Git basic

1. Version control system 을 사용하지 않을 때 발생할 수 있는 문제점

1) 혼자서 프로젝트를 진행할 때 발생할 수 있는 문제점

- 파일의 버전을 바꾸고 싶을 때
- 하드디스크가 크러쉬가 일어날 때 등등

->> 이러한 경우를 방지하기 위해 여러 파일을 만들어야한다.

2) 여러 사람이 프로젝트를 진행할 때 발생할 수 있는 문제점

- 누가 관리할 것인지
- 동시에 동일한 파일을 수정했을 때 어떻게 수정할 것인가
- 잘못 파일을 삭제할 때 어떻게 대응할 것인가 등등

->> 예전에는 클라우드를 공동으로 사용하거나 이메일을 주고받음 등의 방법을 사용했다.

3) 오픈소스에서 발생하는 문제점

- 어느 누구라도 소스코드를 볼 수 있고, 임의적으로 관리할 수 있기에 이에 대한 관리가 더 어렵다.

ex) 리눅스 소스코드가 전세계의 수많은 개발자에 의해 수정된다. 이 경우의 관리를 어떻게 할 것인가?

- 원시적인 방법으로 소스코드를 수정하는 것이 불가능해진다.

=> Version control system 을 사용해 위의 문제점을 해결할 수 있다.

2. Version control system

1) Version control system

- 파일이 변경될 때마다 하나씩 저장소에 저장하는 동작을 한다.
- 이때 저장소는 소프트웨어 개발을 위해 필요한 모든 파일을 저장함
- 저장소의 내용을 직접적으로 수정하는 것이 아니라 **수정된 내용을 보내서 반영하는 것**이 Version control system 의 핵심이다.
- 저장소에 있는 내용을 다운받는 것을 working copy 나 working tree 라고 하고 수정한 것을 반영하는 것이 commit 이라고 한다.

2) 레포지터리에 저장해야하는 것과 저장하면 안되는 것

(1) 레포지터리에 저장해야하는 것

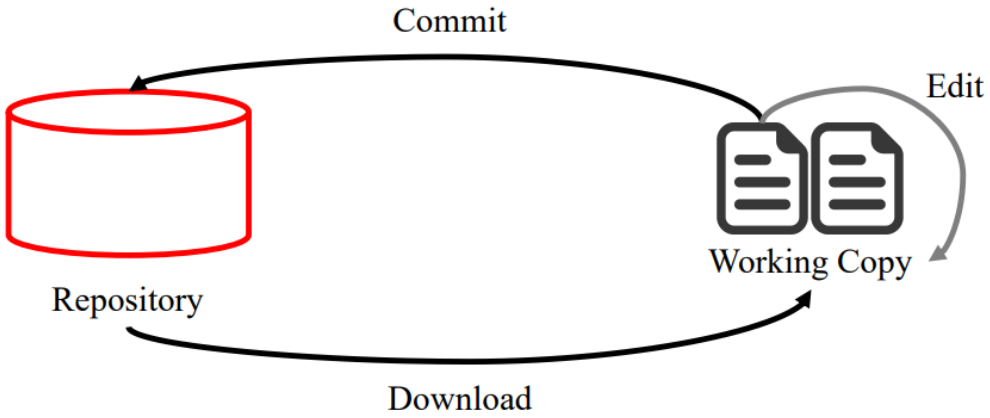
- 소스코드
- 빌딩과 관련된 파일
- 다른 파일들 등등

(2) 레포지터리에 저장하면 안되는 것

- 소스코드를 통해 자동으로 생성할 수 있는 파일

3) 레포지터리에 저장하고 commit 하는 과정

- 레포지터리에 있는 것을 다운로드 받는다.(Working copy, check out)
- 소스코드를 수정한다.(edit)
- 소스코드를 레포지터리에 반영한다.(Repository, check in)



