1-1 버전관리 시스템의 이해

해당 챕터의 내용은 https://git-scm.com/ 공식 사이트 출처 입니다.

1.1 시작하기 - 버전 관리에 관하여

이 장에서는 Git을 시작하는 방법에 대해 설명합니다.

버전 관리에 관하여

"버전 관리"란 무엇이며 왜 신경 써야 할까요?

버전 제어: 시간이 지남에 따라 파일 또는 파일 집합에 대한 변경 사항을 기록하여 나중에 특정 버전을 불러올 수 있도록 하는 시스템

- 버전 관리 시스템을 사용하면
  - 선택한 파일을 이전 상태로 되돌리고,
  - 전체 프로젝트를 이전 상태로 되돌리고,
  - 시간 경과에 따른 변경 사항을 비교하고,
  - 문제를 일으킬 수 있는 항목을 누가 마지막으로 수정했는지, 누가 언제 문제를 발생시켰는지 등을 확인 가능.
  - 작업을 망치거나 파일을 잃어버려도 쉽게 복구 가능.

로컬 버전 제어 시스템

파일에 대한 모든 변경 사항을 리비전 제어하에 유지하는 간단한 데이터베이스가 있는 로컬 VCS.

가장 인기 있는 VCS 도구 중 하나는 RCS라는 시스템.

RCS는 패치 세트(즉, 파일 간의 차이점)를 디스크에 특수 형식으로 보관하는 방식으로 작동하며, 모든 패치를 합산하여 특정 시점의 파일 모양을 다시 만들 수 있습니다.

