



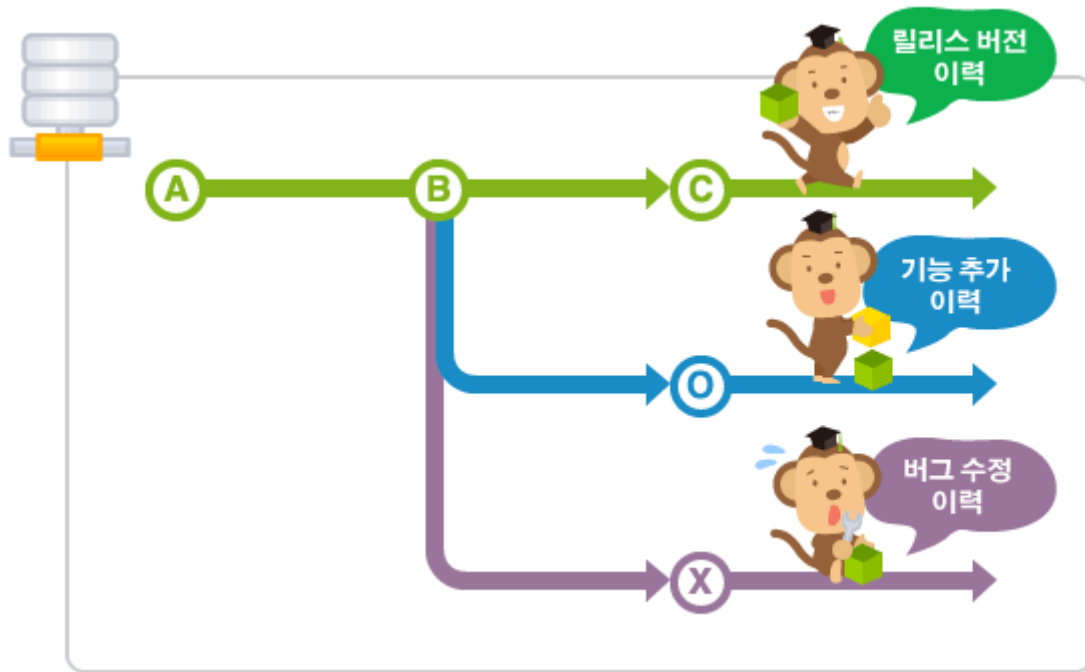
3강 - 브랜치 (branch)

브랜치란?

- 예제 1. 브랜치 목록 보기
 - 예제 2. 새 브랜치 생성하기
 - 예제 3. 하나의 브랜치 병합하기 : fast-forward merge
 - 예제 4. 여러 개의 브랜치 병합하기 : 3-way merge
 - 예제 5. Conflict : 병합 충돌 문제
-

브랜치란?

- 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 합니다.
- 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 있고 이렇게 분리된 작업 영역에서 변경된 내용은 나중에 원래의 버전과 비교해서 하나의 새로운 버전으로 만들어 낼 수 있습니다.



출처 : https://backlog.com/git-tutorial/kr/stepup/stepup1_1.html

예제 1. 브랜치 목록 보기

- 처음에 아래 명령어를 쳐보면 master라는 브랜치가 딱 하나 존재하는 걸 볼 수 있음
- 별 표시는 해당 브랜치에서 현재 작업 중이라는 것을 의미

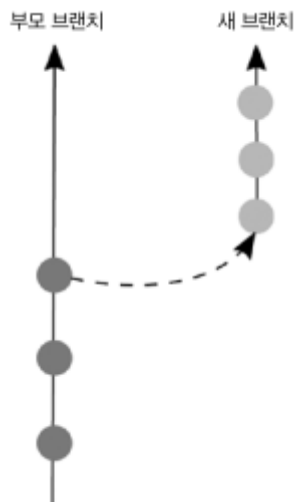
```
$ git branch --list // 기본 브랜치 목록보기 명령어
$ git branch --all  // 원격 브랜치 포함 모든 브랜치 목록 보기
$ git branch        // --list와 같음
```

