[패스트캠퍼스 Day-4 과제 _ iOS스쿨 김지훈]_2016.09.09

Q. 버전관리란 무엇이고, 버전관리 방법에는 어떤 것들이 있는지 알아보기

버전 관리란?

버전 관리란 무엇이며, 왜 이것을 알아야 할까? 버전 관리 시스템은 파일의 변화를 시간에 따라 기록하여 과거 특정 시점의 버전을 다시 불러올 수 있는 시스템이다. 이 책에는 소프트웨어의 소스 코드를 버전 관리하는 예만 나오지만 실제로는 모든 컴퓨터 파일이 버전 관리의 대상이 될 수 있다.

이미지나 레이아웃을 수정할 때마다 각각의 형태를 모두 보존하고 싶은 그래픽 디자이너나 웹 디자이너라면 버전 관리 시스템(Version Control System; VCS)을 사용하는 것이 현명할 수 있다. VCS를 사용하면 개별 파일 혹은 프로젝트 전체를 이전 상태로 되돌리거나 시간에 따른 변경 사항을 검토할 수 있으며, 문제가 되는 부분을 누가 마지막으로 수정했는지, 누가 언제 이슈를 만들어냈는지 등을 알 수 있다. 또한 파일을 잃어버리거나 무언가 잘못되어도 대개 쉽게 복구할 수 있다. 그리고 이 모든 장점을 누리는 데는 큰 노력이 들지 않는다.

로컬 버전 관리 시스템

대부분의 사람들이 버전 관리를 위해 쓰는 방법은 파일을 다른 디렉토리에 복사하는 것이다(똑똑한 사람이라면 디렉토리 이름에 시간을 넣을 것이다). 이 방법은 간단하고 자주 사용되는 방법이지만 실수가 발생하기 쉽다. 어느 디렉토리에서 작업하고 있었는지 잊어버리고 엉뚱한 파일을 덮어쓰거나 의도하지 않았던 위치로 복사할 수도 있다.
이 문제를 해결하기 위해 오래전에 프로그래머들은 간단한 데이터베이스에 파일의 변경 사항을 기록하는 로컬 버전 관리 시스템을 만들었다(그림 1-1 참조).

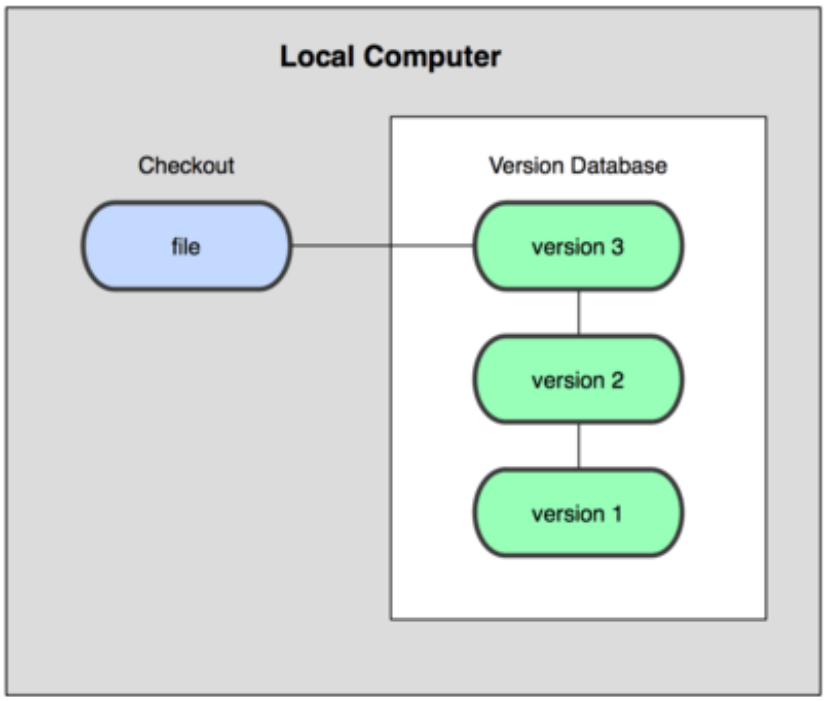

그림 1-1. 로컬 버전 관리 다이어그램

유명했던 VCS 도구들 중 현재에도 널리 쓰이는 것으로 RCS라 불리는 시스템이 있다. 그 예로 Mac OS X 운영체제에서는 개발 도구를 설치하면 RCS가 딸려온다. RCS의 기본적인 동작 방식은 각 리비전들 간의 패치 세트(patch set)라고 하는 데이터의 차이점들을 특별한 형식의 파일에 저장, 특정 시점의 파일 내용을 보고 싶을 때 해당 시점까지의 패치들을 모두 더하여 파일을 만들어내는 것이다.