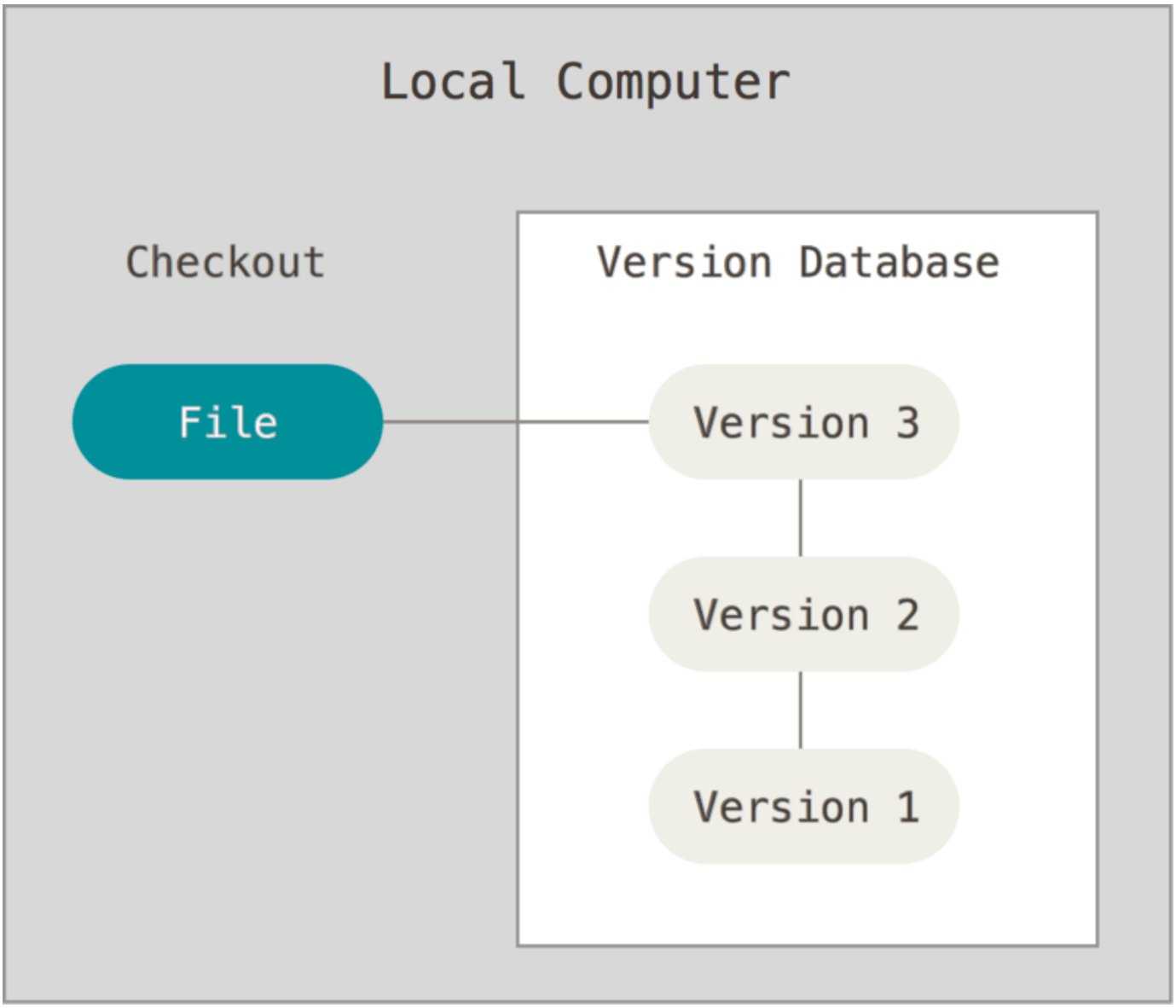

그림 1. 로컬 버전 제어

중앙 집중식 버전 제어 시스템

다음으로 사람들이 직면하는 주요 문제는 다른 시스템의 개발자와 협업해야 한다는 것입니다.

이 문제를 해결하기 위해 중앙 집중식 버전 제어 시스템(CVCS)이 개발되었습니다.

이러한 시스템(예: CVS, Subversion, Perforce)에는 버전이 관리되는 모든 파일이 포함된 단일 서버와 이 중앙 서버에서 파일을 체크아웃하는 여러 클라이언트가 존재.