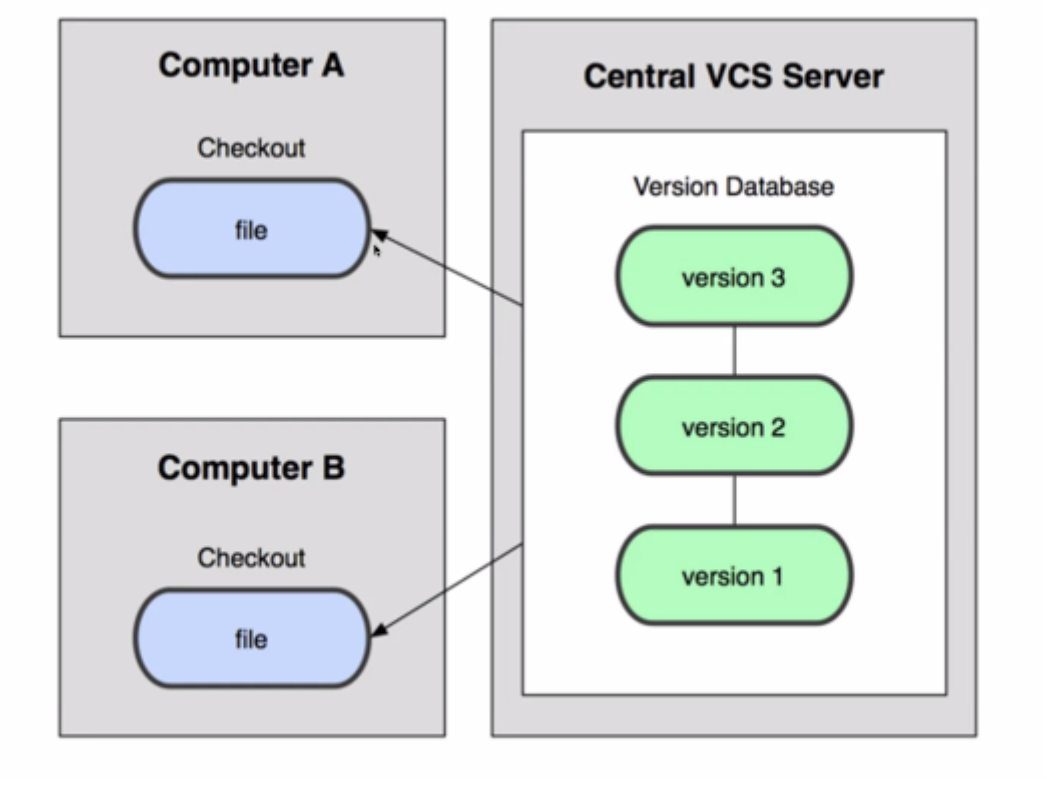
4) Version control 의 유형

- 공통의 자료를 공유한다는 점은 유형구분 없이 동일하다.