중앙집중식 버전 관리 시스템

또 다른 문제는 시스템 외부에 있는 개발자들과 함께 작업하는 것이다. 중앙집중식 버전 관리 시스템(Centralized Version Control System; CVCS)은 이 문제를 해결하기 위해 개발됐다. CVS, Subversion, Perforce와 같은 시스템들이 여기에 속한다. CVCS에서는 버전 관리되는 모든 파일을 저장하는 하나의 서버와, 이 중앙 서버에서 파일들을 가져오는(checkout) 다수의 클라이언트가 존재한다. 오랫동안 사용된 이 방식은 지금까지도 버전 관리의 대표적인 방식이다(그림 1-2 참조).

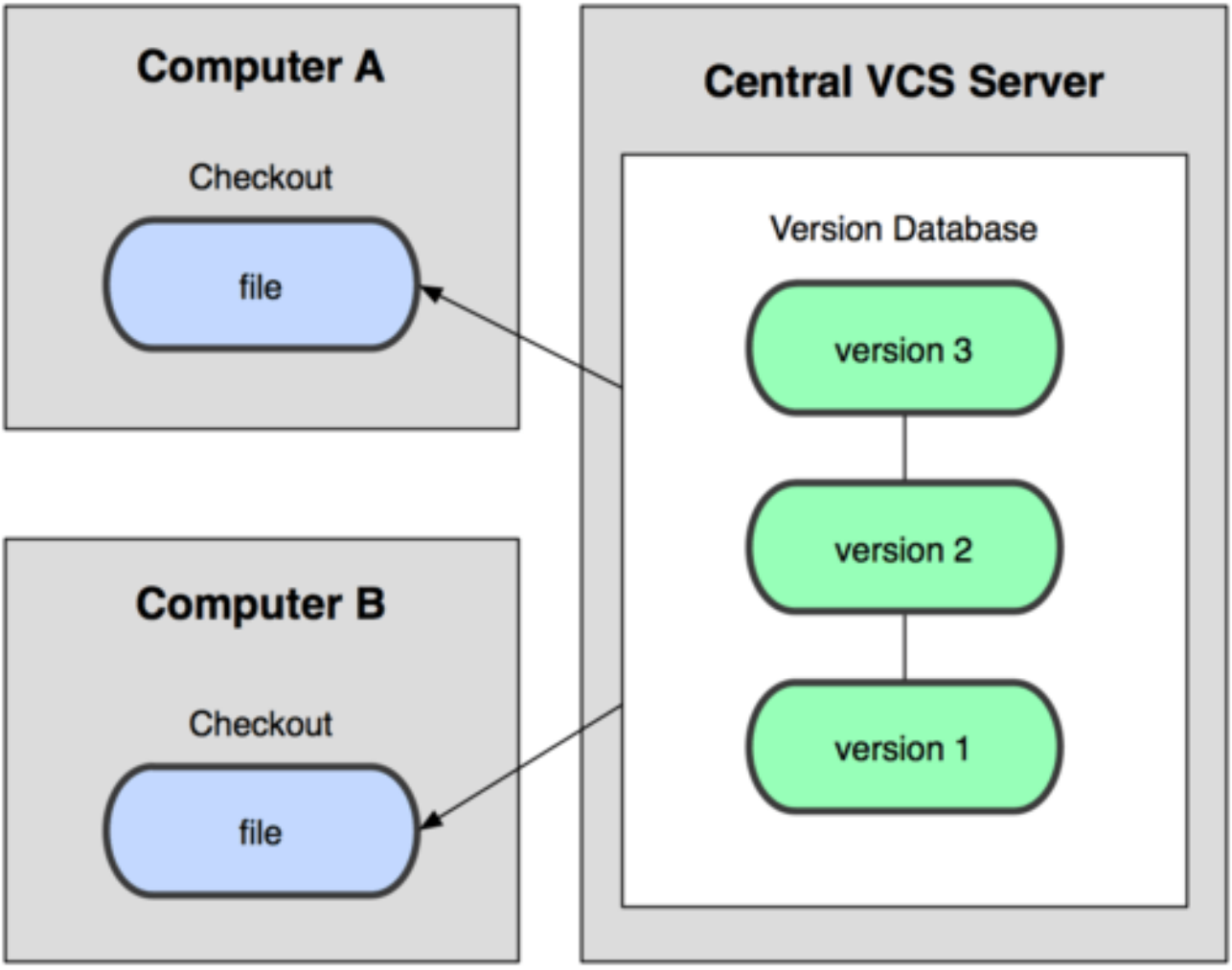

그림 1-2. 중앙집중식 버전 관리 다이어그램

CVCS는 로컬 VCS에 비해 장점이 많다. 누구나 다른 사람들이 무엇을 하고 있는지 알 수 있고, 관리자는 누가 무엇을 할 수 있는지 꼼꼼하게 관리할 수 있다. CVCS를 관리하는 것은 수많은 클라이언트의 로컬 데이터베이스를 관리하는 것보다 훨씬 쉽다.