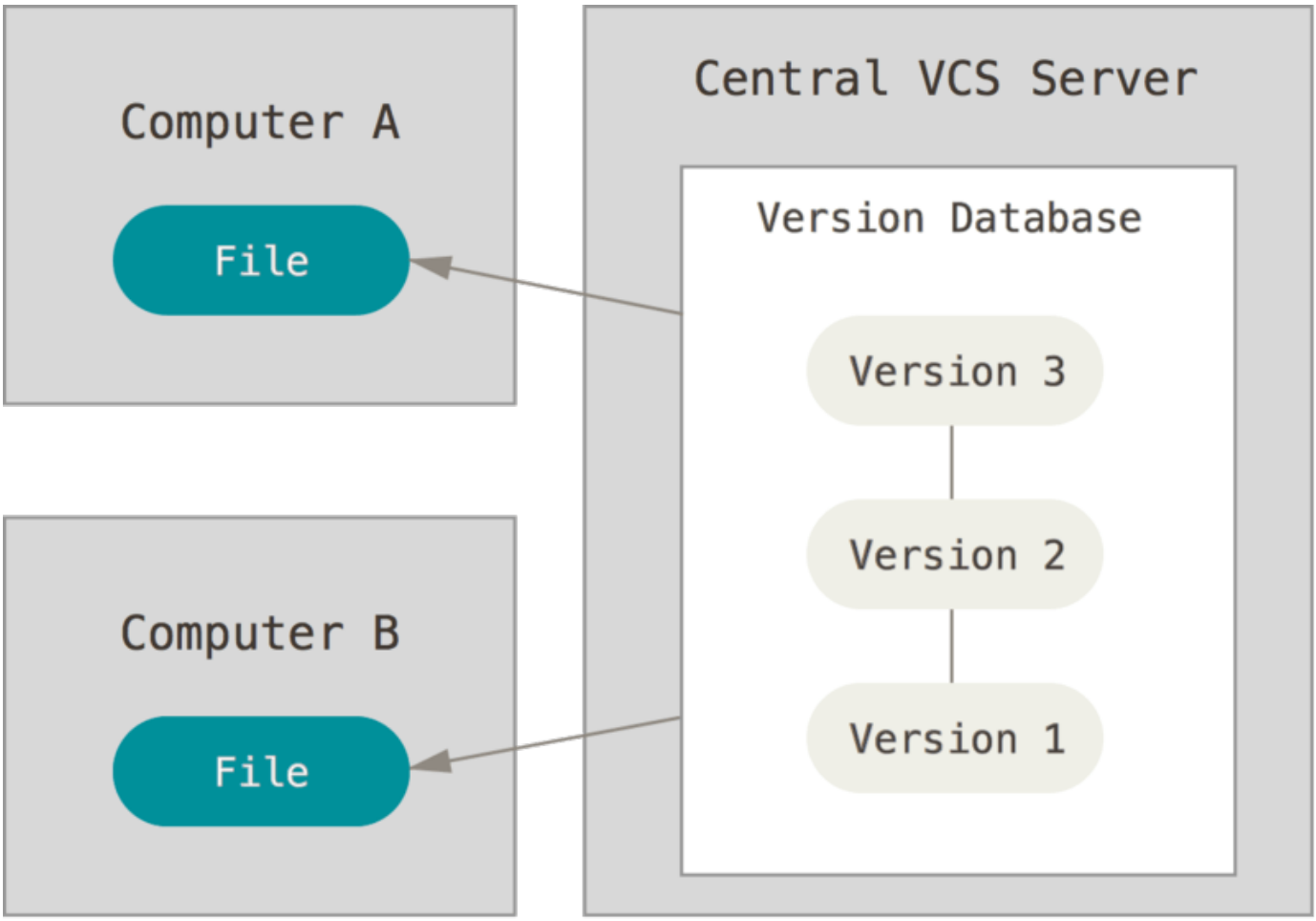

그림 2. 중앙 집중식 버전 관리
이 설정은 특히 로컬 VCS에 비해 많은 이점을 제공합니다.

- 모든 사람이 프로젝트의 다른 모든 사람이 무엇을 하는지 어느 정도 알 수 있습니다.
- 관리자는 누가 무엇을 할 수 있는지 세밀하게 제어할 수 있으며, 모든 클라이언트의 로컬 데이터베이스를 관리하는 것보다 CVCS를 관리하는 것이 훨씬 쉽습니다.

하지만 이 설정에는 몇 가지 심각한 단점도 있습니다.

- 중앙 집중식 서버가 나타내는 단일 장애 지점입니다. 해당 서버가 한 시간 동안 다운되면 그 시간 동안 아무도 공동 작업을 하거나 작업 중인 문서의 버전 변경 내용을 저장할 수 없습니다. 중앙 데이터베이스가 있는 하드 디스크가 손상되고 적절한 백업이 유지되지 않으면 로컬 컴퓨터에 있는 단일 스냅샷을 제외한 프로젝트의 모든 기록을 완전히 잃게 됩니다.
- 로컬 VCS도 이와 동일한 문제를 겪습니다. 프로젝트의 전체 히스토리를 한 곳에 모아두면 모든 것을 잃을 위험이 있습니다.

분산 버전 제어 시스템

분산 버전 제어 시스템(DVCS)이 필요한 이유입니다. DVCS(예: Git, Mercurial, Bazaar 또는 Darcs)에서 클라이언트는

- 파일의 최신 스냅샷만 체크아웃하는 것이 아니라 전체 히스토리를 포함하여 리포지토리를 완전히 미러링합니다.
- 서버가 죽고 해당 서버를 통해 협업하던 시스템이 중단되는 경우, 클라이언트 리포지토리를 서버에 다시 복사하여 복원가능.
- 모든 복제본은 실제로 모든 데이터의 전체 백업.

