Open Source SW Utilization

(524820-2)

송영상(Youngsang Song)

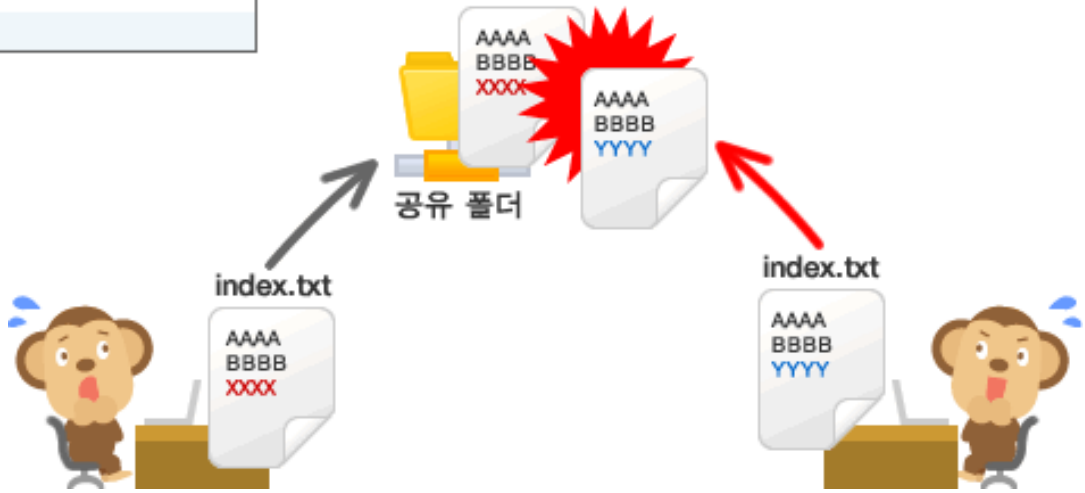
sw.yssong@dankook.ac.kr

Outline

- 형상관리
- CVS
- SVN
- Git
- Perforce(P4D)
- Github
- GitLab

형상 관리


Name
120525_문서_업데이트.txt
120604_문서.txt
120605_문서_수정판.txt
120605_문서_수정판2.txt
120605_문서_최신 복사.txt
120605_문서_최신.txt
120605_문서.txt
1200602_문서.txt
문서_회의용.txt

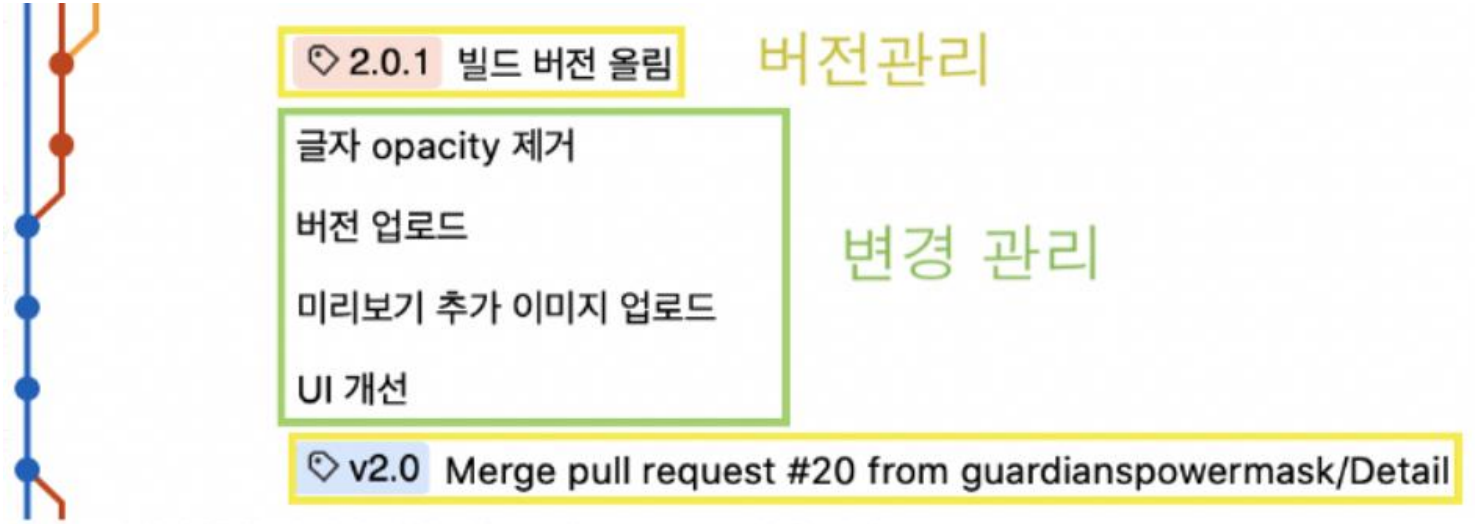

AAAA
BBBB
XXXX
AAAA
BBBB
YYYY
공유 폴더
index.txt
AAAA
BBBB
XXXX
index.txt
AAAA
BBBB
YYYY

형상관리

- **형상관리(Configuration Management)**
  - **SW 형상관리 or 구성관리**
    - 소프트웨어의 변경사항을 체계적으로 추적하고 통제하는 것으로, 형상 관리는 일반적인 단순 버전 관리 기반의 소프트웨어 운용을 좀 더 포괄적인 학술 분야의 형태로 넓히는 근간
    - 형상항목이라는 형태로 작업 산출물을 선정하고, 형상 항목 간의 변경 사항 추적과 통제 정책을 수립하고 관리
  - **SCM(Software Configuration Management)**
    - 소프트웨어 구성 관리란 소프트웨어 소스 코드 뿐 아니라 개발 환경, 빌드 구조 등 전반적인 환경 전반적인 내역에 대한 관리 체계를 정의
    - IEEE Standard Glossary of Software Engineering Terminology(1991)
      - » 형상 항목을 식별하여 그 기능적 물리적 특성을 문서화하고, 그러한 특성에 대한 변경을 제어하고, 변경 처리 상태를 기록 및 보고하고, 명시된 요구사항에 부합하는지 확인하는 일련의 사항에 대해 기술적인 행정적인 지침과 사후 관리를 적용하는 원칙
    - Roger's S Pressman, "Software Engineering, 3rd Edition"(1995)
      - » 전체 소프트웨어 공학 과정에 적용되는 '보호(Umbrella)' 활동이다.
      - » 변경은 언제나 일어날 수 있기 때문에, SCM(Software Configuration Management) 활동은 변경을 알아내고, 제어하고, 적절히 수행되고 있는 것을 확인하기 위해, 변경에 관심을 가지고 있는 사람들에게 이것을 통보하는 것

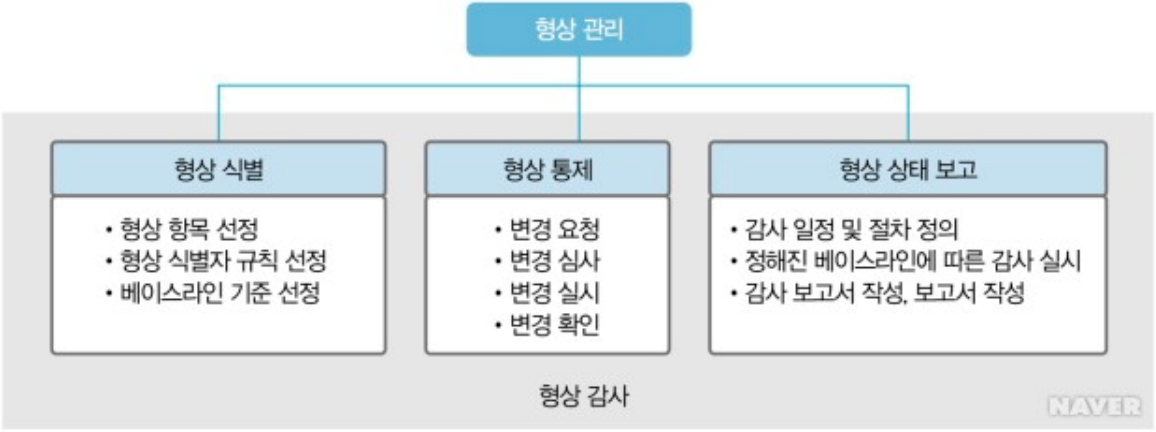
형상관리

- 변경관리/버전관리/형상관리
  - **변경** 관리 – **소스코드** 변경 사항에 대한 관리
  - **버전** 관리 – 변경사항을 '**버전**'이란 개념을 통해 관리.
  - **형상** 관리 – 위의 개념을 포함해 **프로젝트와 관련된 모든 변경사항**을 관리.



형상관리

- **형상관리 구조**
  - **형상 식별** (Configuration Identification)
    - 형상 관리의 **대상**이 무엇인지 **식별**하는 것. 식별 대상을 **형상 항목**(configuration item) 이라 함
  - **형상 제어**(Configuration Control)
    - 형상 항목의 버전(version control)과 변경에 대한 판단을 내리는 것.
  - **형상 감사**(Configuration Audit)
    - 요구대로 형상 항목의 변경이 제대로 이뤄졌는지 살펴보는 것.
  - **형상 상태 보고**(Configuration Status Accounting)
    - 변경된 형상 항목을 관계된 사람들에게 알리는 것.

