- 2대의 git를 사용하는 서버가 있음

서버 1

- work.txt에서 apple이라는 branch를 만들어서 work.txt파일을 수정함

서버 2

- work.txt에서 google이라는 branch를 만들어서 work.txt파일을 수정함
- 이런 상황에서 서버 2에서 아래 명령어로 pull 시도

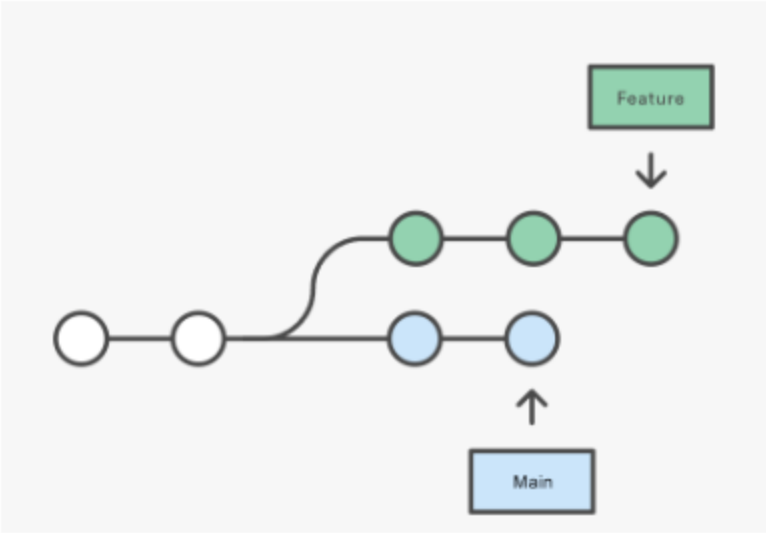
```
gen2@vm3:~/work$ git pull origin apple
remote: 오브젝트 나열하는 중: 5, 완료.
remote: 오브젝트 개수 세는 중: 100% (5/5), 완료.
remote: 오브젝트 압축하는 중: 100% (2/2), 완료.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
오브젝트 묶음 푸는 중: 100% (3/3), 255 바이트 | 255.00 KiB/s, 완료.
ssh://192.168.56.21/home/git/repos2 URL에서
 * branch            apple      -> FETCH_HEAD
 * [새로운 브랜치]   apple      -> origin/apple
힌트: You have divergent branches and need to specify how to reconcile them.
힌트: You can do so by running one of the following commands sometime before
힌트: your next pull:
힌트:
힌트:   git config pull.rebase false  # merge (the default strategy)
힌트:   git config pull.rebase true   # rebase
힌트:   git config pull.ff only       # fast-forward only
힌트:
힌트: You can replace "git config" with "git config --global" to set a default
힌트: preference for all repositories. You can also pass --rebase, --no-rebase,
힌트: or --ff-only on the command line to override the configured default per
힌트: invocation.
fatal: Need to specify how to reconcile divergent branches.
```

- 이런 상황에서 위에서 힌트에 대한 내용을 정리
- **git pull 시, 위와 같은 에러가 발생할 수 있습니다.**
- **해당 에러는, pull 방식을 명시적으로 정하지 않았기 때문에 발생하는 에러입니다.**
- **pull 방식으로는 merge, rebase, fast-forward 방법이 있으며,**
- **해당 포스트에선 각각의 방법에 대해서 다뤄보겠습니다.**

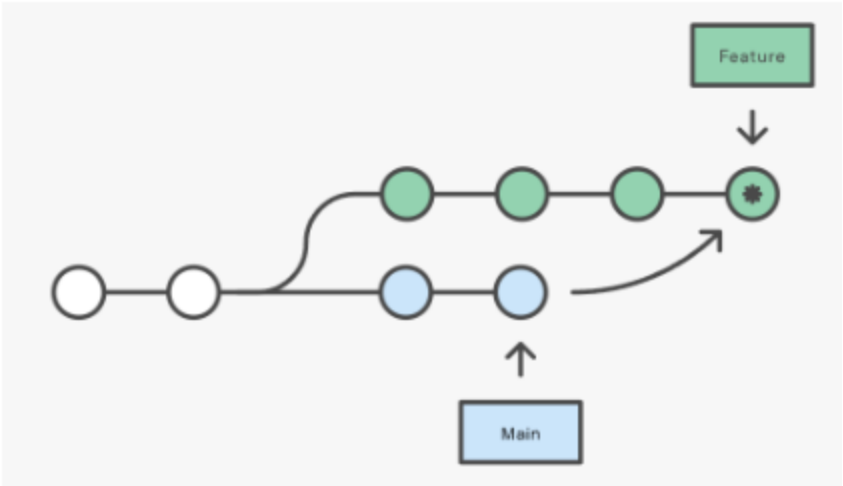
문제 해결

merge (default)

- Local Feature에서 작업을 하는 도중, 다른 누군가가 Remote Main에 새로운 commit을 만들었다고 가정해봅시다.



