Git – 01

목차

Chapter 1. 버전 관리 시스템과 Git

- 버전 관리 시스템의 종류
  - 서버 – 클라이언트 모델
  - 분산 모델
  - CVS
  - 서브버전
  - 머큐리얼

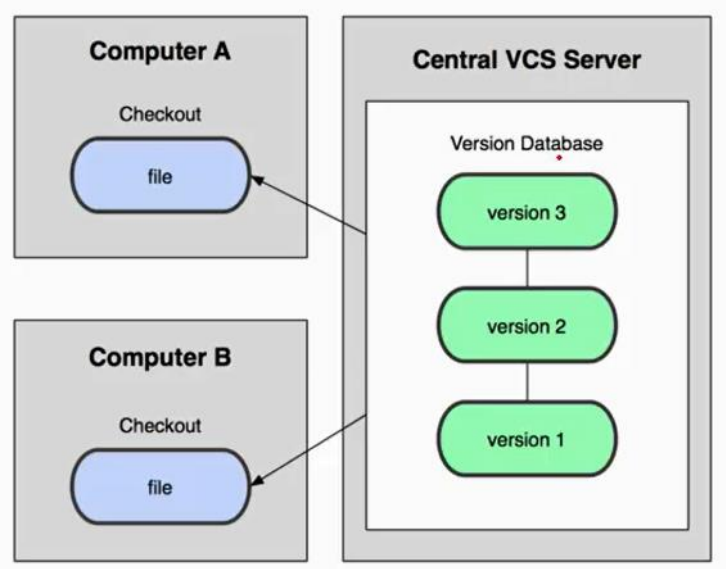
- Git
  - 장점
  - 특징

- Git을 사용해야 하는 이유

버전 관리 시스템

- Version Control System
  - 우리가 손으로 해야 했던 사본 생성, 보존, 복원을 한 번에 해줄 수 있는 도구

- 클라이언트 – 서버 모델



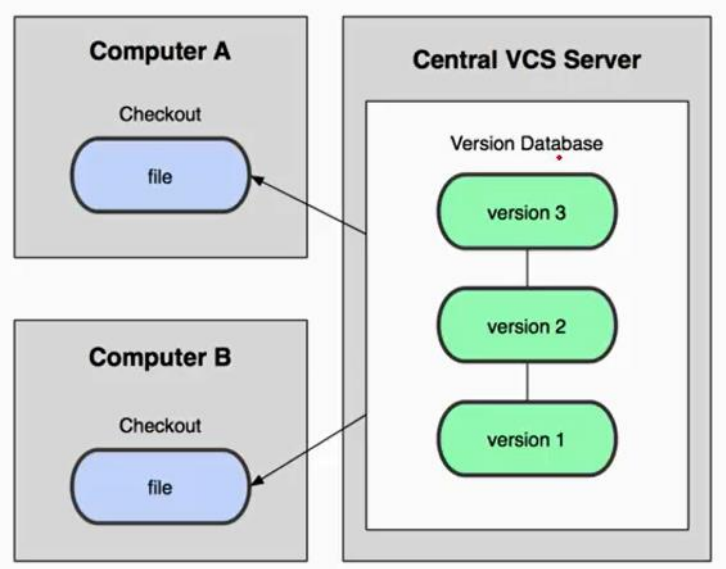
  - 하나의 중앙 저장소를 공유한 후 각각의 클라이언트는 저장소의 일부분만을 갖는 형태

  - 자신이 작업하는 부분만 로컬에 임시로 저장한 후 작업하는 형태

버전 관리 시스템

- Version Control System
  - 우리가 손으로 해야 했던 사본 생성, 보존, 복원을 한 번에 해줄 수 있는 도구

- 클라이언트 – 서버 모델

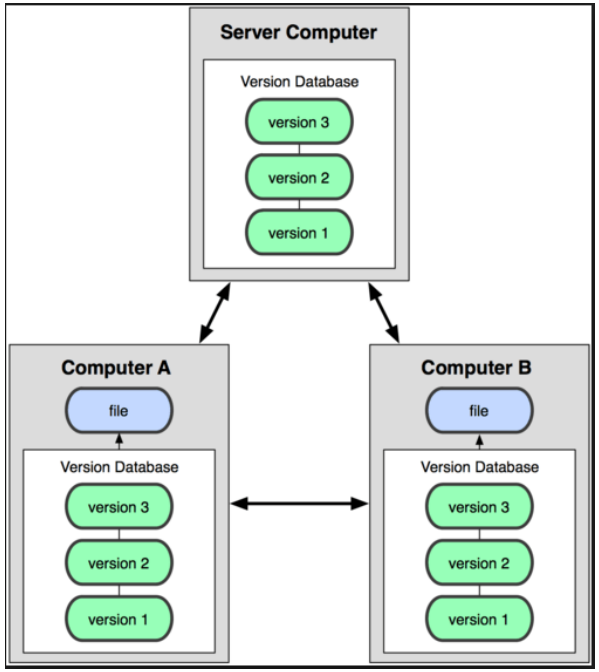


  - 이 모델은 중앙 저장소에서 프로젝트 관리의 모든 것을 처리하므로, 클라이언트에서 할 수 있는 작업이 파일 수정과 서버로 commit하는 것 등등 많지 않다.

  - 만약 서버가 고장난다면 불완전한 로컬의 파일 사본들만 남게 되는 형태이다.