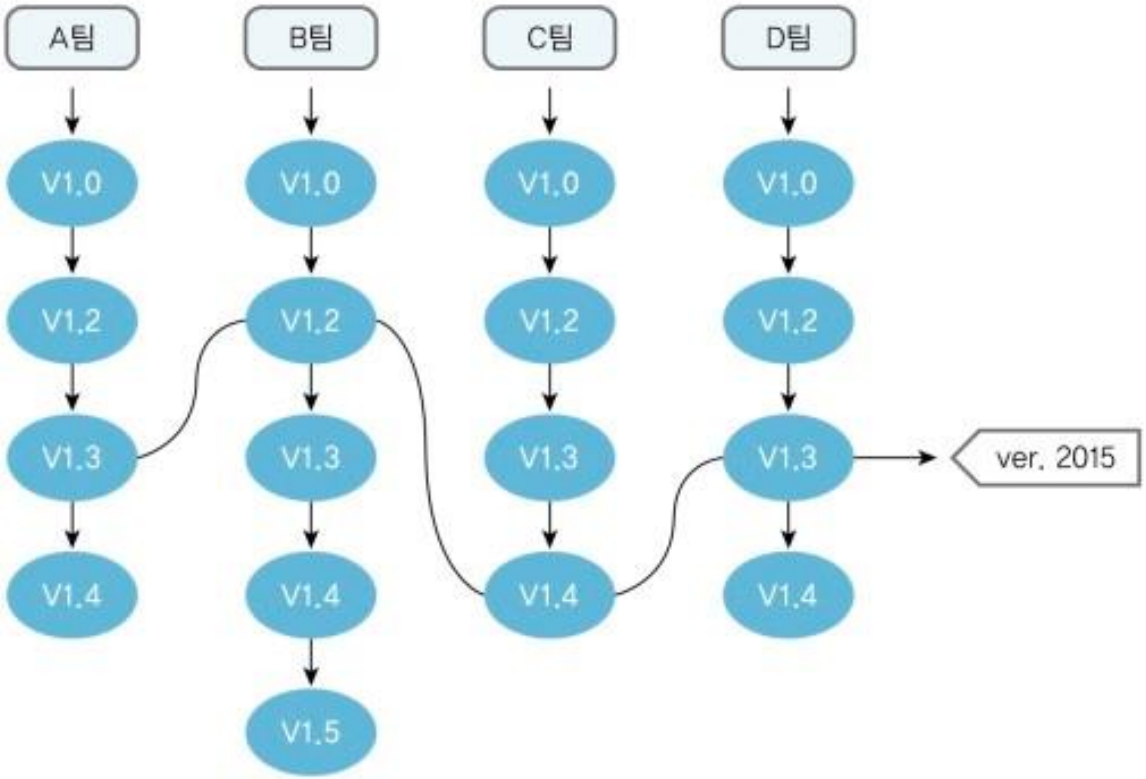


형상관리

- **형상 식별**
  - **어떤 산출물을 형상 관리의 대상으로 할 것인가**
    - **형상 항목** : 개발 단계에서 생산되거나 사용되는 작업이나 산출물
      - » 실행 파일, 문서 형식의 산출물, 원시 코드, 개발 이력, 개발 도구
    - **1) 형상 항목 선정**
      - » 어떤 항목을 관리 대상으로 할 것인지를 정함.
    - **2) 형상 식별자 규칙 선정**
      - » 어떤 프로젝트에서 사용되는 파일인지
      - » 어떤 내용의 문서인지
      - » 버전이 어떻게 되는지
      - » 같은 작업을 하는 소속 팀원들끼리 한눈에 알아볼 수 있도록 이름을 명명하는 규칙
    - **3) 베이스라인 기준 선정**
      - » 베이스라인 : 소프트웨어 개발 과정 중 특정 시점에 만들어진 산출물의 집합을 말한다.

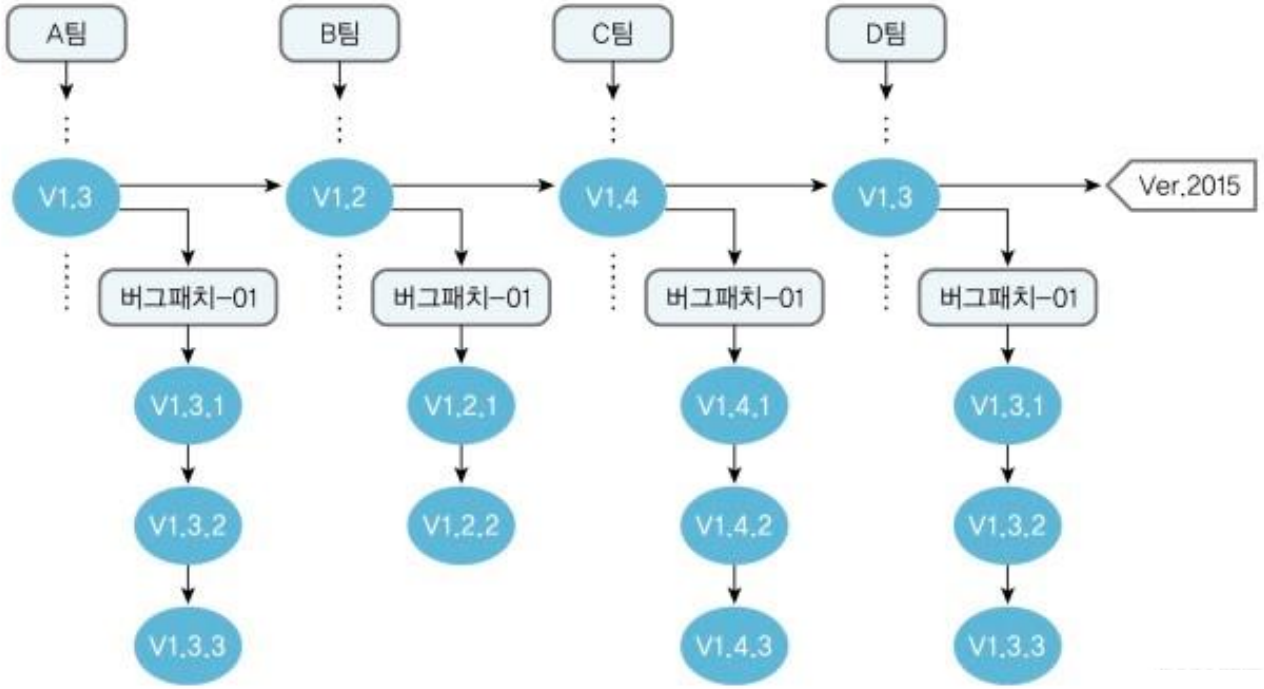
형상관리

- **형상 식별**
  - **Ver. 2015 출시 : 각 팀은 v1.3, v1.2, v1.4, v1.3**
  - **Baseline : v1.3, v1.2, v1.4, v1.3**



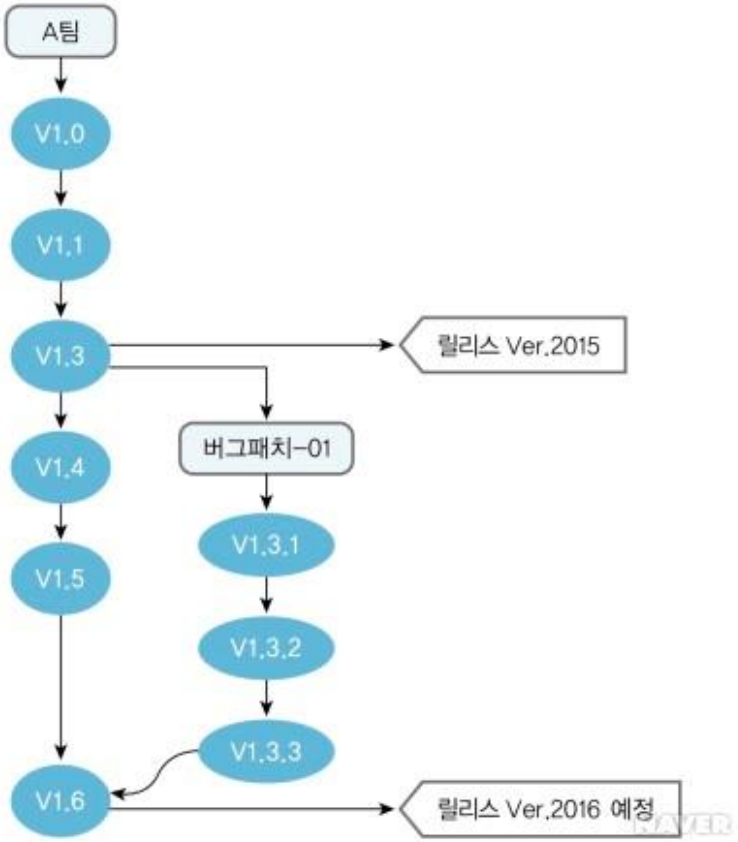
형상관리

- **형상 식별**
  - Ver. 2016 출시 : Ver.2015에 기능 추가
  - Patch : 기존 파일에 덮어쓰면 나중에 수정된 파일과 수정 전 파일을 비교할 수가 없고 수정 전 파일이 필요한 경우에는 난감해짐.
  - 해결하는 방법 = 브랜치



형상관리

- **형상 식별**
  - Ver. 2016의 오류를 해결한 패치 결과 반영 안됨
  - Ver. 2015의 오류가 수정된 패치 결과를 Ver. 2016에 포함.
  - 해결하는 방법 = 브랜치