예제 2. 새 브랜치 생성하기

- 하나의 커밋이 하나의 문제를 해결할 수 있는 작은 프로젝트에서는 master 브랜치에서 이것저것 버전 이력을 쌓는 것이 가능함
- 팀의 규모가 커질수록 공동 작업에 대해 일정한 구조를 갖춰 브랜치 전략을 세우는 것이 필요함
- 먼저 시작점이 될 브랜치로 이동 - switch

```
$ git switch master
```

- 새 브랜치는 부모 브랜치의 커밋에서 이어진다.



- 우리는 food.txt를 원래 흐름(master)과 분리하여 안전하게 작업할 것이다.

```
$ git branch feat/food // 새로운 브랜치 생성
$ git branch           // 브랜치 목록 확인 (master에 *이 붙음)
$ git switch feat/food // 생성된 브랜치로 이동
$ git branch           // 브랜치 목록 확인 (새 브랜치에 *이 붙음)
```

```
$ git status // 변경사항 추적 확인
$ git add food.txt
$ git commit -m "message"
$ git log --oneline --graph --all
```

- 이제 food.txt를 열어서 마구마구 수정해보자

```
$ git log --oneline --graph --all
* 2ec838e (HEAD -> feat/food) Updating food.txt what the new food want to eat
* 1bf2855 (master) Adding food.txt for the pratice1
* a831220 Adding bye.txt for the practice1
* 4eb4bd1 Adding hello.txt for greeting
```

- 현재 로그를 확인해보면 master브랜치는 여전히 이전 커밋에 위치하고 있고 새로운 브랜치는 이력을 쌓아 나가는 것을 볼 수 있음

- 여기서 master로 이동해보면 변경 사항이 적용되지 않은 것을 볼 수 있음

```
$ git switch master
$ notepad food.txt
```

- 다시 새로운 브랜치로 이동해보면 변경사항이 적용된 것을 볼 수 있다.

```
$ git switch feat/food
$ notepad food.txt
```

- 커밋을 몇 개 더 쌓으면서 master 브랜치에는 영향을 주지 않으며 신나게 작업해보자

```
$ mkdir diary // diary 폴더 생성
$ cd diary // diary 폴더로 진입
$ touch my-daily-memory.txt // 파일 생성
$ notepad my-daily-memory.txt // 내용을 작성해보자
$ cd .. // diary폴더 상위폴더로 이동
$ git add --all // 모든 변경사항 추가
$ git commit -m "message" // 버전 생성
$ git log --oneline --all --graph // 로그 확인
```

