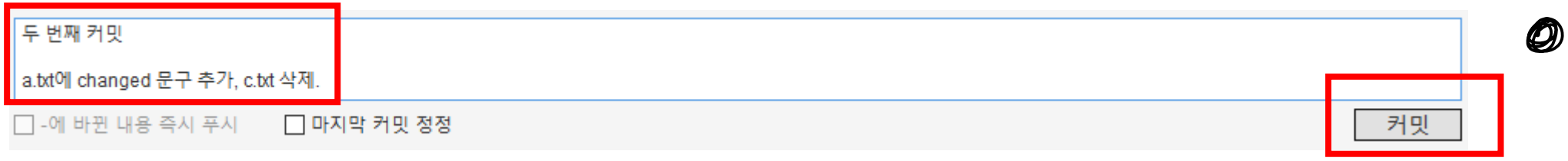


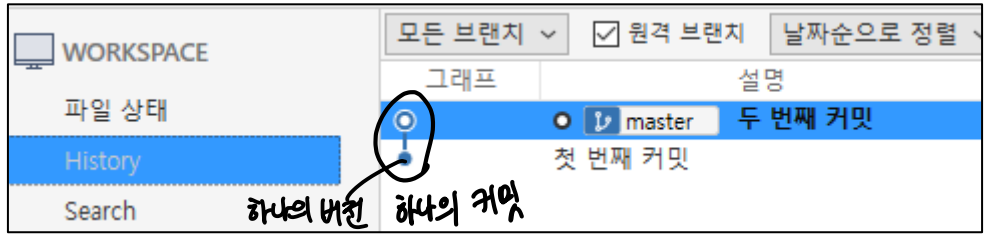
버전 쌓아 올리기

- 방금 만든 첫 번째 버전을 수정한, 또 다른 버전 생성해보기
  - 수정된 파일 스테이지에 올린 후



  - commit message 작성
  - 커밋 하기

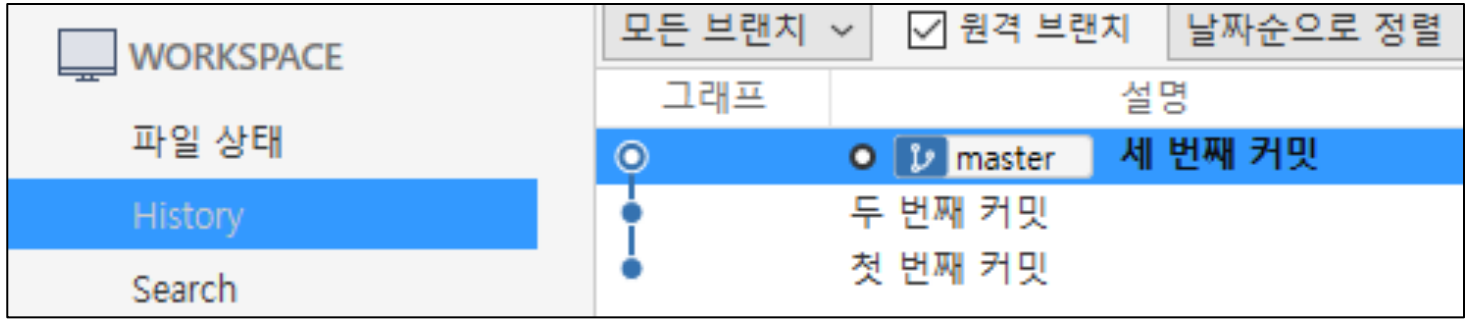




- 그래프 부분 확인해보기
  - 동그라미 두 개가 연결되어 있는 것을 볼 수 있음.
  - 여기서 동그라미 하나는 커밋 하나, 즉 하나의 버전을 나타냄.
  - '두 번째 커밋'의 동그라미가 '첫 번째 커밋'의 동그라미와 연결되어 있는 것은 두 번째 커밋이 첫 번째 커밋에서부터 만들어진 버전임을 나타냄.

