브랜치 (Branch)

브랜치란?

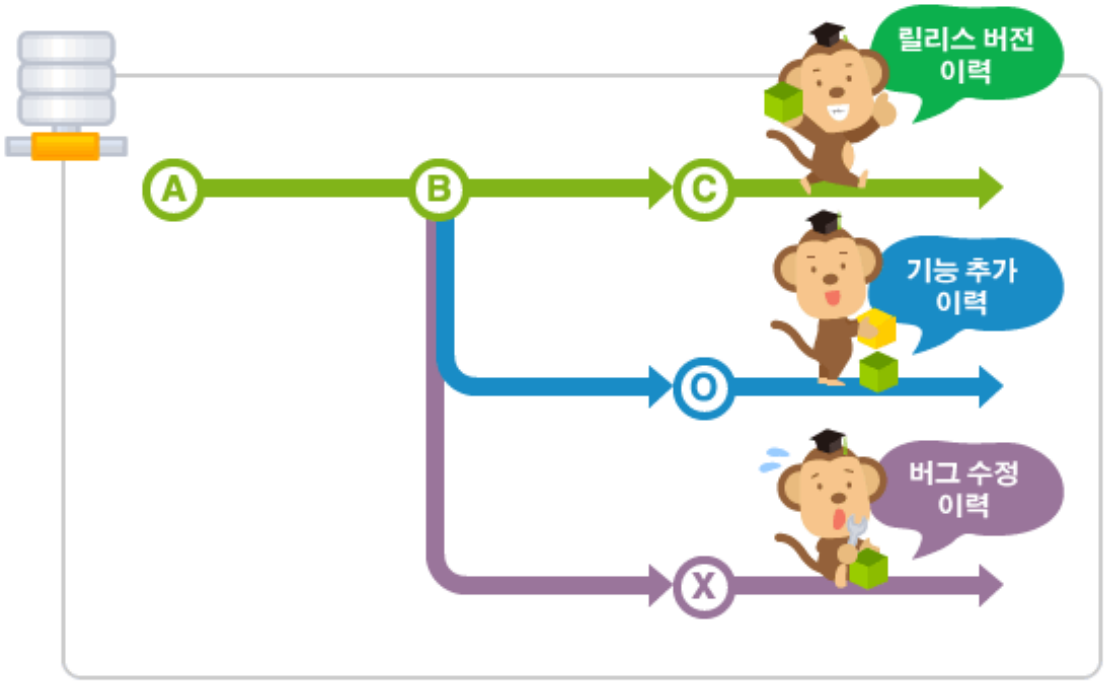
지금까지 Git의 기본적인 사용법에 대해 알아 보았습니다. 발전 편에서는 브랜치의 사용법에 대해 좀 더 자세히 알아보도록 하겠습니다.

소프트웨어를 개발할 때에 개발자들은 동일한 소스코드를 함께 공유하고 다루게 됩니다. 동일한 소스코드 위에서 어떤 개발자는 버그를 수정하기도 하고 또 다른 개발자는 새로운 기능을 만들어 내기도 하죠. 이와 같이 여러 사람이 동일한 소스코드를 기반으로 서로 다른 작업을 할 때에는 각각 서로 다른 버전의 코드가 만들어 질 수 밖에 없습니다.

이럴 때, 여러 개발자들이 동시에 다양한 작업을 할 수 있게 만들어 주는 기능이 바로 '브랜치(Branch)' 입니다. 각자 독립적인 작업 영역(저장소) 안에서 마음대로 소스코드를 변경할 수 있지요. 이렇게 분리된 작업 영역에서 변경된 내용은 나중에 원래의 버전과 비교해서 하나의 새로운 버전으로 만들어 낼 수 있습니다.

브랜치(branch)란?