형상관리

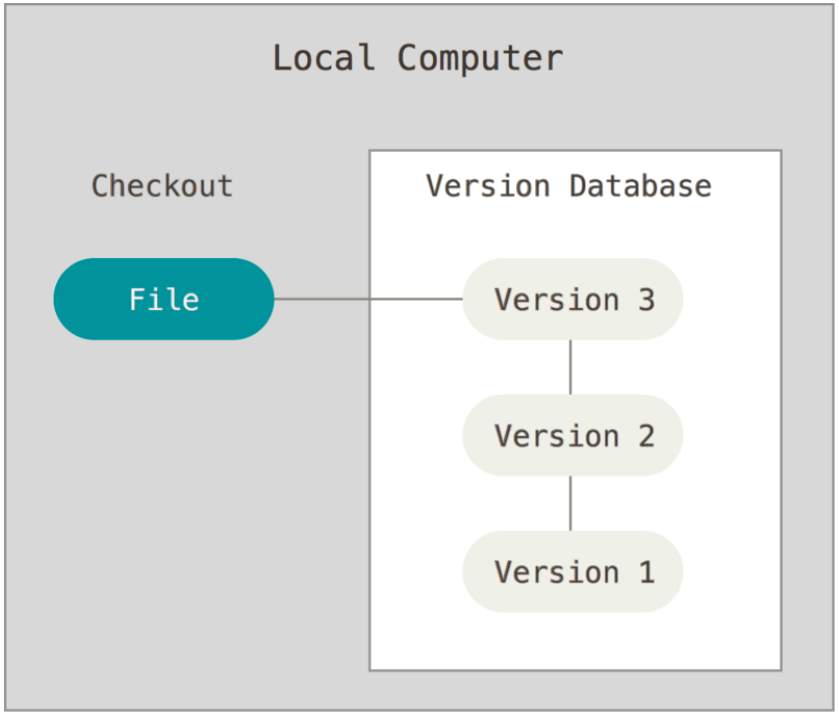
- **형상 통제**
  - 정의 :
    - 형상 항목 목록의 변경 요구를 검토 및 승인해서 현재의 소프트웨어 기준선에 반영될 수 있도록 통제하는 일련의 과정
  - 변경에 대한 요구를 무조건적으로 다 수용 하지 않음
  - 변경하고자 하는 요구를 정해진 양식에 맞추어 작성하고, 이 요청을 수용할 것인지 거절할 것인지 결정
- **형상 상태 보고**
  - **정의 :**
    - 개발 상태에 대한 가시성을 통해 형상을 효율적으로 관리하기 위해서 베이스라인으로 설정된 형상 항목의 구조와 변경 상태를 기록하고, 보고하는 것
  - 프로젝트에서의 변경 횟수
  - 베이스라인의 상태
  - 최근 소프트웨어 항목의 버전, 릴리스 식별자, 릴리스 횟수, 등등

형상관리

- **버전 관리**
  - 변경 사항을 버전으로 관리하는 것을 의미
  - 파일 변화를 시간에 따라 기록했다가 나중에 특정 시점의 버전을 다시 꺼내 올 수 있는 시스템
  - 소프트웨어 소스 코드 뿐 아니라 거의 모든 컴퓨터 파일의 버전을 관리할 수 있음.
  - 방식
    - 로컬 버전 관리
      - » 간단한 데이터베이스를 사용해서 로컬 컴퓨터에서 파일의 변경 정보 관리
    - 중앙 집중식 버전 관리(CVCS, Central Version Control Server)
      - » 파일을 관리하는 서버가 별도로 있고 클라이언트가 중앙 서버에서 파일을 받아서 사용
    - 분산 버전 관리 시스템(DVCS, Distributed Version Control Server)
      - » 저장소를 히스토리와 더불어 전부 복제. **= clone**
      - » 서버에 문제가 생기면 이 복제물로 다시 작업을 시작
      - » 클라이언트 중에서 아무거나 골라도 서버를 복원

형상관리

- **로컬 버전 관리 (Local Version Management)**
  - 간단한 데이터베이스를 사용해서 로컬 컴퓨터에서 파일의 변경 정보 관리
  - 기본적으로 파일에서 변경이 되는 사항만 관리
  - 간단하지만 실수하기 쉽다.

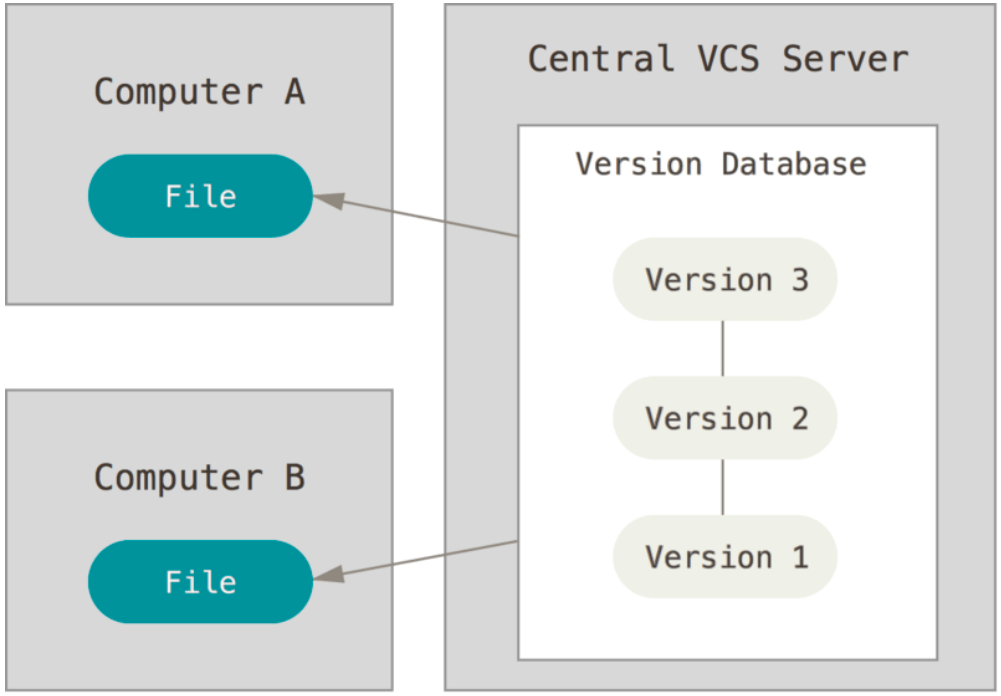


형상관리

- **중앙집중식 버전 관리**
  - **CVCS(Central Version Control Sever)**
  - 파일을 관리하는 서버가 별도로 있고 클라이언트가 중앙 서버에서 파일을 받아서 사용

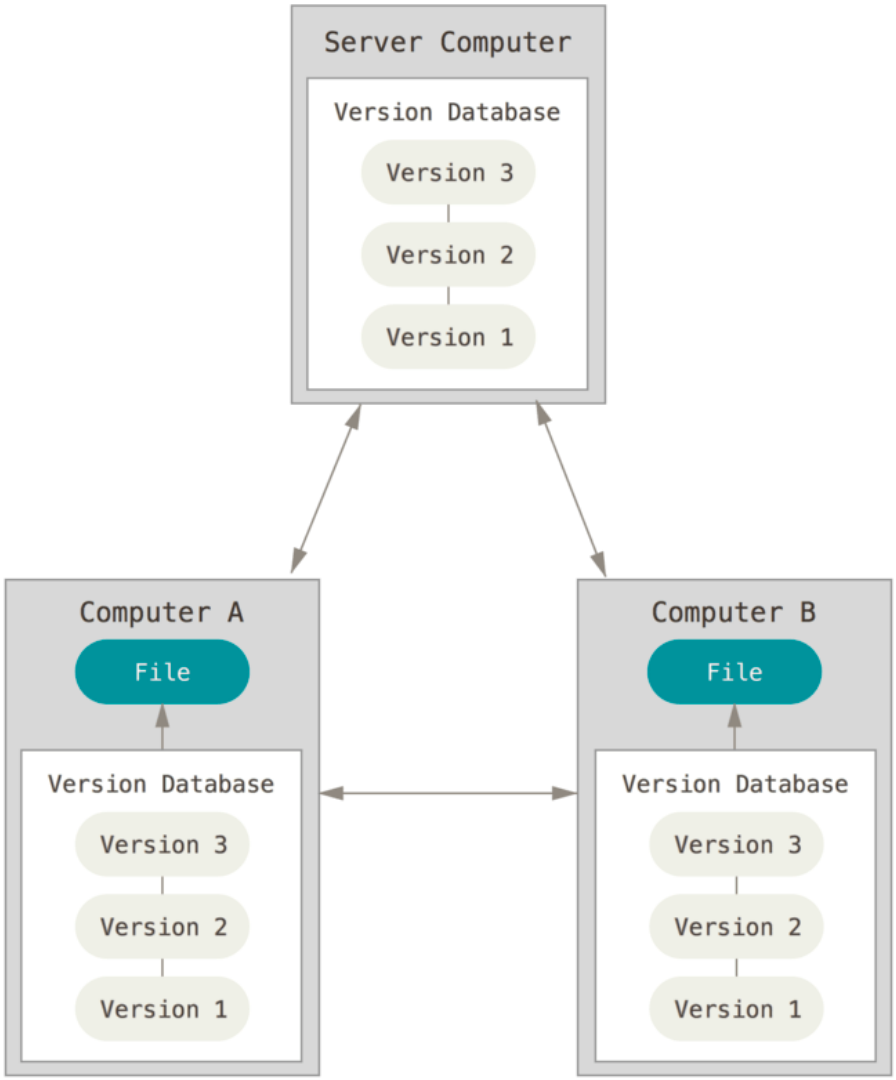
장점
모두 누가 무엇을 하고 있는지 알 수 있게 됨
모든 클라이언트의 로컬 데이터베이스를 관리하는 것보다 VCS 하나를 관리하기가 훨씬 쉽다.