버전 쌓아 올리기: 실습

- 새로운 버전 만들어보기
  - text file d라는 내용을 담은 텍스트 파일 d.txt를 만들기
  - 이를 스테이지에 올린 뒤 커밋
    - 커밋 메시지는 다음과 같이 작성
      - 제목: 세 번째 커밋
      - 본문: d.txt 파일 추가



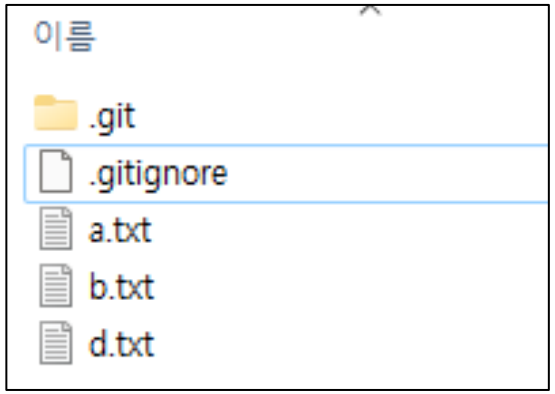
.gitignore로 무시하기

- .gitignore 활용해보기
  - Working directory 안에서 새로운 추가/수정/삭제가 이루어질 때마다 git이 언제나 해당 변경 사항을 알아차린다는 사실을 알았을 것.
  - git으로 변경사항을 추적하고 싶지 않은 파일이나 폴더를 무시할 수 있는 기능
    - 종종 버전 관리 대상에서 제외하고 싶은 파일이나 폴더, 즉 변경 사항이 생기더라도 앞으로도 쭉 버전에 포함하고 싶지 않은 파일이나 폴더가 있을 수 있음.
  - .gitignore 파일은 쉽게 말해 '무시할 파일/폴더 목록'을 적은 파일.
    - git은 .gitignore 파일에 적은 파일이나 폴더에 변경 사항이 생겨도 이를 무시함

.gitignore로 무시하기

- (1) 작업 디렉터리에 .gitignore 파일을 만들기
  - .gitignore 파일은 일반 텍스트 파일처럼 생성해도 좋음
  - git은 정확히 .gitignore라는 파일명을 인식하기 때문에.txt와 같은 확장자는 지우기

- 주의
: 확장자가 안 보일 때 확장자를 지우기 위해서는 윈도우 탐색기에서 파일 확장명 보기에 체크해두어야 함.



