3-1. 브랜치란?

모든 버전 관리 시스템에는 **'branch'**라는 개념이 있다. 일반적으로 **나무가 가지에서 새 줄기를 뻗듯이 여러 갈래로 퍼지는 데이터 흐름**을 가리키는 말로 사용한다.

**\* Branch가 필요한 이유**

우리가 어떤 제품의 사용 설명서를 만든다고 상상해 보자. 사용 설명서의 버전 관리는 깃으로 한다. 제품 출시 전에는 아래처럼 개발 순서에 따라 사용 설명서를 작성하면 된다.

**A -> B -> C -> D**

하지만 제품이 출시되고 나면 문제가 생긴다. **고객사마다 추가로 요구하는 내용이 다르기 때문**이다. **요구 사항을 반영하다 보면 고객사에 따라 제품이 달라질 것이고, 이에 맞춰 사용설명서도 달라져야 한다.** 어떻게 해결해야 할까?

먼저 떠오르는 해결책은 **처음 작업했던 저장소 전체(아래 그림에서 master)를 여러 개 복사해서 각 고객사의 이름(아래 그림에서ms, apple, google)을 붙인 다음 저장소마다 버전 관리를 따로 하는 방법**이다.

**ms: A -> B -> C -> D -> ME**

**google: A -> B -> C -> D -> GE -> GF -> GH**

**apple: A -> B -> C -> D -> AE**

**master: A -> B -> C -> D -> E**

하지만 이 방법은 효율적이지 않다. 고객사마다 디렉터리를 복사하면 **출시 전까지 만들었던 내용은 동일하기 때문에 자료가 중복**된다. 또 버전 관리 시스템의 장점 중 하나는 파일 이름을 더럽히지 않는 것인데, 이 방법은 **고객사마다 디렉터리 이름을 다르게 사용**해야 한다.

마지막으로 **한 가지 중요한 문제**가 더 있다. google에서 작업을 마친 후 그 내용이 apple에서도 필요한 내용이라고 가정해 보자. 단순하게 google에 있는 최신 상태의 코드를 복사해서 apple 디렉터리에 붙여 넣은 다음, apple 디렉터리에서 새로운 버전을 commit 하면 될까?

**ms: A -> B -> C -> D -> ME**

**google: A -> B -> C -> D -> GE -> GF -> GG**

**apple: A -> B -> C -> D -> AE -> AF**

**master: A -> B -> C -> D -> E**

이러면 문제가 발생할 수 있다. ‘GG’ 버전에서 필요한 부분의 코드만 복사해 붙여 넣으려 해도 **‘GE’와 ‘GF’ 버전을 만들 때 수정한 다른 내용이 ‘AE’ 버전에는 반영되어 있지 않으므로 오류가 날 수 있다.** 그렇다고 ‘GG’ 버전의 코드 전체를 그대로 덮어 버리면 ‘D’버전에서 ‘AE’버전을 만들 때 **수정한 내용이 의도치 않게 바뀌거나 사라질 것이다.**

이런 문제를 해결하기 위한 도구가 바로 **‘branch’**이다.

**\* Branch 기능 살펴보기**

(그림이 필요하다면 책을 참고하자.)

**깃으로 버전 관리를 시작하면 기본적으로 master라는 branch가 만들어진다**. 사용자는 commit할 때마다 master branch는 최신 commit을 가리킨다. 즉, **branch는 commit을 가리키는 포인터와 비슷하다**고 생각하면 된다.

**새 branch를 만들면 기존에 저장할 파일을 master branch에 그대로 유지하면서 기존 파일 내용을 수정하거나 새로운 기능을 구현할 파일을 만들 수 있다.** 이렇게 **master branch에서 뻗어 나오는 새 branch를 만드는 것**을 **‘branch 한다’**고 한다.

새 branch에서 원하는 작업을 다 끝냈다면 **새 branch에 있던 파일을 원래 master branch에 합칠 수 있다**. 이렇게 **분기했던 branch를 master branch에 합치는 것**을 ‘**merge 한다’**고 한다.