단점
만약 중앙 서버가 문제가 발생하면 서버가 다운된 동안에는 협업 및 백업 불가능 중앙 서버에 있는 하드디스크가 문제가 생기면 프로젝트들의 히스토리를 잃게 됨
(= 로컬 VCS도 이런 문제가 있음)



형상관리

- **분산 버전 관리 시스템**
  - **DVCS(Distributed Version Control Sever)**
  - 서버에 문제가 생기면 이 복제물로 다시 작업을 시작할 수 있음
  - 클라이언트 중에서 아무거나 골라도 서버를 복원 가능



형상관리

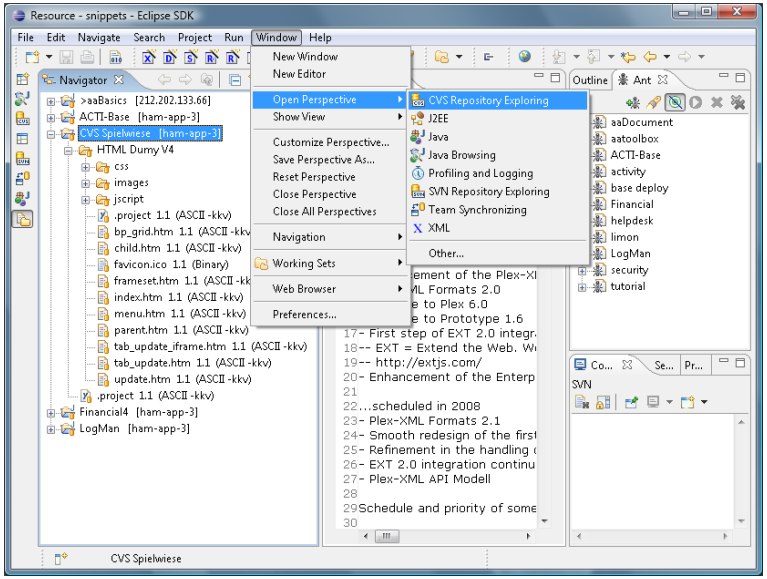
- **형상관리를 잘못했을때 생길 수 있는 어려움**
  - 우선 형상 관리의 **목적**은 **변경을 추적**하고 이를 **처리하는 메커니즘(형상 관리 대상 파악, 베이스라인 지정, 버전 관리, 접근 제어** 등)을 제공
  - 특히 대규모 프로젝트에서는 발생 가능한 **위험이나 혼란을 줄이고** 프로젝트를 **체계적으로 관리**하기 위해 형상관리가 반드시 필요
  - 형상 관리가 제대로 수행되지 않는다면 위와 같은 형상관리의 장점들이 사라 짐

형상관리

- **형상 관리를 위한 도구와 특징**
  - **CVS** (Concurrent Version System)
    - 90년에 출시된 무료 서버-클라이언트 형상관리 시스템
    - 파일 전체를 저장하는 것이 아니라 변경사항만을 저장함으로 용량을 적게 차지하지만 속도가 상대적으로 느림
  - **SVN** (Subversion)
    - 형상관리/소스관리 툴의 일종. 중앙관리만을 지원. 다른 사용자의 커밋과 엉키지 않으며, 커밋 실패 시 롤백 기능을 지원. 안정성에 있어 CVS보다 상대적으로 좋지 않다.
  - **Git**
    - 분산형 버전관리 시스템 . Repository의 완전한 복사본을 로컬에 저장할 수 있다. 처리속도가 빠르지만 대용량 코드 관리에 부적절하다.
  - **Perforce**(P4D)
    - 빠른 속도, 빠른 Merge가 가능하며 큰 리소스 관리에 좋다. 하지만 유료이고 파일명이 바뀌면 히스토리 추적이 곤란하다.

CVS

- **CVS(Concurrent Version System)**
  - 동시 버전 관리 시스템(Concurrent Versioning System)으로도 알려져 있으며, 버전 관리 시스템을 구현
  - 보통 소프트웨어 프로젝트를 진행할 때, 파일로 이뤄진 모든 작업과 모든 변화를 추적하고, 여러 개발자가 협력하여 작업할 수 있게 함
  - CVS는 오픈 소스 프로젝트에서 널리 사용되었으나, 현재는 CVS가 한계를 맞아, CVS를 대체하는 서브버전이 개발 됨

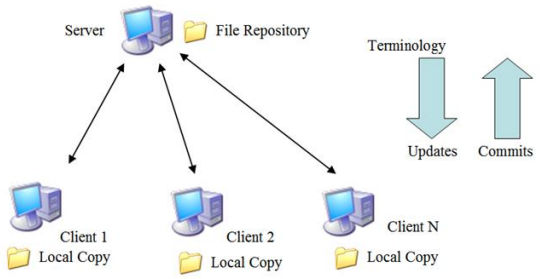


CVS

- **CVS(Concurrent Version System)**
  - 장점
    - 오랜기간 많은 유저를 가졌던 만큼 안정적
    - 하나의 파일에 대한 동시 작업이 가능.
    - Merge, Branch, Tag, Compare 기능을 지원.
    - Unix, Linux, Windows 등 다양한 운영체제를 지원.
    - 파일 전체를 저장하는 것이 아니라 변경사항만을 저장함으로 용량을 적게 차지.
  - 단점
    - CVS 저장소의 파일들은 이름을 바꿀 수 없기 때문에, 제거한 뒤 다시 추가
    - CVS 프로토콜은 디렉토리의 이동이나 이름 변경을 허용하지 않기 때문에 파일을 지우고 다시 추가
    - 아스키 코드를 지원하며, 유니코드는 제한적으로 지원
    - 속도가 상대적으로 느리다.
    - 커밋 실패 시 롤백이 지원되지 않는다.

      *커밋(commit): 수정한 소스를 저장소에 반영한다는 의미
    - CVS 디렉토리가 다른 툴에 비해 지저분한 느낌을 줌

SVN

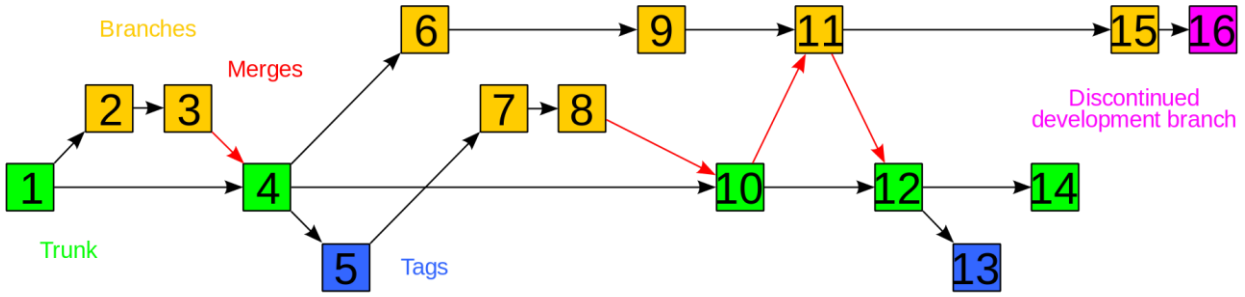


- **SVN(SubVersioN)**
  - **CVCS 방식**
  - CVS의 단점을 보완하기 위해 2000년에 만들어진 소프트웨어
  - **장점**
    - 파일과 디렉토리의 삭제, 이동, 이름 변경, 복사 등을 지원.
    - 소스파일 이외에 이진파일도 효율적으로 저장할 수 있음.
    - 디렉토리도 버전 관리를 할 수 있으며, 디렉토리 전체를 빠르게 옮기거나 복사할 수 있고, 리비전 기록도 그대로 유지.
    - 저장소의 크기에 상관 없이 일정한 시간 안에 가지치기나 태그를 할 수 있음.
    - 처리 속도가 상대적으로 빠름.
  - 단점
    - CVS보다 상대적으로 안정성이 부족
    - .svn 디렉토리로 인해 저장소가 다소 지저분한 느낌
    - 잦은 커밋으로 인해 리비전 번호가 크게 증가할 수 있다.
    - 소스코드는 Diff를 통해 Merge가 가능하지만, 이진파일은 어느 한쪽을 버릴 수 밖에 없다.
    - 개별 개발자만의 개발 이력을 가질 수 없다.

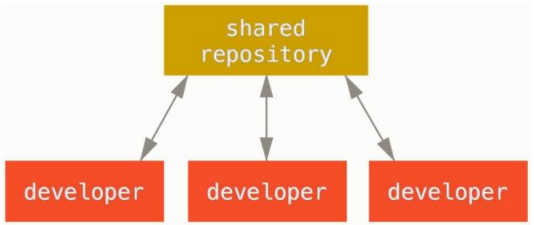
SVN

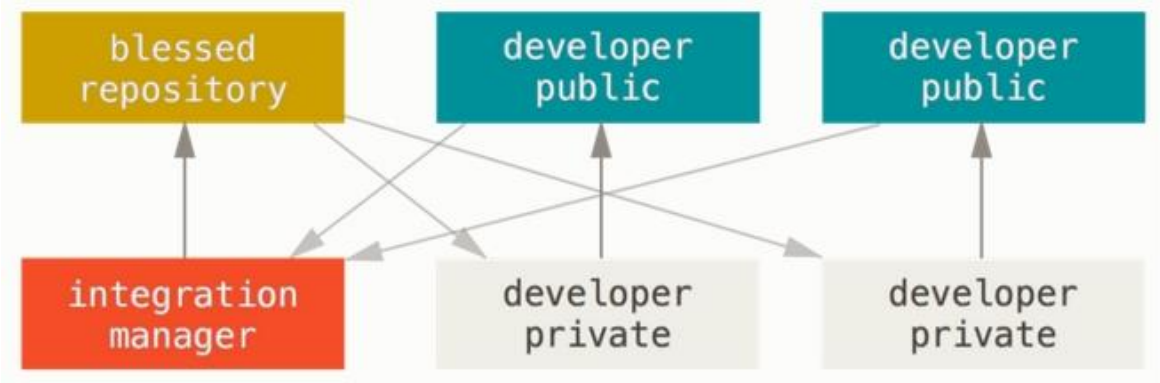
용어

- Repository: 프로젝트 파일 및 변경 정보가 저장되는 장소
- Import: 빈 Repository에 맨 처음 파일들을 채우는 것
- Export: 버전 관리 파일들을 뺀 순수 파일만 빼내는 것
- Checkout: 저장소에서 최신 버전의 소스코드를 최초로 받아오는 것 / Repository에서 프로젝트 관련 파일들을 받아온다
- Update: 로컬 저장소에 있는 파일들을 저장소의 최신 버전으로 받아 오기
- Commit: 로컬 저장소의 변경된 내용을 서버로 전송 / Checkout한 파일의 수정사항을 갱신
- Revert: 로컬 저장소의 내용을 이전 상태로 돌림
- Add: 버전관리 대상으로 파일 등록
- Trunk: 개발 소스를 commit 했을 때 개발 소스가 모이는 곳 / 프로젝트에서 가장 중심이 되는 디렉토리, 소스와 파일 포함
- **Branch**: trunk에서 분리/복사한 소스로 버전별 배포판을 만들거나 trunk와 별도로 운영환경을 위한 안정화된 소스 관리 목적으로 사용
- Tag: 특정 시점의 상태 보존 목적으로 사용 장기적으로 1.0, 1.1 등 버전 별로 소스 코드를 따로 저장, 특정 시점에서 프로젝트의 스냅샷을 찍어두는 것