버전 관리 시스템

- 분산 모델
  - 분산 모델은 프로젝트에 참여하는 모든 클라이언트가 전체 저장소에 대한 개별적인 로컬 저장소를 가지고 작업하는 형태이다.



  - '클라이언트 – 서버' 모델과 달리 분산 모델의 클라이언트는 각자가 온전한 전체 저장소의 사본을 로컬에 가지게 된다.

  - 서버 뿐만 아니라 프로젝트에 참여하는 모든 컴퓨터 각각에 로컬 저장소가 있으므로 위의 그림처럼 저장소와 하는 모든 상호작용이 로컬 저장소에 반영될 수 있다.

버전 관리 시스템

- CVS – Concurrent Versions System
  - 서버 – 클라이언트 방식의 가장 오래된 버전 관리 시스템
    - 서버의 저장소에 프로젝트의 원본이 있고, 클라이언트 각각은 서버에서 파일을 가져다가(Check out) 로컬 저장소에서 변경한 뒤 변경된 내역을 서버에 다시 보낸다.
  - 파일 각각의 버전을 관리하고 추적할 수 있다.
    - 하지만 파일 이름이 변경되거나 이동되는 것을 자동으로 추적하지 못하는 등 생각보다 많은 한계가 존재한다.
  - 서버에 사고가 생기면, 프로젝트를 복구하는 방법이 매우 제한적이다.

버전 관리 시스템

- Subversion – SVN
  - CVS의 여러 단점을 개선한 클라이언트 – 서버 모델의 자유 소프트웨어 버전 관리 시스템
  - CVS와 완전하게 호환되는 동시에 더 나은 기능을 제공하는 것을 목표로 한다.
  - CVS와 비교하여 각각의 commit이 원자적이며 파일 이름 변경, 복사, 이동, 삭제 등의 작업 내역을 유지하는 히스토리를 만든다는 장점이 있다.
  - 또한, CVS가 만드는 branch와 비교하여 SVN에서의 branching은 더욱 가벼운 작업이 되었다.

버전 관리 시스템

- Mercurial
  - 분산 모델의 버전 관리 시스템
  - 각각의 클라이언트가 전체 저장소를 갖는 분산 모델이므로 Git과 기본 개념에서는 크게 다르지 않다.
  - 머큐리얼이 시스템에 필요한 모든 기능을 한 번에 통합을 제공한다면, Git은 필요한 기능을 골라서 사용한다는 차이가 있다.
  - 파이썬으로 개발되었고, 명령어 대부분이 SVN과 공통으로 사용하는 부분이 많다.
  - 또한 Git과는 다르게 프로젝트의 커밋 내역은 변경 불가능한 사항으로 다룬다.
  - 태그나 브랜치 등을 다루는 방식이 Git과 다르다.

Git

- Git
  - 2005년, Linus Torvalds에 의해 개발된 분산 버전 관리 시스템
  - BitKeeper라고 불리는 분산 버전 관리 시스템 도입 이후, 프리 소프트웨어 문제로 리눅스 커뮤니티와 틀어진 것이 Git의 개발 동기
  - 리눅스 커널 버전 관리를 위해 2주만에 개발되었으며, 실제 개발 3일이 지났을 무렵 Git 자체의 버전을 관리하기 시작했다.
- 장점(대중성과 검증된 안정성)
  - 전 세계의 수많은 사용자가 사용한다.
  - Git을 사용한 저장소 공유 사이트인 GitHub 웹 사이트가 존재한다.
  - 사용자 수에서 나오는 엄청난 숫자의 튜토리얼과 프로젝트가 존재한다.

Git

- 특징
  - 크게 봤을 때, master 저장소 서버와 master 저장소의 완전한 사본을 가지는 클라이언트 저장소로 구성되어 있다.
    - 서버와 클라이언트 모두 완전한 저장소를 가지고 있다.
    - 저장소를 하나의 프로젝트로 봐도 무방하다.

  - 아래와 같은 기능을 지원하며, 아래에 포함되지 않은 더 강력한 기능들이 있다.
    - 로컬 및 원격 저장소 생성
    - 로컬 저장소에 파일 생성 및 추가
    - 수정 내역을 로컬 저장소에 제출
    - 파일 수정 내역 추적
    - 원격 저장소에 제출된 수정 내역을 로컬 저장소에 적용
    - master에 영향을 끼치지 않는 branch 생성
    - branch 사이의 병합
    - branch를 병합하는 도중의 충돌 감지

Git을 사용해야 하는 이유

- 협업을 할 때 모두가 동일한 버전의 프로젝트 리소스를 사용하기가 힘들다.
  - 누구를 기준으로 '최신' 버전을 맞출 것인가?
  - 파일을 주고 받는 것은 어떻게 해야할 것인가?
  - 어떤 팀원이 작업하고 있던 컴퓨터가 통째로 고장이라면?

- 어느 순간 누가 담당했는지 모를 코드들이 생긴다.
  - 코드들이 어떤 역할을 하는지 정확하게 파악할 수 없는 지경에 이를 수 있다.

- 언제 누가 무엇을 했느냐를 기록할 필요가 있다.

- 즉, 팀원 사이의 버전 맞춤, 할당된 작업의 결과물 관리, 특정 결과물이 누구의 것인지 추적하는 것과 같은 일이 Git을 이용함으로써 가능하다.