예제 3. 하나의 브랜치 병합하기 : fast-forward merge

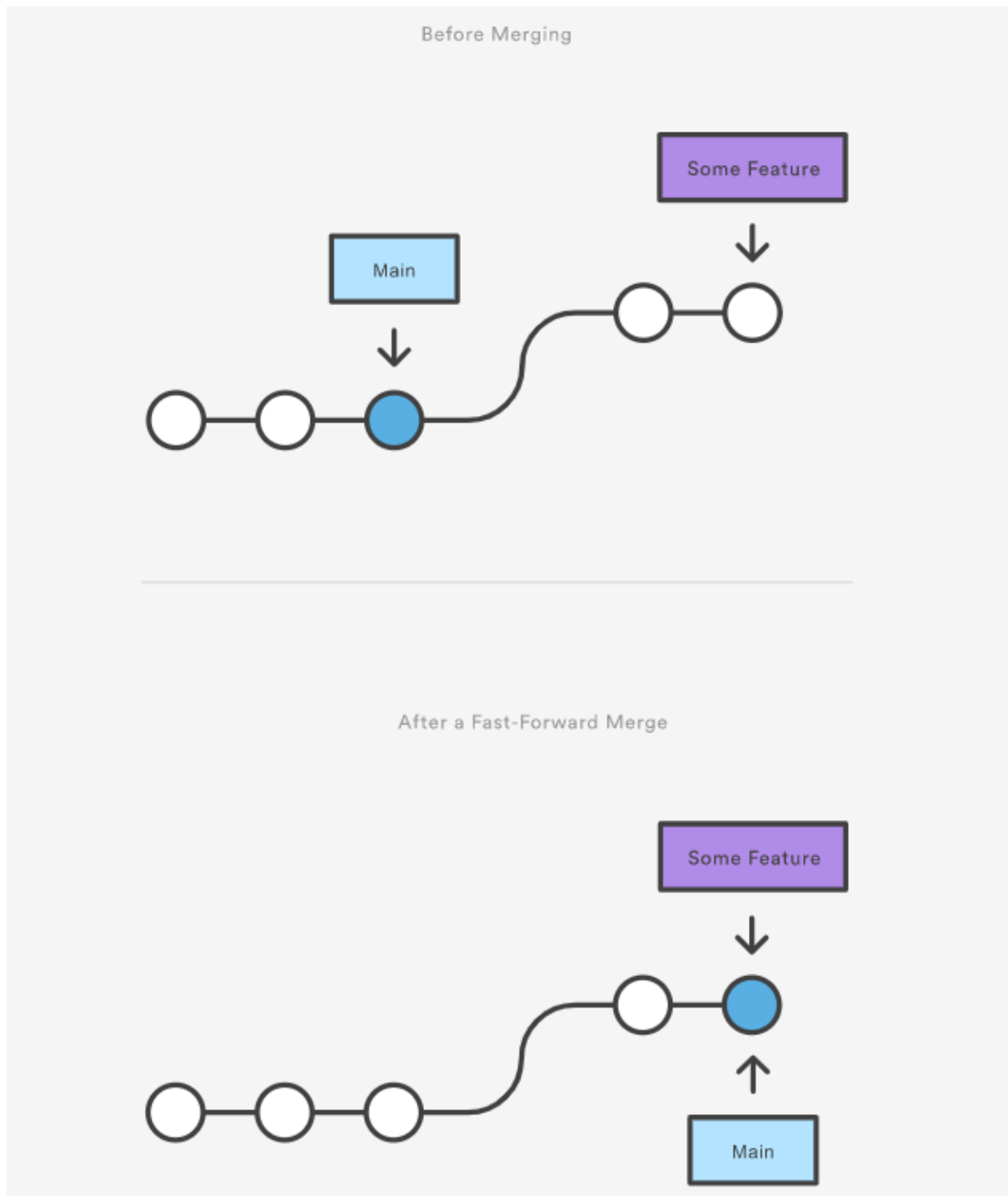
- 충분히 작업이 끝나고 검토가 완료된 브랜치 작업은 이제 메인 브랜치인 master에 합쳐 줘야 함
- 우리는 새 브랜치를 통해 메인 작업물인 master에 영향을 주지 않으며 작업할 수 있었음
- 특정 브랜치에 다른 브랜치를 합치는 작업을 merge 라고 함

```
$ git switch master      // 병합 원본으로 이동
$ git merge feat/food    // 새 브랜치를 병합
$ git log --oneline --all --graph // 로그 확인
$ git branch -d feat/food // 병합이 끝난 브랜치 삭제
```