- **Git**
  - 소스코드를 효과적으로 관리하기 위해 개발된 '분산형 버전 관리 시스템' (원래는 Linux 소스코드를 관리할 목적으로 개발)
  - **형상 관리 도구(Configuration Management Tool) 중 하나.**
  - Git은 소프트웨어를 개발하는 기업의 핵심 자산인 소스코드를 효과적으로 관리할 수 있게 해주는 **무료, 공개소프트웨어.**
  - SVN보다 여러 장점이 있어 SVN을 쓰던 개발 조직들이 Git을 선호



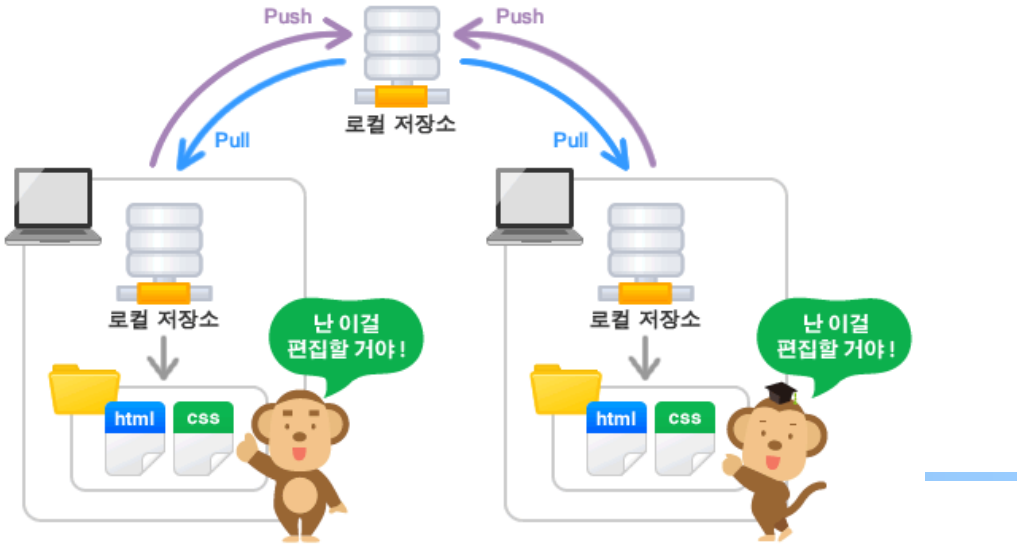


SVN vs. Git

	SVN	git
사용법	간편하고 어렵지 않다.	다소 복잡하고 초보자가 이해하기 어렵다
기능	버전관리에 최적화된 간편한 기능	다양한 기능이 존재. 이런 기능이 있어?라고 할정도로 버전 관리에 필요한 모든 기능이 존재함.
프로세스	중앙 집중식	분산 관리식
소스 충돌 위험	매우 높음	권한 설정을 통해 충돌 위험 감소
저장소 백업 여부	저장소 백업이 용이 하지 못함.	git 저장소만 있으면 리모트 복구 언제든지 가능. 매우 용이
다수 작업 관리	관리에 한계가 존재	수백~수천의 프로그래머의 분산 작업에 매우 용이
작업 내용 복구	다소 불편함.	예전 리비전으로 복구가 매우 편리
브렌치 생성	다소 불편함.	로컬에서 브렌치 생성 및 태그 생성이 매우 편리하다.

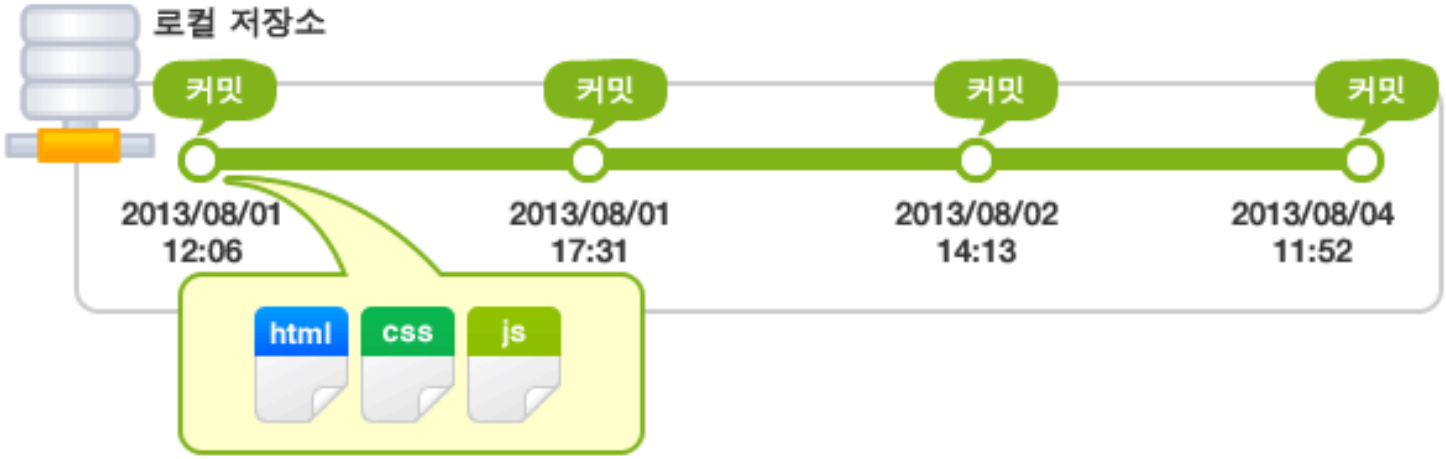
Git

- **Git Repository**
  - 파일이나 폴더를 저장해 두는 곳
  - 파일이 변경 이력 별로 구분되어 저장
  - 비슷한 파일이라도 실제 내용 일부 문구가 서로 다르면 다른 파일로 인식하여 파일을 변경 사항 별로 구분해 저장
  - **Repository**
    - 원격 저장소(Remote Repository): 파일이 원격 저장소 전용 서버에서 관리되며 여러 사람이 함께 공유하기 위한 저장소
    - 로컬 저장소(Local Repository): 내 PC에 파일이 저장되는 개인 전용 저장소



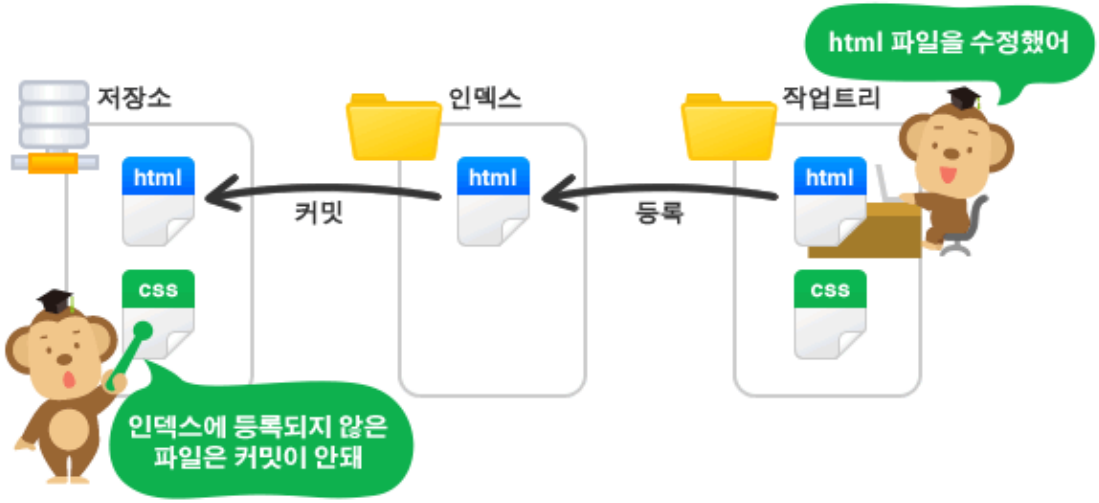
Git

- **Commit**
  - 파일 및 폴더의 추가/변경 사항을 저장소에 기록
  - 이전 커밋 상태부터 현재 상태까지의 변경 이력이 기록된 커밋(혹은 리비전)이 생성
  - 커밋은 시간순으로 저장되며, 최근 커밋부터 거슬러 올라가면 과거 변경 이력과 내용 확인 가능
  - 버그 수정, 기능 추가 등 특별한 의미가 있는 업데이트를 작업 별로 구분해서 각각 커밋하면, 나중에 이력을 보고 특정 변경 내용을 찾기 용이



Git

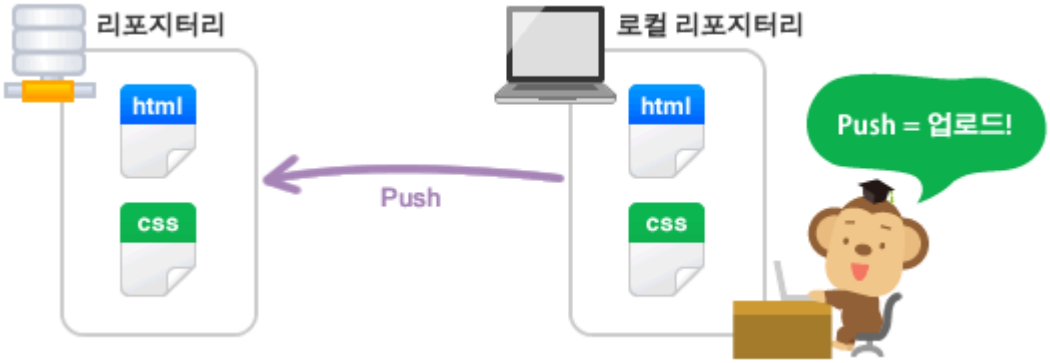
- **Work Tree**
  - 흔히 말하는 폴더를 '작업 트리'(Work Tree)라고 부름
- **Index**
  - 커밋을 실행하기 전의 저장소와 작업 트리 사이에 존재하는 공간