그러나 CVCS는 심각한 단점이 있다. 중앙 서버가 잘못되면 모든 것이 잘못된다는 점이다. 서버가 다운될 경우 서버가 다시 복구될 때까지 다른 사람과의 협업도, 진행 중이던 작업을 버전 관리하는 것도

불가능해진다. 중앙 데이터베이스가 저장된 하드디스크에 오류가 발생하고 백업도 없다면, 사람들이 각자 자신의 컴퓨터에 가지고 있던 스냅샷 외에는 그동안 쌓인 프로젝트의 이력을 모두 잃게 된다. 로컬 VCS 시스템도 같은 문제가 있다. 프로젝트의 모든 이력이 한곳에만 있을 경우 이것은 피할 수 없는 문제다.

분산 버전 관리 시스템

분산 버전 관리 시스템(Distributed Version Control System; DVCS)은 앞서 말한 문제를 해결하기 위해 개발되었다. Git, Mecurial, Bazaar, Darcs 등 DVCS에서는 클라이언트가 파일들의 마지막 스냅샷을 가져오는 대신 저장소(repository)를 통째로 복제한다. 따라서 서버에 문제가 생겨도 어느 클라이언트든 복제된 저장소를 다시 서버로 복사하면 서버가 복구된다. 체크아웃(checkout)을 할 때마다 전체 백업이 일어나는 셈이다(그림 1-3 참조).

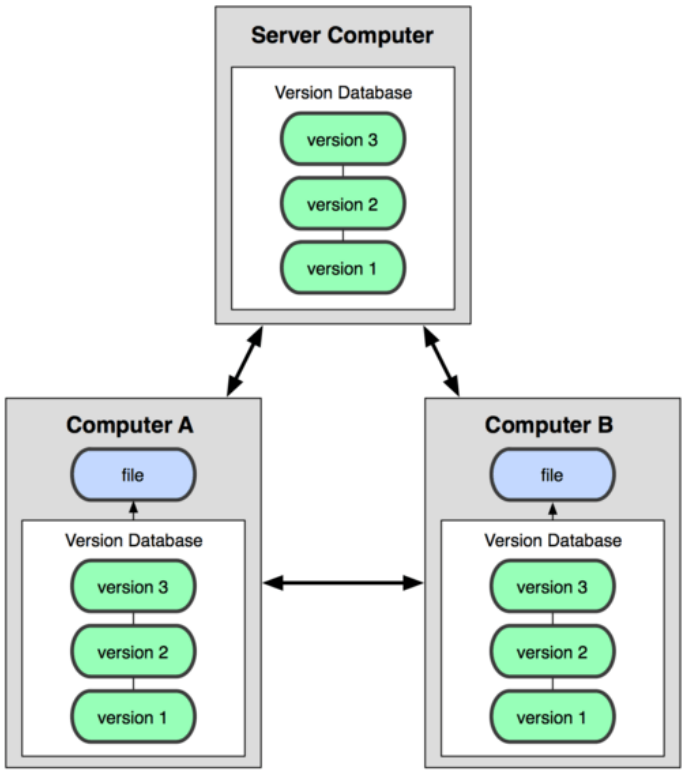

그림 1-3. 분산 버전 관리 시스템 다이어그램

게다가 대부분의 DVCS에서는 다수의 원격 저장소(remote repository)를 갖는 것이 가능하기 때문에 동시에 여러 그룹과 여러 방법으로 함께 작업할 수 있다. 이로 인해 계층 모델(hierarchical model) 등 중앙집중 시스템에서는 할 수 없는 다양한 작업 방식(workflow)들을 사용해볼 수 있다.