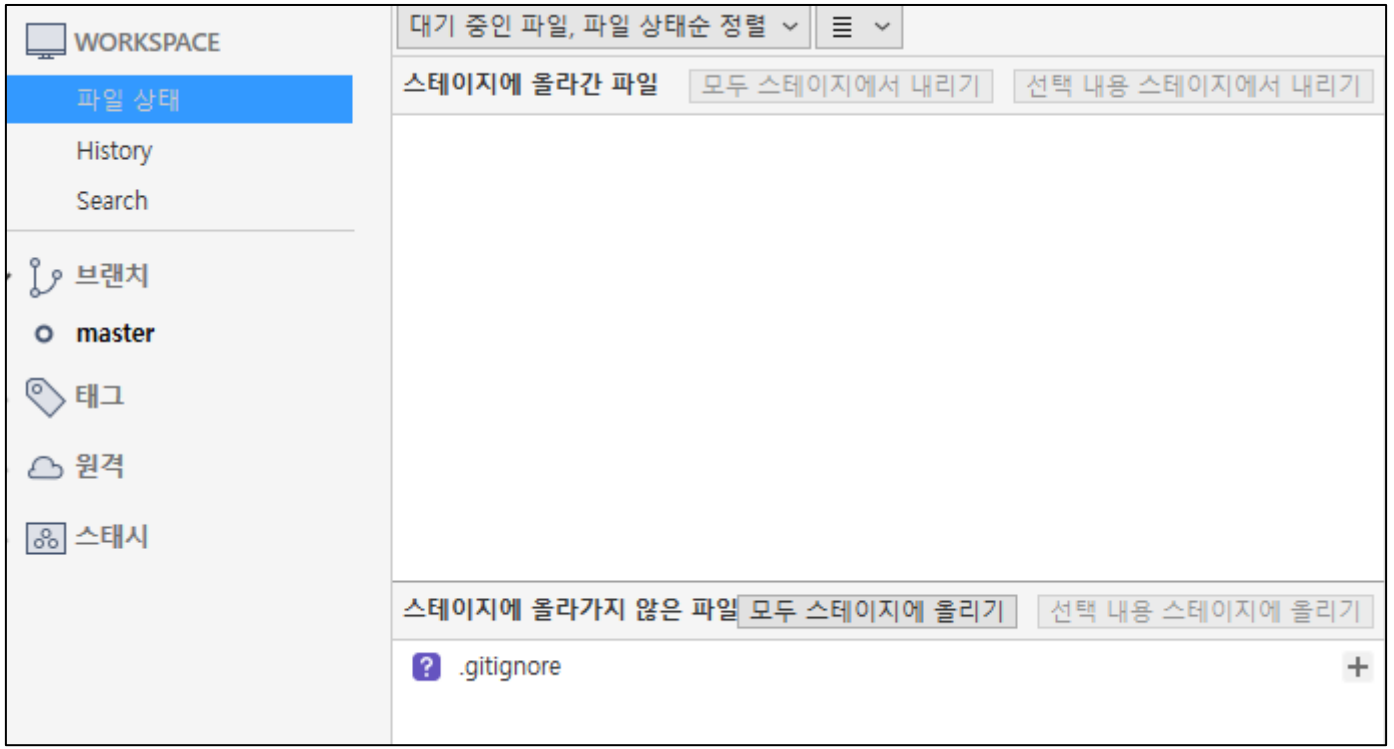
.gitignore로 무시하기

- (2) 생성된 .gitignore 안에 e.txt를 적은 뒤 저장해보기
  - git이 작업 디렉터리 내 앞으로 e.txt 파일을 무시하겠다는 의미