- HEAD 포인터가 master로 이동한 것을 확인

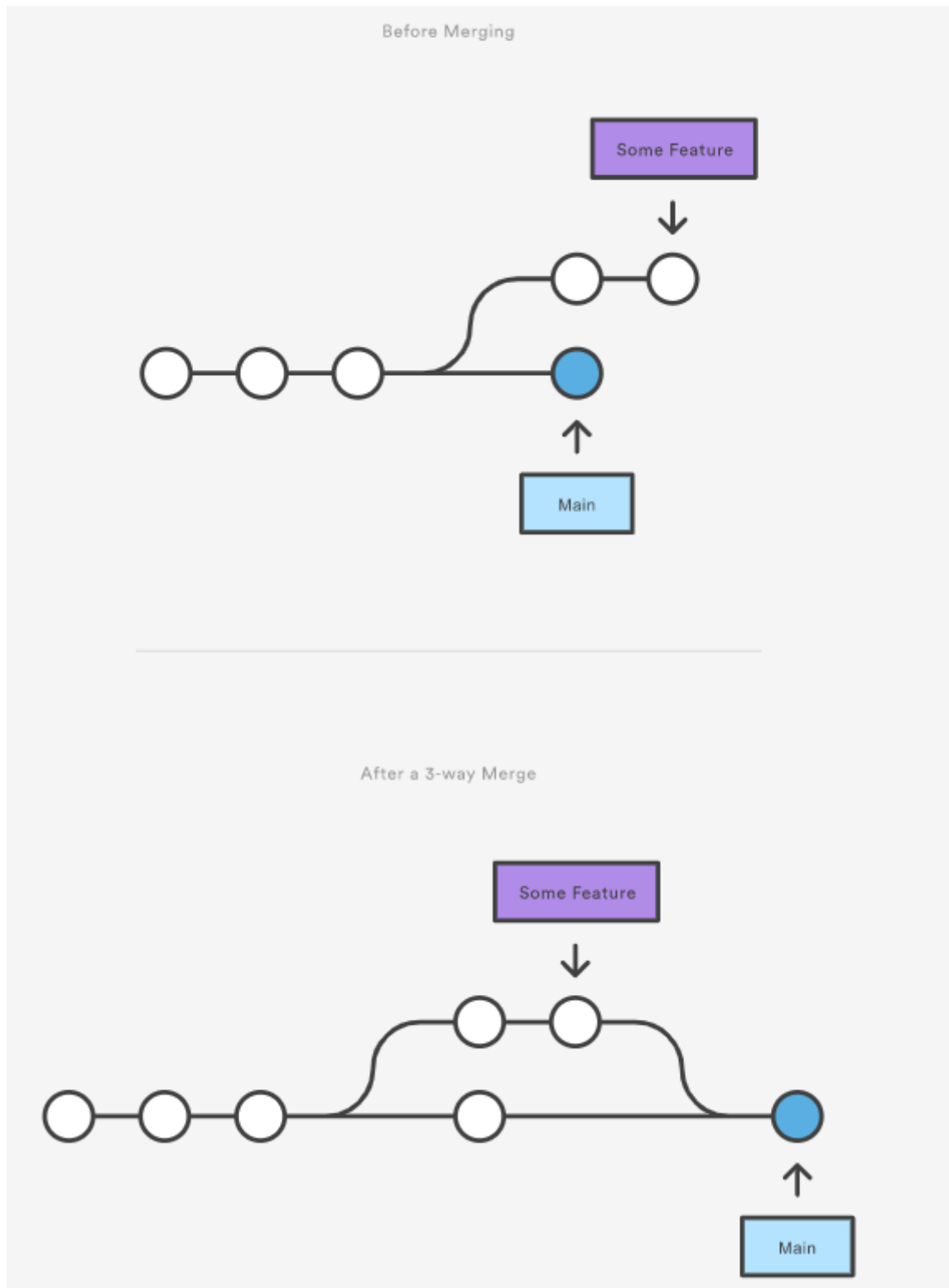
```
$ git log --oneline --all --graph
* 0224f9a (HEAD -> master, feat/food) Adding diary directory and make new
file into this directory
* 2ec838e Updating food.txt what the new food want to eat
* 1bf2855 Adding food.txt for the pratice1
* a831220 Adding bye.txt for the practice1
* 4eb4bd1 Adding hello.txt for greeting
```