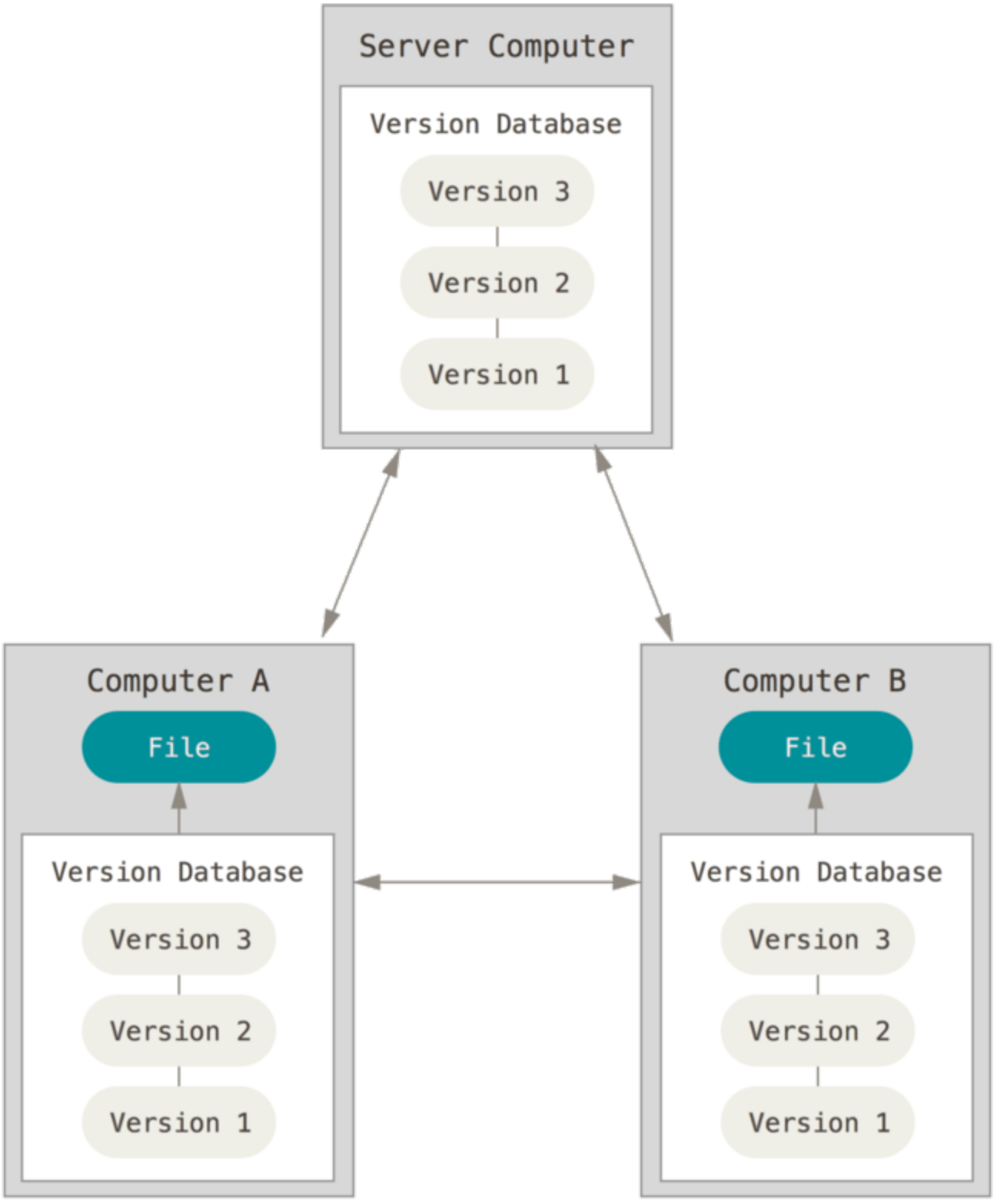

그림 3. 분산 버전 제어
또한 이러한 시스템 중 상당수는

- 작업할 수 있는 원격 리포지토리를 여러 개 보유하는 데 매우 능숙하므로 동일한 프로젝트 내에서 서로 다른 방식으로 여러 그룹과 동시에 공동 작업할 수 있습니다.
- 이를 통해 계층적 모델과 같이 중앙 집중식 시스템에서는 불가능한 여러 유형의 워크플로를 설정할 수 있습니다.

짧게 보는 Git의 역사

Linux 커널은 상당히 큰 범위의 오픈 소스 소프트웨어 프로젝트입니다. Linux 커널 유지 관리 초기(1991~2002년)에는 소프트웨어에 대한 변경 사항이 패치와 아카이브 파일로 전달되었습니다. 2002년에 Linux 커널 프로젝트는 BitKeeper라는 독점적인 DVCS를 사용하기 시작했습니다.