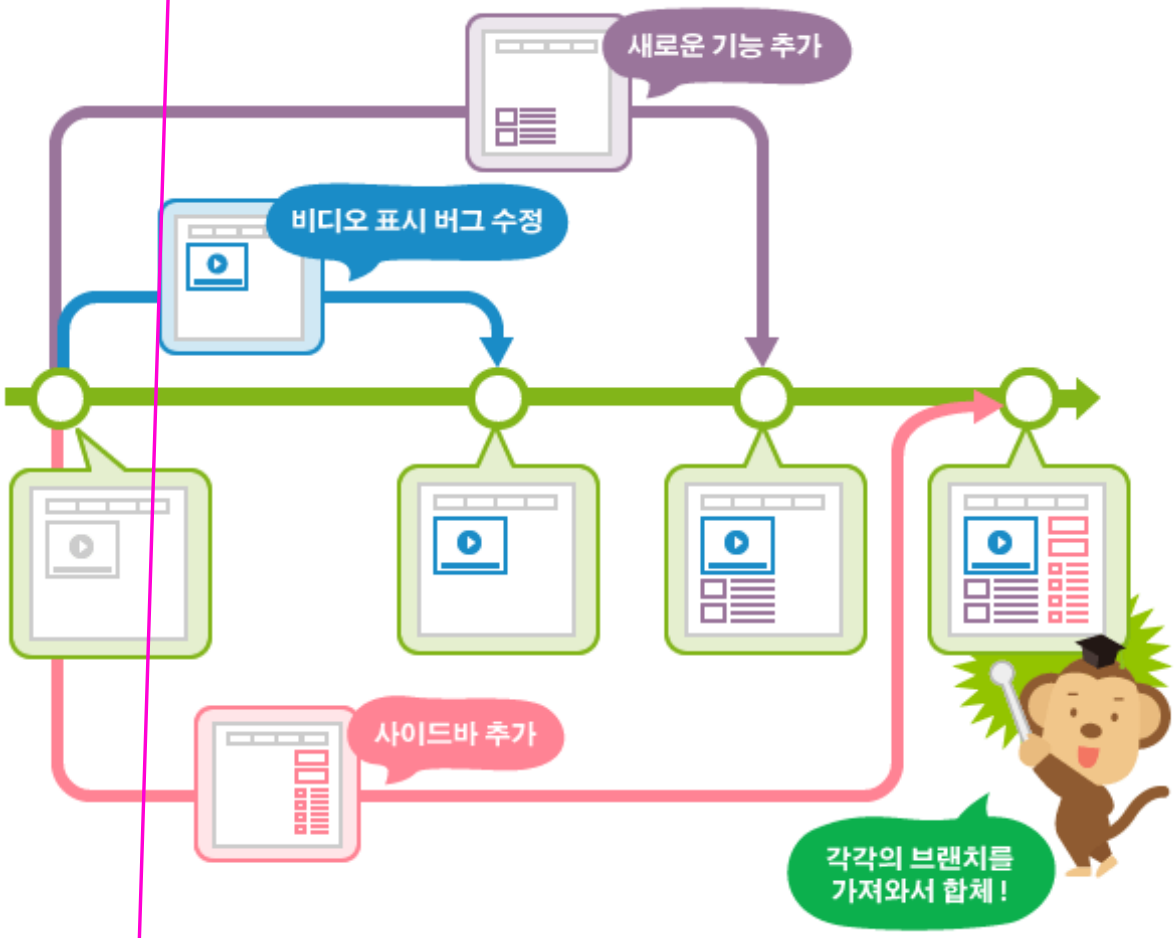
브랜치란 독립적으로 어떤 작업을 진행하기 위한 개념입니다. 필요에 의해 만들어지는 각각의 브랜치는 다른 브랜치의 영향을 받지 않기 때문에, 여러 작업을 동시에 진행할 수 있습니다.



또한 이렇게 만들어진 브랜치는 다른 브랜치와 병합(Merge)함으로써, 작업한 내용을 다시 새로운 하나의 브랜치로 모을 수 있습니다.

아래 그림을 보면, 브랜치를 사용하여 동시에 여러 작업을 진행할 때의 작업 흐름을 한눈에 파악할 수 있습니다.