- (3) e.txt 생성하기
  - 내용 예: text file e

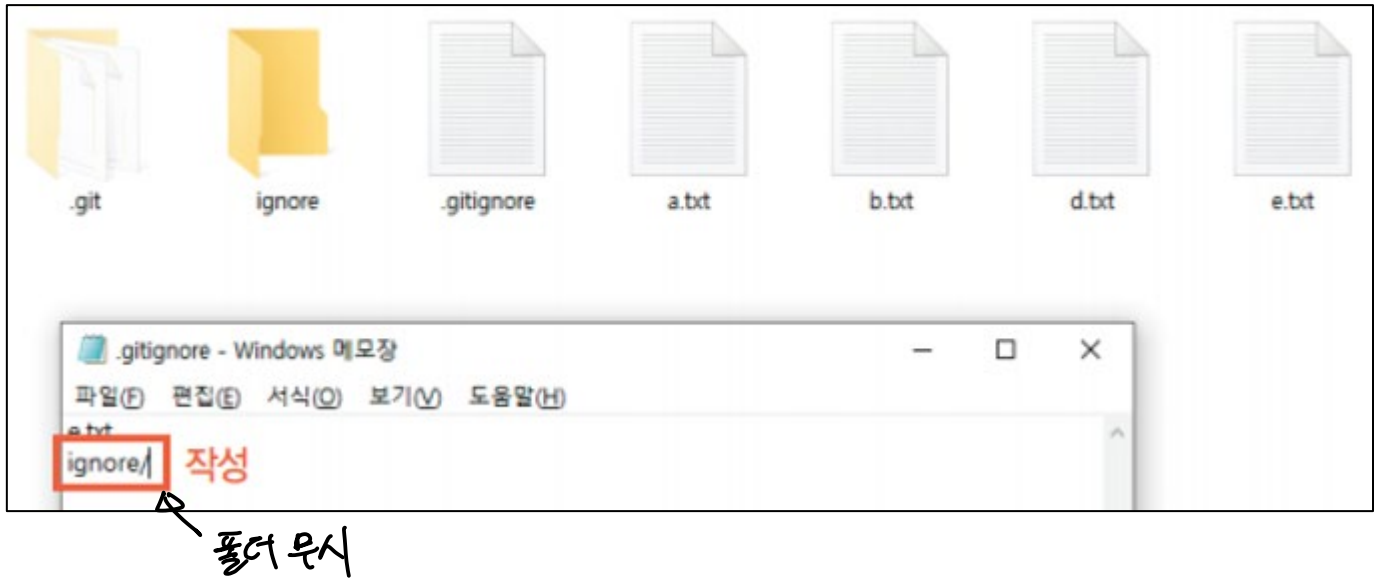
.gitignore로 무시하기

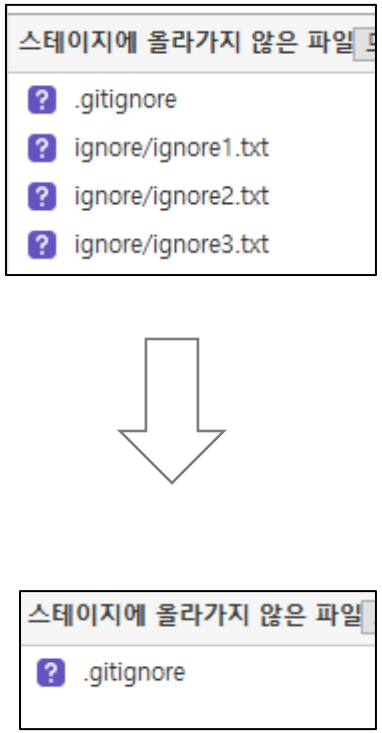
- (4) git이 e.txt 무시 확인



.gitignore로 무시하기

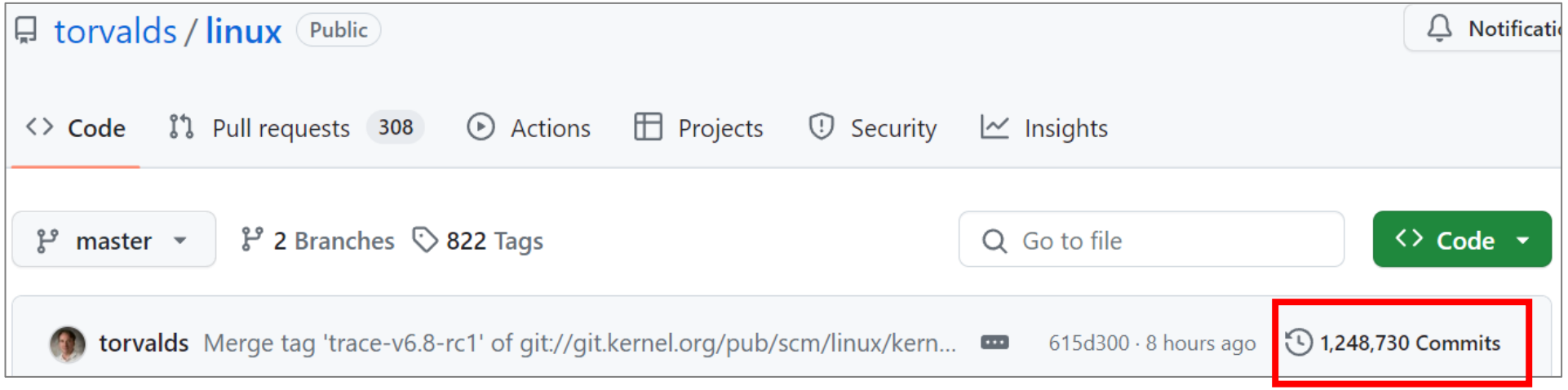
- (5) 파일 외에 폴더에도 적용해보기
  - ignore 폴더 생성하기
  - ignore 폴더 안에 임의의 파일들 생성해보기
    - (예) ignore1.txt, ignore2.txt, ignore3.txt
  - .gitignore파일에 e.txt 다음 줄에 ignore/라고 작성