2005년에 Linux 커널을 개발한 커뮤니티와 BitKeeper를 개발한 상업 회사 간의 관계가 깨지면서 도구의 무료 지위가 취소되었습니다. 이로 인해 리눅스 개발 커뮤니티(특히 리눅스 창시자인 리누스 토발즈)는 비트키퍼를 사용하면서 얻은 교훈을 바탕으로 자체 도구를 개발하게 되었습니다.

1.3 시작하기 - Git이란 무엇인가요?

What is Git?

Git이란 간단히 말해 무엇일까요?

- Git이 무엇이고 어떻게 작동하는지 기본 사항을 이해
- Git의 사용자 인터페이스는 다른 VCS 와 상당히 유사하지만, Git은 정보를 저장하고 생각하는 방식이 매우 다르므로 이러한 차이점을 이해하면 사용 중에 혼란을 피하는 데 도움이 됩니다.

Snapshots, Not Differences

Git과 다른 VCS(서브버전 및 프렌즈 포함)의 가장 큰 차이점은 Git이 데이터를 생각하는 방식이다. 개념적으로 대부분의 다른 시스템은 정보를 파일 기반 변경 목록으로 저장합니다. 이러한 다른 시스템(CVS, Subversion, Perforce, Bazaar 등)은 저장하는 정보를 파일 집합과 시간이 지남에 따라 각 파일에 적용된 변경 사항으로 생각합니다(이를 일반적으로 델타 기반 버전 제어라고 함).

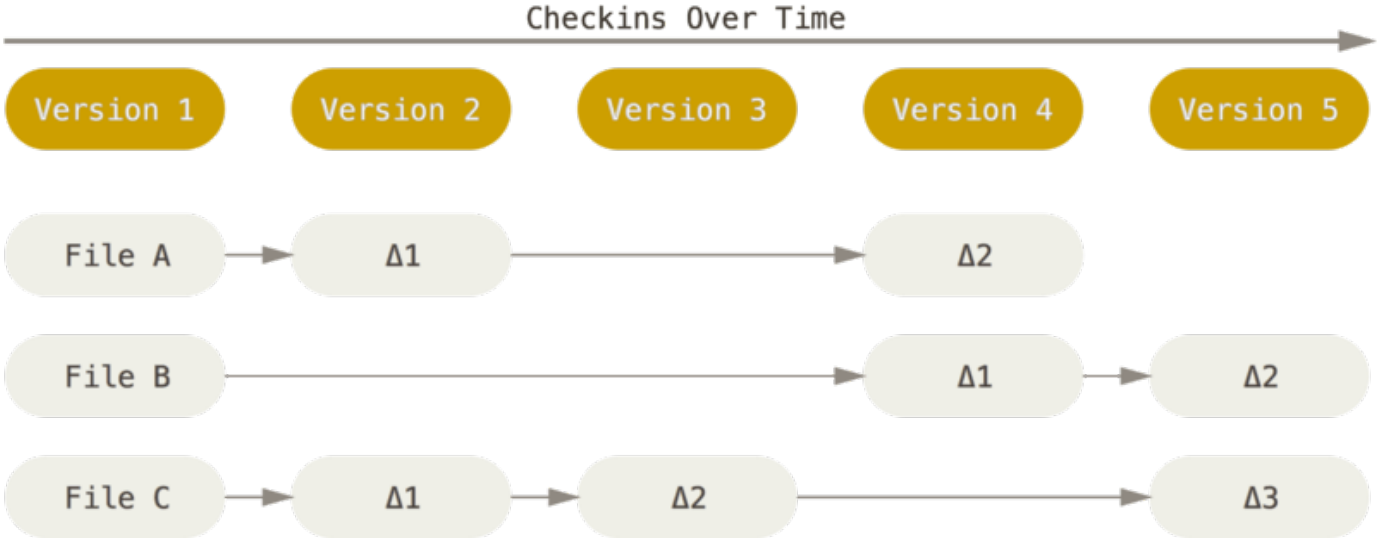

그림 4. 데이터를 각 파일의 기본 버전에 대한 변경 사항으로 저장하기

Git은 데이터를

- Git을 사용하면 커밋하거나 프로젝트의 상태를 저장할 때마다 기본적으로 모든 파일이 그 순간에 어떻게 보이는지 사진을 찍고 해당 스냅샷에 대한 참조를 저장.
- 효율을 높이기 위해 파일이 변경되지 않은 경우 Git은 파일을 다시 저장하지 않음, 대신 이전에 저장한 동일한 파일에 대한 링크만 저장함.
- 결론: Git은 데이터를 스냅샷의 스트림처럼 생각합니다.