- '커밋' 작업은 '작업 트리'에 있는 변경 내용을 저장소에 바로 기록하는 것이 아니라 그 사이 공간인 '인덱스'에 파일 상태를 기록(stage – 스테이징)
- 저장소에 변경 사항을 기록하기 위해서는, 기록하고자 하는 모든 변경 사항들이 '인덱스'에 존재

Git

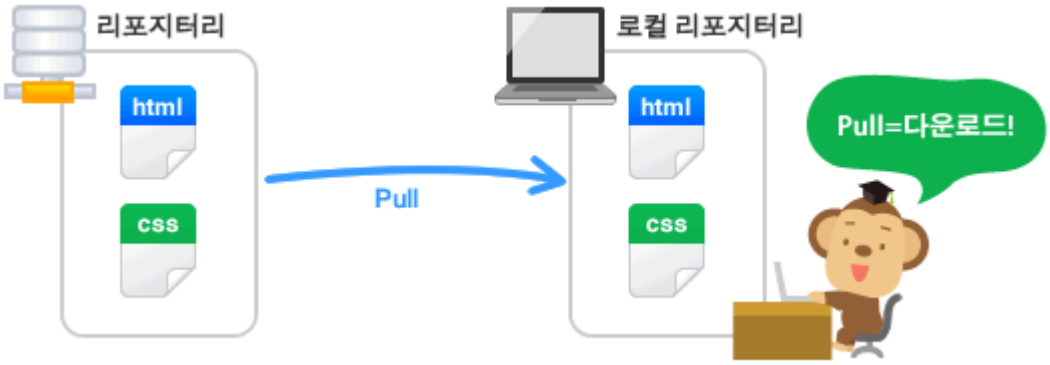
- **Push**
  - 로컬 저장소의 변경 이력을 원격 저장소에 업로드
  - 원격 저장소에 내 변경 이력이 업로드되어, 원격 저장소와 로컬 저장소가 동일한 상태가 됨



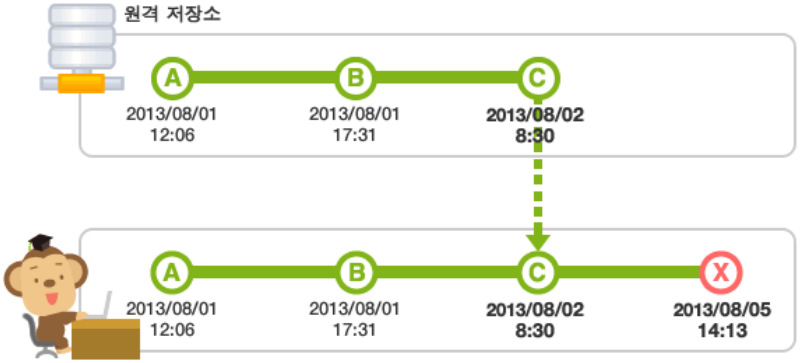
- **Clone**
  - 원격 저장소를 복제
  - 복제란 원격 저장소의 내용을 통째로 다운로드
  - 복제한 저장소를 다른 PC에서 로컬 저장소로 사용할 수 있음

Git

- **Pull**
  - 원격 저장소에서 로컬 저장소로 업데이트
  - 원격 저장소에서 최신 변경 이력을 다운로드하여 내 로컬 저장소에 그 내용을 적용

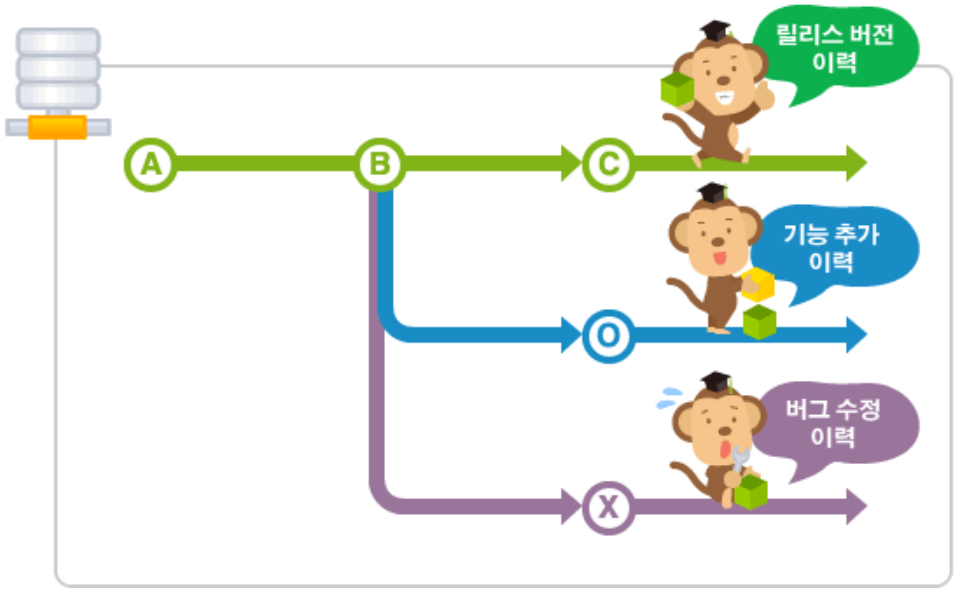


- **Merge**
  - pull 을 실행한 후 다른 사람이 push 를 하여 원격 저장소를 업데이트 해버린 경우 error 발생
  - 병합(merge)이라는 작업을 진행하여 다른 사람의 업데이트 이력을 내 저장소에도 갱신
  - 병합하지 않은 채로 이력을 덮어쓰게 되면 다른 사람이 push 한 업데이트 내역이 사라져 버림



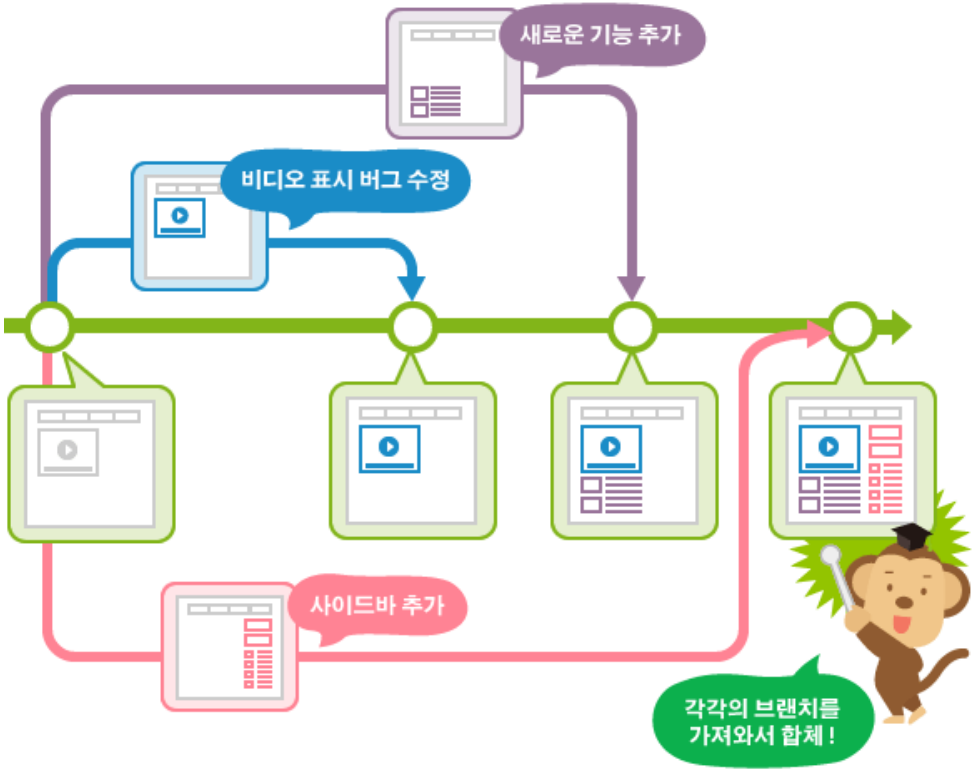
Git

- **Branch**
  - 독립적으로 작업을 진행
  - 여러 사람이 동일한 소스코드를 기반으로 서로 다른 작업을 할 때에는 각각 서로 다른 버전의 코드가 생성
  - 여러 개발자들이 동시에 다양한 작업을 할 수 있게 만들어 주는 기능이 바로 '브랜치(Branch)’



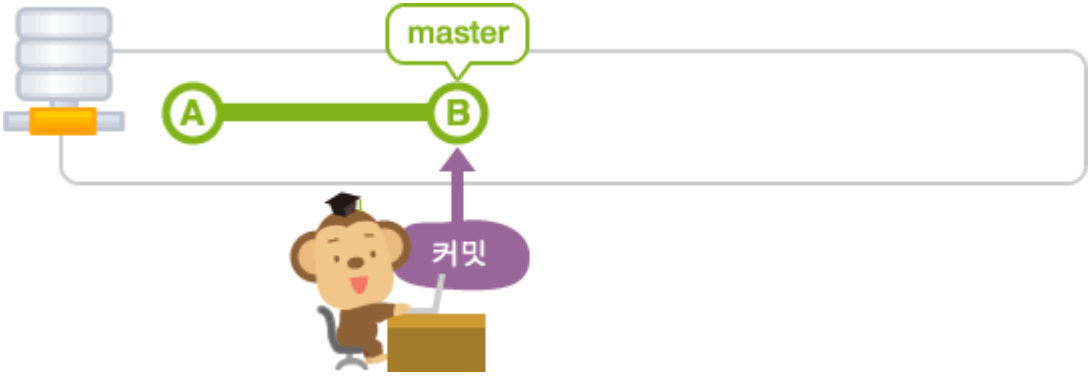
Git

- **Branch 작업**
  - Main Branch에서 자신의 작업 전용 Branch생성
  - 각자 작업 후 Main Branch 에 자신의 변경 사항 적용
  - 작업 단위로 결과를 모아서 적용, 문제 발생 시 원인을 찾아 수정 가능

