Git 기초

Git의 핵심은 뭘까?

이 질문은 Git을 이해하는데 굉장히 중요하다. GIt이 무엇이고 어떻게 동작하는지 이해한다면 쉽게 Git을 효과적으로 사용할 수 있다. GIt을 배우려면 Subversion이나 Perforce 같은 다른 VCS를 사용했던 경험을 지워버려야 한다. Git은 미묘하게 달라서 다른 VCS에서 쓰던 개념으로는 헷갈릴 거다. 사용자 인터페이스는 매우 비슷하지만, 정보를 취급하는 방식이 다르다. 이런 차이점을 이해하면 GIt을 사용하는 것이 어렵지 않다.

델타가 아니라 스냅샷

Subversion과 Subversion 비슷한 놈들과 Git의 가장 큰 차이점은 데이터를 다루는 방법에 있다. 큰 틀에서 봤을 때 대부분의 VCS시스템이 관리하는 정보는 파일들의 목록이다. CVS, Subversion, Perforce, Bazaar등의 시스템은 파일의 집합으로 정보를 관리한다. 각 파일의 변화를 시간순으로 관리한다.



Git은 이런식으로 데이터를 저장하지도 취급하지도 않는다. 대신 GIt의 데이터는 파일 시스템의 스냅샷이라 할 수 있으며 크기가 아주 작다. Git은 커밋하거나 프로젝트의 상태를 저장할 떄마다 파일이 존재하는 그 순간을 중요하게 여긴다. 파일이 달라지지 않았으면 **GIt은 성능을 위해서 파일을 저장하지 않는다. 단지 이전 상태의 파일에 대한 링크만 저장한다.**



이것이 Git이 다른 VCS와 구분되는 점이다. 이점 때문에 Git은 다른 시스템들이 과거로부터 답습해왔던 버전 컨트롤의 개념과 다르다는 것이고 많은 부분을 새로운 관점에서 바라본다. Git은 강력한 도구를 지원하는 작은 파일 시스템이다. Git은 단순한 VCS가 아니다. 이제 3장에서 설명할 Git 브랜치를 사용하면 얻게 되는 이득이 무엇인지 설명한다.

거의 모든 명령을 로컬에서 실행

**거의 모든 명령이 로컬 파일과 데이터만 사용하기 때문에 네트워크에 있는 다른 컴퓨터는 필요 없다.** 대부분의 명령어가 네트워크의 속도에 영향을 받는 CVCS에 익숙하다면 Git이 매우 놀라울 것이다. Git의 이런 특징에서 나오는 미칠듯한 속도는 오직 Git느님만이 구사할 수 있는 초인적인 능력이다. **프로젝트의 모든 히스토리가 로컬 디스크에 있기 때문에 모든 명령을 순식간에 실행된다.**

예를 들어 Git은 프로젝트의 히스토리를 조회할 때 서버없이 조회한다. 그냥 로컬 데이터베이스에서 히스토리를 읽어서 보여준다. 그래서 눈 깜짝할 사이에 히스토리를 조회할 수 있다. 어떤 파일의 현재 버전과 한 달 전의 상태를 비교해보고 싶을 때도 Git은 그냥 한 달 전의 파일과 지금파일을 로컬에서 찾는다. 파일을 비교하기 위해 리모트에 있는 서버에 접근하고 나서 예전 버전을 가져올 필요가 없다.

즉 오프라인 상태에서도 비교할 수 있다. 비행기나 기차 등에서 작업하고 네트워크에 접속하고 있지 않아도 커밋할 수 있다. 다른 VCS 시스템에서는 불가능한 일이다. Perforce는 서버에 연결할 수 없을 때 할 수 있는 일이 별로 없다. Subversion이나 CVS에서도 마찬가지다. 데이터베이스에 접근할 수 없어서 파일을 편집할 수는 있지만, 커밋할 수 없다. 매우 사소해 보이지만 실제로 이 상황에 부닥쳐보면 느껴지는 차이가 매우 크다.