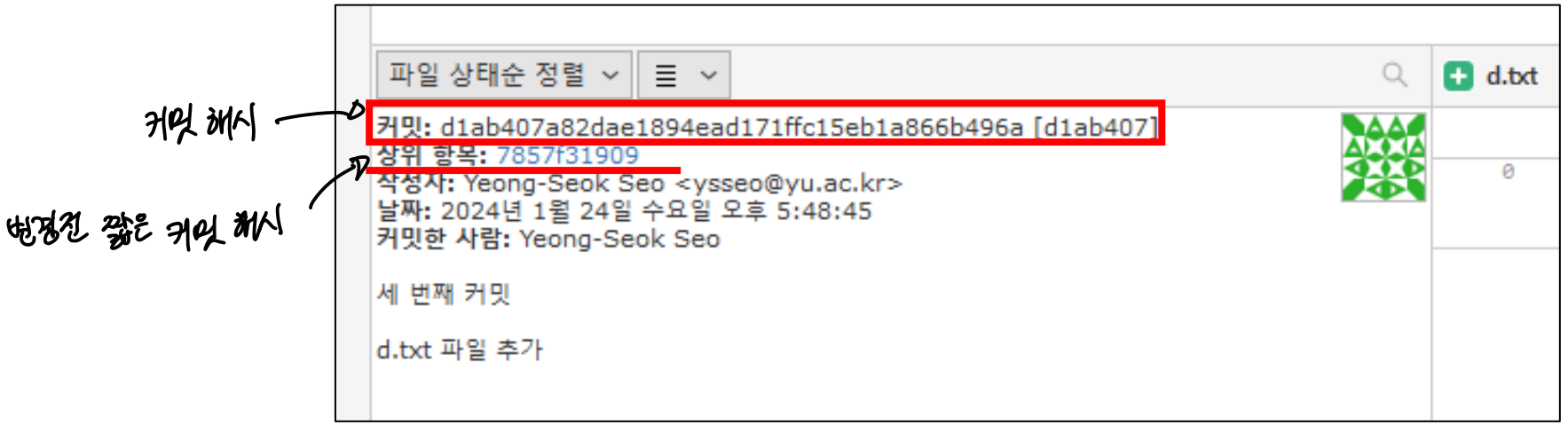
commit에 대한 분석

- commit 살펴보기
  - 현업에서 개발되는 대규모의 SW의 경우 다수의 commit들이 모여서 만들어짐.
    - Linux의 경우, https://github.com/torvalds/linux



commit에 대한 분석

- commit 구분
  - 각 커밋에는 고유한 커밋 해시가 있음
  - 커밋 해시란 마치 학번, 사번과 같이 각 커밋이 가진 고유한 ID
    - 특정 커밋(특정 변경사항)을 지칭할 때 사용



