
Open source Software
오픈소스 소프트웨어
24. 병합 충돌과 해결

학습 개요

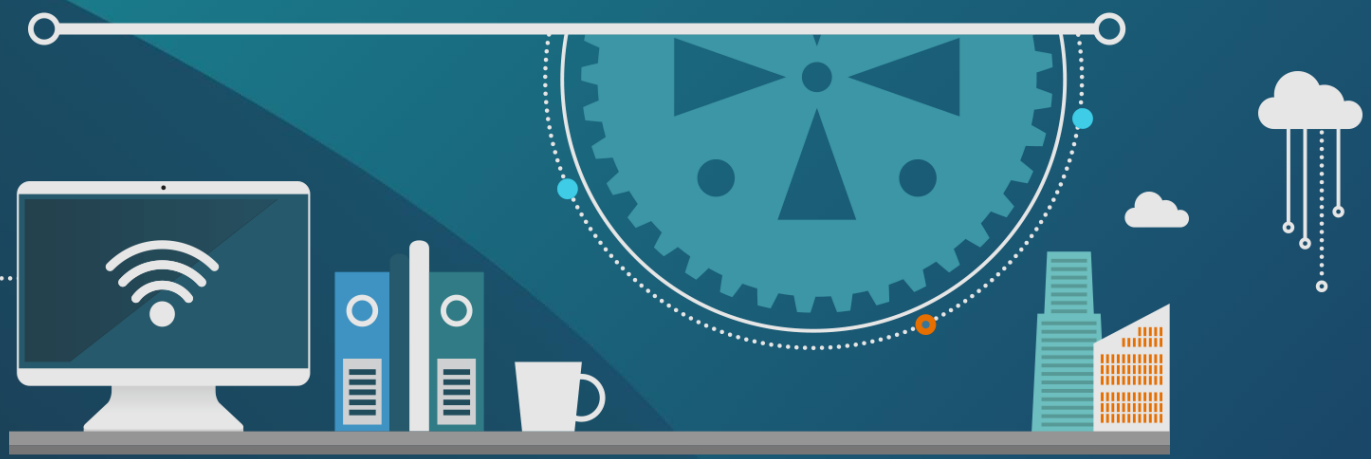
1. 병합 충돌 발생과 이해
2. 충돌 발생한 코드의 수정
3. 수정한 파일 add, commit

학습 목표

1. 3-way 병합에서 충돌이 발생한 것을 인지하고 발생 사유를 이해할 수 있다.
2. 충돌이 발생한 파일 내부의 표기 방식을 이해하고 코드를 수정할 수 있다.
3. 충돌을 완전히 해결하기 위해 코드 수정 이후 add 후 commit해 3-way 병합을 수행할 수 있다.

LESSON 01

병합 충돌



1 병합 충돌

병합에서 충돌 발생 표기

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git merge feat/list
Auto-merging basic.py
CONFLICT (content): Merge conflict in basic.py
Automatic merge failed; fix conflicts and then commit the result.

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main|MERGING)
$ git lo
2ecae57 (HEAD -> main) add divmod, main
56361e0 Add print list of range, feat/list
5be70c1 Add print list, feat/list
834e72e Add print literals, main
fab1006 Create basic.py, main

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main|MERGING)
$ git st
On branch main
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 modified:   basic.py