- 이때 만약, Local Feature에서 Remote Main을 pull한다면,
- **merge 방식의 경우, Local Feature에 추가적인 merge commit을 생성하게 됩니다.**

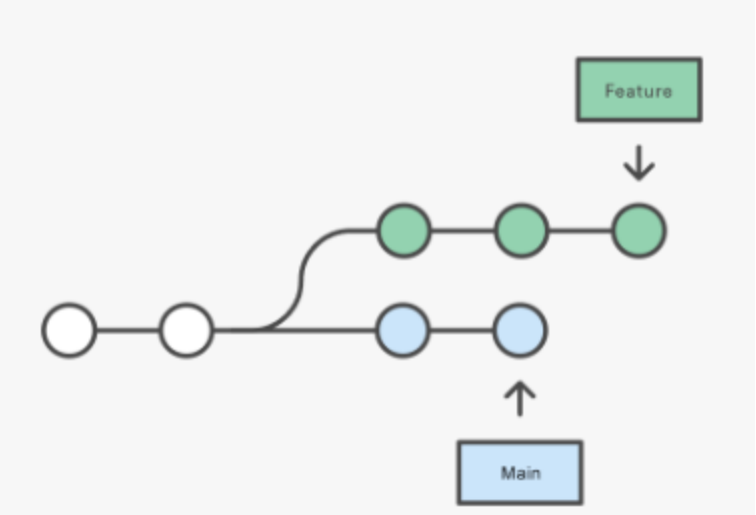


- 아래는 merge 방식 적용 코드 입니다.

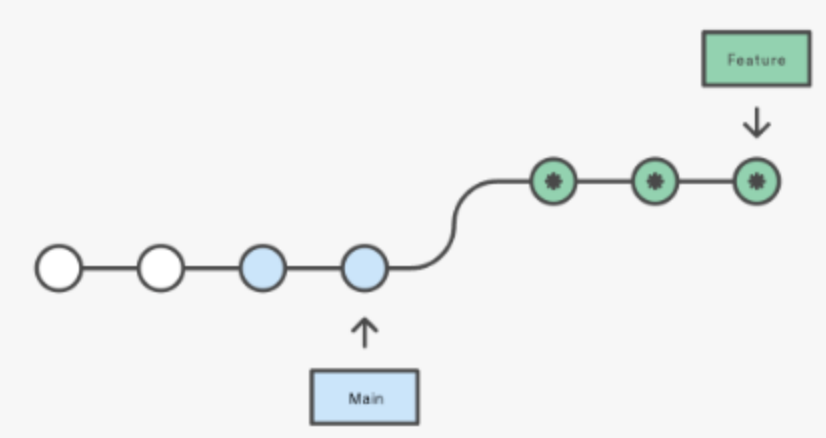
```
# 해당 repo에만 적용
git config pull.rebase false
# 모든 repo에 적용
git config --global pull.rebase false
```

rebase

- Local Feature에서 작업을 하는 도중, 다른 누군가가 Remote Main에 새로운 commit을 만들었다고 가정해봅시다.



- 이때 만약, Local Feature에서 Remote Main을 pull한다면,
- **rebase 방식의 경우, Local Feature에 추가적인 merge commit을 생성하지 않고,**
- **Remote Main commit에 Local Feature의 commit을 연결합니다.**



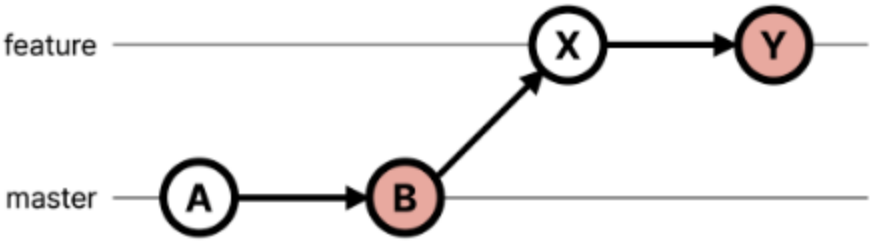
- 아래는 rebase 방식 적용 코드 입니다.

```
# 해당 repo에만 적용
git config pull.rebase true
# 모든 repo에 적용
git config --global pull.rebase true
```