Git의 무결성

**Git은 모든 데이터를 저장하기 전에 체크섬(또는 해시)을 구하고 체크섬으로 데이터를 관리한다.**

체크섬 없이 어떤 파일이나 디렉토리도 변경할 수 없다. 체크섬은 Git에서 사용하는 가장 기본적인 (Atomic) 데이터 단위이자 Git의 기본 철학이다. Git없이 체크섬을 다룰 수 없어서 파일의 상태도 알 수 없고 심지어 데이터를 잃어버릴 수도 없다.

GIt은 SHA-1해시를 사용하여 체크섬을 만든다. 만든 체크섬은 40자 길이의 16진수 문자열이다. 파일의 내용이나 디렉토리 구조를 이용하여 체크섬을 구한다. SHA-1은 아래처럼 생겼다.

24b9da6552252987aa493b52f8696cd6d3b00373

Git은 모든 것을 해시로 식별하기 때문에 이런 값은 여기저기서 보인다. **실제로 Git은 파일을 이름으로 저장하지 않고 해당 파일의 해시로 저장한다.**

Git은 데이터를 추가할 뿐

**Git으로 무얼 하든 데이터를 추가한다, 되돌리거나 데이터를 삭제할 방법이 없다.** 다른 VCS처럼 Git도 커밋하지 않으면 변경사항을 잃어버릴 수 있다. 하지만, 일단 스냅샨을 커밋하고 나면 데이터를 잃어버리기 어렵다.

GIt을 사용하면 프로젝트가 심각하게 망가질 걱정 없이 매우 즐겁게 여러 가지 실험을 해 볼 수 있다. 9장을 보면 Git이 데이터를 어떻게 저장하고 손실을 복구해야 할지 알 수 있다.

세 가지 상태

이 부분은 중요하기에 집중해서 읽어야 한다. Git을 공부하기 위해 반드시 짚고 넘어가야 할 부분이다**. Git은 파일을 Committed, Modified, Staged 이렇게 세 가지 상태로 관리한다**. **Committed란 데이터가 로컬 데이터 베이스에 안전하게 저장됐다는 것을 의미한다. Modified는 수정한 파일을 아직 로컬 데이터베이스에 커밋하지 않은 것을 말한다. Staged란 현재 수정한 파일을 곧 커밋할 것이라고 표시한 상태를 의미한다.**

이 세 가지 상태는 Git 프로젝트의 세 가지 단계와 연결돼 있다. Git 디렉토리, 워킹 디렉토리, Staging Area 이렇게 세 가지 단계를 이해하고 넘어가자.



Git 디렉토리는 Git이 프로젝트의 메타데이터와 객체 데이터베이스를 저장하는 곳을 말한다. **Git 디렉토리가 Git의 핵심이다. 다른 컴퓨터에 있는 저장소를 Clone 할 때 Git 디렉토리가 만들어진다.**

**워킹 디렉토리는 프로젝트의 특정 버전을 Checkout 한 것이다**. Git 디렉토리는 지금 작업하는 디스크에 있고 그 디렉토리에 압축된 데이터베이스에서 파일을 가져와서 워킹 디렉토리를 만든다.

**Staging Area는 Git 디렉토리에 있다. 단순한 파일이고 곧 커밋할 파일에 대한 정보를 저장한다. 종종 인덱스라고 불리기도 하지만, Staging Area라는 명칭이 표준이 되어가고 있다.**

**Git으로 하는 일은 기본적으로 아래와 같다.**

* **워킹 디렉토리에서 파일을 수정한다.**
* **Staging Area에 파일을 Stage 해서 커밋할 스냅샷을 만든다.**
* **Staging Area에 있는 파일들을 커밋해서 Git 디렉토리에 영구적인 스냅샷으로 저장한다.**

Git 디렉토리에 있는 파일들은 Commited 상태이다. 파일을 수정하고 Staging Area에 추가했다면 Staged이다. 그리고 Checkout하고 나서 수정했지만, 아직 Staging Area에 추가하지 않으면 Modifed이다. 2장에서 이 상태에 대해 좀 더 자세히 배운다. 특히 Staging Area를 어떻게 이용하는지 혹은 아예 생략하는 방법도 설명한다.