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

충돌 발생 의미

1 병합 충돌

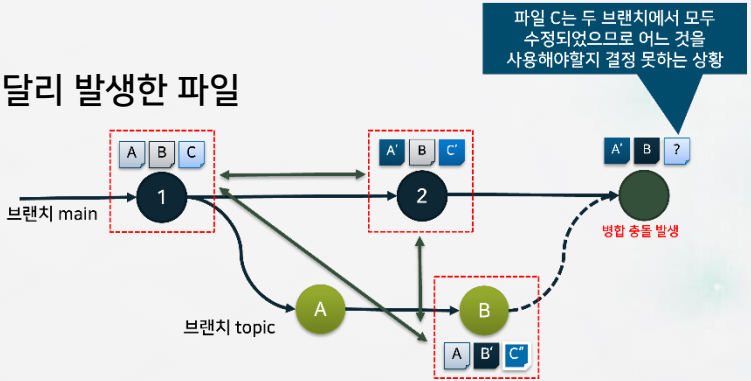
병합 충돌(conflict) 이해

3-way 상태에서 두 브랜치의 동일 조상인 커밋 1을 기준

- 병합할 두 브랜치 마지막 커밋을 비교

충돌의 기준

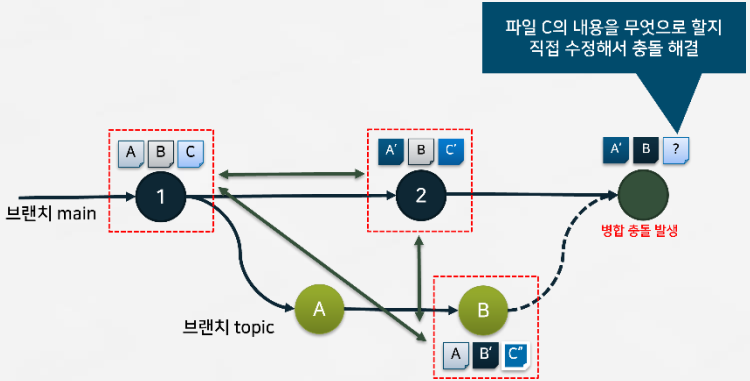
- 파일 충돌 없음
  - 수정되지 않거나 한쪽 브랜치에서만 수정되면
    - ➡ 파일 A는 브랜치 main에서만 수정
    - ➡ 파일 B는 브랜치 topic에서만 수정
- 파일 충돌 발생
  - 두 브랜치 모두에서 변경 사항이 달리 발생한 파일
    - ➡ 파일 C는 양쪽에서 수정



1 병합 충돌

충돌 해결

- 충돌이 발생하면 해당 파일의 충돌을 먼저 해결
  - 충돌 해결
    - 직접 파일 내용을 수정 후 저장
      - ➡ 충돌이 발생한 파일 C를 직접 편집
      - ➡ C′ 또는 C″ 또는 아예 다르게
    - 계속해서 add, commit 진행
    - 필요하면 합병된 이전 브랜치 삭제



1 병합 충돌

충돌 발생 코드 표시

충돌한 파일 내부 표시

- 3개의 표시로 구분
  - <<<<<<< HEAD
  - 현재 브랜치 HEAD의 수정 내용
  - ==========
  - 병합되는 브랜치 feat/list의 수정 내용
  - >>>>>>> feat/list

수정 후에는 노란색 바탕 3개의 표시는 모두 삭제하도록

```
print('branch basic')
print(1, 2, 3)
print([1, 2, 3])
<<<<<<< HEAD
print(list(range(1, 11)))    # 4번째 커밋

a, b = divmod(20, 3)         # 5번째 커밋
=======
print(list(range(1, 11)))      # 4번째 커밋

print(tuple(range(1, 11, 2))) # 5번째 커밋
>>>>>>> feat/list
```