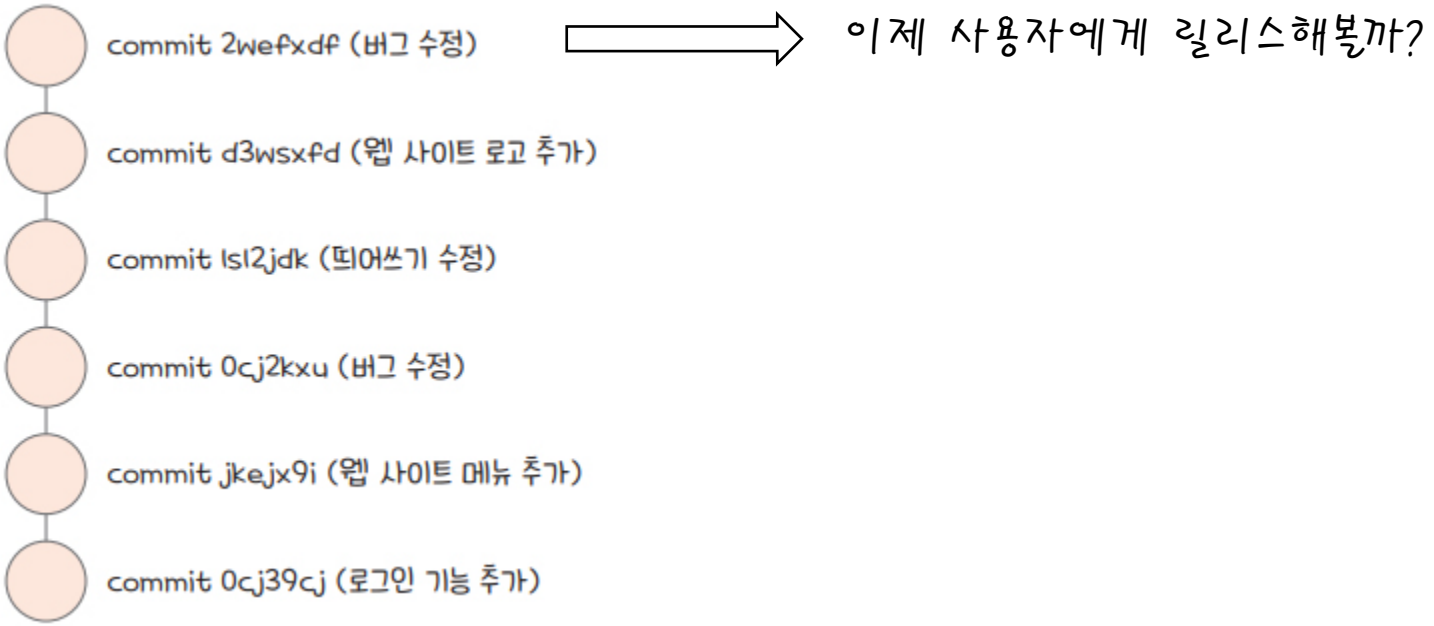
  - "상위 항목"에서는 특정 커밋을 지칭하기 위해 짧은 커밋 해시를 사용
    - 상위항목: 어떤 커밋을 변경해서 만들어진 커밋인지 나타내는 항목

commit에 대한 분석

- commit과 release
  - SW를 개발해나가는 과정에서 다양한 commit이 쌓이게 됨.
    - 로그인 기능, 글쓰기 기능과 같이 새로운 기능을 추가하는 커밋,
    - 버그를 수정하는 커밋,
    - 아주 사소한 변경 사항만 담은 커밋 등