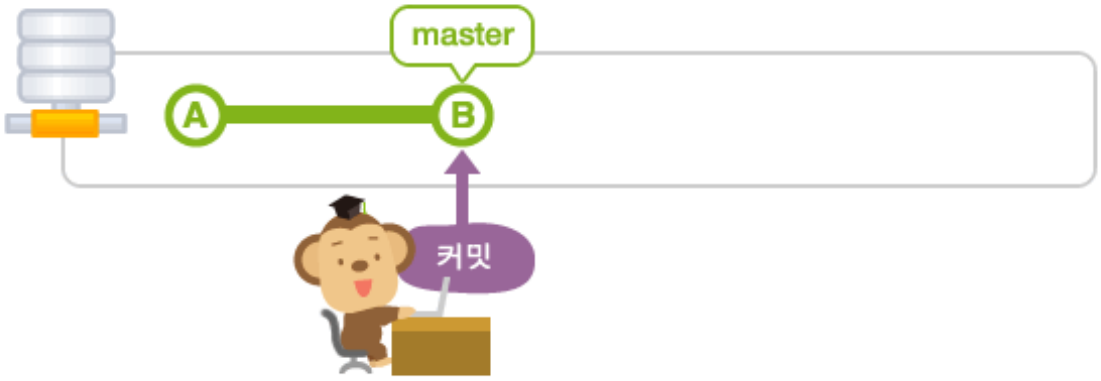
여러 명이서 동시에 작업을 할 때에 다른 사람의 작업에 영향을 주거나 받지 않도록, 먼저 메인 브랜치에서 자신의 작업 전용 브랜치를 만듭니다. 그리고 각자 작업을 진행한 후, 작업이 끝난 사람은 "메인 브랜치"에 자신의 브랜치의 변경 사항을 적용합니다. 이렇게 함으로써 다른 사람의 작업에 영향을 받지 않고 독립적으로 특정 작업을 수행하고 그 결과를 하나로 모아 나가게 됩니다. 이러한 방식으로 작업할 경우 '작업 단위', 즉 브랜치로 그 작업의 기록을 중간 중간에 남기게 되므로 문제가 발생했을 경우 원인이 되는 작업을 찾아내거나 그에 따른 대책을 세우기 쉬워집니다.



master 브랜치

저장소를 처음 만들면, Git은 바로 'master'라는 이름의 브랜치를 만들어 둡니다. 이 새로운 저장소에 새로운 파일을 추가 한다거나 추가한 파일의 내용을 변경하여 그 내용을 저장(커밋, Commit)하는 것은 모두 'master' 라는 이름의 브랜치를 통해 처리할 수 있는 일이 됩니다.

'master'가 아닌 또 다른 새로운 브랜치를 만들어서 '이제부터 이 브랜치를 사용할거야!'라고 선언(체크아웃, checkout)하지 않는 이상, 이 때의 모든 작업은 'master' 브랜치에서 이루어 집니다.



2. 새 브런치 생성하기

`git branch` 명령으로 새로운 브랜치를 만들 수 있습니다.
기존에 작업하고 있던 `master` 브랜치에서 `testing` 이라는 브랜치를 만들어보겠습니다.

```
$ git branch testing
```

위 명령을 실행하면 새로 만든 브랜치도 지금 작업하고 있던 마지막 커밋을 가리킵니다.

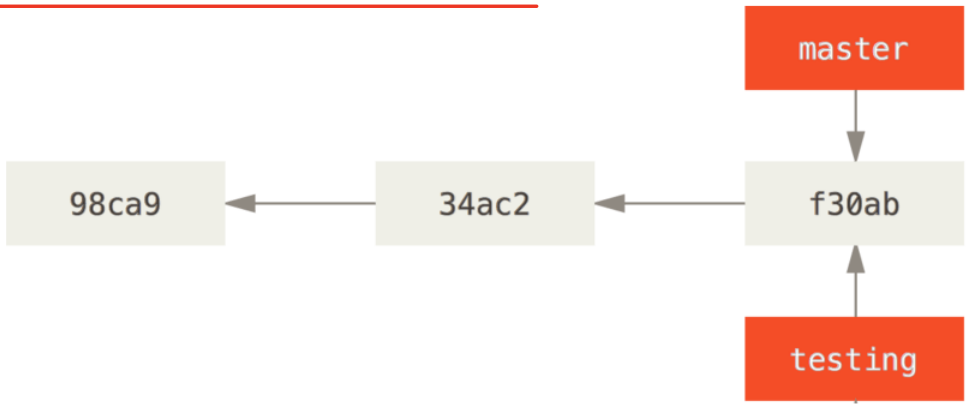


그림 1. 한 커밋 히스토리를 가리키는 두 브랜치

지금 작업 중인 브랜치가 무엇인지 Git은 어떻게 파악할까요?

다른 버전 관리 시스템과는 달리 Git은 `HEAD` 라는 특수한 포인터가 있습니다. 이 포인터는 지금 작업하는 로컬 브랜치를 가리킵니다. 브랜치를 새로 만들었지만, Git은 아직 `master` 브랜치를 가리키고 있습니다. 이렇듯 `git branch` 명령은 브랜치를 만들기만 하고 HEAD가 가리키는 브랜치를 옮기지 않습니다.