1 병합 충돌

충돌 발생 vscode 내부 표시

충돌한 파일을 vscode로 열기

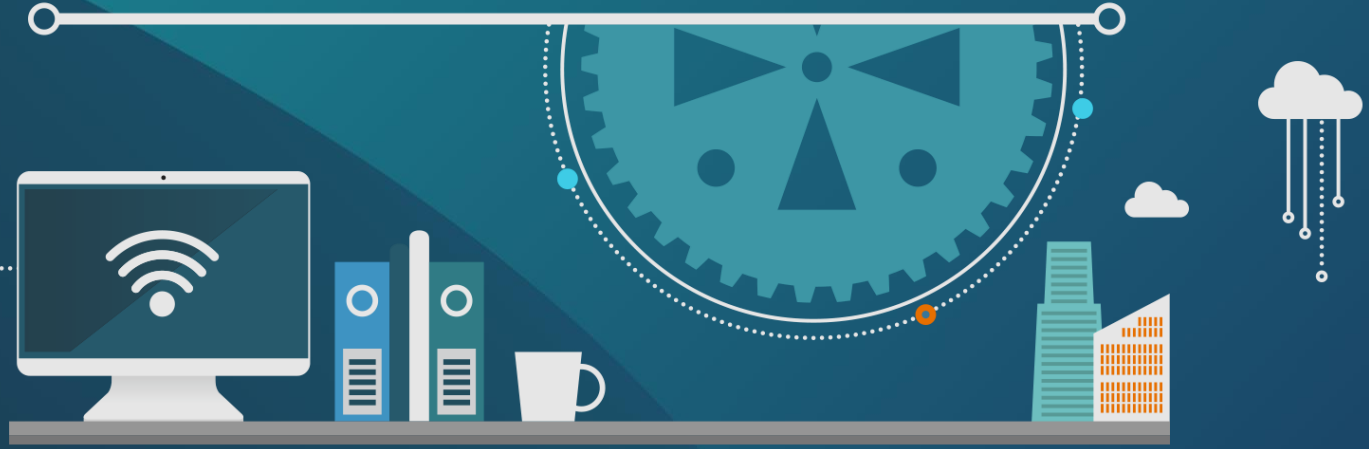
- 색상과 함께 3개의 표시로 구분
  - <<<<<<< HEAD
  - 현재 브랜치 HEAD의 수정 내용
  - ==========
  - 병합되는 브랜치 feat/list의 수정 내용
  - >>>>>>> feat/list

Vscode에서 충돌 해결 편의를 위해 표시되는 메뉴

```
print('branch basic')
print(1, 2, 3)
print([1, 2, 3])
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< HEAD (Current Change)
print(list(range(1, 11)))   # 4번째 커밋

a, b = divmod(20, 3)        # 5번째 커밋
=======
print(list(range(1, 11)))   # 4번째 커밋

print(tuple(range(1, 11, 2)))   # 5번째 커밋
>>>>>>> feat/list (Incoming Change)

```

LESSON 02

충돌 해결



2 충돌해결

병합 취소 후 다시 병합

병합 취소

- $ git merge --abort

다시 병합

- $ git merge feat/list

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main|MERGING)
$ git merge --abort

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git merge feat/list
Auto-merging basic.py
CONFLICT (content): Merge conflict in basic.py
Automatic merge failed; fix conflicts and then commit the result.
```

2 충돌해결

충돌 해결

직접 수정 후 저장

- 충돌 표시 모두 제거

```
print('branch basic')
print(1, 2, 3)
print([1, 2, 3])
print(list(range(1, 11)))          # 4번째 커밋

a, b = divmod(20, 3)               # 5번째 커밋
print(a, b)                        # 5번째 커밋
print(tuple(range(1, 11, 2)))      # 충돌 해결
```

추가, 커밋 다시

- $ git commit -am 'Resolve conflict, main'

2 충돌해결

충돌 발생 시 상태

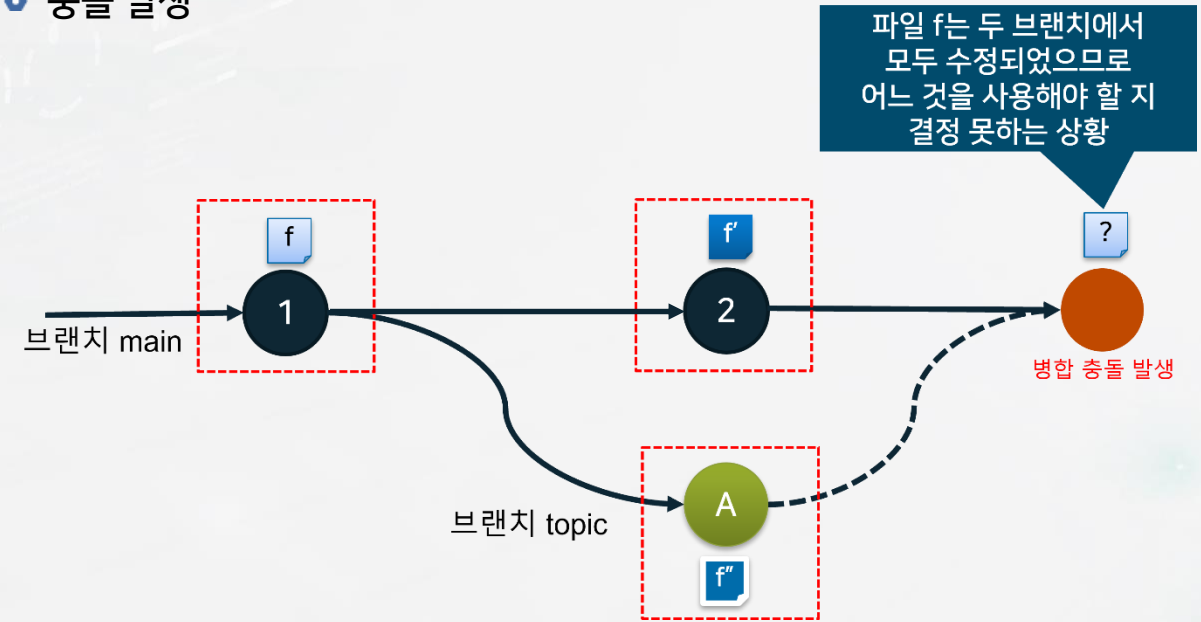
충돌 후 커밋

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main|MERGING)
$ git st
On branch main
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 modified:   basic.py

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

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main|MERGING)
$ git commit -am 'Resolve conflict, main'
[main c1d4ca6] Resolve conflict, main
```