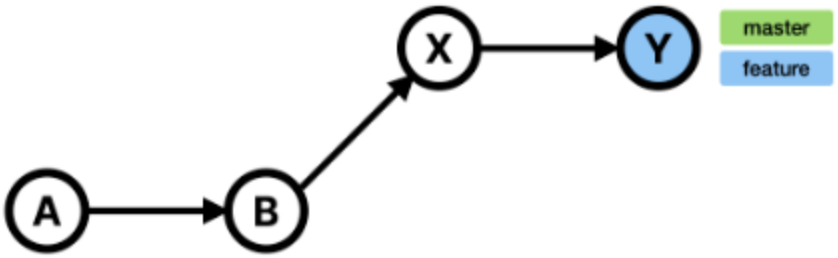
fast-forward

- **현재 브랜치랑 병합할 브랜치가 fast-forward일 경우에만 Pull이 가능한 방식입니다.**
  - **분기한 브랜치의 커밋 히스토리가 기존 브랜치의 커밋 히스토리를 포함하고 있다면 fast-forward 관계입니다.**

fast-forward 관계 (O)

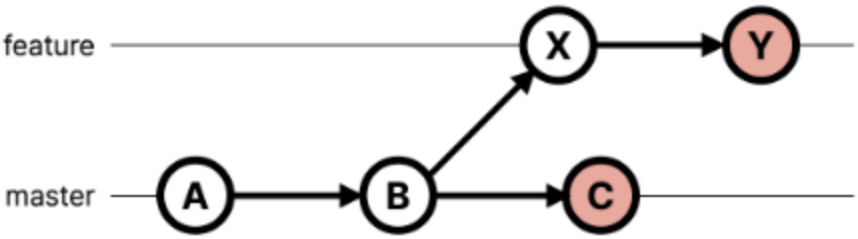


- master에서 feature로 분기된 브랜치의 커밋 히스토리는 A-B-X-Y입니다.
- 그리고 기존 브랜치인 master의 커밋 히스토리는 A-B입니다.
- 따라서, 분기된 브랜치인 feature의 히스토리는 기존 master 브랜치의 히스토리를 포함하고 있으므로, 이는 Fast-Forward 관계입니다.

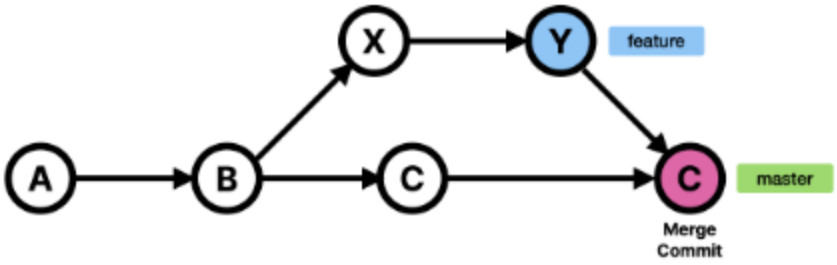


- 이런 관계에서 master 브랜치에서 git merge [feature브랜치명]을 실행한다면,
- merge에 대한 새로운 commit이 생기지 않고 HEAD의 위치만 변하게 됩니다.

fast-forward 관계 (X)



- 분기된 feature 브랜치의 커밋 히스토리는 A-B-X-Y입니다.
- 그리고 기존 브랜치인 master의 커밋 히스토리는 A-B-C입니다.
- feature의 히스토리가 master의 히스토리를 모두 담고 있지 않습니다.



- 이런 경우에, master 브랜치에서 git merge [feature브랜치명]을 실행한다면,
- merge에 대한 새로운 commit이 생기게 됩니다.
- 아래는 fast-forward 방식 적용 코드 입니다.

```
# 해당 repo에만 적용
git config pull.ff only

# 모든 repo에 적용
git config --global pull.ff only
```

결론

- **merge**
  - merge commit 생성됨
- **rebase**
  - merge commit 생성되지 않음
- **ff**
  - 현재 브랜치랑 병합할 브랜치가 fast-forward일 경우에만 Pull 허용함