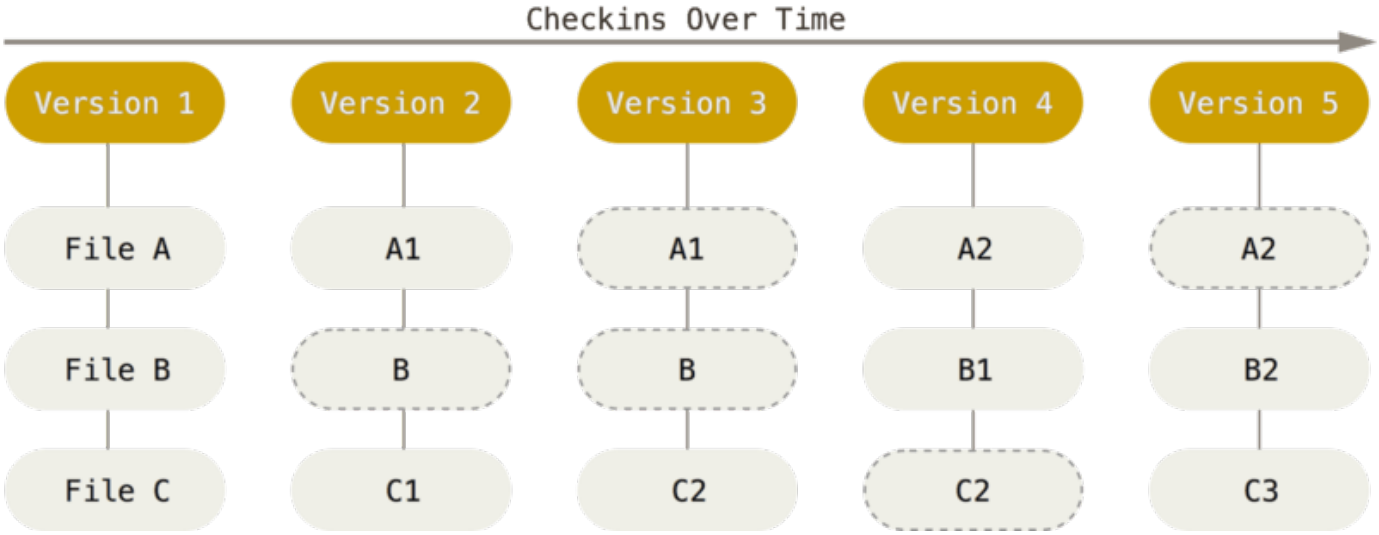

그림 5. 시간이 지남에 따라 프로젝트의 스냅샷으로 데이터 저장하기

이는 Git과 거의 모든 다른 VCS 간의 중요한 차이점입니다.

거의 모든 작업이 로컬에서 이루어집니다.

- Git의 대부분의 작업은 로컬 파일과 리소스만 있으면 작동하며, 일반적으로 네트워크에 있는 다른 컴퓨터의 정보는 필요하지 않습니다.
- 프로젝트의 전체 히스토리를 로컬 디스크에 바로 저장할 수 있기 때문에 대부분의 작업이 거의 즉각적으로 이루어집니다.

예를 들어, 프로젝트의 히스토리를 찾아보기 위해 서버로 나가서 히스토리를 가져와서 표시할 필요 없이 로컬 데이터베이스에서 바로 읽어오기만 하면 됩니다. 즉, 거의 즉시 프로젝트 기록을 볼 수 있습니다. 현재 버전의 파일과 한 달 전의 파일 사이에 발생한 변경 사항을 확인하려면 원격 서버에 요청하거나 원격 서버에서 이전 버전의 파일을 가져와 로컬에서 계산할 필요 없이 Git에서 한 달 전의 파일을 조회하여 로컬 차이 계산을 수행할 수 있습니다.