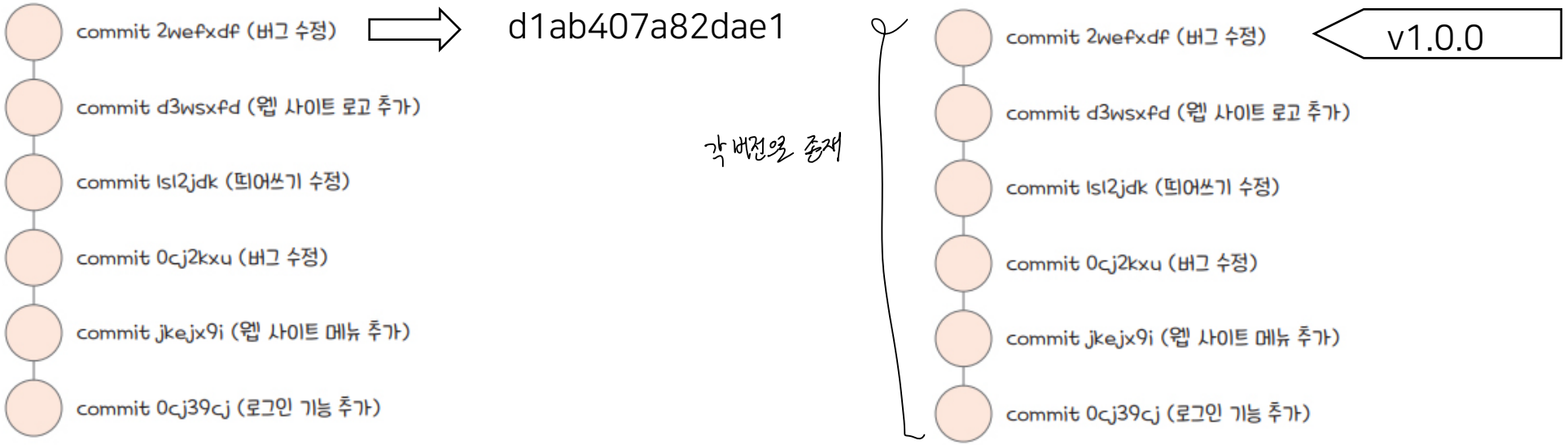
commit에 대한 분석

- commit과 release (cont'd)
  - Commit이 점점 쌓이며 SW 개발이 완성되어 감.
  - SW가 충분히 개발됐다고 판단되면 마침내 사용자에게 결과물을 선보일 것인데, 개발한 SW를 사용자에게 선보이는 것을 release라고 함.
    - 즉, 커밋이 쌓이면 언젠간 사용자에게 릴리스하게 됨



commit에 대한 분석

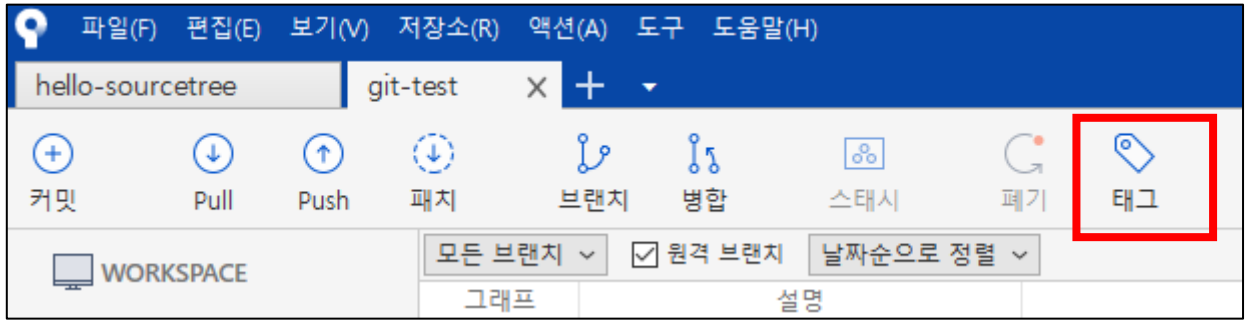
- commit과 release (cont'd)
  - 사용자에게 release할 버전 설정은 어떻게 할까?
    - 커밋 해시 이용? >> 가독성이 떨어져서 추후 수많은 커밋 중 유의미한 버전이 무엇인지 찾기 어려움'
    - 일반적으로 가독성이 높은 태그(tag) 이용
      - 특정 커밋에 붙일 수 있는 꼬리표
      - 분기점이 되는 특정 커밋에 태그 사용: 주로 버전을 명시.