(1) 중앙화

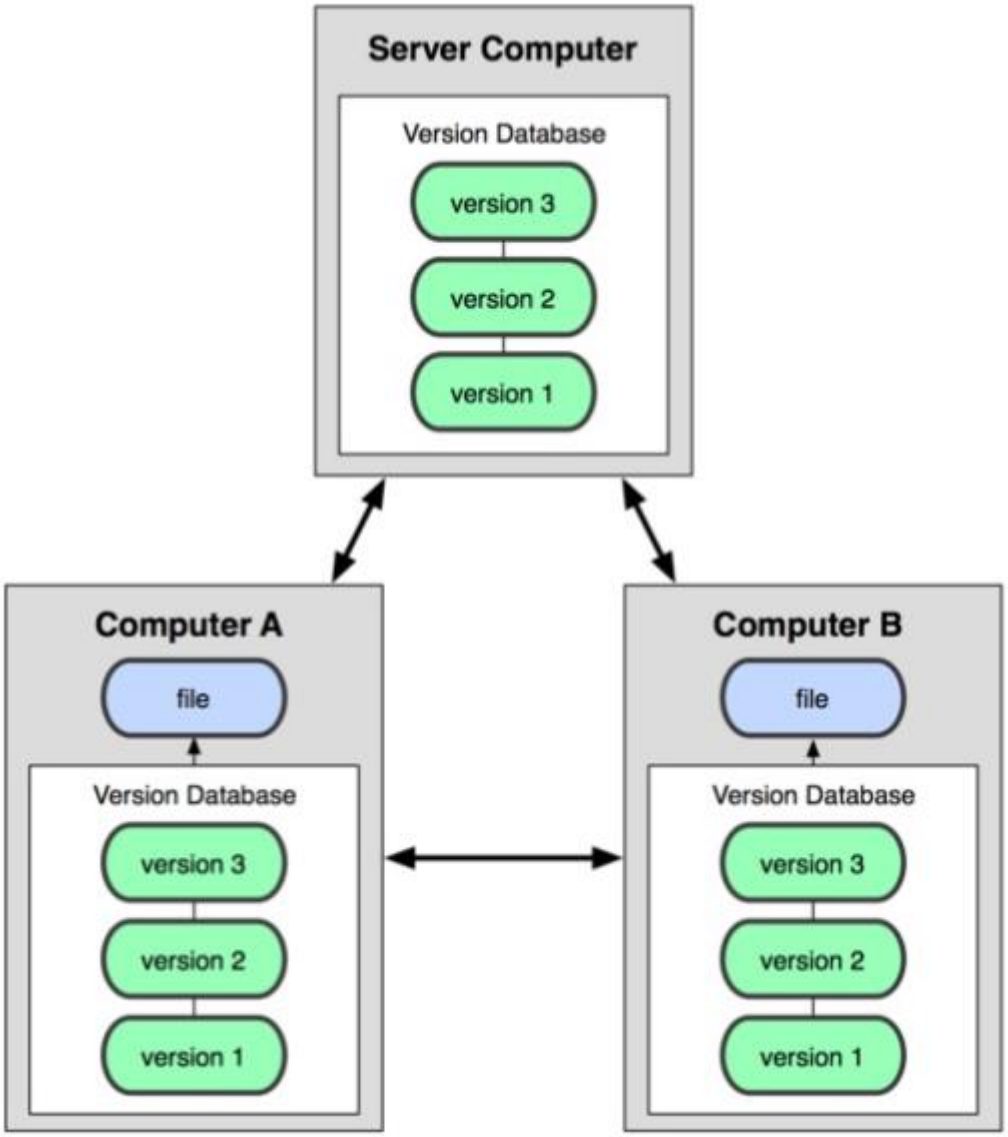
- 하나의 저장소(서버)를 관리하여 모든 파일이 해당 저장소에 저장된다.
- 파일을 다운받을 때 모든 정보가 아니라 파일만 checkout 할 수 있다.

- 비교적 관리하기 쉽고, 복잡하지 않다.
- **디스크 크러쉬가 발생할 때 모든 자료가 날라간다는 단점이 존재**한다.



(2) 분산화

- 모든 참가자가 모든 정보(과거 정보 등등도 포함)를 가지고 있다.
- Push / Pull 을 통해 저장소 간에 작업한 내용을 공유할 수 있다.
- 본인의 저장소를 가지고 있기 때문에 로컬에서 작업할 수 있고, 이에 따라 속도도 빠르고 간단하게 처리 가능하다.
- 외부 저장소가 날아가더라도 다른 참여자가 저장소를 가지고 있기 때문에 큰 문제 발생하지는 않는다.
- **수정사항이 발생할 때마다 서로 저장소끼리 주고받아야하기에 복잡하다는 단점**이 있다.



일반적으로는 원격 저장소를 중앙에 두고 진행하기도 한다.

3. Git

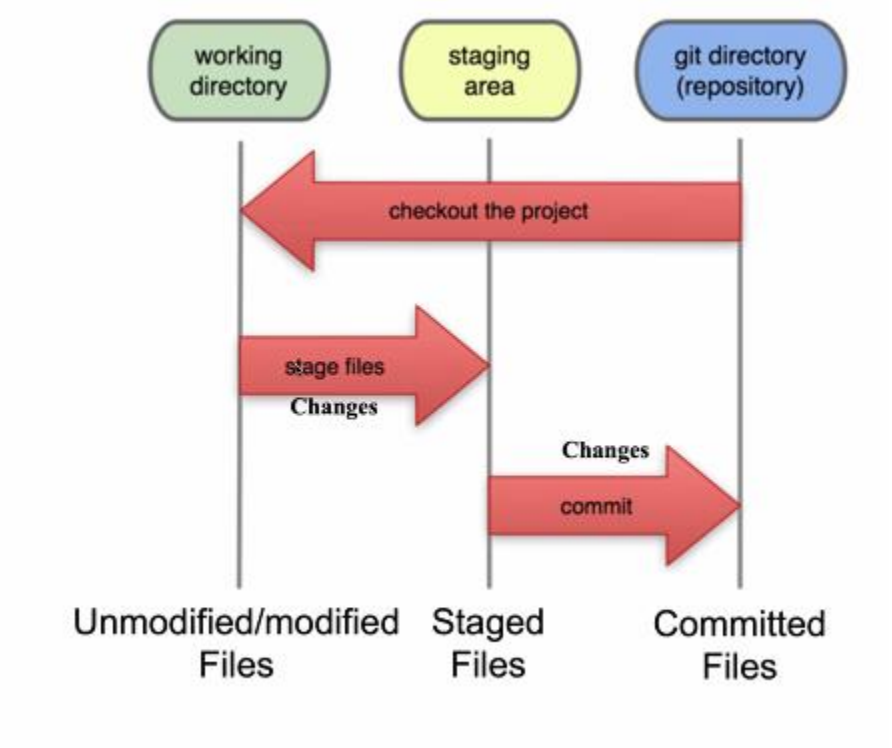
1) Git 이란?