이는 또한 오프라인 상태이거나 VPN을 사용하지 않는 경우 할 수 없는 일이 거의 없다는 것을 의미합니다. 비행기나 기차를 타고 이동하면서 간단한 작업을 하고 싶을 때, 네트워크에 연결될 때까지 로컬 사본에 커밋하고 업로드할 수 있습니다. 집에 돌아가서 VPN 클라이언트가 제대로 작동하지 않더라도 계속 작업할 수 있습니다.

무결성을 갖춘 Git

Git의 모든 항목은 저장되기 전에 체크섬을 거친 다음 해당 체크섬으로 참조됩니다. 즉, Git이 모르는 상태에서 파일이나 디렉터리의 내용을 변경하는 것은 불가능합니다. 이 기능은 가장 낮은 수준에서 Git에 내장되어 있으며 Git의 철학에 필수적인 요소입니다. 전송 중에 정보가 손실되거나 파일이 손상되어도 Git이 이를 감지할 수 없습니다.

Git이 이 체크섬을 위해 사용하는 메커니즘을 SHA-1 해시라고 한다. 이는 16진수 문자(0-9 및 a-f)로 구성된 40자 문자열로, Git의 파일 또는 디렉토리 구조의 내용을 기반으로 계산됩니다. SHA-1 해시는 다음과 같이 생겼습니다:

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

이 해시값은 Git에서 매우 많이 사용되기 때문에 곳곳에서 볼 수 있습니다. 실제로 Git은 데이터베이스에 파일 이름이 아닌 콘텐츠의 해시값으로 모든 것을 저장합니다.

일반적으로 데이터만 추가하는 Git

Git에서 작업을 수행하면 거의 모든 작업이 Git 데이터베이스에 데이터만 추가합니다. 시스템에서 되돌릴 수 없는 작업을 수행하거나 어떤 식으로든 데이터를 지우도록 하는 것은 어렵습니다. 다른 VCS와 마찬가지로 아직 커밋하지 않은 변경 사항을 잃거나 엉망으로 만들 수 있지만, 스냅샷을 Git에 커밋한 후에는 특히 정기적으로 데이터베이스를 다른 리포지토리로 푸시하는 경우 손실하기가 매우 어렵습니다.

따라서 작업을 심각하게 망칠 위험 없이 실험할 수 있는 장점이 있습니다. Git이 데이터를 저장하는 방법과 손실된 데이터를 복구하는 방법에 대해 자세히 알아보려면 Undoing Things 를 참조하세요.

세 가지 상태

나머지 학습 과정이 순조롭게 진행되기를 원한다면 Git에 대해 기억해야 할 주요 사항이 있습니다. Git에는 수정된 상태, 스테이징된 상태, 커밋된 상태의 세 가지 주요 파일 상태가 있습니다:

- Modified 상태는 파일을 변경했지만 아직 데이터베이스에 커밋하지 않았음을 의미합니다.
- Staged 상태는 수정된 파일을 현재 버전에서 다음 커밋 스냅샷으로 이동하도록 표시했음을 의미합니다.
- Committed 상태는 데이터가 로컬 데이터베이스에 안전하게 저장되었음을 의미합니다.

이로써 Git 프로젝트의 세 가지 주요 섹션인 워킹 디렉터리, 스테이징 영역, Git 디렉터리를 살펴볼 수 있습니다.