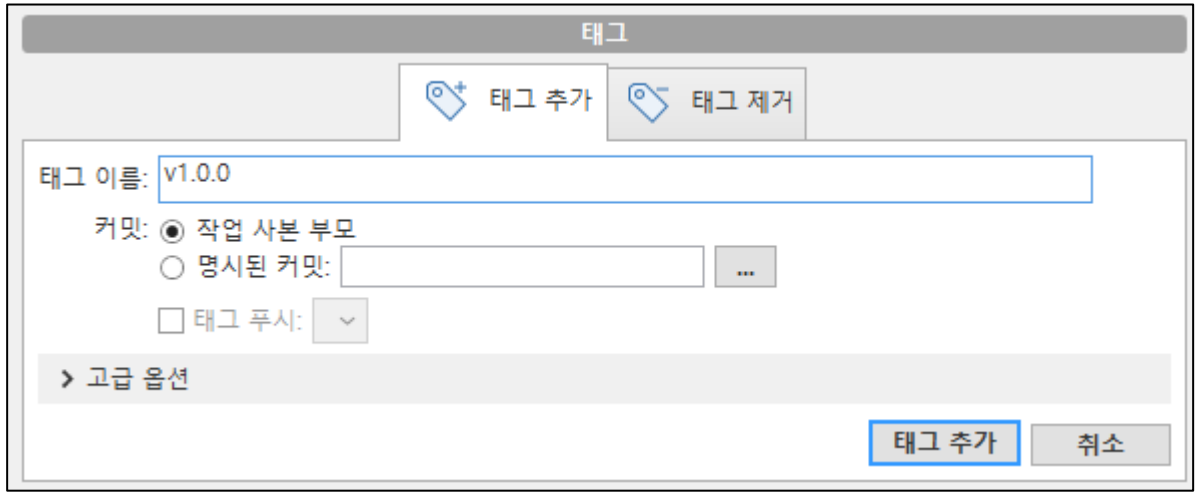
commit에 대한 분석

commit과 release (cont'd)

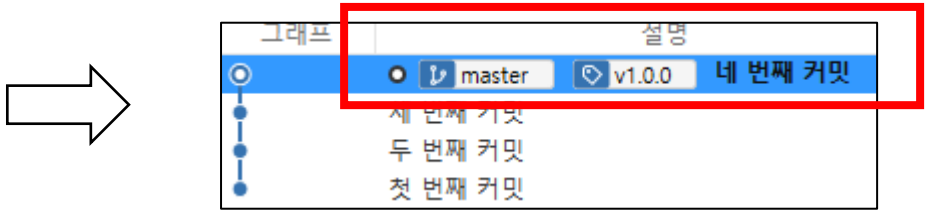
- Sourcetree에서는 "태그" 버튼 클릭



- 태그 추가 or 태그 제거 가능



- 작업 사본 부모 : 최근에 만든 커밋에 태그를 붙이고 싶을 때
- 명시된 커밋: 특정 커밋에 태그를 붙이고 싶을 때



commit에 대한 분석

- 버전 표기법에 대한 이해
  - 사용자에게 선보이는 버전 표기 방식은 소프트웨어마다 다름.
    - <연도.월.일> 형태
    - <숫자.숫자.숫자> 형태
    - <숫자.숫자> 형태 등등
  - 개발자가 정하기 나름.
  - 어떤 소프트웨어는 이 버전을

Looking for latest release of a version branch?

Node.js Version	Codename	Release Date	npm
v21.6.1	-	2024-01-22	v10.2.4
v20.11.0	Iron	2024-01-09	v10.2.4
v19.9.0	-	2023-04-10	v9.6.3
v18.19.0	Hydrogen	2023-11-29	v10.2.3
v17.9.1	-	2022-06-01	v8.11.0
v16.20.2	Gallium	2023-08-08	v8.19.4

commit에 대한 분석

- 버전 표기법에 대한 이해
  - 예) 세자리 표기법 vX. Y.Z
    - X: 가장 앞에 나오는 숫자는 주(Major) 버전이라고 부름
      - 이는 가장 중요한 버전으로, 일반적으로 새롭게 내놓은 버전이 기존에 내놓은 버전과 호환되지 않을 정도로 큰 변화가 있을 때 증가
    - Y: 두 번째 숫자는 부(Minor) 버전이라 부름
      - 일반적으로 새롭게 내놓은 버전이 기존에 내놓은 버전과 문제없이 호환되지만, 새로운 기능을 추가했을 때 증가
    - Z: 마지막 숫자는 수(Patch) 버전이라 부름
      - 일반적으로 기존에 내놓은 버전과 문제없이 호환되며 버그를 수정한 정도의 작은 변화가 있을 때 증가

요약

- 하나의 버전을 만드는 과정
  - Working directory 내의 파일을 변경(추가, 삭제 등)
  - 변경한 내용 중 버전에 포함할 파일을 stage에 올리기
  - 커밋하기 => 버전 생성
  - .gitignore를 작성해 버전 관리를 하지 않을 파일이나 폴더를 자동으로 걸러낼 수 있었음
- Release 하기
  - 개발한 SW를 고객들에게 오픈하기
  - 태그를 통하여 버전 명시