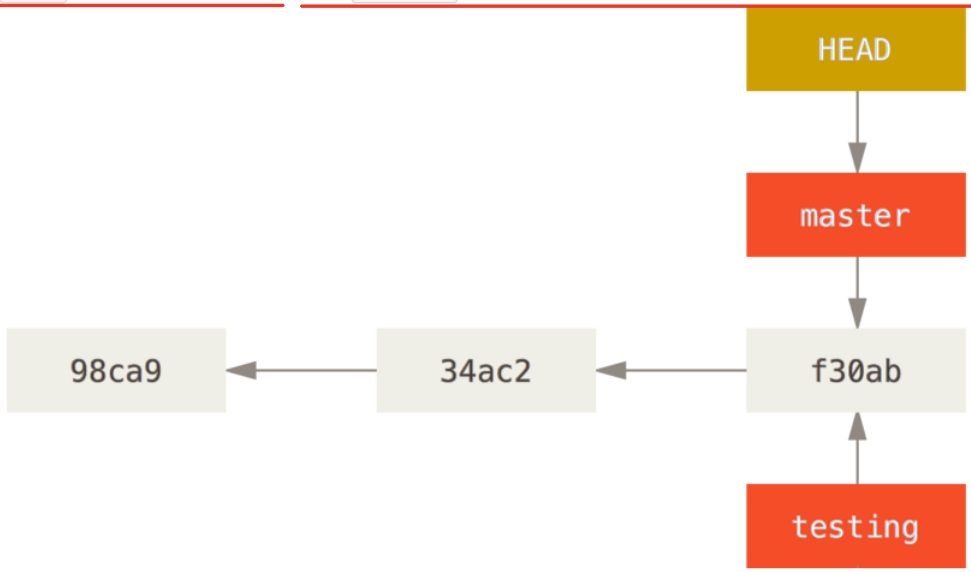


그림 2. 현재 작업 중인 브랜치를 가리키는 HEAD

3. 브랜치 이동하기

`git branch` 명령은 브랜치를 생성할 뿐 HEAD 가 가리키는 브랜치를 옮기지는 않는다고 했습니다.
그렇다면 브랜치는 어떻게 이동할 수 있을까요?
`git checkout` 명령으로 다른 브랜치로 이동할 수 있습니다. 한번 `testing` 브랜치로 바꿔보겠습니다.

```
$ git checkout testing
```

이렇게 하면 HEAD는 testing 브랜치를 가리킵니다.

그림 3. HEAD가 testing 브랜치를 가리킴

자, 이제 브랜치의 핵심을 살펴볼 수 있습니다.
커밋을 새롭게 한 번 해보겠습니다.

```
$ vim test.rb
$ git commit -a -m 'made a change'
```

그림 4. HEAD가 가리키는 testing 브랜치가 새 커밋을 가리킴

새로 커밋해서 `testing` 브랜치는 앞으로 이동했습니다. 하지만, `master` 브랜치는 여전히 이전 커밋을 가리킵니다. 그렇다면 이번엔 `master` 브랜치로 되돌아가보겠습니다.

```
$ git checkout master
```