2 충돌해결

3-way 병합 수행

- 파일 f를 두 브랜치에서 동시에 수정
  - 충돌 발생



2 충돌해결

병합 전까지 수행

순서	절차	명령
❶	저장소 mconf 생성 후 이동	`$ git init mconf` `$ cd mconf`

❷	파일 f 생성 후 커밋	`$ echo 111 > f`
		`$ git add f`
		`$ git commit -m 1`

❸	브랜치 topic 생성 후 이동	`$ git switch -c topic` `Switched to a new branch 'topic'`
❹	파일 f 수정 후 커밋	`$ echo aaa >> f` `$ git commit -am A`
❺	브랜치 main으로 이동	`$ git checkout main` `Switched to branch 'main'`
❻	파일 f 수정 후 커밋	`$ echo 222 >> f` `$ git commit -am 2`

❼	현재 모든 브랜치 로그 이력 보기	`$ git log --graph –oneline --all`
		`* 004b455 (HEAD -> main) 2`
		`| * 003ee3c (topic) A`
		`|/`
		`* 96d9857 1`

2 충돌해결

병합 시 충돌 발생

충돌 후 코딩

순서	절차	명령

❽	병합 실행해 충돌 발생	`$ git merge topic`
		`Auto-merging f`
		`CONFLICT (content): Merge conflict in f`
		`Automatic merge failed; fix conflicts and then commit the r esult.`

❾	코드 수정	`$ code f`

C: > [smart git] > mconf > f

```
1   111
    Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
2   <<<<<<< HEAD (Current Change)
3   222
4   =======
5   aaa
6   >>>>>>> topic (Incoming Change)
7
```

2 충돌해결

원하는 코드로 수정

- 모든 소스로 구성

f

C: > [smart git] > mconf > f

```
1   111
2   222
3   aaa
```

2 충돌해결

코드 수정 후 반드시 커밋이 필요

커밋 수행

순서	절차	명령
⑩	충돌 해결을 위한 커밋	`$ git commit -am 2A`

⑪	병합 후 로그 이력 그래프	`$ git log --graph --oneline --all`
		`* 4e1c6f5 (HEAD -> main) 2A`
		`|\`
		`| * 003ee3c (topic) A`
		`* | 004b455 2`
		`|/`
		`* 96d9857 1`

⑫	병합된 파일 확인	`$ cat f`
		`111`
		`222`
		`aaa`

Summary

3-way 충돌 발생

- $ git merge hotfix

충돌한 파일을 인지하고 파일 수정

- $ code file

수정 후 다시 add, commit

- $ git commit -am 'msg'

충돌 이후 병합 취소

- $ git merge --abort