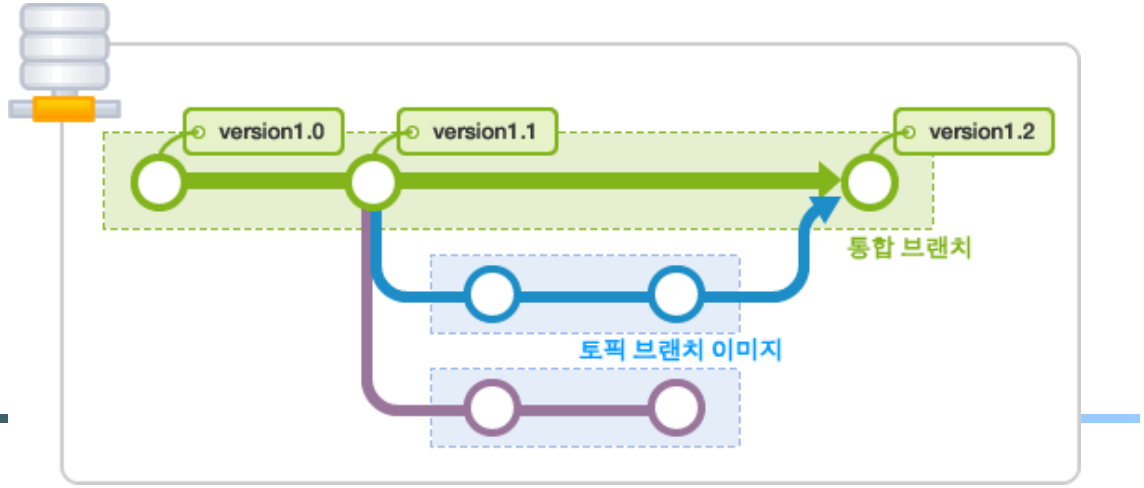


Git

- **Master**
  - 저장소를 처음 만들면, Git은 바로 'master'라는 이름의 브랜치를 생성

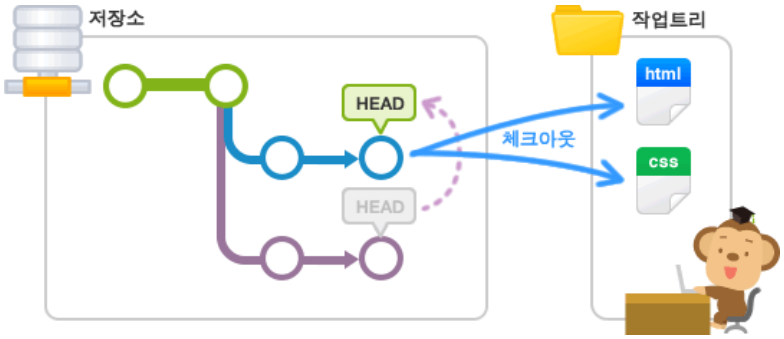


- **Branch 생성**
  - **통합 브랜치(Integration Branch)**
    - 언제든지 배포할 수 있는 버전을 만들 수 있어야 하는 브랜치
    - 늘 안정적인 상태를 유지하는 것이 중요
    - 안정적인 상태란 그 어플리케이션의 모든 기능이 정상적으로 동작하는 상태
  - **토픽 브랜치(Topic Branch)**
    - 기능 추가나 버그 수정과 같은 단위 작업을 위한 브랜치
    - 필요한 수만큼 토픽 브랜치 생성 가능
    - 토픽 브랜치는 보통 통합 브랜치로부터 만들어 내며, 토픽 브랜치에서 특정 작업이 완료되면 다시 통합 브랜치에 병합
    - 토픽 브랜치는 '피처 브랜치(Feature branch)' 라고 부르기

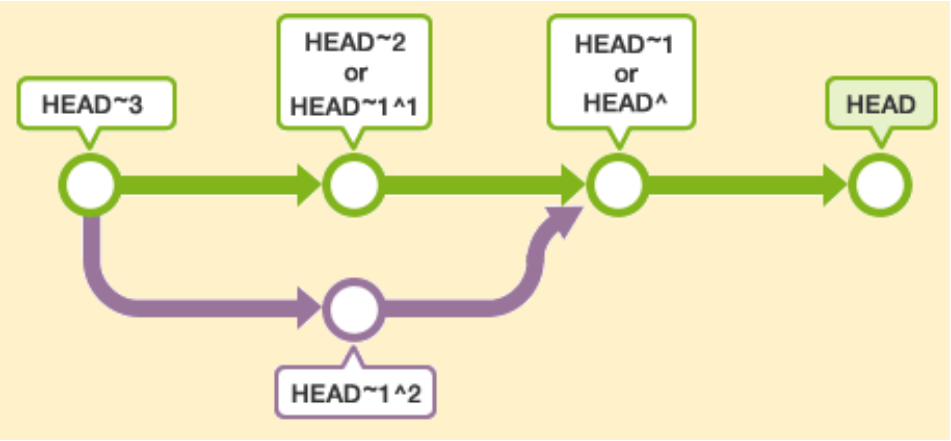


Git

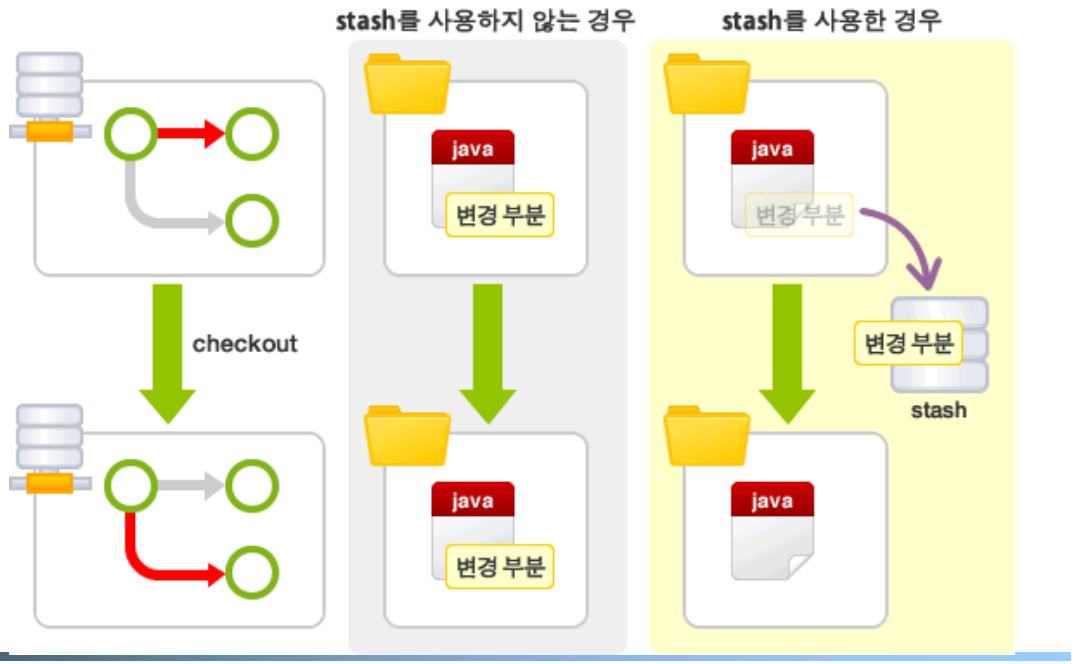
- **Branch 전환**
  - 초기 master branch가 선택
  - Checkout명령을 통해 branch 전환



- **Head**
  - 현재 사용 중인 브랜치의 선두 부분을 나타내는 이름
  - '~(틸드)'와 숫자를 'HEAD' 뒤에 붙여 앞의 커밋을 가리킬 수 있음.
  - '^(캐럿)'은, 브랜치 병합에서 원본이 여러 개가 있는 경우 몇 번째 원본인지를 지정



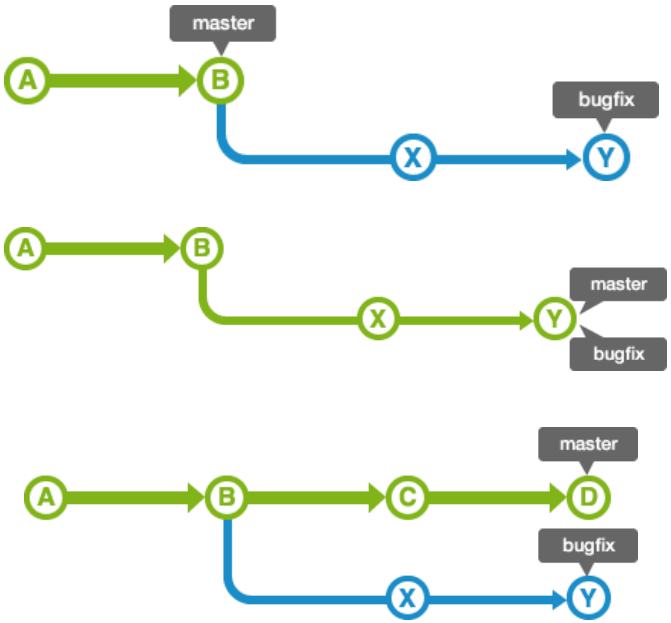
- **Stash**
  - 파일의 변경 내용을 일시적으로 기록해두는 영역
    - 커밋하지 않은 변경 내용이나 새롭게 추가한 파일이 인덱스와 작업 트리에 남아 있는 채로 다른 브랜치로 전환(checkout)하면, 그 변경 내용은 기존 브랜치가 아닌 전환된 브랜치에서 커밋할 수 있음
    - 일시적으로 변경 내용을 다른 곳에 저장하여 충돌을 피하게 한 뒤 체크아웃
  - 작업 트리와 인덱스 내에서 아직 커밋하지 않은 변경을 일시적으로 저장