- 리눅스를 효율적으로 개발하기 위해 만든 버전 컨트롤 시스템이다.
- **빠르고, 비선형적인 개발이 되고, 분산화되고, 많은 사람이 참여하는 버전 컨트롤 시스템을 만드는 것이 git 의 목표**다.

- Resilience / Speed / Space / Simplicity / Large userbase 의 장점이 있다.
- 배우는데 시간이 오래걸린다는 단점이 있다.
- 로컬과 서버에 저장소가 따로 있어 원격에 있는 서버에 반영을 해야한다.

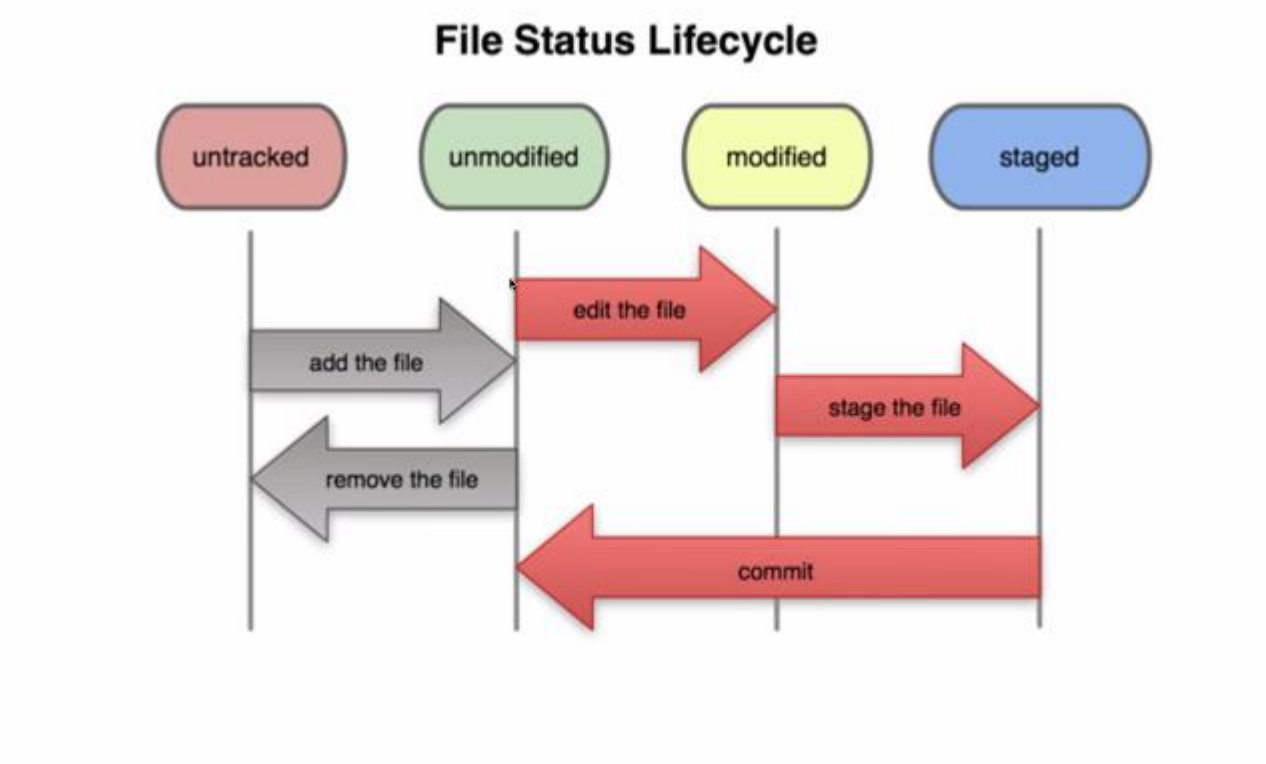
2) Local git project(레포지터리)의 세 가지 영역

(1) Working directory(working tree): 실제 작업을 하는 영역
(2) Staging area: commit 을 하기 전에 어떤 파일을 저장할건지 선택하여 변경한 내역들이 있는 영역
(3)Repository(local): commit 을 해서 staging area 에 있는 것을 저장하는 영역



3) File 상태

(1) Untracked: 레포지터리에 처음 등장하는 파일 상태
(2) Unmodified: 레포지터리에 있는 파일과 동일한 경우의 파일 상태