[번외] 형상관리와 버전관리

PM, PMO, 스프트웨어 공학(SE), 형상 관리, 버전관리, 테스트 케이스 작성과 테스트, 정적 코드 검사, Unit test..... 개발자들이 그리 탐탁치 않게 여기는 것들이 아닌가 싶습니다. 필요한 기능을 빨리 개발해 내면 되지 실질적으로는 별로 도움이 되지 않는것 같은 복잡한 절차를 왜 자꾸 만들지? 하는 의문이기도 합니다. 이러한 의문이 어떤 경우에는 설득력이 있는 경우도 있습니다. 절차라는 것이 진짜 형식일 뿐인 경우도 있으니 까요. 그런나 1인 개발 체제를 넘어서서 여러 사람이 함께하는 협업 개발 환경으로 전환되거나, 소프트웨어의 생존 기간이 길어지고 업데이트와 업그레이드 과정이 반복되다 보면, 위에서 그리 탐탁치 않게 여기던 것들이 "아, 이래서 필요한 것이었구나!"하는 공감을 하게 되곤 합니다. 그중에 대표적인 것이 형상 관리, 버전 관리 입니다.
소프트웨어 형상 관리(SCM : Software Configuration Management)는 Configuration을 Change and Configuration 이라 할 정도로 소프트웨어의 "변화"에 초점을 두고 있습니다. 몰론 SCM이 등장하게 된 배경에는 소프트웨어 품질 관리와 생산성 향상이라는 목적이 있습니다. 어떠한 소프트웨어도 개발 이후에 "변화"를 만나지 않는 것은 없습니다. 많은 사람이 사용하는 소프트웨어(SW, Software), 다양한 운영체제(OS, Operating System)에서 사용할 수 있는 SW, 오랜 기간 꾸준하게 사용되는 살아있는 SW 일수록 "변화"는 숙명과도 같은 것입니다. 이런 "변화"를 체계적으로 준비해서 관리하려는 것이 바로 SCM입니다. 개발자들이 SCM에 대하여 긍정적 시각을 갖지 못하는 배경에는 많은 경우 ISO, SPICE, CMMI와 같은 인증에 떠밀려서 실제로 실무에는 현실화되지 않은 것까지 형식적으로 자료를 제출하는 경험 때문이 아닌가 싶기도 합니다. 경영자는 어떻게 해서든 인증을 받아 제품과 기업의 활로를 찾아보려하고 개발자는 형식적인 틀에 떠밀리고 싶지 않고.....이 간격이 좁히려는 노력이 있는 소프트웨어 개발 조직이 좋은 품질의 SW를 만들어 낼것임에는 틀림없습니다.
형상 관리의 대상은 소스 코드와 다양한 프로그램 자원 뿐만아니라 SW 생애 주기(Life Cycle)에 걸친 모든 정보를 포함합니다. 가장 기본이 되는 스펙(Spcification)과 설계(Design) 자료를 비롯하여 테스트 케이스(Test case), 제품 빌드 시스템, 제품 출시후의 사용자 피드백과 버그 트래킹 정보에 이르기 까지 변화하는 모든 정보를 포함합니다. 소프트웨어 형상 관리의 한 요소가 바로 버전 관리 시스템입니다.
다양한 문서와 소스코드, 이미지등의 프로그램 자원(Resource), 트래킹 정보에 이르기 까지 모든 변화 과정을 기록 관리하여 누가 언제 무엇을 수정했는지 추적할 수 있도록 해줍니다. 여러 사용자가 원활하게 협업할 수 있는 것 또한 버전 관리 시스템 도입으로 누릴 수 있는 효과입니다. 버전 관리(Version control) 시스템은 리비전 관리(Revision control), 소스 관리(Source control)로도 불립니다. 문서 및 코드의 변화는 각각 리비전이라는 숫자에 의해 인식할 수 있으며 최초의 관리 대상은 리비전 1로 변화의 단위마다 리버전은 1씩 증가하게 됩니다. 이 리비전을 기준으로 특정 리비전으로 돌아가거나 일부를 합치는 등의 작업을 할 수 있습니다.
형상 관리 시스템과 버전 관리 시스템을 혼동하거나 비슷하게 여기는 가장 큰 이유는 버전 관리 시스템을 지원하는 도구 때문이지 않나 싶습니다. 대표적인 버전 관리 시스템으로는 오픈 소스인 CVS(Concurrent Version System)와 Subversion, 마이크로소프트의 VSS(Visual Source Safe), IBM의 Rational Clearcase 등이 있고 최근에는 Git가 많이 사용되고 있습니다. 이전에는 하나의 저장소(리포지토리, Repository)를 두는 중앙 집중식의 버전 관리 시스템을 주로 사용했으나 깃(Git), Mercurial의 경우에는 분산형 버전 관리 시스템으로 중앙 집중식 도구의 단점을 해소한 특성이 있습니다.
버전 관리 기능은 앞서 언급한 전문적인 버전 관리 시스템 뿐만아니라 워드와 같은 오피스 프로그램이나 위키와 같은 협업 문서 작성 시스템에서도 채용하고 있습니다. 아래의 그림은 위키 시스템에서 제공하는 리비전 체계를 기반으로 리비전간 위키 문서의 차이를 비교한 모습입니다.