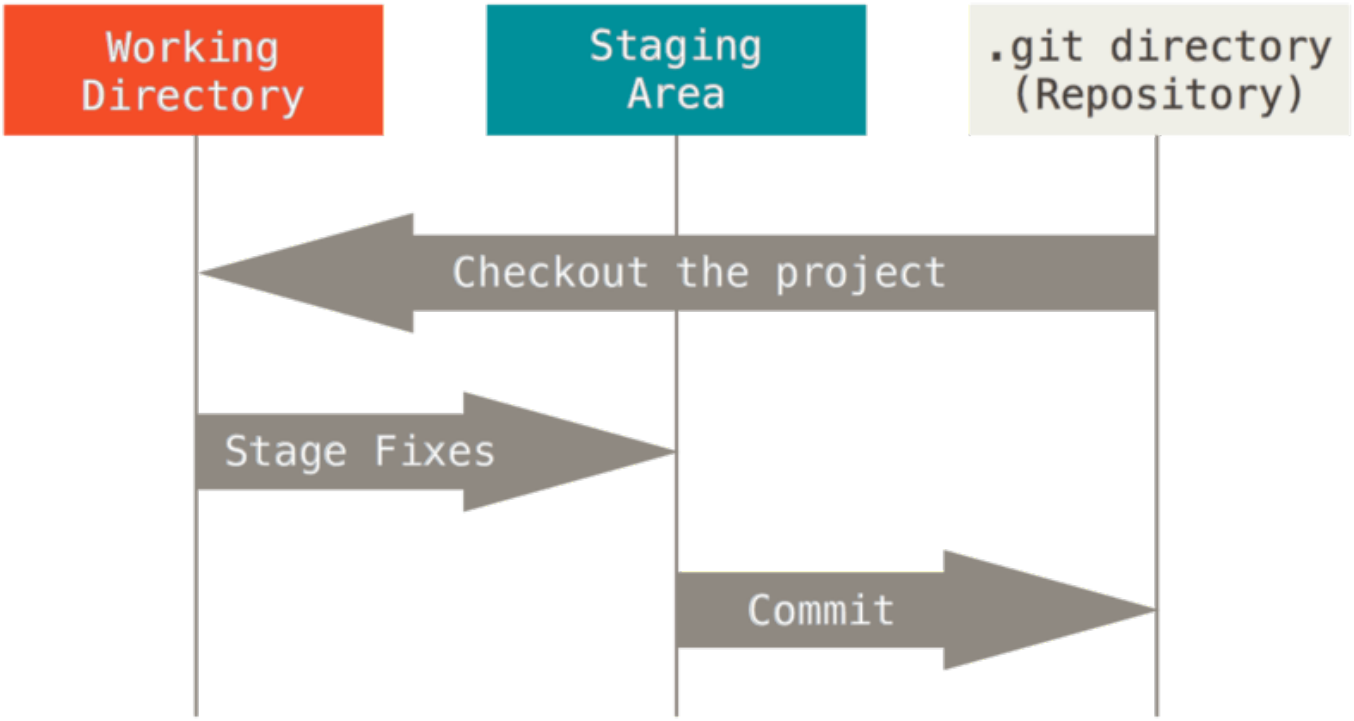

그림 6. 워킹 디렉터리, 스테이징 영역 및 Git 디렉터리

- 워킹 디렉터리
  - 프로젝트의 한 버전에 대한 단일 체크아웃입니다.
  - Git 디렉터리의 압축된 데이터베이스에서 꺼내어 디스크에 배치하여 사용하거나 수정할 수 있습니다.
- 스테이징 영역
  - 일반적으로 Git 디렉터리에 포함된 파일로, 다음 커밋에 들어갈 내용에 대한 정보를 저장합니다.
  - Git 전문 용어로는 "인덱스"라고 하지만 "스테이징 영역"이라는 표현도 잘 어울립니다.
- Git 디렉터리
  - Git이 프로젝트의 메타데이터와 오브젝트 데이터베이스를 저장하는 곳입니다.
  - 이것은 Git에서 가장 중요한 부분이며, 다른 컴퓨터에서 리포지토리를 복제할 때 복사되는 부분입니다.

기본 Git 워크플로우는 다음과 같습니다:

1. 작업 트리에서 파일을 수정합니다.
2. 다음 커밋에 포함할 변경 사항만 선택적으로 스테이징하면 스테이징 영역에 해당 변경 사항만 추가됩니다.
3. 커밋을 수행하면 스테이징 영역에 있는 파일을 그대로 가져와서 해당 스냅샷을 Git 디렉터리에 영구적으로 저장합니다.

파일의 특정 버전이 Git 디렉터리에 있으면 커밋된 것으로 간주합니다. 파일이 수정되어 스테이징 영역에 추가되었다면 스테이징된 것입니다. 그리고 체크아웃된 이후 변경되었지만 스테이징되지 않은 경우 수정된 것으로 간주합니다. Git 기초에서는 이러한 상태와 이러한 상태를 활용하거나 스테이징 부분을 완전히 건너뛰는 방법에 대해 자세히 알아보도록 합니다.