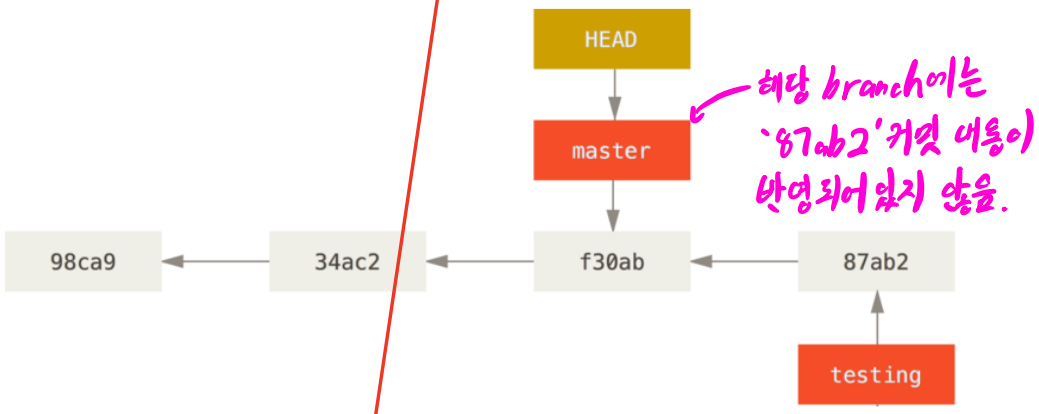


그림 5. HEAD가 checkout 한 브랜치로 이동함

방금 실행한 명령이 한 일은 두 가지입니다.
`master` 브랜치가 가리키는 커밋을 HEAD가 가리키게 하고, 워킹 디렉토리의 파일도 그 시점으로 되돌려 놓았습니다. 앞으로 커밋을 하면 다른 브랜치의 작업들과 별개로 진행되기 때문에 `testing` 브랜치에서 임시로 작업하고 원래 `master` 브랜치로 돌아와서 하던 일을 계속할 수 있습니다.

브랜치를 이동하면 워킹 디렉토리의 파일이 변경됩니다.
이전에 작업했던 브랜치로 이동하면 워킹 디렉토리의 파일은 그 브랜치에서 가장 마지막으로 했던 작업 내용으로 변경됩니다. 파일 변경
를 이동시키는게 불가능한 경우 Git은 브랜치 이동 명령을 수행하지 않습니다.
이번에는 `master` 브랜치에서 파일을 수정하고 커밋을 해보겠습니다.

```
$ vim test.rb
$ git commit -a -m 'made other changes'
```

이렇게 되면 프로젝트 히스토리는 분리되어 진행합니다. *(브랜치가 갈라집니다.)*
두 작업 내용은 서로 독립적으로 각 브랜치에 존재합니다. 커밋 사이를 자유롭게 이동하다가 때가 되면 두 브랜치를 Merge 하면 됩니다

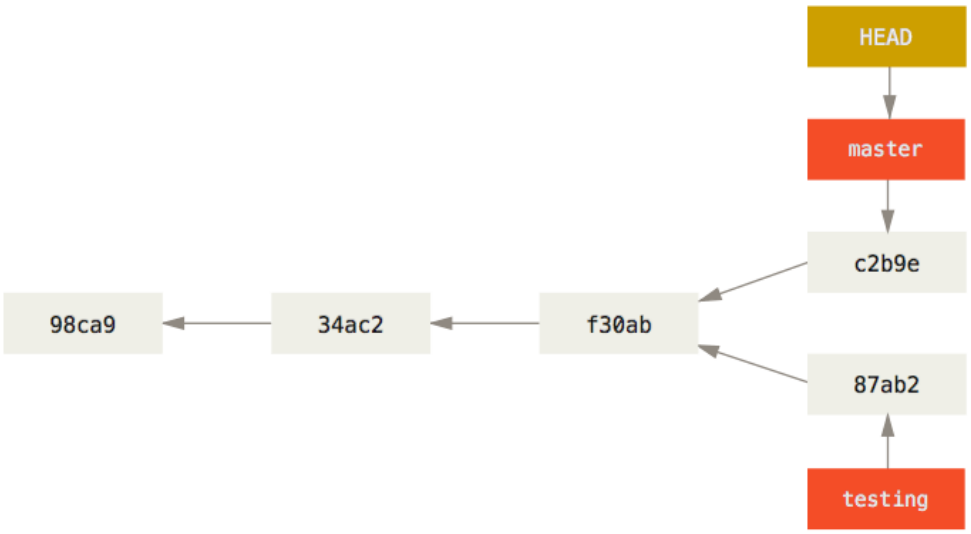


그림 6. 갈라지는 브랜치

Git의 엄청난 장점!!

실제로 Git의 브랜치는 어떤 한 커밋을 가리키는 40글자의 SHA-1 체크섬 파일에 불과하기 때문에 만들기도 쉽고 지우기도 쉽습니다. **새로 브랜치를 하나 만드는 것은 41바이트 크기의 파일(40자와 줄 바꿈 문자) 하나 만드는 것에 불과합니다.**

브랜치가 필요할 때 프로젝트를 통째로 복사해야 하는 다른 버전 관리 도구와 Git의 차이는 극명합니다. 통째로 복사하는 작업은 프로젝트 크기에 따라 다르겠지만 수십 초에서 수십 분까지 걸립니다. 그에 비해 Git은 순식간입니다.
게다가 커밋을 할 때마다 이전 커밋의 정보를 저장하기 때문에 Merge 할 때 어디서부터(Merge Base) 합쳐야 하는지 압니다. 이런 특징은 개발자들이 수시로 브랜치를 만들어 사용하게 합니다.