Git

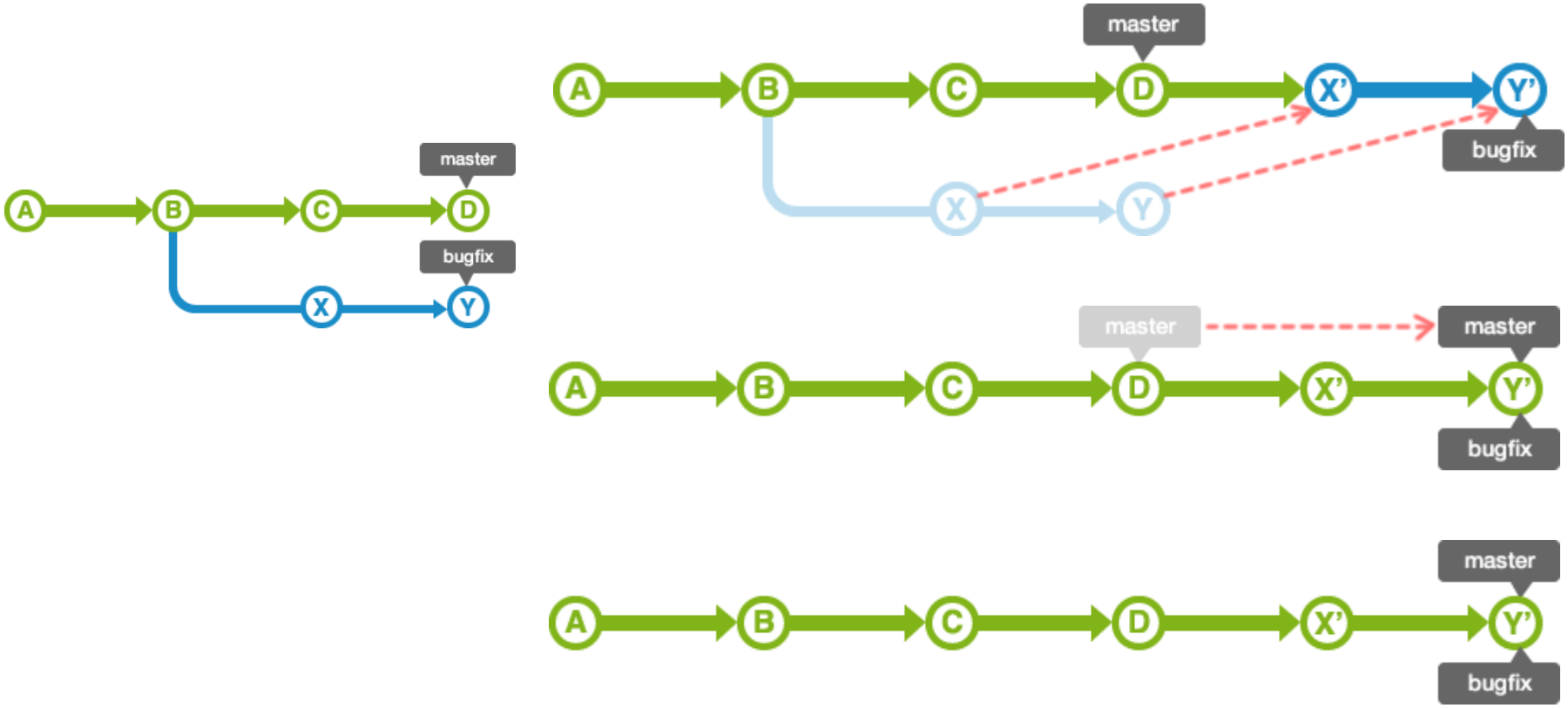
- **Branch통합**
  - 작업이 완료된 토픽 브랜치는 최종적으로 통합 브랜치에 병합
  - **Branch 통합 방법**
    - **Merge**
    - **Rebase**
  - 위의 방법이 무엇이냐에 따라 통합 후의 브랜치의 이력이 크게 달라짐

- Merge
  - 변경 내용의 이력이 모두 그대로 남아 있기 때문에 이력이 복잡해짐.



Git

- **Rebase**
  - 이력은 단순해지지만, 원래의 커밋 이력이 변경됨. 정확한 이력을 남겨야 할 필요가 있을 경우에는 사용하면 안됨.



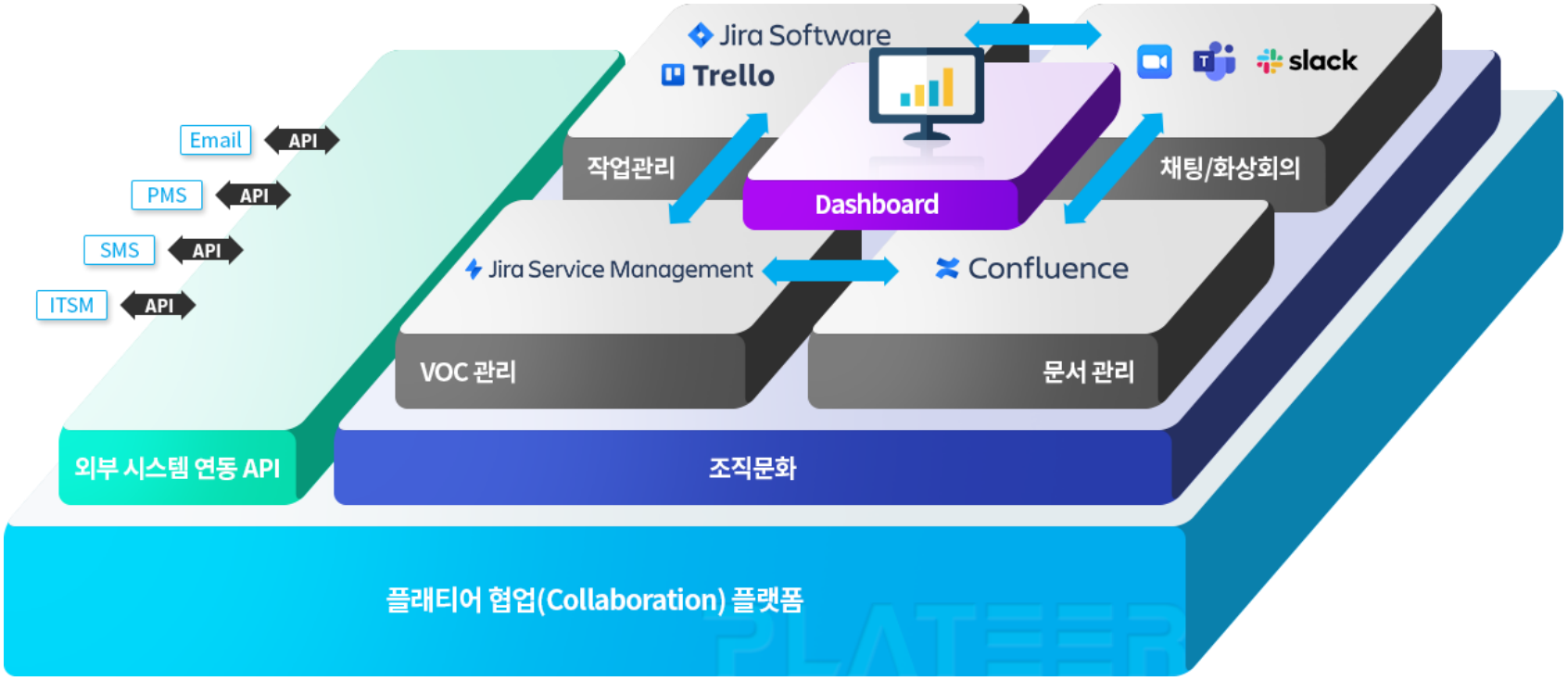
Git

- 장점
  - Repository의 완전한 복사본을 로컬에 저장할 수 있다.
  - 처리 속도가 빠르다.
  - 일시적인 작업에 대한 이력 관리가 쉽다.
  - Branch merge를 할 경우 리비전을 지정하지 않아도 되므로 편하다.(해당 branch가 언제 생겨났는지 자동적으로 파악된다.)
  - 이미 커밋한 것도 수정이 가능하다.
  - 장소에 구애 받지 않고 협업이 가능하다. 로컬에 저장이 가능하기 때문에 offline 작업이 가능하다. (웹 업계)

- 단점
  - 이전 VCS(Version Control System)과는 다른 동작 방식(개별 로컬 파일을 가질 수 있음)을 갖고 있기 때문에, 첫 사용 시 난해할 수 있다.
  - 대용량 코드 관리에 부적절하다.
  - GUI 툴이 빈약하다.
  - 한눈에 diff를 보기 어렵다.

Perforce

- **P4D**
  - 2014년 출시된 형상관리 툴
  - 코드와 바이너리 파일의 변환을 추적하기 위해 제작
  - GUI로 구성되어 다양한 플렛폼을 지원



Perfoce

- **장점**
  - 빠른 속도, 빠른 Merge
  - 히스토리 검색이 편하다.
  - P4diff가 편리하다.
  - 리비전 넘버링 인터페이스가 편리하다.
  - 큰 리소스 관리에 좋다. 바이너리 파일 처리가 매우 빠르다 (게임 업계)

- **단점**
  - 파일명이 바뀌면 히스토리 추적이 곤란하다.
  - CLI(Command Line Interface)가 상대적으로 안좋다.
  - 유료

Summary

Git 

Git-based

- GitHub 
- Gitlab 
- BitBucket 

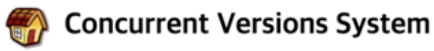
SVN 

Mercurial 

Dimensions CM 

Perforce

Bazaar 

CVS 

Reference

- https://ko.wikipedia.org/wiki/%EA%B5%AC%EC%84%B1_%EA%B4%80%EB%A6%AC
- https://smoothroutine.tistory.com/104
- https://velog.io/@dewgang/%ED%98%95%EC%83%81%EA%B4%80%EB%A6%AC%EC%97%90-%EB%8C%80%ED%95%98%EC%97%AC
- https://sujinnaljin.medium.com/software-engineering-%ED%98%95%EC%83%81-%EA%B4%80%EB%A6%AC%EC%97%90-%EB%8C%80%ED%95%98%EC%97%AC-932d14f6f341
- https://backlog.com/git-tutorial/kr/intro/intro1_1.html