- 이렇듯 이전 상태 브랜치가 병합을 통해 최신 상태로 반영되는 것을 fast-forward라고 함



예제 4. 여러 개의 브랜치 병합하기 : 3-way merge

- fast-forward 병합 전략은 단순히 이전 브랜치인 master의 위치만 새 브랜치의 커밋으로 이동하면 되지만 동시에 여러 브랜치에서 작업을 하는 경우엔 불가능함
- 이를테면 master 브랜치에서 갈라져 나온 2개의 브랜치 A, B가 있다고 가정하면 A 브랜치에서 작업을 하는 동시에 B 브랜치에서도 작업이 진행되기 때문에 A를 먼저 master에 병합할 때는 ff전략이 가능하지만 A가 병합된 이후에 B를 병합할 때는 ff전략이 불가능하다.
- 이럴 땐 A가 병합된 master 브랜치의 커밋과 B브랜치의 최종커밋을 합치면서 새로운 병합 커밋(merge commit)을 만들어 줘야 한다.



- 브랜치를 2개 생성하고 하나씩 추가 작업을 진행합니다


```
// master 브랜치에서 feat/hobby 브랜치를 만들고 이동
$ git switch -c feat/hobby master
$ notepad hobby.txt // hobby.txt를 만들고 내용을 적는다
$ git add -all
$ git commit -m "message"

// master 브랜치에서 fix/food 브랜치를 만들고 이동
$ git switch -c fix/food master
// food.txt를 수정 후 애드 커밋
$ git add -all
$ git commit -m "message"

$ git log --oneline --all --graph // 로그 확인
```

- 첫번째 feat/hobby브랜치를 master에 병합 : fast-forward 병합

```
$ git switch master // 병합 원본 브랜치로 이동
$ git merge feat/hobby // 하위 브랜치 병합
$ git log --oneline --all --graph
```

- 두번째 fix/food브랜치를 master에 병합 : 3-way 병합

```
$ git merge fix/food // 하위 브랜치 병합
// 3-way 병합은 반드시 병합 커밋을 작성해서 master의 최종커밋과 fix/food의 최종커밋을 합쳐야 함
$ git log --oneline --all --graph
$ git branch -d feat/hobby feat/food // 병합이 끝난 브랜치 삭제
```


```
$ git log --oneline --all --graph
```

```
* 884dd91 (HEAD -> master) Merge branch 'fix/food' : 병합 커밋 !
| \
| * 0b4a6f3 (fix/food) Updating food.txt
* | a933bd0 (feat/hobby) Adding hobby.txt
|/
* 0224f9a Adding diary directory and make new file into this directory
* 2ec838e Updating food.txt what the new food want to eat
* 1bf2855 Adding food.txt for the pratice1
* a831220 Adding bye.txt for the practice1
* 4eb4bd1 Adding hello.txt for greeting
```

예제 5. Conflict : 병합 충돌 문제

- 병합시 같은 라인의 코드를 작업하면 서로 충돌이 남
- 먼저 master에서 새 분기(feat/team)를 만들어 team.txt를 만들고 커밋해보자

```
$ git switch -c feat/team master // master분기에서 새 브랜치 생성 후 이동
$ notepad team.txt // team.txt 생성후 라인을 3줄 작성해보세요
```