(3) Modified: 레포지터리에 있는 파일에서 수정된 파일 상태
(4) Staged: modified 파일을 staging 해서 commit 을 하기위해 선택한 파일 상태



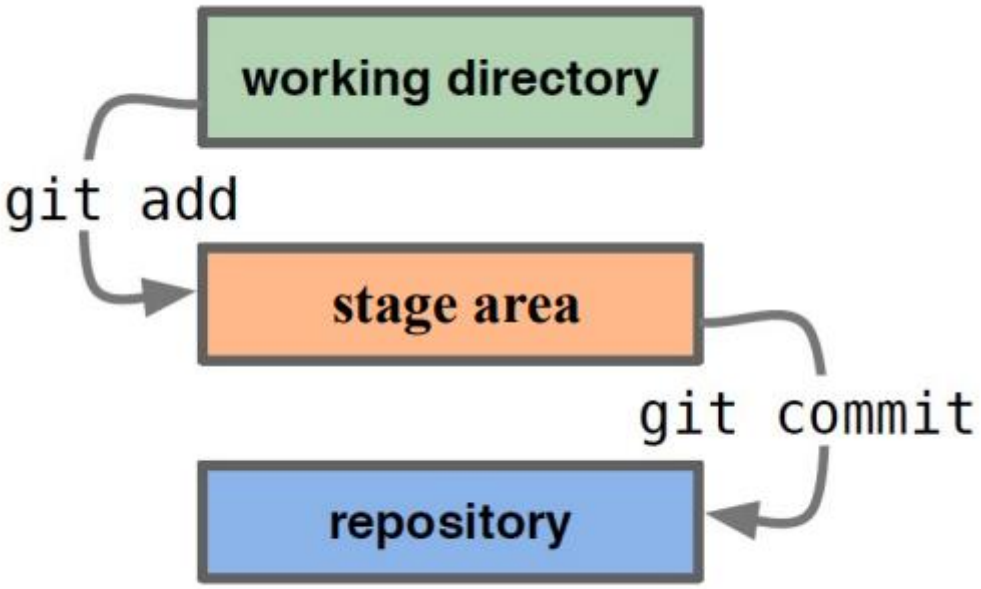
staged 상태에서 commit 을 한다면 unmodified 된다.

4) Checksum

- 전송받은 것이 올바르게 됐는지를 확인하기 위해 사용한다.
- **중앙화**는 서버에 있는 레포지터리에 저장할 경우에는 숫자를 통해 버전을 관리하는 것을 의미한다.
- **분산화**는 각자의 레포지터리에서 commit 마다 고유한 아이디를 부여해 버전을 관리하는 것을 의미한다. 각자 레포지터리를 사용해 작업을 진행하기에 숫자로 버전을 관리할 수 없기에 레포지터리의 내용이 이전에 커밋한 내용과 다르다면 16 진수로 해쉬해서 이를 표현한다.

5) Basic flow

- Init or clone a repo
- Edit files
- Stage the changes
- Review your changes
- Commit the changes


working directory
git add
stage area
git commit
repository