 team.txt - Windows 메모장

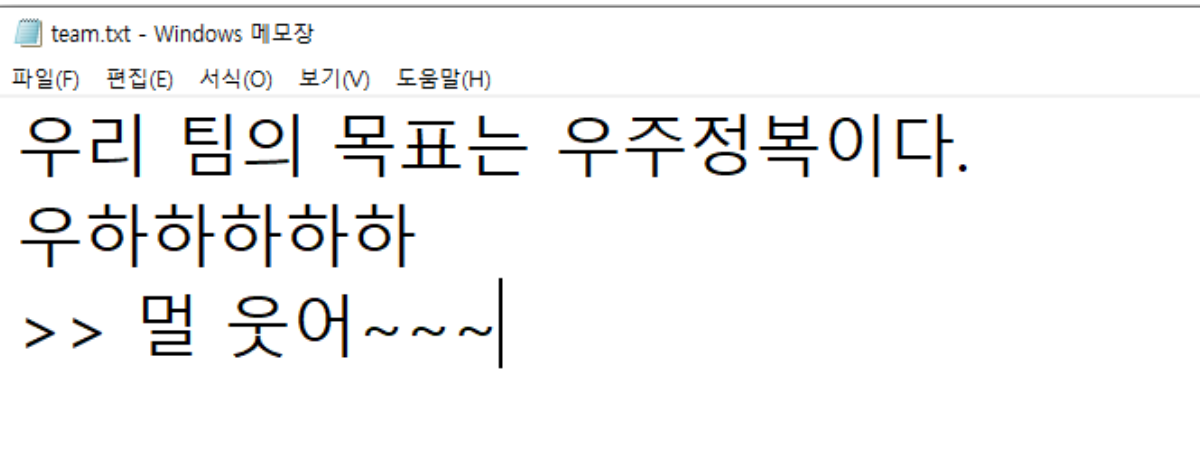
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

우리 팀의 목표는 지구정복이다.
우하하하하하
크하하하하하?!

```
$ git add --all
$ git commit -m "message"
```

- 그 다음 다시 master에서 새 브랜치(fix/team)를 만들어 team.txt를 생성하고 커밋

```
$ git switch -c fix/team master // master분기에서 새 브랜치 생성 후 이동
$ notepad team.txt // team.txt 생성후 라인을 3줄 작성해보세요
```



```
$ git add --all
$ git commit -m "message"
```

- master branch로 이동하여 feat/team브랜치와 fix/team 브랜치를 순서대로 병합해보자

```
$ git switch master
$ git merge feat/team // fast-forward merge
$ git merge fix/team // 3-way merge
```

- 여기서 master는 이제 문제상황을 발견하는데 자기브랜치에서 갈라져나간 2개의 브랜치가 서로 같은 파일을 생성하여 다른 작업을 했기 때문에 뭐가 맞는건지 몰라서 Conflict를 발생시킨다.

```
$ git merge fix/team
Auto-merging team.txt
CONFLICT (add/add): Merge conflict in team.txt
Automatic merge failed; fix conflicts and then commit the result.
```

- `\$ git status`를 입력하여 상태를 보자

```
PC - 08@DESKTOP-M0L64VA MINGW64 /e/git-practice (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both added:      team.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

- 이 때 당황하지 말고 conflict가 발생한 파일인 team.txt를 열어보자
- git에서 친절하게 충돌이 난 영역을 구분해서 보여준다

<<<<<<< HEAD

우리 팀의 목표는 지구정복이다

우하하하하하

크하하하하하?!

=====

우리 팀의 목표는 우주정복이다.

우하하하하하

>> 멀 웃어~~~

>>>>>>> fix/team|

- 수동으로 둘 중에 뭐가 맞는건지 수정해준다.
- 그리고 저장하고 닫아준다.

우리 팀의 목표는 지구정복하고 나서 우주정복이다

우하하하하하

웃는 걸로 싸우지 말자~~|

- 이제 두 커밋의 충돌을 평화롭게 해결했으니 conflict 해결 merge commit을 날려주자

```
$ git add team.txt
$ git commit -m "message"
$ git log --oneline --all --graph
```

```
$ git log --oneline --all --graph
* 09a0e3e (HEAD -> master) Merge commit with resolving conflict in team
.txt
| \
| * 12be775 (fix/team) Adding team.txt
* | 05706bc (feat/team) Adding team.txt
|/
* 884dd91 Merge branch 'fix/food' : 병합 커밋!
| \
| * 0b4a6f3 Updating food.txt
* | a933bd0 Adding hobby.txt
|/
* 0224f9a Adding diary directory and make new file into this directory
* 2ec838e Updating food.txt what the new food want to eat
* 1bf2855 Adding food.txt for the practice1
```